Course : B. Sc. (h) Computer Science

Year : III

Semester : VI

Name : Bharat Sharma

College Rollno : CSC/20/50

University Rollno : 20059570040

Information Security Practical File

Submitted to :- Mr. Rajeev Rai

INDEX

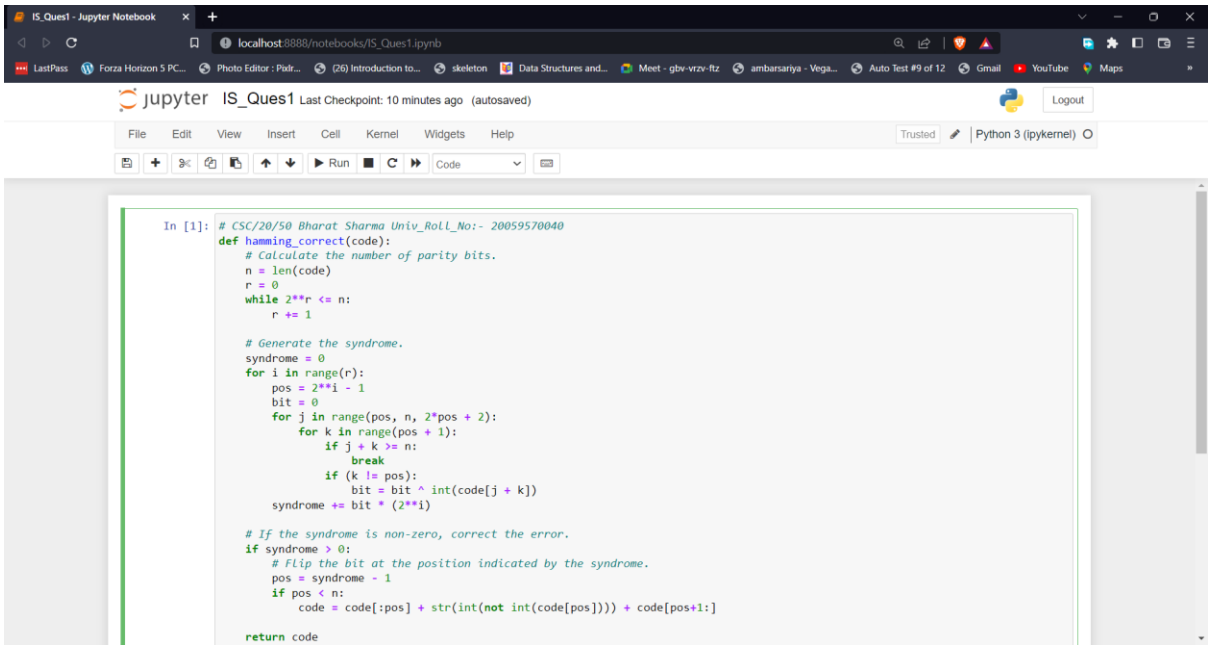
11.	**Ques11.**Implement a stream cipher technique	20

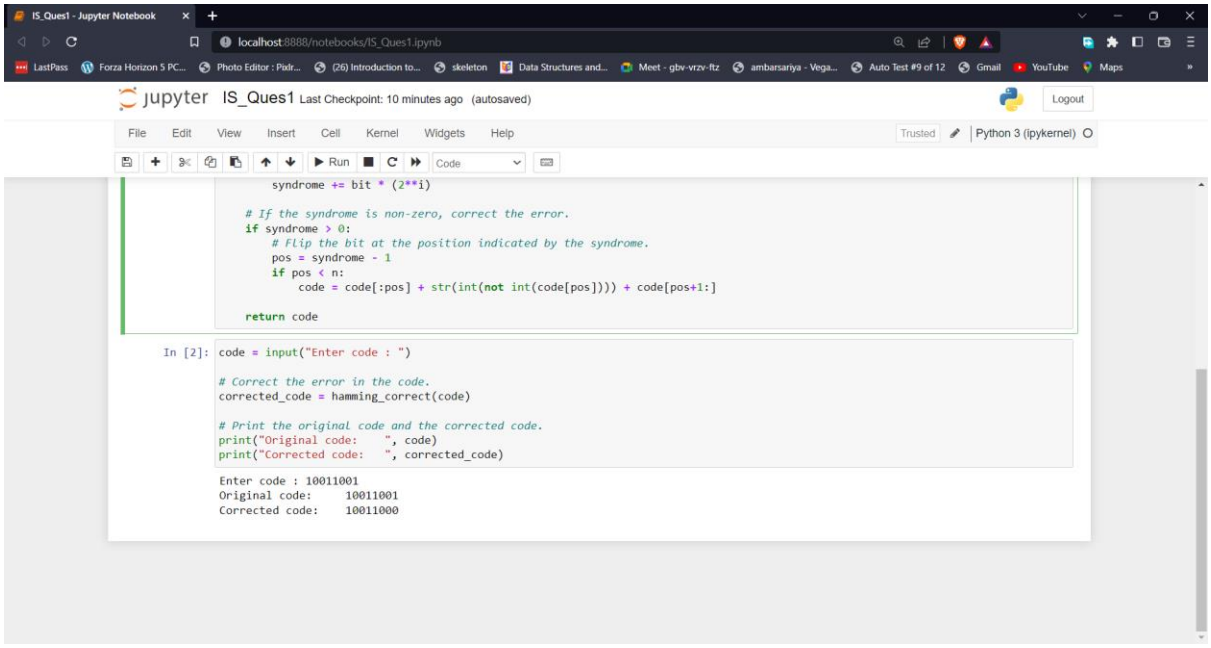
Ques 1. Implement the error correcting code.

