**PRACTICAL NO. 3**

**Q1)** Write program to implement the following Substitution cipher techniques:

1. **Rail Fence Cipher**

**SOURCE CODE :**

def railencrypt(st,k):

c = 0

x = 0

m =[[0] \* (len(st)) for i in range(k)]

for r in range(len(st)):

m[c][r] = st[r]

if x == 0:

if c == (k-1):

x = 1

c -= 1

else:

c += 1

else:

if c == 0:

x = 0

c += 1

else:

c -= 1

result = []

for i in range(k):

for j in range(len(st)):

if m[i][j] != 0:

result.append(m[i][j])

print("CipherText:","" . join(result))

def raildecrypt(st,k):

c , x = 0 , 0

m =[[0] \* (len(st)) for i in range(k)]

for r in range(len(st)):

m[c][r] = 1

if x == 0:

if c == (k-1):

x = 1

c -= 1

else:

c += 1

else:

if c == 0:

x = 0

c += 1

else:

c -= 1

result = []

c , x = 0 , 0

for i in range(k):

for j in range(len(st)):

if m[i][j] == 1:

m[i][j] = st[x]

x += 1

for r in range(len(st)):

if m[c][r] != 0:

result.append(m[c][r])

if x == 0:

if c == (k-1):

x = 1

c -= 1

else:

c += 1

else:

if c == 0:

x = 0

c += 1

else:

c -= 1

print("PlainText:","" . join(result))

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

string = input("Enter the string:")

string = string.upper()

key = int(input("Enter the Key:"))

n = int(input("1.Encryption\n2.Decryption\nEnter Your choice:"))

if(n == 1):

railencrypt(string,key)

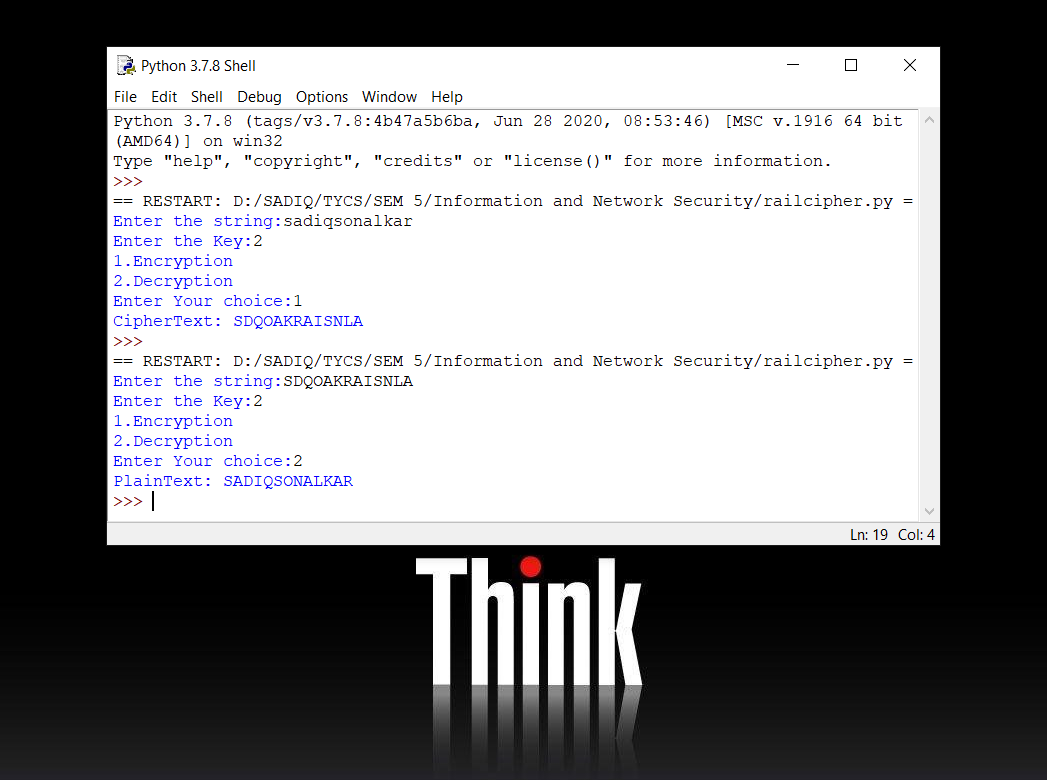
elif(n == 2):

raildecrypt(string,key)

else:

print("Wrong Choice")

