**PRACTICAL NO. :1**

**Aim :** Implement Caesar cipher encryption/decryption.

**Theory:**

The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It’s simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter so- me fixed number of positions down the alphabet. For example with a shift of 1, A would be rep- laced by B, B would become C, and so on.

The encryption can be represented using modular arithmetic by first transforming the letters int- o to numbers, according to the scheme, A = 0, B = 1,…, Z = 25. Encryption of a letter by a shift n can be described mathematically as.

E=(X+K) mod 26

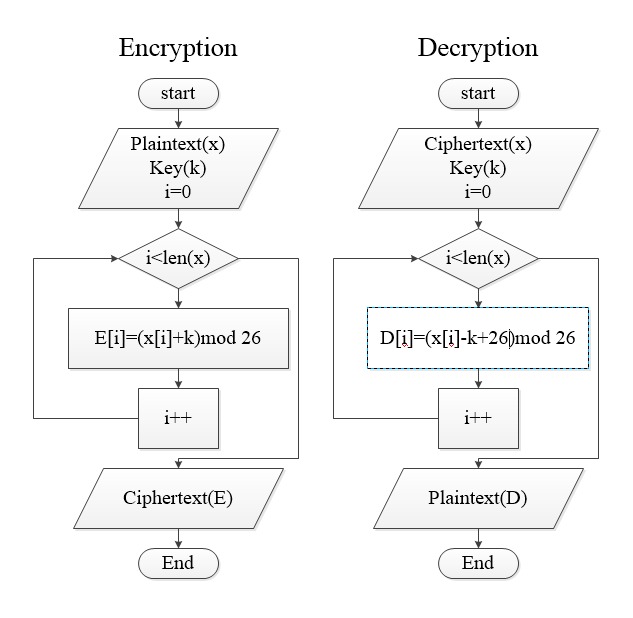
(For encryption)

D=((Y-K)+26) mod 26

(For decryption.)

Where, X=Plain text ,K=Key,Y=Cipher text

**Flowchart:**

****

**PROGRAM:**

def encrypt(msg, s):

result = ""

for str in msg.split():

for i in str:

if (i.isupper()):

result += chr(((ord(i)+s-65) % 26)+65)

else:

result += chr(((ord(i)+s-97) % 26)+97)

result+=" "

return result

def deycrypt(msg, s):

result = ""

for str in msg.split():

for i in msg:

if (i.isupper()):

result += chr((((ord(i)-s-65)+26) % 26)+65)

else:

result += chr((((ord(i)-s-97)+26) % 26)+97)

result=" "

return result

print("darshit")

print("\_\_\_\_\_\_\_\_ Caesar cipher\_\_\_\_\_\_\_\_")

choice=int(input("Enter choice\n1:Encryption\n2:Decryption\n"))

if (choice==1):

s\_msg = input("Enter plain text:-")

print("\nencrypted message is {}".format(encrypt(s\_msg,int(input("enter swap value ")))))

elif(choice==2):

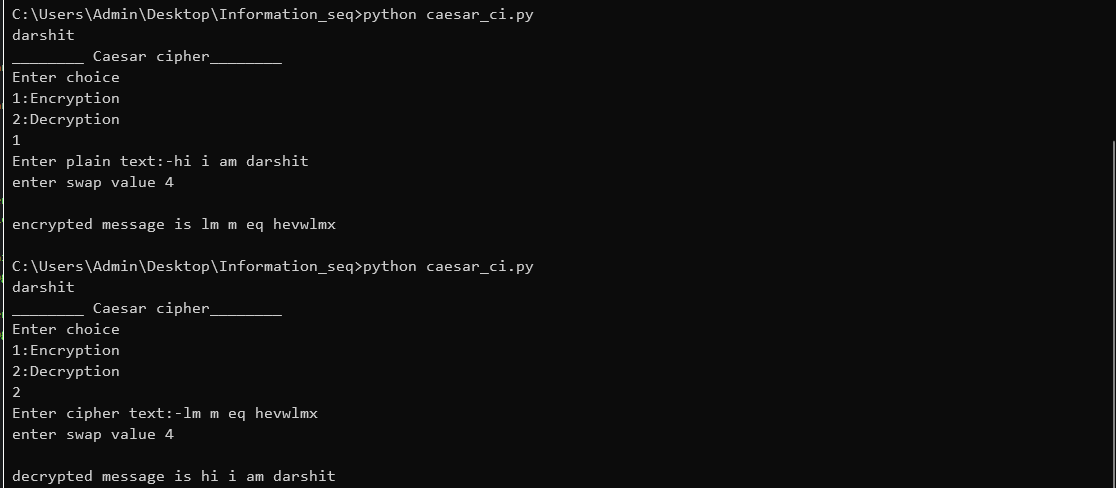
s\_msg=input("Enter cipher text:-")

print("\ndecrypted message is {}".format(deycrypt(s\_msg, int(input("enter swap value ")))))

