Sourec code: Caesar cipher.

def encrypt(message, key):  
 cipher = **""** for i in message:  
 if i.isupper():  
 cipher += chr((ord(i) + key - 65) % 26 + 65)  
 elif i.islower():  
 cipher += chr((ord(i) + key - 97) % 26 + 97)  
 else:  
 cipher+=**" "** return cipher  
  
message = input(**"Enter the message:"**)  
print(**"Cipher:"**, encrypt(message, 3))  
  
*# Decryption part*def decrypt(cipher, key):  
 message = **""** for i in cipher:  
 if i.isupper():  
 message += chr((ord(i) - key - 65) % 26 + 65)  
 elif i.islower():  
 message += chr((ord(i) - key - 97) % 26 + 97)  
 else:  
 message+=**" "** return message  
  
cipher = input(**"Enter the cipher:"**)  
print(**"Message: "**, decrypt(cipher, 3))

Output:

C:\Users\Mehedi\AppData\Local\Programs\Python\Python39\python.exe "C:\4-1\torun sir\ceaser3.py"

Enter the message:mehedi

Cipher: phkhgl

Enter the cipher:phkhgl

Message: mehedi

Process finished with exit code 0

Source code: Mono-Alphabetic cipher.

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 = plain\_text.index(ch.upper())  
 cipher = cipher + key[index]  
 except:  
 cipher = cipher + ch  
print("Cipher: ", cipher)  
decrypted\_mess = ''  
for ch in cipher:  
 try:  
 index = key.index(ch.upper())  
 decrypted\_mess = decrypted\_mess + plain\_text[index]  
 except:  
 decrypted\_mess = decrypted\_mess + ch  
print("Decrypted Message: ", decrypted\_mess)

Output:

C:\Users\Mehedi\AppData\Local\Programs\Python\Python39\python.exe "C:\4-1\torun sir\ceaser3.py"

Enter message: mehedi

Plain Text: ['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']

Key: ['D', 'B', 'X', 'P', 'U', 'A', 'T', 'I', 'O', 'S', 'R', 'C', 'K', 'E', 'V', 'F', 'Q', 'J', 'L', 'W', 'H', 'Y', 'N', 'G', 'Z', 'M']

Cipher: KUIUPO

Decrypted Message: MEHEDI

Process finished with exit code 0

Source Code: Playfair cipher.

global cipher   
global cipher\_text  
def create\_matrix(key):  
 key = key.upper()  
 matrix = [[0 for i in range(5)] for j in range(5)]  
 letters\_added = []  
 row = 0  
 col = 0  
 for letter in key:  
 if letter not in letters\_added:  
 matrix[row][col] = letter  
 letters\_added.append(letter)  
 else:  
 continue  
 if col == 4:  
 col = 0  
 row = row + 1  
 else:  
 col = col + 1  
 for letter in range(65,91):  
 if letter == 74:  
 continue  
 if chr(letter) not in letters\_added:  
 letters\_added.append(chr(letter))  
 index = 0  
 for i in range(5):  
 for j in range(5):  
 matrix[i][j] = letters\_added[index]  
 index = index + 1  
 return matrix  
  
def separate\_same\_letters(message):  
 index = 0  
 while(index<len(message)):  
 letter1 = message[index]  
 if index == len(message) -1:  
 message = message + 'X'  
 index = index + 2  
 continue  
 letter2 = message[index + 1]  
 if letter1 == letter2:  
 message = message[:index + 1] + "X" + message[index + 1:]  
 index = index + 2  
 return message  
  
def indexOf(letter,matrix):  
 for i in range(5):  
 try:  
 index = matrix[i].index(letter)  
 return (i, index)  
 except:  
 continue  
def playfair(key, message, encrypt = True):  
 inc = 1  
 if encrypt == False:  
 inc = -1  
 matrix = create\_matrix(key)  
 message = message.upper()  
 message = message.replace(' ','')  
 message = separate\_same\_letters(message)  
 cipher\_text = ''  
 cipher = ''  
 for(letter1, letter2) in zip(message[0::2], message[1::2]):  
 row1,col1 = indexOf(letter1, matrix)  
 row2,col2 = indexOf(letter2, matrix)  
 if row1 == row2:  
 cipher\_text += matrix[row1][(col1 + inc)%5] + matrix[row2][(col2 + inc)%5]  
 cipher = cipher\_text  
 elif col1 == col2:  
 cipher\_text += matrix[(row1+inc)%5][col1] + matrix[(row2 + inc)%5][col2]  
 cipher = cipher\_text  
 else:  
 cipher\_text += matrix[row1][col2] + matrix[row2][col1]  
 cipher = cipher\_text  
  
  
 return cipher\_text  
  
message = input('Enter plain text: ')  
key = input('Enter key: ')  
print('Encrypting: ')  
print(playfair(key,message))  
encrypt\_message = playfair(key, message)  
print('Decrypting: ')  
print(playfair(key,encrypt\_message, False))

