

**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

|  |  |  |
| --- | --- | --- |
| Sno | Practical Questions | Page No |
| 1 | **Ques1.** Implement the error correcting code. | 4 |
| 2 | **Ques2.** Implement the error detecting code. | 6 |
| 3 | **Ques3.** Implement caeser cipher substitution operation. | 8 |
| 4 | **Ques4.** Implement monoalphabetic and polyalphabetic cipher substitution operation. | 9 |
| 5 | **Ques5.** Implement playfair cipher substitution operation. | 11 |
| 6 | **Ques6.** Implement hill cipher substitution operation. | 13 |
| 7 | **Ques7.** Implement rail fence cipher transposition operation. | 15 |
| 8 | **Ques8.** Implement row transposition cipher transposition operation. | 16 |
| 9 | **Ques9.**Implement product cipher transposition operation. | 17 |
| 10. | **Ques10.**Illustrate the Ciphertext only and Known Plaintext attacks. | 18 |
| 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)

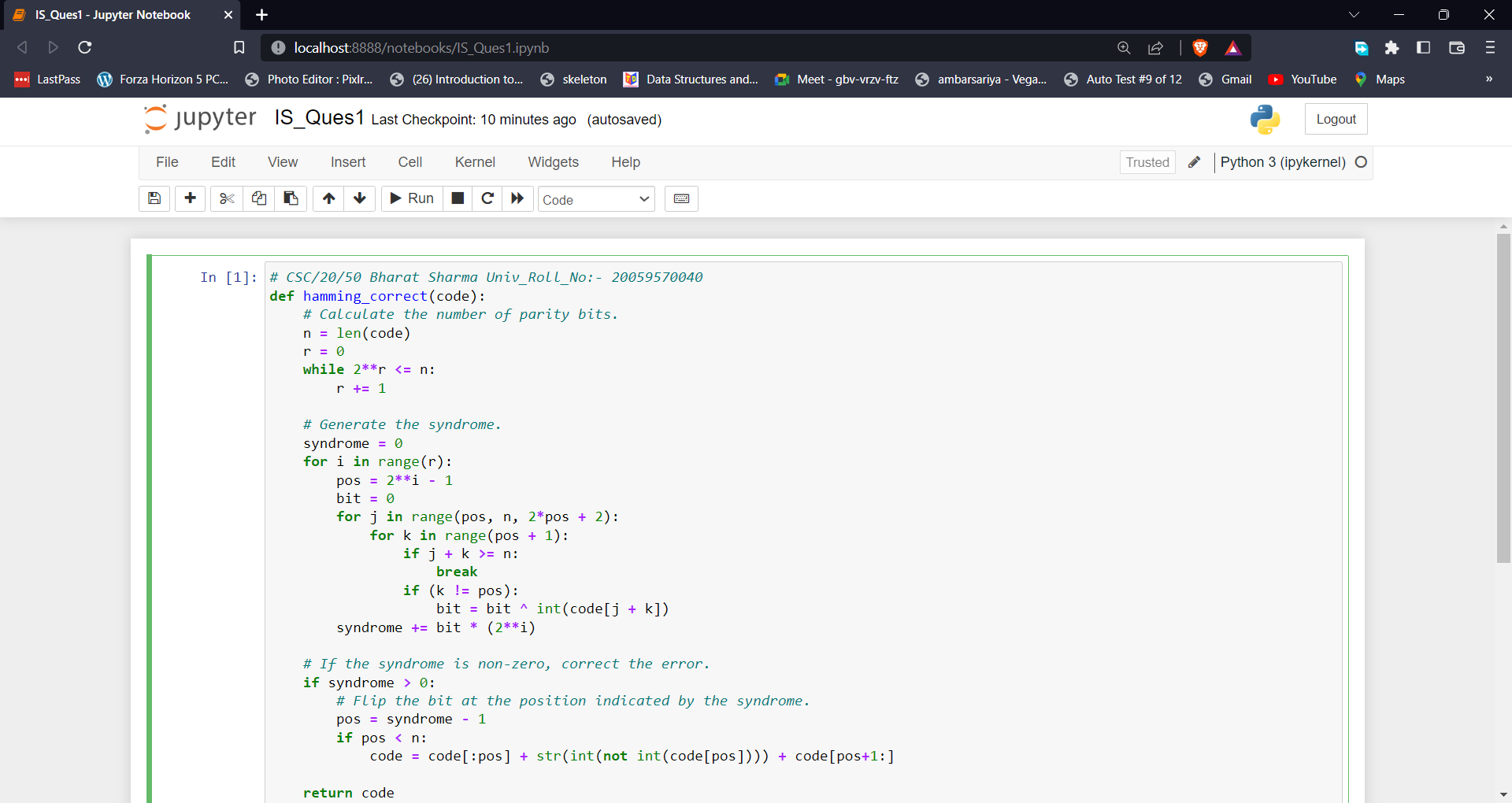
# Print the original code and the corrected code.