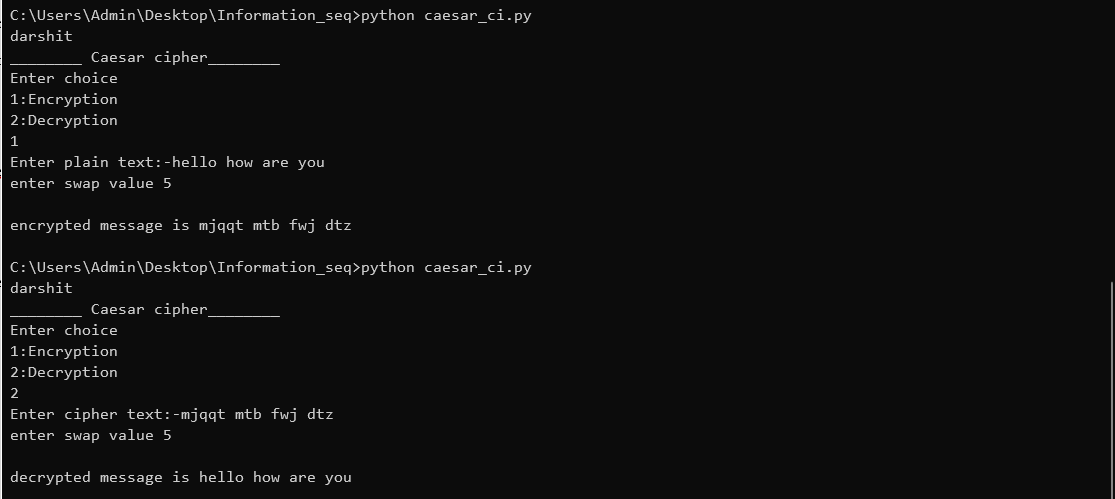
else:

print("Invalid choice")

**OUTPUT:**



**Problem Solving:**



|  |  |
| --- | --- |
| Performance |  |
| Lab problem |  |
| Context solving |  |
| **Total** |  |

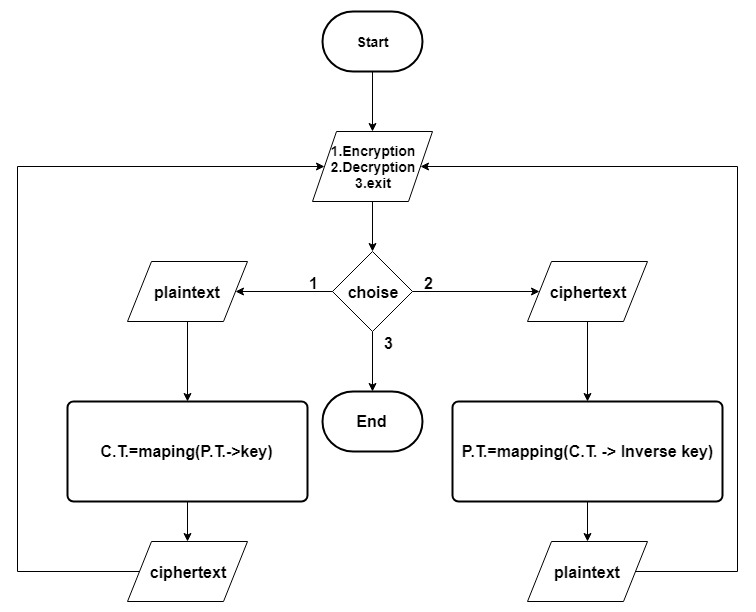
**PRACTICAL NO. :2**

**Aim :** Implement monoalphabetic cipher encryption/decryption.

**Theory:**

Monoalphabetic cipher is a substitution cipher in which for a given key, the cipher alphabet for each plain alphabet is fixed throughout the encryption process. For example, if ‘A’ is encrypted as ‘D’, for any number of occurrence in that plaintext, ‘A’ will always get encrypted to ‘D’.