OUTPUT:

C:\Users\Mehedi\AppData\Local\Programs\Python\Python39\python.exe "C:\4-1\torun sir\Lab3\_Playfair\_Cipher.py"

Enter plain text: MEHEDI

Enter key: ICE

Encrypting:

NCGALD

Decrypting:

MEHEDI

Process finished with exit code 0

Source code: Hill cipher.

import numpy as np  
from egcd import egcd  
from egcd import egcd # pip install egcd  
  
alphabet = "abcdefghijklmnopqrstuvwxyz"  
  
letter\_to\_index = dict(zip(alphabet, range(len(alphabet))))  
index\_to\_letter = dict(zip(range(len(alphabet)), alphabet))  
  
  
def matrix\_mod\_inv(matrix, modulus):  
 *"""We find the matrix modulus inverse by  
 Step 1) Find determinant  
 Step 2) Find determinant value in a specific modulus (usually length of alphabet)  
 Step 3) Take that det\_inv times the det\*inverted matrix (this will then be the adjoint) in mod 26  
 """* det = int(np.round(np.linalg.det(matrix))) # Step 1)  
 det\_inv = egcd(det, modulus)[1] % modulus # Step 2)  
 matrix\_modulus\_inv = (  
 det\_inv \* np.round(det \* np.linalg.inv(matrix)).astype(int) % modulus  
 ) # Step 3)  
  
 return matrix\_modulus\_inv  
  
  
def encrypt(message, K):  
 encrypted = ""  
 message\_in\_numbers = []  
  
 for letter in message:  
 message\_in\_numbers.append(letter\_to\_index[letter])  
  
 split\_P = [  
 message\_in\_numbers[i: i + int(K.shape[0])]  
 for i in range(0, len(message\_in\_numbers), int(K.shape[0]))  
 ]  
  
 for P in split\_P:  
 P = np.transpose(np.asarray(P))[:, np.newaxis]  
  
 while P.shape[0] != K.shape[0]:  
 P = np.append(P, letter\_to\_index[" "])[:, np.newaxis]  
  
 numbers = np.dot(K, P) % len(alphabet)  
 n = numbers.shape[0] # length of encrypted message (in numbers)  
  
 # Map back to get encrypted text  
 for idx in range(n):  
 number = int(numbers[idx, 0])  
 encrypted += index\_to\_letter[number]  
  
 return encrypted  
  
  
def decrypt(cipher, Kinv):  
 decrypted = ""  
 cipher\_in\_numbers = []  
  
 for letter in cipher:  
 cipher\_in\_numbers.append(letter\_to\_index[letter])  
  
 split\_C = [  
 cipher\_in\_numbers[i: i + int(Kinv.shape[0])]  
 for i in range(0, len(cipher\_in\_numbers), int(Kinv.shape[0]))  
 ]  
  
 for C in split\_C:  
 C = np.transpose(np.asarray(C))[:, np.newaxis]  
 numbers = np.dot(Kinv, C) % len(alphabet)  
 n = numbers.shape[0]  
  
 for idx in range(n):  
 number = int(numbers[idx, 0])  
 decrypted += index\_to\_letter[number]  
  
 return decrypted  
  
  
# message = 'my life is potato'  
message = str(input('Enter plain text: '))  
message = message.lower()  
K = np.matrix([[3, 3], [2, 5]])  
# K = np.matrix([[6, 24, 1], [13,16,10], [20,17,15]]) # for length of alphabet = 26  
# K = np.matrix([[3,10,20],[20,19,17], [23,78,17]]) # for length of alphabet = 27  
Kinv = matrix\_mod\_inv(K, len(alphabet))  
  
encrypted\_message = encrypt(message, K)  
decrypted\_message = decrypt(encrypted\_message, Kinv)  
  
print("Original message: " + message)  
print("Encrypted message: " + encrypted\_message)  
print("Decrypted message: " + decrypted\_message)  
  
alphabet='abcdefghijklmnopqrstuvwxyz'  
letterToIndex=dict(zip(alphabet))

Output:

