**PRACTICAL NO. 2**

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

1. **Verman Cipher**

**SOURCE CODE :**

pt = input("PLAINTEXT:\t")

otp = input("key:\t")

ptList = list()

otpList = list()

uPt = pt.upper()

updatedOtp = otp.upper()

for character1 in uPt:

temp1 = ord(character1) - 65

ptList.append(temp1)

for character2 in updatedOtp:

temp1 = ord(character2) - 65

otpList.append(temp1)

for number in ptList:

if (number < 0):

ptList.remove(number)

for number in otpList:

if (number < 0):

otpList.remove(number)

tempList = [i + j for i, j in zip(ptList, otpList)]

ctList = []

for item in tempList:

if (item > 25):

item -= 26

ctList.append(item)

else:

ctList.append(item)

sadiq = ""

for number in ctList:

number += 65

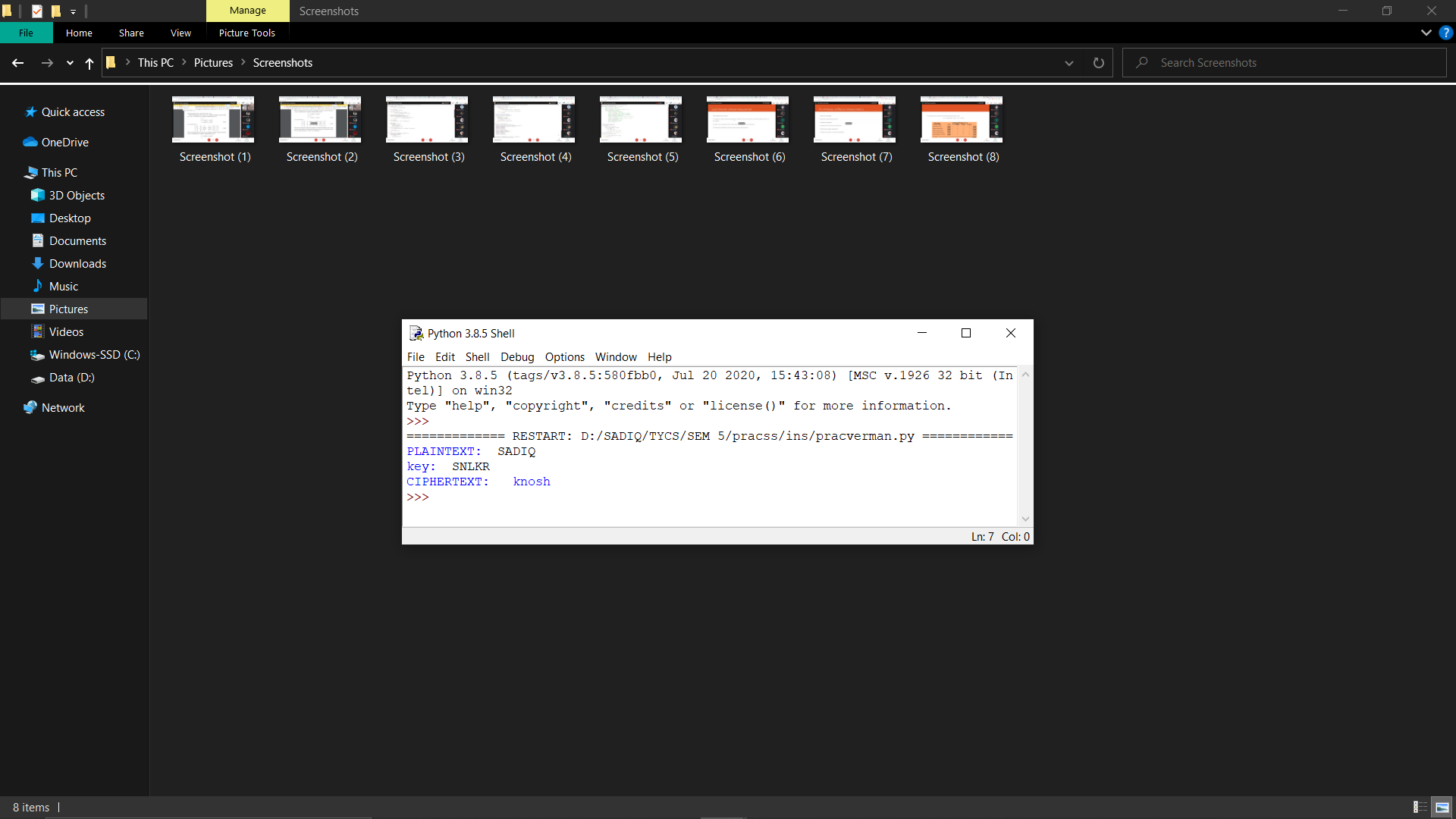
temp = chr(number)