Ans:-

```
# CSC/20/50 Bharat Sharma Univ_Roll_No:- 20059570040
def hamming_correct(code):
    # Calculate the number of parity bits.
    n = len(code)
    r = 0
    while 2**r <= n:
        r += 1
    # Generate the syndrome.
    syndrome = 0
    for i in range(r):
        pos = 2**i - 1
        bit = 0
        for j in range(pos, n, 2*pos + 2):
            for k in range(pos + 1):
                if j + k >= n:
                    break
                if (k != pos):
                    bit = bit ^ int(code[j + k])
        syndrome += bit * (2**i)
    # If the syndrome is non-zero, correct the error.
    if syndrome > 0:
        # Flip the bit at the position indicated by the syndrome.
        pos = syndrome - 1
        if pos < n:
            code = code[:pos] + str(int(not int(code[pos]))) + code[pos+1:]
    return code
code = input("Enter code : ")
# Correct the error in the code.
corrected_code = hamming_correct(code)
# Print the original code and the corrected code.
print("Original code:    ", code)
print("Corrected code:   ", corrected_code)
```

OUTPUT:-



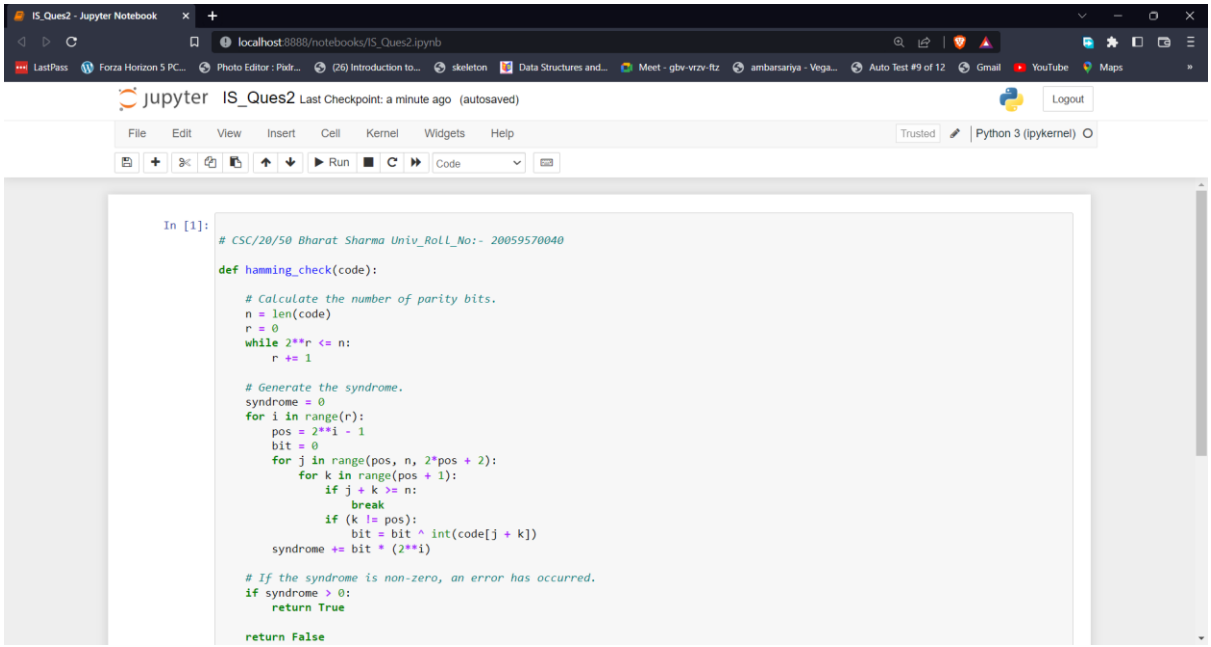

IS_Ques1 - Jupyter Notebook
localhost:8888/notebooks/IS_Ques1.ipynb
jupyter IS_Ques1 Last Checkpoint: 10 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
syndrome += bit * (2**i)
# If the syndrome is non-zero, correct the error.
if syndrome > 0:
# Flip the bit at the position indicated by the syndrome.
pos = syndrome - 1
if pos < n:
code = code[:pos] + str(int(not int(code[pos]))) + code[pos+1:]
return code
In [2]:
code = input("Enter code : ")
# Correct the error in the code.
corrected_code = hamming_correct(code)
# Print the original code and the corrected code.
print("Original code: ", code)
print("Corrected code: ", corrected_code)
Enter code : 10011001
Original code: 10011001
Corrected code: 10011000

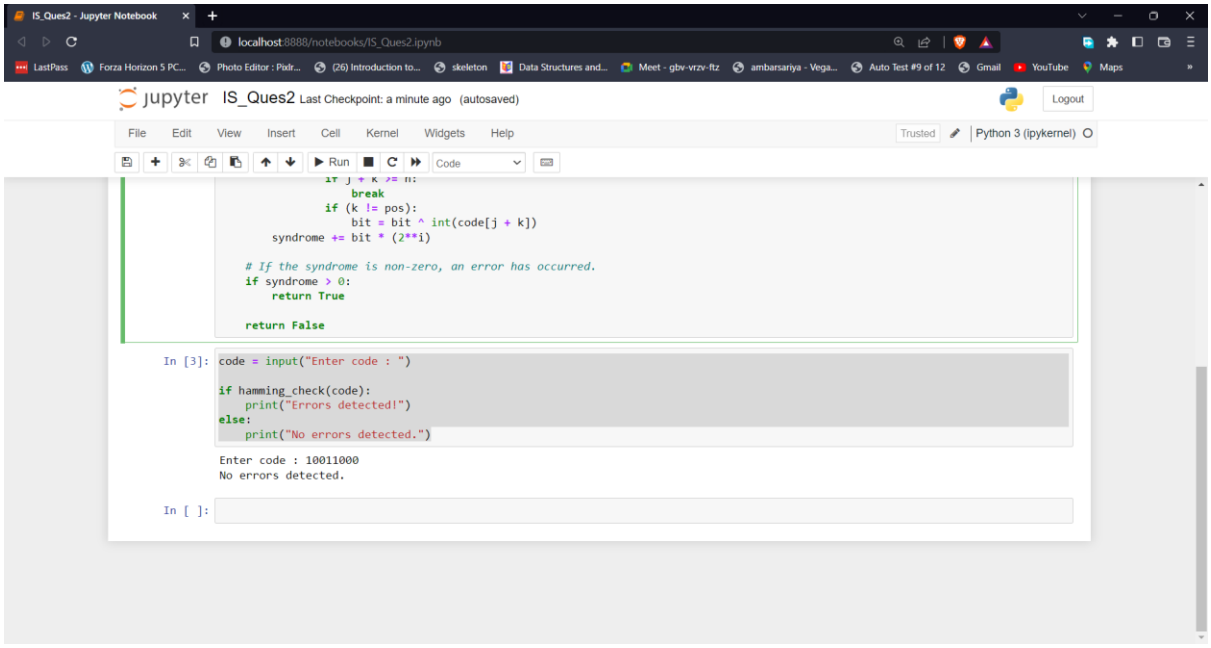
Ques 2. Implement the error detecting code.

