Aim:

To implement encryption and decryption using the Playfair cipher.

PROGRAM:  
A. PLAYFAIR ENCRYPTION CIPHER:

# Python program to implement Playfair Cipher

# Function to convert the string to lowercase

def toLowerCase(text):

return text.lower()

# Function to remove all spaces in a string

def removeSpaces(text):

newText = ""

for i in text:

if i == " ":

continue

else:

newText = newText + i

return newText

# Function to group 2 elements of a string

# as a list element

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

# Function to fill a letter in a string element

# If 2 letters in the same string matches

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']

# Function to generate the 5x5 key square matrix

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)

# Get 2 letter cipherText

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 = 'instruments'

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

PlainTextList = Diagraph(FillerLetter(text\_Plain))

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

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

key = "Monarchy"

print("Key text:", key)

key = toLowerCase(key)

Matrix = generateKeyTable(key, list1)

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

CipherList = encryptByPlayfairCipher(Matrix, PlainTextList)

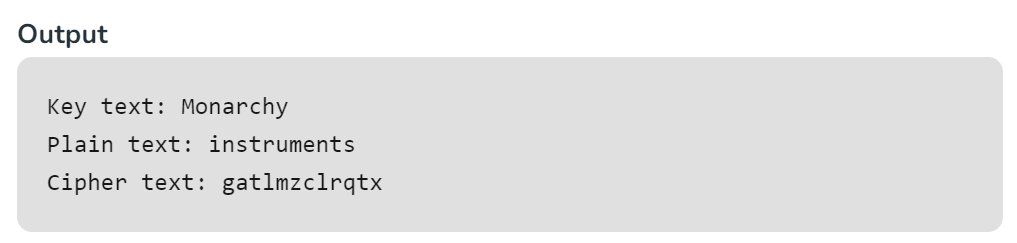
CipherText = ""

for i in CipherList:

CipherText += i

print("CipherText:", CipherText)**ules is true**: Form a rectangle with the two letters and take the letters on the horizon

OUTPUT:



B. PLAYFAIR DECRYPTION CIPHER:

import numpy as np

def to\_lower\_case(text):

return text.lower()

def remove\_spaces(text):

return text.replace(" ", "")

def generate\_key\_table(key):

key = remove\_spaces(to\_lower\_case(key))

key = key.replace('j', 'i')

key = ''.join(dict.fromkeys(key)) # Remove duplicate letters

alphabet = "abcdefghiklmnopqrstuvwxyz" # 'j' is excluded

key\_table = [c for c in key if c in alphabet]

for char in alphabet:

if char not in key\_table:

key\_table.append(char)

key\_table = np.array(key\_table).reshape(5, 5)

return key\_table

def search(key\_table, a, b):

if a == 'j':

a = 'i'

if b == 'j':

b = 'i'

p1 = p2 = None

for i in range(5):

for j in range(5):

if key\_table[i, j] == a:

p1 = (i, j)

elif key\_table[i, j] == b:

p2 = (i, j)

return p1, p2

def decrypt(cipher, key):

key\_table = generate\_key\_table(key)

deciphered = []

for i in range(0, len(cipher), 2):

p1, p2 = search(key\_table, cipher[i], cipher[i+1])

if p1[0] == p2[0]:

deciphered.append(key\_table[p1[0], (p1[1]-1)%5])

deciphered.append(key\_table[p2[0], (p2[1]-1)%5])

elif p1[1] == p2[1]:

deciphered.append(key\_table[(p1[0]-1)%5, p1[1]])

deciphered.append(key\_table[(p2[0]-1)%5, p2[1]])

else:

deciphered.append(key\_table[p1[0], p2[1]])

deciphered.append(key\_table[p2[0], p1[1]])

return ''.join(deciphered)

# Driver code

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

key = "Monarchy"

print("Key Text:", key)

cipher = "gatlmzclrqtx"

print("Ciphertext:", cipher)

decrypted\_text = decrypt(cipher, key)

print("Deciphered text:", decrypted\_text)

OUTPUT:

