Problem NO:- 01

Problem Name:- Implement ceasar cipher algorithm to encrypt and decrypt messages.

Theory:-

The Ceasar Cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet. For example,

plain: meet me after the toga party

Cipher: PHHW PH DIWHU WKH WRID SDUWB

Note that the alphabet is wrapped around, so that the letter following Z is A. We can define the transformation by listing all possibilities, as follows:

plain: a b c d e f g h i j k l m n o p q r s t u v w x y z

cipher: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Let us assign a numerical equivalent to each letter:

a	b	c	d	e	f	g	h	I	J	K	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Then the algorithm be expressed as follow. For each plaintext letter P, substitute the ciphertext letter C:

$$C = E(3, P) = (P + 3) \bmod 26$$

A shift may be of any amount, so that the general ceasear algorithm is

$$C = F(K, P) = (P + K) \bmod 26$$

where K takes on a value in the range 1 to 25. The decryption algorithm is simply,

$$P = D(K, C) = (C - K) \bmod 26$$

Source code:-

```
def encrypt_func(txt, s):
result = ""
for i in range (len(txt)):
    char = txt[i]
if (char.isupper()):
    result += chr((ord(char) + (s) - 65)%26+65)

else:
   result+=chr((ord (char) + (s) - 97)%26+97)

return result

txt = input("enter the txt:")
s  = int (input ("enter the key:"))

Print (" plain txt :" + txt)
Print (" Shift pattern :" + str(s))
Print ("cipher :" + encrypt_func (txt, (s)))

def decrypt_func(result, s):
txt = " ".
for i in range (length (result)):
```

```
char = result[i]
if (char.isupper()):
    txt += chr((ord(char) - s - 65) % 26 + 65)
else:
    txt += chr((ord(char) - s - 97) % 26 + 97)
return txt
result = input("enter the result :")
s = int(input("enter the key:"))
Print("plain txt :" + result)
Print(" shift patter :" + str(s))
Print(" Cipher :" + decrypt_func(result,
                                 s))
```

Problem No:- 02

Problem Name:- Implement Mono-Alphabetic algorithm to encrypt and decrypt messages.

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'. It uses a fixed key which consist of the 26 letters of a shuffled alphabet.

Plain text alphabet	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Cipher text alphabet (key)	M	U	A	L	V	O	Z	K	R	N	J	X	Q	D	F	S

With the above key all 'A' letter in the plaintext will be encoded to an 'm'.

This type of cipher is a form of Symmetric encryption as the same key can be used to both encrypt and decrypt message.

Source code:-

```
import random
plain_text=[]
key =[]
for i in range (65, 65+26);
    plain_text.append (chr(i))
    key.append (chr(i))
message =input ("Enter message:")
random.shuffle (key)
print ("plain Text:" plain_text)
print ("key:" key)
cipher =''
for ch in message:
```

```
try:
    index = key.index(ch.upper())
    decrypted_mess = decrypted_mess + plain_
                     text[index]
except:
    decrypt_mess = decrypt_mess + ch
print("Decrypted message:", decrypted_mesg)
```

Problem No: 03

Problem Name:- Implement poly- Alphabetic cipher algorithm to encrypt and decrypt messages.

Theory:-

A polyalphabetic cipher substitution, using multiple substitution alphabets. The vigenere cipher is probably by the best-known example of a poly-alphabetic cipher, though it is a simplied special case. The Enigma machine is more complex but is still fundamentally a polyalphabetic.

One of the simplest polyalphabetic cipher is vigenere cipher, we can express the vigenere cipher in the following ~~cipher~~ manner. Assume a sequence of plaintext letters $P = P_0, P_1, P_2 \dots P_{n-1}$, and a key consisting

of the sequence of letters $K = K_0 K_1 K_2 \ldots K_{m-1}$ where typically $m < n$. The sequence of ciphertext letters $C = C_0, C_1, C_2 \ldots C_{n-1}$ is calculated as follows:

$$C = C_0, C_1, C_2, \ldots C_{n-1} = E(K, P) = E[(K_0, K_1, K_2, \ldots K_{m-1}), (P_0, P_1, P_2, \ldots, P_{n-1})]$$

$$= (P_0 + K_0) \bmod 26, (P_1 + K_1) \bmod 26, \ldots,$$
$$(P_{m-1} + K_{m-1}) \bmod 26, (P_m + K_0) \bmod 26,$$
$$(P_{m+1} + K_1) \bmod 26, \ldots, (P_{2m-1} + K_{m-1}) \bmod 26$$

A general equation of the encryption process is.

$$C_i = (P_i + K_{i \bmod m}) \bmod 26$$

Decryption is a generalization of the following equation:

$$P_i = (C_i - K_{i \bmod m}) \bmod 26.$$

Source Code:-

Problem No: 04

Problem Name:- Implement playfair cipher algorithm to encrypt and decrypt message.

Theory:-

The best-known multiple-letter encryption is the playfair, which treats diagram in the plaintext as single units and translates these units into ciphertext diagram.