Ans:-

```
# CSC/20/50 Bharat Sharma Univ_Roll_No:- 20059570040
def hamming_check(code):

    # Calculate the number of parity bits.
    n = len(code)
    r = 0
    while 2**r <= n:
        r += 1

    # Generate the syndrome.
    syndrome = 0
    for i in range(r):
        pos = 2**i - 1
        bit = 0
        for j in range(pos, n, 2*pos + 2):
            for k in range(pos + 1):
                if j + k >= n:
                    break
                if (k != pos):
                    bit = bit ^ int(code[j + k])
        syndrome += bit * (2**i)

    # If the syndrome is non-zero, an error has occurred.
    if syndrome > 0:
        return True

    return False
code = input("Enter code : ")

if hamming_check(code):
    print("Errors detected!")
else:
    print("No errors detected.")
```

OUTPUT:-



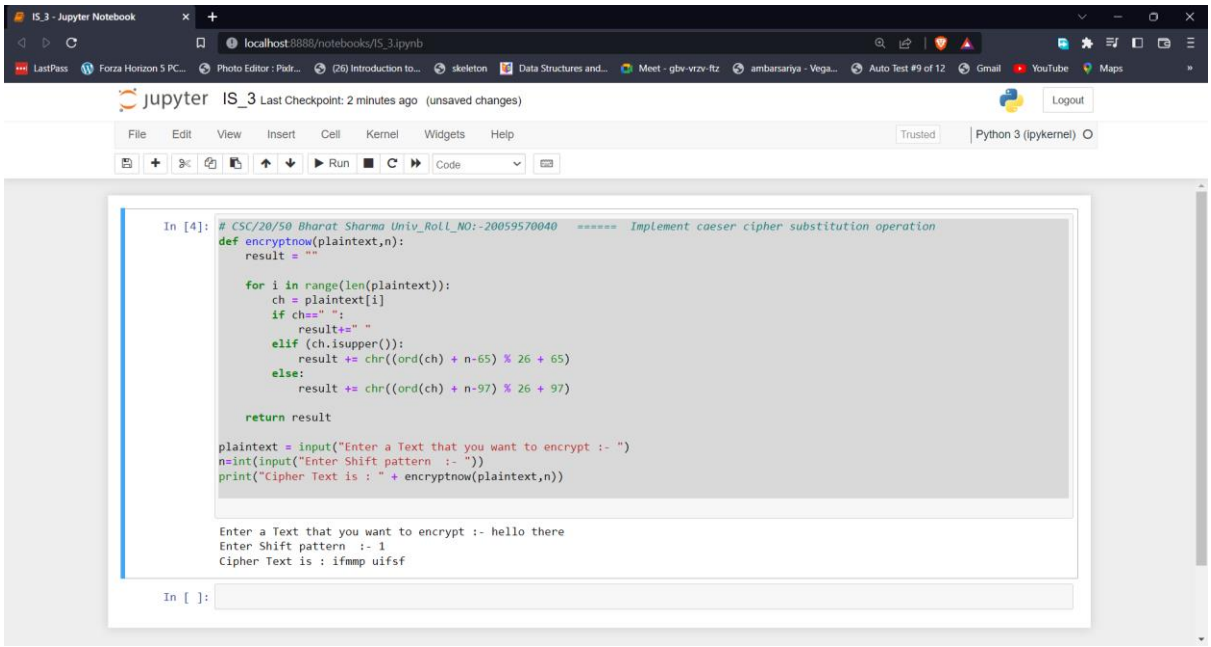

IS_Ques2 - Jupyter Notebook
localhost:8888/notebooks/IS_Ques2.ipynb
jupyter IS_Ques2 Last Checkpoint: a minute ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
if j + k >= n:
break
if (k != pos):
bit = bit ^ int(code[j + k])
syndrome += bit * (2**i)
# If the syndrome is non-zero, an error has occurred.
if syndrome > 0:
return True
return False
In [3]:
code = input("Enter code : ")
if hamming_check(code):
print("Errors detected!")
else:
print("No errors detected.")
Enter code : 10011000
No errors detected.
In [ ]:

Ques3. Implement caeser cipher substitution operation.

Ans

```
# CSC/20/50 Bharat Sharma Univ_Roll_NO:-20059570040   ======  Implement caeser cipher substitution operation
def encryptnow(plaintext,n):
    result = ""

    for i in range(len(plaintext)):
        ch = plaintext[i]
        if ch==" ":
            result+=" "
        elif (ch.isupper()):
            result += chr((ord(ch) + n-65) % 26 + 65)
        else:
            result += chr((ord(ch) + n-97) % 26 + 97)

    return result

plaintext = input("Enter a Text that you want to encrypt :- ")
n=int(input("Enter Shift pattern  :- "))
print("Cipher Text is : " + encryptnow(plaintext,n))
```