tempstr = str(temp)

sadiq = sadiq + tempstr

print("CIPHERTEXT:\t", sadiq.lower())

**OUTPUT :**



1. **Playfair Cipher**

**SOURCE CODE :**

key = input("Enter key : ")

key = key.replace(" ", "")

key = key.upper()

def matrix(x, y, initial):

return [[initial for i in range(x)] for j in range(y)]

result = list()

for c in key:

if c not in result:

if c == 'J':

result.append('I')

else:

result.append(c)

flag = 0

for i in range(65, 91):

if chr(i) not in result:

if i == 73 and chr(74) not in result:

result.append("I")

flag = 1

elif flag == 0 and i == 73 or i == 74:

pass

else:

result.append(chr(i))

k = 0

my\_matrix = matrix(5, 5, 0)

for i in range(0, 5):

for j in range(0, 5):

my\_matrix[i][j] = result[k]

k += 1

def locindex(c):

loc = list()

if c == 'J':

c = 'I'

for i, j in enumerate(my\_matrix):

for k, l in enumerate(j):

if c == l:

loc.append(i)

loc.append(k)

return loc

def encrypt():

msg = str(input("ENTER MSG : "))

msg = msg.upper()

msg = msg.replace(" ", "")

i = 0

for s in range(0, len(msg) + 1, 2):

if s < len(msg) - 1:

if msg[s] == msg[s + 1]:

msg = msg[:s + 1] + 'X' + msg[s + 1:]

if len(msg) % 2 != 0:

msg = msg[:] + 'X'

print("CIPHER TEXT:", end=' ')

while i < len(msg):

loc = list()

loc = locindex(msg[i])

loc1 = list()

loc1 = locindex(msg[i + 1])

if loc[1] == loc1[1]:

print("{}{}".format(my\_matrix[(loc[0] + 1) % 5][loc[1]], my\_matrix[(loc1[0] + 1) % 5][loc1[1]]), end=' ')

elif loc[0] == loc1[0]:

print("{}{}".format(my\_matrix[loc[0]][(loc[1] + 1) % 5], my\_matrix[loc1[0]][(loc1[1] + 1) % 5]), end=' ')

else:

print("{}{}".format(my\_matrix[loc[0]][loc1[1]], my\_matrix[loc1[0]][loc[1]]), end=' ')

i = i + 2

def decrypt():

msg = str(input("ENTER CIPHER TEXT:"))

msg = msg.upper()

msg = msg.replace(" ", "")

print("PLAIN TEXT:", end=' ')

i = 0

while i < len(msg):

loc = list()

loc = locindex(msg[i])

loc1 = list()

loc1 = locindex(msg[i + 1])

if loc[1] == loc1[1]:

print("{}{}".format(my\_matrix[(loc[0] - 1) % 5][loc[1]], my\_matrix[(loc1[0] - 1) % 5][loc1[1]]), end=' ')

elif loc[0] == loc1[0]:

print("{}{}".format(my\_matrix[loc[0]][(loc[1] - 1) % 5], my\_matrix[loc1[0]][(loc1[1] - 1) % 5]), end=' ')

else:

print("{}{}".format(my\_matrix[loc[0]][loc1[1]], my\_matrix[loc1[0]][loc[1]]), end=' ')

i = i + 2

while (1):

choice = int(input("\n 1.Encryption \n 2.Decryption: \n 3.EXIT \n Enter Your Choice: \n "))

if choice == 1:

encrypt()

elif choice == 2:

decrypt()

elif choice == 3:

exit()

else:

print("Choose correct choice")

**OUTPUT :**