**Flowchart:**

****

**PROGRAM:**

def encrypt(msg,mono\_dict):

result=""

for i in msg:

result+=mono\_dict[i]

return result

def decrypt(msg,mono\_dict\_invers):

result=""

for i in msg:

result+=mono\_dict\_invers[i]

return result

print("Darshit Pipariya")

print("\_\_\_\_\_\_\_\_ Monoalphabetic cipher\_\_\_\_\_\_\_\_")

mono\_dict={'0':'A','1':'D','2':'C','3':'B','4':'E','5':'G','6':'H','7':'F','8':'I','9':'K','a':'J','b':'N','c':'O','d':'M','e':'L','f':'R','g':'Q','h':'P','i':'U','j':'S','k':'T','l':'X','m':'Z','n':'Y','o':'V','p':'W','q':'0','r':'2','s':'1','t':'3','u':'4','v':'7','w':'5','x':'9','y':'8','z':'6','A':'x','B':'z','C':'y','D':'w','E':'v','F':'q','G':'s','H':'r','I':'t','J':'u','K':'p','L':'a','M':'c','N':'d','O':'b','P':'e','Q':'h','R':'g','S':'f','T':'j','U':'i','V':'k','W':'n','X':'m','Y':'l','Z':'o',' ':' '}

mono\_dict\_invers=dict([(value, key) for key, value in mono\_dict.items()])

while(True):

choice=int(input("Enter choice\n1:Encryption\n2:Decryption\n3:Exit\n"))

if (choice==1):

s\_msg = input("Enter plain text:-")

print("\nencrypted message is {}".format(encrypt(s\_msg,mono\_dict)))

elif(choice==2):

s\_msg=input("Enter cipher text:-")

print("\ndecrypted message is {}".format(decrypt(s\_msg,mono\_dict\_invers)))

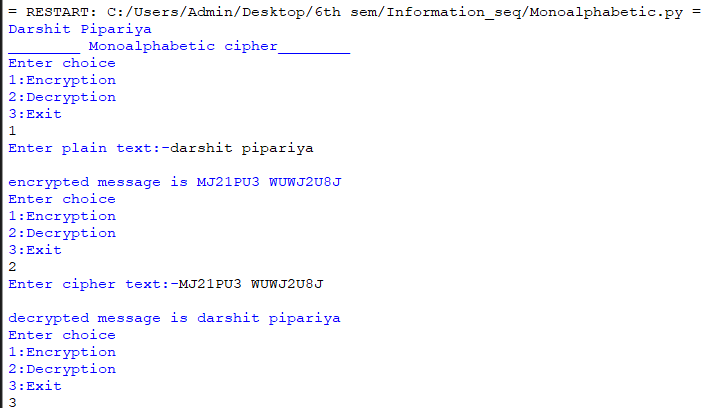
elif(choice==3):

break

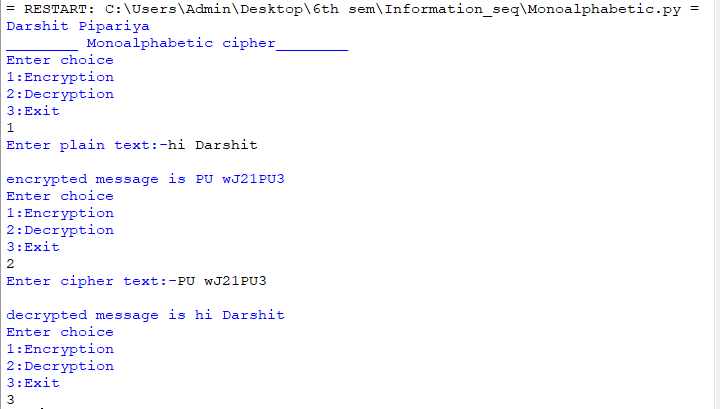
else:

print("Invalid choice")

**Output:**



**Problem Solving:**



|  |  |
| --- | --- |
| Performance |  |
| Lab problem |  |
| Context solving |  |
| **Total** |  |

**PRACTICAL NO. :3**

**Aim :** Implement playfair cipher encryption-decryption.

**Theory:**

The Playfair cipher uses a 5 by 5 table containing a [key word or phrase](https://en.wikipedia.org/wiki/Key_(cryptography)). Memorization of the keyword and 4 simple rules was all that was required to create the 5 by 5 table and use the cipher.