OUTPUT:-

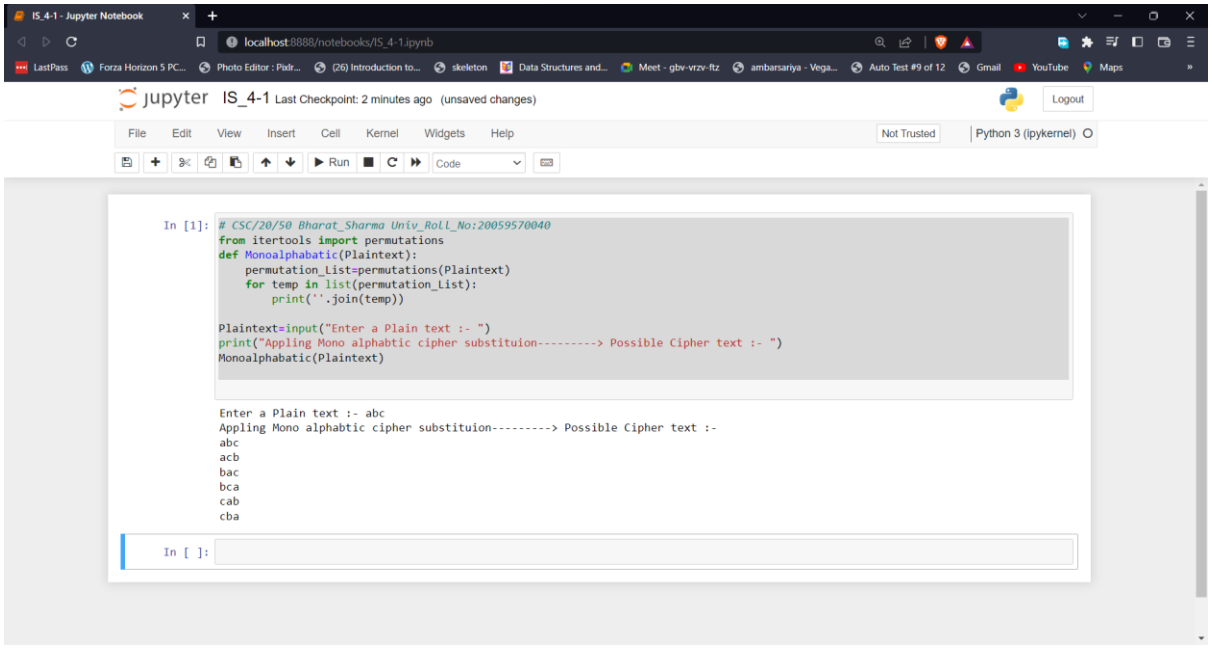


Ques4. Implement monoalphabetic and polyalphabetic cipher substitution operation..

Ans ---Monoalphabetic cipher substitution

```
# CSC/20/50 Bharat_Sharma Univ_Roll_No:20059570040
from itertools import permutations
def Monoalphabatic(Plaintext):
    permutation_List=permutations(Plaintext)
    for temp in list(permutation_List):
        print(''.join(temp))

Plaintext=input("Enter a Plain text :- ")
print("Appling Mono alphabtic cipher substituion---------> Possible Cipher text :- ")
Monoalphabatic(Plaintext)
```

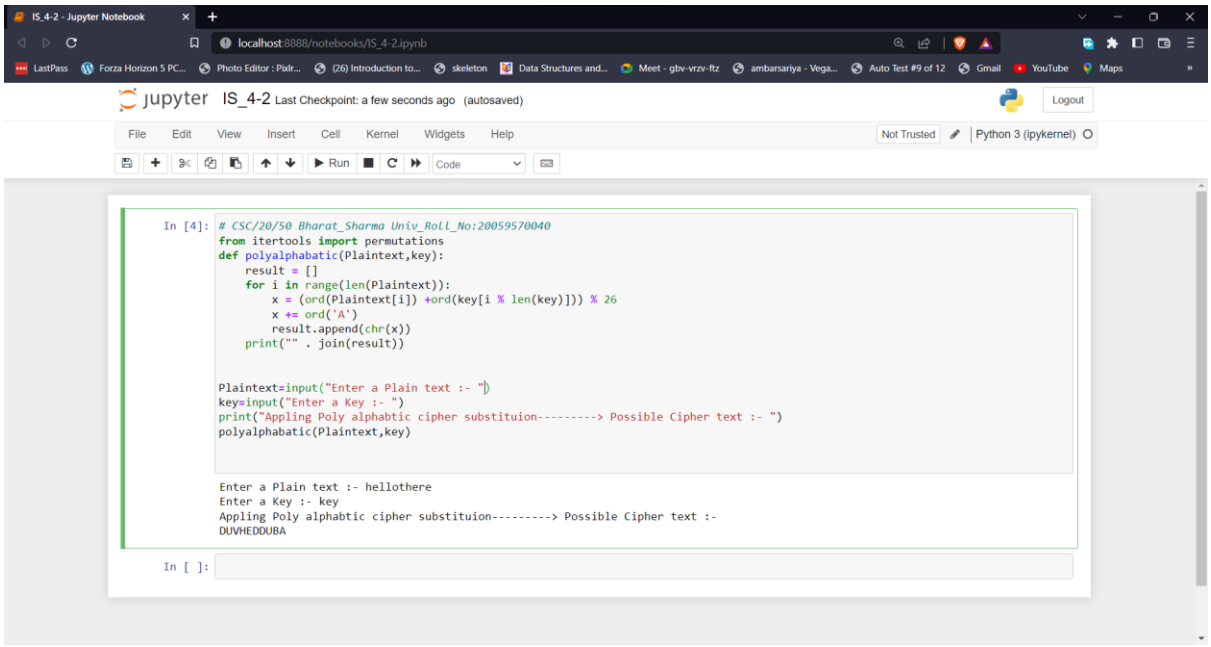
OUTPUT:-



Ans ---Polyalphabetic cipher substitution

```
# CSC/20/50 Bharat_Sharma Univ_Roll_No:20059570040
from itertools import permutations
def polyalphabatic(Plaintext,key):
    result = []
    for i in range(len(Plaintext)):
        x = (ord(Plaintext[i]) +ord(key[i % len(key)])) % 26
        x += ord('A')
        result.append(chr(x))
    print("" . join(result))


Plaintext=input("Enter a Plain text :- ")
key=input("Enter a Key :- ")
print("Appling Poly alphabtic cipher substituion---------> Possible Cipher text :- ")
polyalphabatic(Plaintext,key)
```