**OUTPUT :**



1. **Simple Columnar Cipher**

**SOURCE CODE :**

# Python3 implementation of

# Columnar Transposition

import math

key = input("Enter Key : \n")

# Encryption

def encryptMessage(msg):

    cipher = ""

    # track key indices

    k\_indx = 0

    msg\_len = float(len(msg))

    msg\_lst = list(msg)

    key\_lst = sorted(list(key))

    # calculate column of the matrix

    col = len(key)

    # calculate maximum row of the matrix

    row = int(math.ceil(msg\_len / col))

    # add the padding character '\_' in empty

    # the empty cell of the matix

    fill\_null = int((row \* col) - msg\_len)

    msg\_lst.extend('\_' \* fill\_null)

    # create Matrix and insert message and

    # padding characters row-wise

    matrix = [msg\_lst[i: i + col]

            for i in range(0, len(msg\_lst), col)]

    # read matrix column-wise using key

    for \_ in range(col):

        curr\_idx = key.index(key\_lst[k\_indx])

        cipher += ''.join([row[curr\_idx]

                        for row in matrix])

        k\_indx += 1

    return cipher

# Decryption

def decryptMessage(cipher):

    msg = ""

    # track key indices

    k\_indx = 0

    # track msg indices

    msg\_indx = 0

    msg\_len = float(len(cipher))

    msg\_lst = list(cipher)

    # calculate column of the matrix

    col = len(key)

    # calculate maximum row of the matrix

    row = int(math.ceil(msg\_len / col))

    # convert key into list and sort

    # alphabetically so we can access

    # each character by its alphabetical position.

    key\_lst = sorted(list(key))

    # create an empty matrix to

    # store deciphered message

    dec\_cipher = []

    for \_ in range(row):

        dec\_cipher += [[None] \* col]

    # Arrange the matrix column wise according

    # to permutation order by adding into new matrix

    for \_ in range(col):

        curr\_idx = key.index(key\_lst[k\_indx])

        for j in range(row):

            dec\_cipher[j][curr\_idx] = msg\_lst[msg\_indx]

            msg\_indx += 1

        k\_indx += 1

    # convert decrypted msg matrix into a string

    try:

        msg = ''.join(sum(dec\_cipher, []))

    except TypeError:

        raise TypeError("This program cannot",

                        "handle repeating words.")

    null\_count = msg.count('\_')

    if null\_count > 0:

        return msg[: -null\_count]

    return msg

# Driver Code

msg = input("Enter Plain Text : \n")

cipher = encryptMessage(msg)

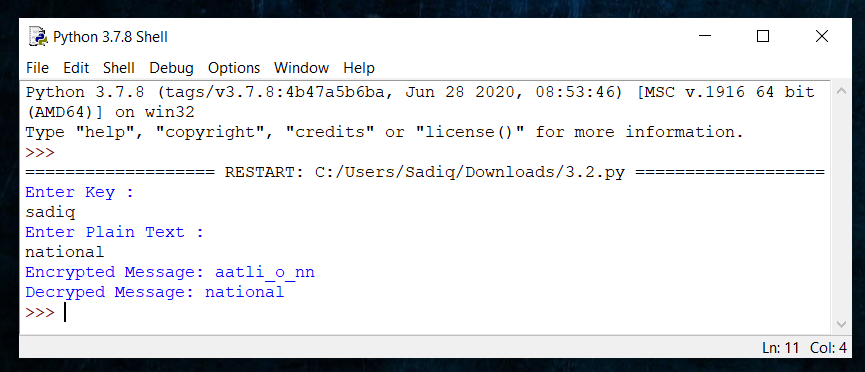
print("Encrypted Message: {}".

            format(cipher))

print("Decryped Message: {}".

    format(decryptMessage(cipher)))

**OUTPUT :**

****