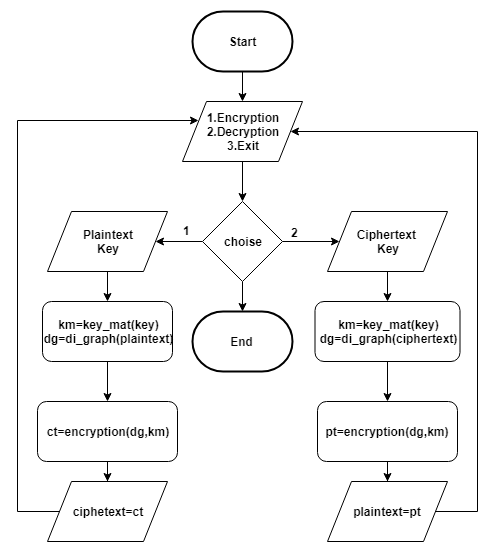
To generate the key table, one would first fill in the spaces in the table (a modified [Polybius square](https://en.wikipedia.org/wiki/Polybius_square)) with the letters of the keyword (dropping any duplicate letters), then fill the remaining spaces with the rest of the letters of the alphabet in order (usually omitting "J" or "Q" to reduce the alphabet to fit; other versions put both "I" and "J" in the same space). The key can be written in the top rows of the table, from left to right, or in some other pattern, such as a spiral beginning in the upper-left-hand corner and ending in the center. The keyword together with the conventions for filling in the 5 by 5 table constitute the cipher key.

To encrypt a message, one would break the message into digrams (groups of 2 letters) such that, for example, "HelloWorld" becomes "HE LL OW OR LD". These digrams will be substituted using the key table. Since encryption requires pairs of letters, messages with an odd number of characters usually append an uncommon letter, such as "X", to complete the final digram. The two letters of the digram are considered opposite corners of a rectangle in the key table. To perform the substitution, apply the following 4 rules, in order, to each pair of letters in the plaintext:

1. If both letters are the same (or only one letter is left), add an "X" after the first letter. Encrypt the new pair and continue. Some variants of Playfair use "Q" instead of "X", but any letter, itself uncommon as a repeated pair, will do.
2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively (wrapping around to the left side of the row if a letter in the original pair was on the right side of the row).
3. If the letters appear on the same column of your table, replace them with the letters immediately below respectively (wrapping around to the top side of the column if a letter in the original pair was on the bottom side of the column).
4. If the letters are not on the same row or column, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair. The order is important – the first letter of the encrypted pair is the one that lies on the same **row** as the first letter of the plaintext pair.

To decrypt, use the *inverse* (opposite) of the last 3 rules, and the first as-is (dropping any extra "X"s or "Q"s that do not make sense in the final message when finished).

**Flowchart:**



**PROGRAM:**

def key\_genarator(key):

    alphabet='abcdefghiklmnopqrstuvwxyz'

    key=key.replace(" ","")

    table=[]

    for char in key.lower():

        if char not in table:

            if char=='j':

                char='i'

            table.append(char)

    for char in alphabet:

        if char not in table:

            table.append(char)

    j=0

    final=[]

    for i in range(5):

        l=[]

        for k in range(5):

            if(j<len(table)):

                l.append(table[j])

                j+=1

        final.append(l)

    return final

def to\_diagraph(text):

    table=[]

    text=text.replace(" ","")

    for i in range(len(text)-1):

        table.append(text[i])

        if(text[i]==text[i+1]):

            if(text[i]=='x'):

                table.append('y')

            else:

                table.append('x')

    table.append(text[-1])

    if(len(table)%2!=0):

        if(text[-1]=='x'):

                table.append('y')

        else:

                table.append('x')

    l1=[]

    for i in range(0,len(table),2):

        l1.append([table[i],table[i+1]])

    return l1

def encryption():

    plaintext=input("Enter plaintext: " )

    key=input("Enter key: ")

    diagraph=to\_diagraph(plaintext)

    key\_matrix=key\_genarator(key)

    cipher=[]

    #find co-ordinates of dia-graph in key metrix

    for d in diagraph:

        e1,e2=d[0],d[1]

        for i in range(len(key\_matrix)):

            if(e1 in key\_matrix[i]):

                j=key\_matrix[i].index(e1)

                e1\_x,e1\_y=i,j

            if(e2 in key\_matrix[i]):

                j=key\_matrix[i].index(e2)

                e2\_x,e2\_y=i,j

        if(e1\_x==e2\_x):