OUTPUT:-



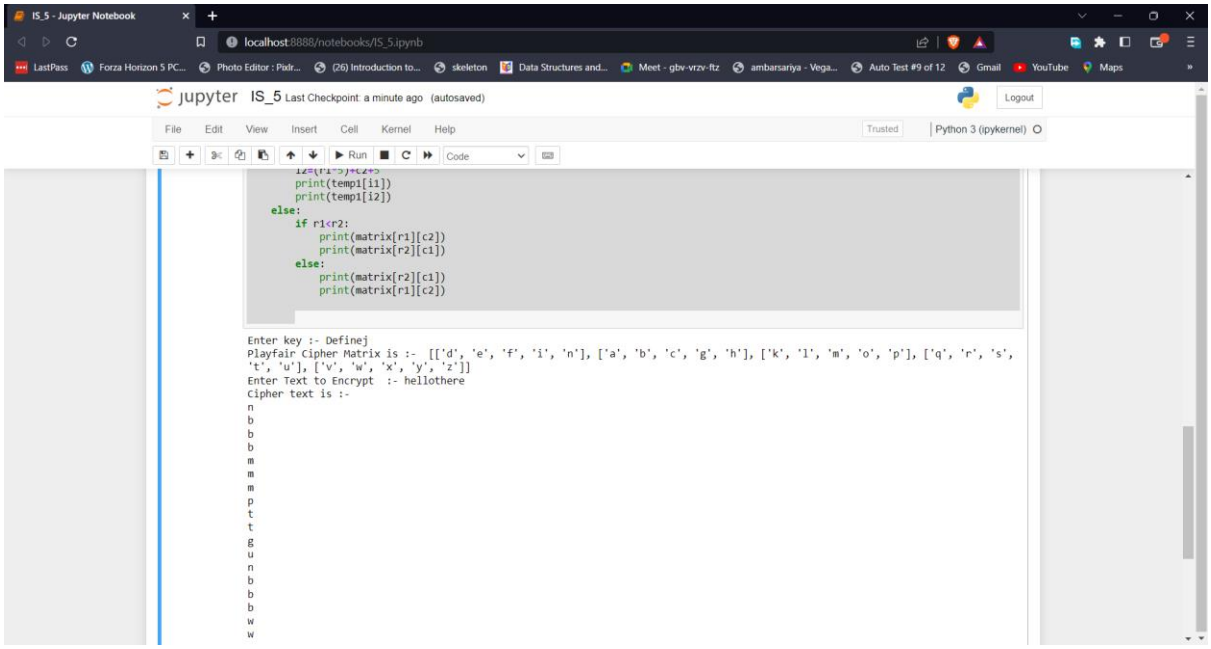
Ques 5. Implement play fair cipher substitution operation.

Ans:-

```
# Bharat_Sharma CSC/20/50 Univ_ROll_No:20059570040
import string
import numpy as np
key=input("Enter key :- ")
key=key.lower()
key=key.replace(" ","")

temp=[]
for alpha in key:
    if alpha not in temp:
        temp.append(alpha)
key=temp
matrix=[]
for i in string.ascii_lowercase[:26]:
    if i not in key:
        key.append(i)
i=key.index('i')
j=key.index('j')
if i<j:
    del key[j]
else:
    del key[i]
temp1=key
while key!=[]:
    matrix.append(key[:5])
    key=key[5:]

print("Playfair Cipher Matrix is :- ",matrix)
def getpoistion(alpha,matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if matrix[i][j] == alpha:
                return (i, j)


text=input("Enter Text to Encrypt  :- ")
text=text.lower()
text=text.replace(" ","")
text1=[]
for i in text:
    if i not in temp1:
        if i=='j':
            text1.append('i')
        else:
            text1.append('j')
    else:
        text1.append(i)

text=text1
print("Cipher text is :- ")
for i in range(len(text)):
    r1,c1=getpoistion(text[i],matrix)
    r2,c2=getpoistion(text[i+1],matrix)
    i=i+1
    if r1==r2:
        i1=(r1*5)+c1+1
        i2=(r1*5)+c2+1
        i1=i1%25
        i2=i2%25
        print(temp1[i1])
        print(temp1[i2])
    elif c1==c2:
        i1=(r1*5)+c1+5
        i2=(r1*5)+c2+5
        print(temp1[i1])
        print(temp1[i2])
    else:
        if r1<r2:
            print(matrix[r1][c2])
            print(matrix[r2][c1])
        else:
            print(matrix[r2][c1])
```

```
        print(matrix[r1][c2])
```

OUTPUT:-

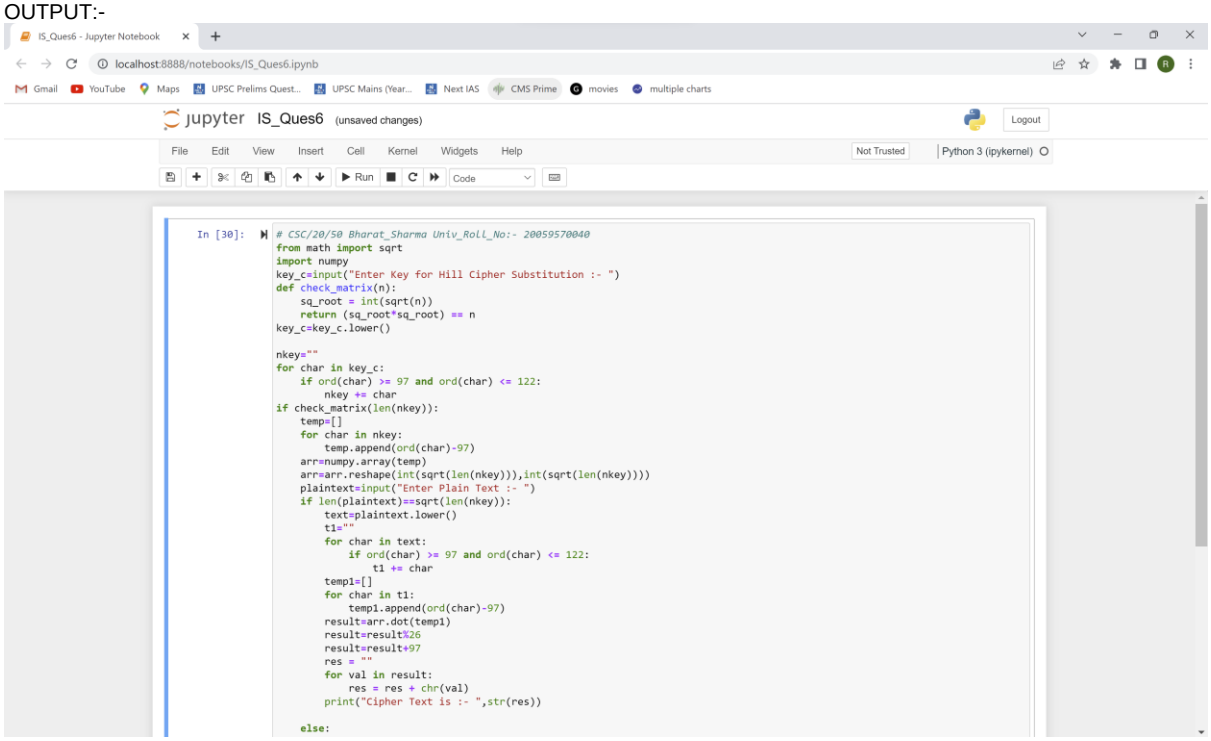


Ques 6. Implement hill cipher substitution operation..