The playfair algorithm is based on the use of 5×5 matrix of letter constructed using a keyword. Here is an example, solved by Lord peter Wimsey in Dorothy Sayer's Have His Carcase.

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

In this case, the keyword is monarchy. The matrix is constructed by filling in the letters of the keyword from left to right and from top to bottom and then filling in the remainder of the matrix with the remaining letters in alphabetic order. The letter I and J count as one letter. plaintext is encrypted two letters at a time, according to the following rules:

1. Repeating plaintext letters that are in the same pair are seperated with a filler letter, such as x, so that ballon would be treated as ba lx lo on.

2. Two plaintext letters that fail in the same row of the matrix are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, ar is encrypted as RM

3. Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column ~~are each~~ circularly follow the last.

4. Otherwise, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupaid by the other plaintext letter. Thus, hs becomes BP and ea becomes IM

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 (c)
flag = 0
for i in rang(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 rang (0,5):
     my_matrix [i] [j] = result[K]
     K+=1
```

```
def encrypt():
msg = str(input("Enter message:"))
msg = str(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])
if loc[1]==loc[1]:
    Print("{} {}".format(my_matrix[
```

```
[loc[0]+1)%5][loc[1]], my_matrix
[(loc1[1]]), 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"))
if choice ==1:
    encrypt()
elif choie ==2:
    decrypt()
elif choice ==3:
    exit()
else:
    print("choose connect choice")
```

Problem NO:- 05

Problem Name:- Implement Hill Cipher algorithm to encrypt and decrypt messages.

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 is multiplied by an invertible nxn matrix, against ~~used for~~ modulo 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 nxn matrices

Source Code:-

```
def getkeymatrix (key, n):
 k=0
 for i in range(n):
   for j in range(n):
      key matrix [i] [j] = ord(key[k])%65
      k+=1

def encrypt (messagevector, n):
   for i in range(n):
     for j in range(1):
       ciphertext [i][j] = 0
       for x in range(n):
         ciphermatrix[i][j] += (keyMatrix
                  [i][x] * message vector[x][j])
   CipherMatrix [i][j] = CipherMatrix[i][j]%26

def Hillcipher (message, key, n):
   getkeymatrix (key, n)
```

```
for i in range(n):
    messagevector[i][0] = ord(message[i])%65
    encrypt(message vector, n)
Print("ciphertext:", "".join(ciphertext)
message = input("Enter plaintext:")
key = input("Enter key:"):
n = len(message)
Key Matrix = [[0] * n for i in range(n)]
message vector = [[0] for i in range(n)]
HillCipher(message, key, n)
```

Problem NO: 06

Problem Name:- Implement VERNAM CIPHER algorithm to encrypt and decrypt messages.

Theory:-

The Ultimate defence against such a cryptanalysis is to choose a keyword that is as long as the plaintext and has no statistical relationship to it. Such a system was introduced by an AT & T engineer named Gilbert Vernam in 1918. His system works on binary data rather than letters. The system can be expressed succinctly as follows:

$$C_i = P_i \oplus K_i$$

where,

P_i = ith binary digit of plaintext

K_i = ith binary digit of key

C_i = ith binary digit of ciphertext

⊕ = exclusive - or (XOR) operation

Source Code:-

```
def Vernam (plain, key, flag):
    result = ""
    for i in rang (len (plain)):
        Char = plain[i]
    if (flag):
        result += Chr ((ord (char) - 97 + ord (key[i]
                  -97) % 26 + 97
    else:
        result += Chr (( ord (char) - ord (key[i]
                  + 26) % 26 - 97
    return result
if __name__ == "__main__":
    key = ''.join (input ("Enter key:").lower()
                  .spilt ())
    plain = ''.join (input ("Enter text:")
                  .lower (). split ())
if (len (key) != len (plain):
    Print ("Invalid key!")
    exit (None)
```

```
Ciphertext = vernam (plain, key, True)
Print (" cipher Text :" CipherText)
key = ''.join (input (Enter key:''))
plain = ''.join (input (" Enter ciphertext:"))
Print (''plaintext :'', vernam (cipherText
                    key, false.
```

Problem NO: 07

Problem Name: Implement Rail fence algorith to encrypt and decrypt messages.

Theory:-

The simplest ~~such~~ cipher is the rail fence technique, in which the plain-text is written down as a sequence of diagonals and then read off as a sequence of rows. For example, to encipher the message "meet me after the toga party" with rail fence of depth 2, we write the following:

```
m . e m a t r h t g p r y
  e t e f e t e o a a t
```

The encrypted message is

MEMATRHTTGPRYETEFETEOAAT

A more complex scheme is to write the message in a rectangle, row by row and read the message off, column

by Column but permute the order of the columns. The order of the columns then becomes they key to the algorithm.

key	4	3	1	2	5	6	7
plaintext	a	t	t	a	c	k	p
	o	s	t	p	o	n	e
	d	u	n	t	i	l	t
	w	o	a	m	x	y	z

Ciphertext TTNAAPTMTSUOAODWCOIXKN
LYPETZ