            # if row is same

            e1=key\_matrix[e1\_x][(e1\_y+1)%5]

            e2=key\_matrix[e2\_x][(e2\_y+1)%5]

        elif(e1\_y==e2\_y):

            # if column is same

            e1=key\_matrix[(e1\_x+1)%5][e1\_y]

            e2=key\_matrix[(e2\_x+1)%5][e2\_y]

        else:

            e1=key\_matrix[e1\_x][e2\_y]

            e2=key\_matrix[e2\_x][e1\_y]

        cipher.append(e1)

        cipher.append(e2)

    cipher="".join( i for i in cipher)

    print("Ciphertext:-{}".format(cipher))

def decryption():

    ciphertext=input("Enter ciphertext: ")

    key=input("Enter key:")

    diagraph=to\_diagraph(ciphertext)

    key\_matrix=key\_genarator(key)

    plaintext=[]

    #find co-ordinates of dia-graph in key metrix

    for d in diagraph:

        e1,e2=d[0],d[1]

        for i in range(len(key\_matrix)):

            if(e1 in key\_matrix[i]):

                j=key\_matrix[i].index(e1)

                e1\_x,e1\_y=i,j

            if(e2 in key\_matrix[i]):

                j=key\_matrix[i].index(e2)

                e2\_x,e2\_y=i,j

        if(e1\_x==e2\_x):

            # if row is same

            e1=key\_matrix[e1\_x][(e1\_y-1)%5]

            e2=key\_matrix[e2\_x][(e2\_y-1)%5]

        elif(e1\_y==e2\_y):

            # if column is same

            e1=key\_matrix[(e1\_x-1)%5][e1\_y]

            e2=key\_matrix[(e2\_x-1)%5][e2\_y]

        else:

            e1=key\_matrix[e1\_x][e2\_y]

            e2=key\_matrix[e2\_x][e1\_y]

        plaintext.append(e1)

        plaintext.append(e2)

    plaintext="".join( i for i in plaintext)

    plaintext=plaintext.replace('x','')

    print("Plaintext:- {} ".format(plaintext))

if \_\_name\_\_ == "\_\_main\_\_":

    print("\_\_\_\_\_\_\_\_ Playfair cipher\_\_\_\_\_\_\_\_")

    while True:

        choice=int(input("Enter choice\n1:Encryption\n2:Decryption\n3:Exit\n"))

        if (choice==1):

            encryption()

        elif(choice==2):

            decryption()

        elif(choice==3):

            break

        else:

            print("Invalid choice")

**Output:**



**Problem Solving:**



|  |  |
| --- | --- |
| Performance |  |
| Lab problem |  |
| Context solving |  |
| **Total** |  |

**PRACTICAL NO. :4**

**Aim :** Implement polyalphabetic cipher encryption-decryption.

**Theory:**

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of [polyalphabetic substitution](https://en.wikipedia.org/wiki/Polyalphabetic_cipher). A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets .The encryption of the original text is done using the *[Vigenère square or Vigenère table](https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher" \l "/media/File:Vigen%C3%A8re_square_shading.svg)*.

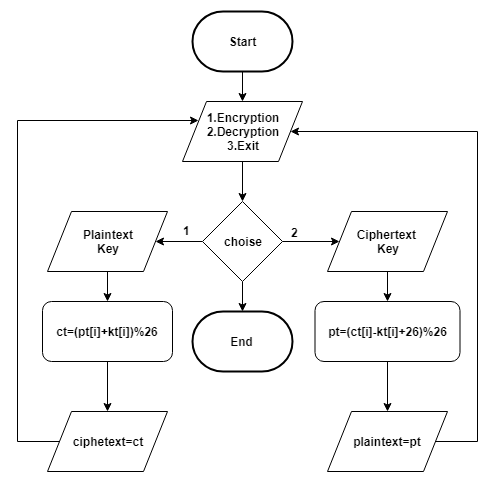
* The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible[Caesar Ciphers](https://www.geeksforgeeks.org/caesar-cipher/).
* At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
* The alphabet used at each point depends on a repeating keyword.