Ans:-

```
# CSC/20/50 Bharat_Sharma Univ_Roll_No:- 20059570040
from math import sqrt
import numpy
key_c=input("Enter Key for Hill Cipher Substitution :- ")
def check_matrix(n):
    sq_root = int(sqrt(n))
    return (sq_root*sq_root) == n
key_c=key_c.lower()
nkey=""
for char in key_c:
    if ord(char) >= 97 and ord(char) <= 122:
        nkey += char
if check_matrix(len(nkey)):
    temp=[]
    for char in nkey:
        temp.append(ord(char)-97)
    arr=numpy.array(temp)
    arr=arr.reshape(int(sqrt(len(nkey))),int(sqrt(len(nkey))))
    plaintext=input("Enter Plain Text :- ")
    if len(plaintext)==sqrt(len(nkey)):
        text=plaintext.lower()
        t1=""
        for char in text:
            if ord(char) >= 97 and ord(char) <= 122:
                t1 += char
        temp1=[]
        for char in t1:
            temp1.append(ord(char)-97)
        result=arr.dot(temp1)
        result=result%26
        result=result+97
        res = ""
        for val in result:
            res = res + chr(val)
        print("Cipher Text is :- ",str(res))
    else:
        print("Plain text of Wrong length ")
else:
    print("Key is not valid ")
```

OUTPUT:-



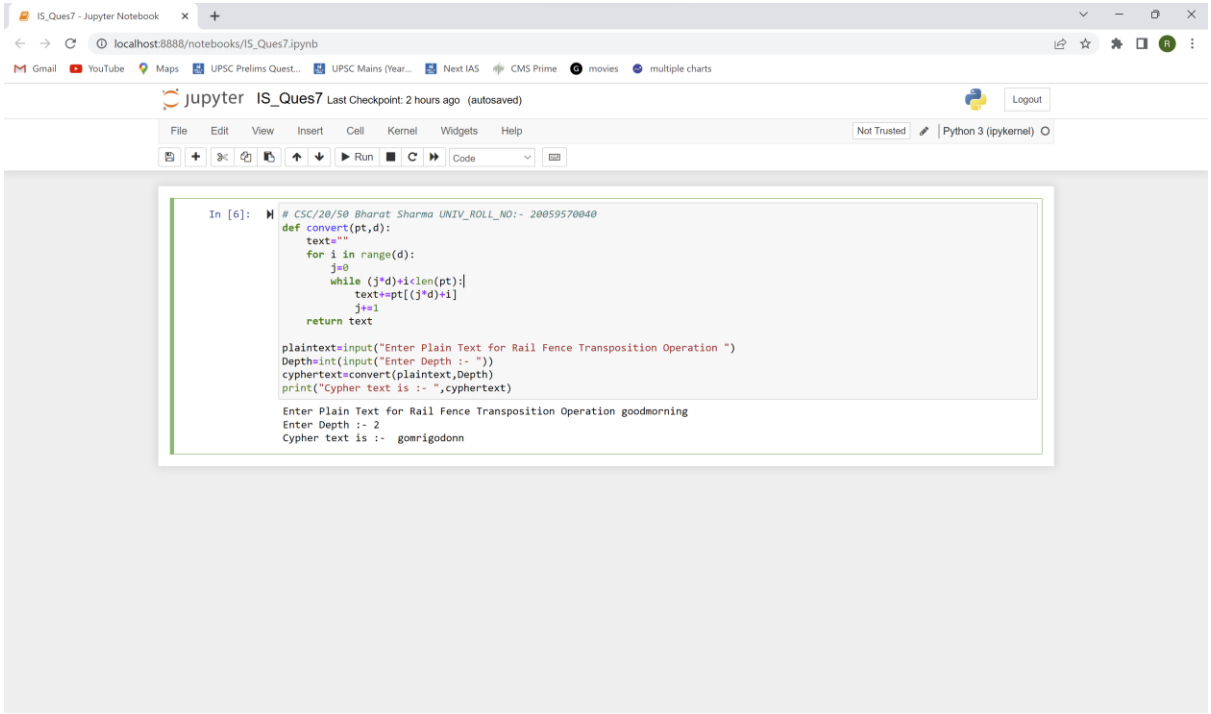

print("Plain text of Wrong length ")
else:
print("Key is not valid ")
Enter Key for Hill Cipher Substitution :- gybnqkurp
Enter Plain Text :- act
Cipher Text is :- poh
In [4]:

Ques 7. Implement rail fence cipher transposition operation.

Ans:-

```
# CSC/20/50 Bharat Sharma UNIV_ROLL_NO:- 20059570040
def convert(pt,d):
    text=""
    for i in range(d):
        j=0
        while (j*d)+i<len(pt):
            text+=pt[(j*d)+i]
            j+=1
    return text

plaintext=input("Enter Plain Text for Rail Fence Transposition Operation ")
Depth=int(input("Enter Depth :- "))
cyphertext=convert(plaintext,Depth)
print("Cypher text is :- ",cyphertext)
```

OUTPUT:-

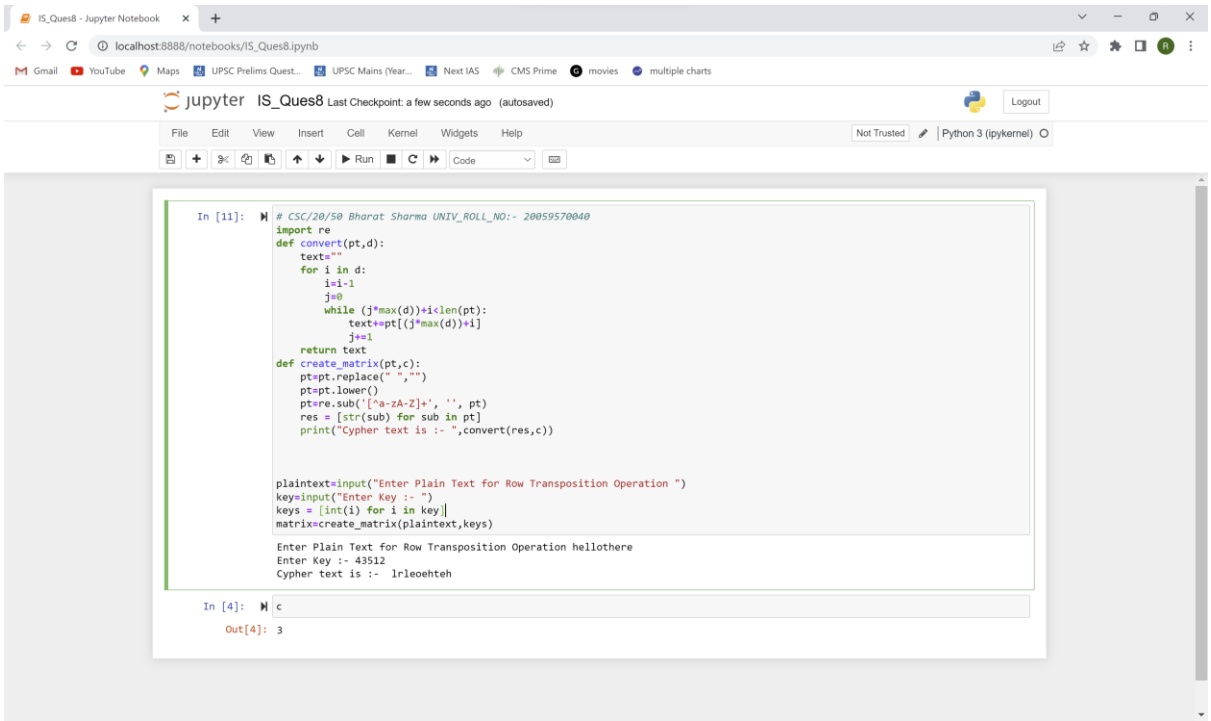


Ques 8. Implement row transposition cipher transposition operation.

Ans:-

```
# CSC/20/50 Bharat Sharma UNIV_ROLL_NO:- 20059570040
import re
def convert(pt,d):
    text=""
    for i in d:
        i=i-1
        j=0
        while (j*max(d))+i<len(pt):
            text+=pt[(j*max(d))+i]
            j+=1
    return text
def create_matrix(pt,c):
    pt=pt.replace(" ","")
    pt=pt.lower()
    pt=re.sub('[^a-zA-Z]+', '', pt)
    res = [str(sub) for sub in pt]
    print("Cypher text is :- ",convert(res,c))


plaintext=input("Enter Plain Text for Row Transposition Operation ")
key=input("Enter Key :- ")
keys = [int(i) for i in key]
matrix=create_matrix(plaintext,keys)
```

OUTPUT:-

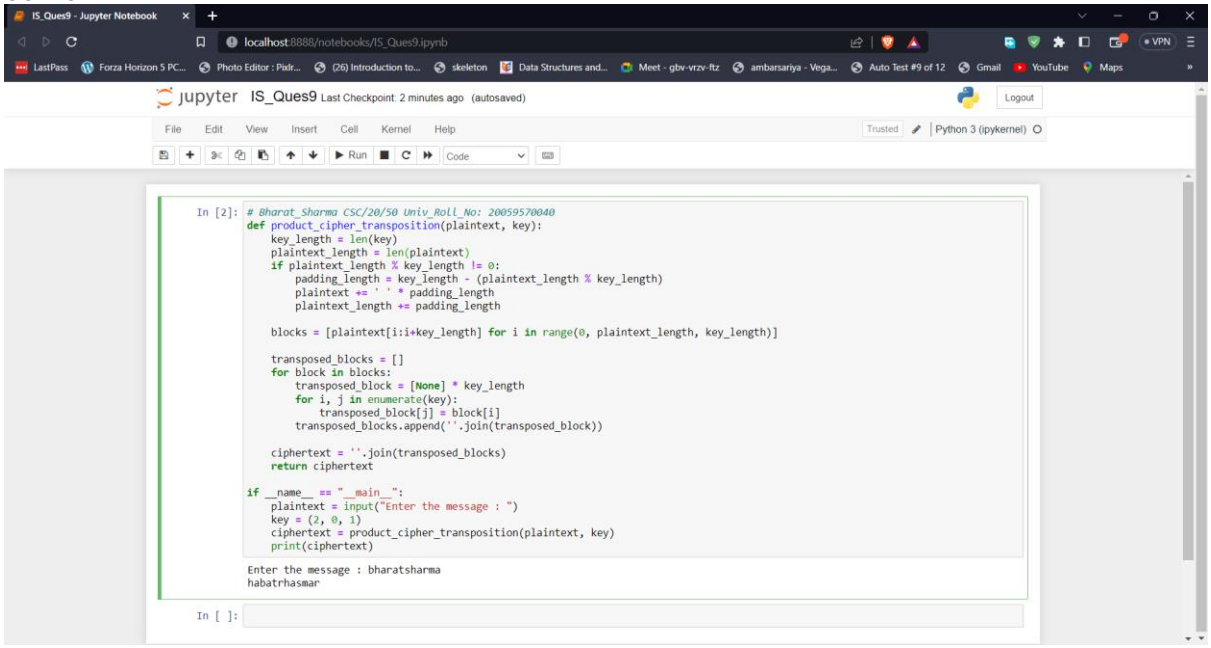


Ques 9. Implement product cipher transposition operation.

Ans:-

```
# Bharat_Sharma CSC/20/50 Univ_Roll_No: 20059570040
def product_cipher_transposition(plaintext, key):
    key_length = len(key)
    plaintext_length = len(plaintext)
    if plaintext_length % key_length != 0:
        padding_length = key_length - (plaintext_length % key_length)
        plaintext += ' ' * padding_length
        plaintext_length += padding_length

    blocks = [plaintext[i:i+key_length] for i in range(0, plaintext_length, key_length)]

    transposed_blocks = []
    for block in blocks:
        transposed_block = [None] * key_length
        for i, j in enumerate(key):
            transposed_block[j] = block[i]
        transposed_blocks.append(''.join(transposed_block))

    ciphertext = ''.join(transposed_blocks)
    return ciphertext

if __name__ == "__main__":
    plaintext = input("Enter the message : ")
    key = (2, 0, 1)
    ciphertext = product_cipher_transposition(plaintext, key)
    print(ciphertext)
```