Enter plain text: help

Original message: help

Encrypted message: hiat

Decrypted message: help

Process finished with exit code 1

Source Code: Poly-Alphabetic cipher

def generateKey(string, key):  
 key = list(key)  
 if len(string) == len(key):  
 return(key)  
 else:  
 for i in range(len(string)-len(key)):  
 key.append(key[i%len(key)])  
 return("".join(key))  
def encrypt(string, key):  
 cipher = []  
 for i in range(len(string)):  
 x = (ord(string[i])+ord(key[i]))%26 + 65  
 cipher.append(chr(x))  
 return("".join(cipher))  
def decrypt(cipher, key):  
 plain=[]  
 for i in range(len(cipher)):  
 x = (ord(cipher[i])-ord(key[i]))%26 + 65  
 plain.append(chr(x))  
 return("".join(plain))  
string = input('Enter message: ')  
keyword = input('Enter key: ')  
key = generateKey(string, keyword)  
print('Cipher: ',encrypt(string, key))  
cipher = encrypt(string,key)  
print('Plain: ', decrypt(cipher, key))

OUTPUT:

C:\Users\Mehedi\AppData\Local\Programs\Python\Python39\python.exe "C:\4-1\torun sir\polyalphabetic.py"

Enter message: ATTACK MEHEDI

Enter key: ICE

Cipher: IVXIEOBOIPGHQ

Plain: ATTACKTMEHEDI

Process finished with exit code 0

Source Code: Vernam cipher.

import random  
  
  
def generate\_key(length):  
 key = ""  
 for i in range(length):  
 key += chr(random.randint(65, 90)) # ASCII codes for A-Z  
 return key  
  
  
def encrypt(plaintext, key):  
 ciphertext = ""  
 cipherCode = []  
 for i in range(len(plaintext)):  
 x = ord(plaintext[i]) ^ ord(key[i])  
 cipherCode.append(x)  
 ciphertext += chr(x % 26 + 65)  
 return ciphertext, cipherCode  
  
  
def decrypt(cipherCode, key):  
 plaintext = ""  
 for i in range(len(cipherCode)):  
 x = (cipherCode[i] ^ ord(key[i]))  
 plaintext += chr(x)  
 return plaintext  
  
  
plaintext = input("Enter the message:")  
key = generate\_key(len(plaintext)) # Generate a random key  
  
ciphertext, cipherCode = encrypt(plaintext, key)  
print("Ciphertext:", ciphertext)  
decryptedtext = decrypt(cipherCode, key)  
print("Decrypted text:", decryptedtext)

OUTPUT:

C:\Users\Mehedi\AppData\Local\Programs\Python\Python39\python.exe "C:\4-1\torun sir\mehedi.py"

Ciphertext: PDTKFVEJQGI

Decrypted text: Information

Process finished with exit code 0

Source Code: Brute force attack cipher

def brute\_force\_decrypt(ciphertext):  
 for shift in range(26):  
 decrypted\_text = caesar\_decrypt(ciphertext, shift)  
 print(f"Shift {shift}: {decrypted\_text}")  
  
def caesar\_decrypt(ciphertext, shift):  
 decrypted\_text = ""  
 for char in ciphertext:  
 if char.isalpha():  
 if char.islower():  
 decrypted\_text += chr((ord(char) - shift - ord('a')) % 26 + ord('a'))  
 else:  
 decrypted\_text += chr((ord(char) - shift - ord('A')) % 26 + ord('A'))  
 else:  
 decrypted\_text += char  
 return decrypted\_text  
  
ciphertext =input("Enter the cipher:")  
print("Brute Force Decryption for Caesar Cipher:")  
brute\_force\_decrypt(ciphertext)

OUTPUT:

|  |  |
| --- | --- |
| C:\Users\Mehedi\AppData\Local\Programs\Python\Python39\python.exe "C:\4-1\torun sir\BruteForce.py"  Brute Force Decryption for Caesar Cipher:  Shift 0: hello  Shift 1: gdkkn  Shift 2: fcjjm  Shift 3: ebiil  Shift 4: dahhk  Shift 5: czggj  Shift 6: byffi  Shift 7: axeeh  Shift 8: zwddg  Shift 9: yvccf  Shift 10: xubbe  Shift 11: wtaad  Shift 12: vszzc  Shift 13: uryyb  Shift 14: tqxxa  Shift 15: spwwz | Shift 16: rovvy  Shift 17: qnuux  Shift 18: pmttw  Shift 19: olssv  Shift 20: nkrru  Shift 21: mjqqt  Shift 22: lipps  Shift 23: khoor  Shift 24: jgnnq  Shift 25: ifmmp  Process finished with exit code 0 |