**Encryption**  
The first letter of the plaintext, G is paired with A, the first letter of the key. So use row G and column A of the Vigenère square, namely G. Similarly, for the second letter of the plaintext, the second letter of the key is used, the letter at row E and column Y is C. The rest of the plaintext is enciphered in a similar fashion

**Decryption**

Decryption performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letter in this row, and then using the column’s label as the plaintext. For example, in row A (from AYUSH), the ciphertext G appears in column G, which is the first plaintext letter. Next we go to row Y (from AYUSH), locate the ciphertext C which is found in column E, thus E is the second plaintext letter.

**Flowchart:**

****

**PROGRAM:**

def Encryption():

plaintext = input("Enter plaintext: ")

key = input("Enter key: ")

ciphertext = ""

pt = plaintext.replace(' ', '')

k = key.replace(' ', '')

lower\_alpha = "abcdefghijklmnopqrstuvwxyz"

upper\_alpha = lower\_alpha.upper()

for i in range(len(pt)):

if(pt[i].isupper() and k[i % len(k)].isupper()):

ciphertext += chr((upper\_alpha.index(pt[i]) +

upper\_alpha.index(k[i % len(k)])) % (26)+65)

if(pt[i].isupper() and k[i % len(k)].islower()):

ciphertext += chr((upper\_alpha.index(pt[i]) +

lower\_alpha.index(k[i % len(k)])) % (26)+65)

if(pt[i].islower() and k[i % len(k)].islower()):

ciphertext += chr((lower\_alpha.index(pt[i]) +

lower\_alpha.index(k[i % len(k)])) % (26)+97)

if(pt[i].islower() and k[i % len(k)].isupper()):

ciphertext += chr((lower\_alpha.index(pt[i]) +

upper\_alpha.index(k[i % len(k)])) % (26)+97)

print(ciphertext)

def Decryption():

ciphertext = input("Enter ciphertext: ")

key = input("Enter key: ")

plaintext = ""

ct = ciphertext.replace(' ', '')

k = key.replace(' ', '')

lower\_alpha = "abcdefghijklmnopqrstuvwxyz"

upper\_alpha = lower\_alpha.upper()

for i in range(len(ct)):

if(ct[i].isupper() and k[i % len(k)].isupper()):

plaintext += chr((upper\_alpha.index(ct[i]) -

upper\_alpha.index(k[i % len(k)])) % (26)+65)

if(ct[i].isupper() and k[i % len(k)].islower()):

plaintext += chr((upper\_alpha.index(ct[i]) -

lower\_alpha.index(k[i % len(k)])) % (26)+65)

if(ct[i].islower() and k[i % len(k)].islower()):

plaintext += chr((lower\_alpha.index(ct[i]) -

lower\_alpha.index(k[i % len(k)])) % (26)+97)

if(ct[i].islower() and k[i % len(k)].isupper()):

plaintext += chr((lower\_alpha.index(ct[i]) -

upper\_alpha.index(k[i % len(k)])) % (26)+97)

print(plaintext)

if \_\_name\_\_ == "\_\_main\_\_":

print("\_\_\_\_\_\_\_\_ Polyalphabetic cipher\_\_\_\_\_\_\_\_")

while True:

choice = int(

input("Enter choice\n1:Encryption\n2:Decryption\n3:Exit\n"))

if (choice == 1):

Encryption()

elif(choice == 2):

Decryption()

elif(choice == 3):

break

else:

print("Invalid choice")

**OUTPUT:**



**Problem Solving:**



|  |  |
| --- | --- |
| Performance |  |
| Lab problem |  |
| Context solving |  |
| **Total** |  |

**PRACTICAL NO. :5**

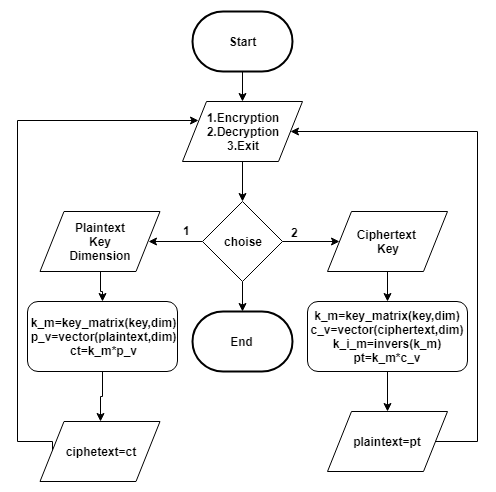
**Aim :** Implement hill cipher encryption-decryption.

**Theory:**

Hill cipher is a polygraphic substitution cipher based on linear algebra.Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26)

**Flowchart:**

****

**PROGRAM:**

import numpy as np

def message(msg, dim):

    ms = []

    msg = msg.replace(' ', '')

    ind = 0

    lower\_alpha = "abcdefghijklmnopqrstuvwxyz"

    upper\_alpha = lower\_alpha.upper()

    for index in range(0, len(msg), dim):

        messageVector = [[''] for i in range(dim)]

        for j in range(dim):

            if(index < len(msg)):

                if(msg[index].isupper()):

                    messageVector[j][0] = upper\_alpha.index(msg[index])

                    index += 1

                else:

                    messageVector[j][0] = lower\_alpha.index(msg[index])

                    index += 1

            else:

                messageVector[j][0] = ind

                ind += 1

                index += 1

        ms.append(messageVector)

    return ms

def key\_matrix(key, dim):

    k\_matrix = []

    k = key.replace(' ', '')

    lower\_alpha = "abcdefghijklmnopqrstuvwxyz"

    upper\_alpha = lower\_alpha.upper()

    index = 0

    ind = 0

    for i in range(dim):

        l = []

        for j in range(dim):

            if(index < len(k)):

                if(k[index].isupper()):

                    l.append(upper\_alpha.index(k[index]))

                    index += 1

                else:

                    l.append(lower\_alpha.index(k[index]))

                    index += 1

            else:

                l.append(ind)

                ind += 1

        k\_matrix.append(l)

    return k\_matrix

def deter(k\_matrix):

    a = np.array(k\_matrix)

    return int(round(np.linalg.det(a)))

def inver\_mat(k\_matrix):

    a = np.array(k\_matrix)

    a = np.linalg.inv(a)

    d = deter(k\_matrix)

    a = deter(k\_matrix) \* a

    x = 1

    while True:

        if((d\*x) % 26 == 1):

            break

        x += 1

    a = (a\*x) % 26

    x = []

    for i in a.tolist():

        b = []

        for j in i:

            b.append(int(round(j)))

        x.append(b)

    return x

def mat\_mul(k\_matrix, message\_matrix):

    arr1 = np.array(k\_matrix)

    arr2 = np.array(message\_matrix)

    arr\_result = np.matmul(arr1, arr2)

    arr\_result = arr\_result % 26

    return arr\_result.tolist()

def encryption():

    plaintext = input("Enter plaintext: ")

    key = input("Enter key: ")

    dim = int(input("Enter dim: "))

    k\_m = key\_matrix(key, dim)

    p\_ve = message(plaintext, dim)

    cipher = []

    c = []

    ct = ""

    for i in p\_ve:

        cipher.extend(mat\_mul(k\_m, i))

    lower\_alpha = "abcdefghijklmnopqrstuvwxyz"

    upper\_alpha = lower\_alpha.upper()

    for i in cipher:

        for j in i:

            ct += lower\_alpha[j]

    print(ct)

def decryption():

    ciphertext = input("Enter ciphertext: ")

    key = input("Enter key: ")

    dim = int(input("Enter dim: "))

    k\_m = key\_matrix(key, dim)

    if (deter(k\_m) == 0 or deter(k\_m) == 13 or deter(k\_m) % 2 == 0):

        print("Decryption not possible")

    else:

        k\_m = inver\_mat(k\_m)

        p\_ve = message(ciphertext, dim)

        cipher = []

        c = []

        ct = ""

        for i in p\_ve:

            cipher.extend(mat\_mul(k\_m, i))

        lower\_alpha = "abcdefghijklmnopqrstuvwxyz"

        for i in cipher:

            for j in i:

                ct += lower\_alpha[j]

        print(ct)

if \_\_name\_\_ == "\_\_main\_\_":

    print("\_\_\_\_\_\_\_\_ Hill cipher\_\_\_\_\_\_\_\_")

    while True:

        choice = int(

            input("Enter choice\n1:Encryption\n2:Decryption\n3:Exit\n"))

        if (choice == 1):

            encryption()

        elif(choice == 2):

            decryption()

        elif(choice == 3):

            break

        else:

            print("Invalid choice")

**OUTPUT:**



**Problem Solving:**



|  |  |
| --- | --- |
| Performance |  |
| Lab problem |  |
| Context solving |  |
| **Total** |  |