OUTPUT:-

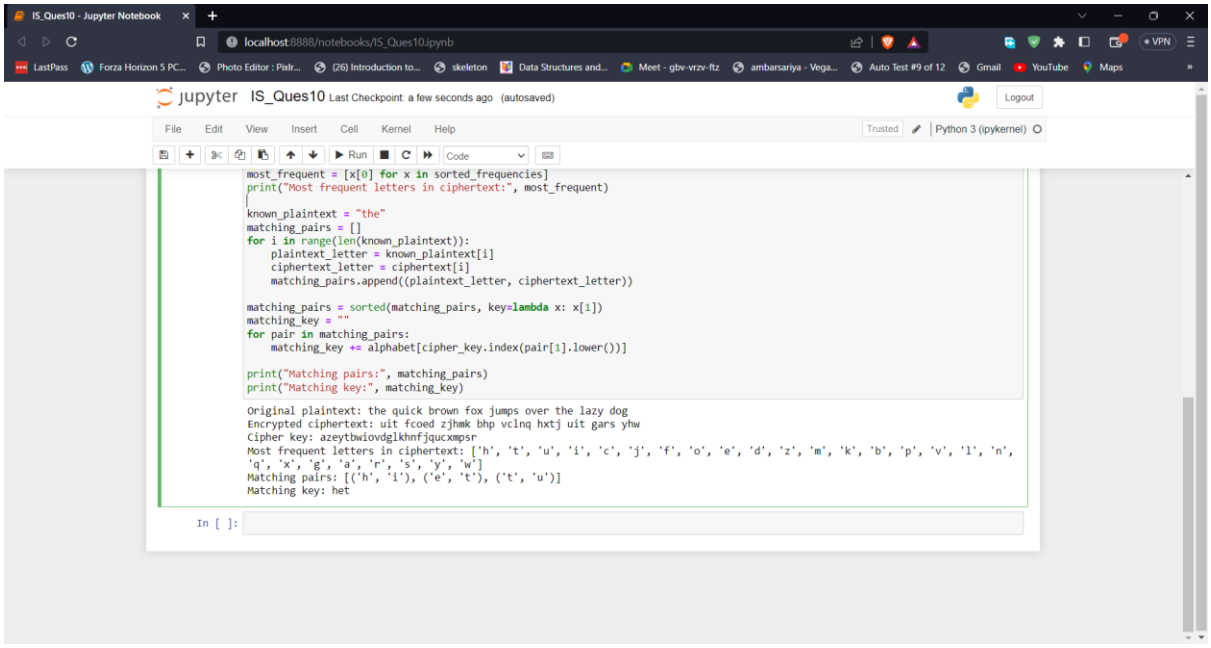


Ques 10. . Illustrate the Ciphertext only and Known Plaintext attacks

Ans:-

```
# Bharat_Sharma CSC/20/50 Univ_Roll_No: 20059570040
import random
import string

alphabet = list(string.ascii_lowercase)
cipher_key = alphabet.copy()
random.shuffle(cipher_key)
cipher_key = ''.join(cipher_key)

plaintext = "the quick brown fox jumps over the lazy dog"
ciphertext = ""
for letter in plaintext:
    if letter.isalpha():
        ciphertext += cipher_key[alphabet.index(letter.lower())]
    else:
        ciphertext += letter

print("Original plaintext:", plaintext)
print("Encrypted ciphertext:", ciphertext)
print("Cipher key:", cipher_key)

frequencies = {}
for letter in ciphertext:
    if letter.isalpha():
        if letter.lower() in frequencies:
            frequencies[letter.lower()] += 1
        else:
            frequencies[letter.lower()] = 1

sorted_frequencies = sorted(frequencies.items(), key=lambda x: x[1], reverse=True)
most_frequent = [x[0] for x in sorted_frequencies]
print("Most frequent letters in ciphertext:", most_frequent)

known_plaintext = "the"
matching_pairs = []
for i in range(len(known_plaintext)):
    plaintext_letter = known_plaintext[i]
    ciphertext_letter = ciphertext[i]
    matching_pairs.append((plaintext_letter, ciphertext_letter))

matching_pairs = sorted(matching_pairs, key=lambda x: x[1])
matching_key = ""
for pair in matching_pairs:
    matching_key += alphabet[cipher_key.index(pair[1].lower())]

print("Matching pairs:", matching_pairs)
print("Matching key:", matching_key)
```

OUTPUT:-


IS_Ques10 - Jupyter Notebook
localhost:8888/notebooks/IS_Ques10.ipynb
jupyter IS_Ques10 Last Checkpoint: a few seconds ago (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code
most_frequent = [x[0] for x in sorted_frequencies]
print("Most frequent letters in ciphertext:", most_frequent)
known_plaintext = "the"
matching_pairs = []
for i in range(len(known_plaintext)):
plaintext_letter = known_plaintext[i]
ciphertext_letter = ciphertext[i]
matching_pairs.append((plaintext_letter, ciphertext_letter))
matching_pairs = sorted(matching_pairs, key=lambda x: x[1])
matching_key = ""
for pair in matching_pairs:
matching_key += alphabet[cipher_key.index(pair[1].lower())]
print("Matching pairs:", matching_pairs)
print("Matching key:", matching_key)
Original plaintext: the quick brown fox jumps over the lazy dog
Encrypted ciphertext: uit fcoed zjhmk bhp vclnq hxtj uit gars yhw
Cipher key: azeytbwiovdglkhnfjqucxmpsr
Most frequent letters in ciphertext: ['h', 't', 'u', 'i', 'c', 'j', 'f', 'o', 'e', 'd', 'z', 'm', 'k', 'b', 'p', 'v', 'l', 'n', 'q', 'x', 'g', 'a', 'r', 's', 'y', 'w']
Matching pairs: [('h', 'i'), ('e', 't'), ('t', 'u')]
Matching key: het
In [ ]:

Ques 11. Implement a stream cipher technique.

Ans:-

```
# Bharat_Sharma CSC/20/50 Univ_Roll_No: 20059570040
def rc4_keystream(key):
    """Generate a pseudorandom keystream using the RC4 algorithm."""
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        yield S[(S[i] + S[j]) % 256]

def stream_cipher(plaintext, key):
    keystream = rc4_keystream(key)
    ciphertext = []
    for byte in plaintext:
        keystream_byte = next(keystream)
        ciphertext_byte = byte ^ keystream_byte
        ciphertext.append(ciphertext_byte)
    return bytes(ciphertext)

if __name__ == "__main__":
    plaintext = b"Hello, world!"
    key = b"secretkey"
    ciphertext = stream_cipher(plaintext, key)
    print("Cipher Text ==> ",ciphertext)
    decrypted_plaintext = stream_cipher(ciphertext, key)
    print("Deciphered Text ==> ",decrypted_plaintext)
```

OUTPUT:-