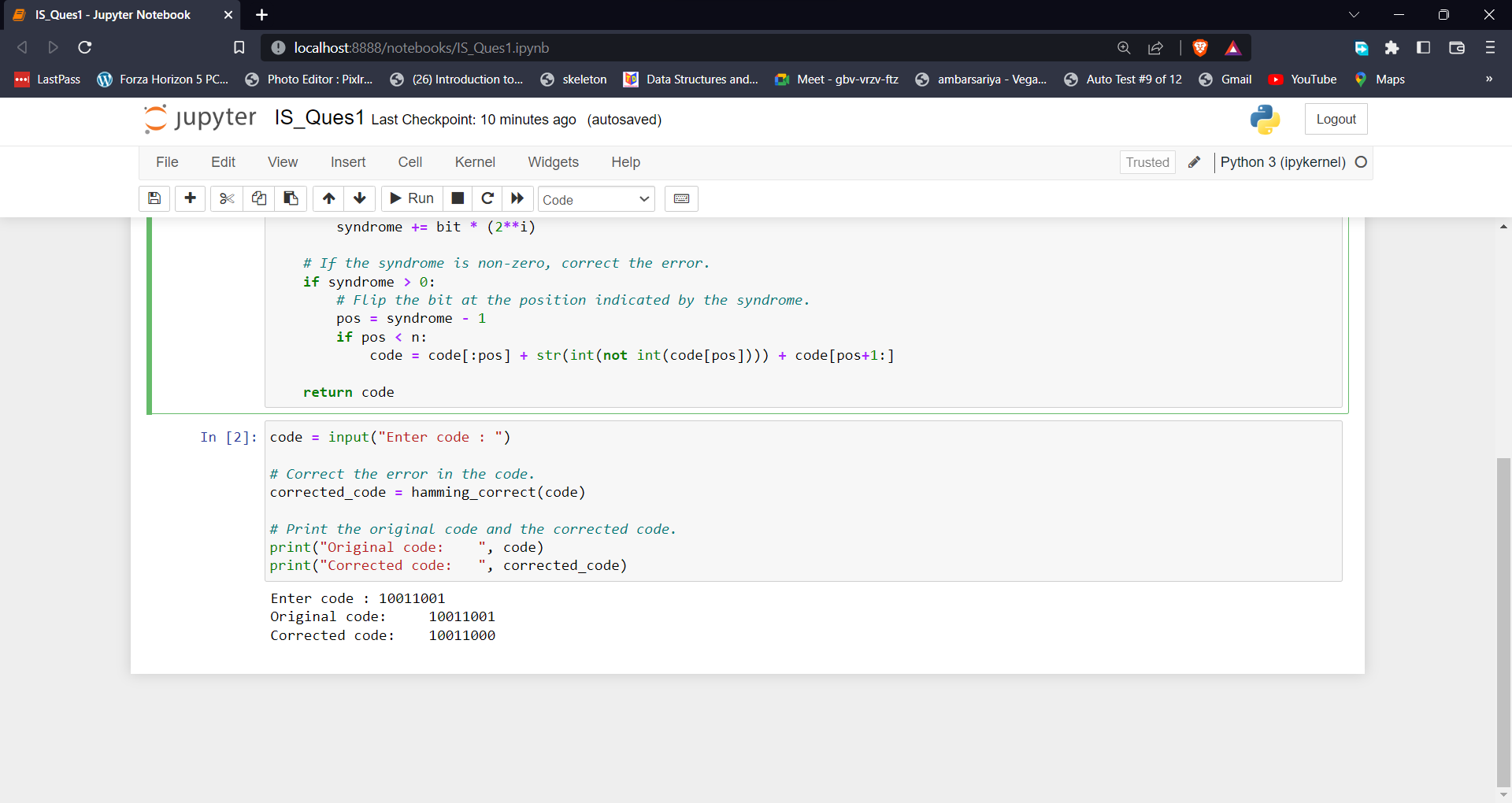
print("Original code: ", code)

print("Corrected code: ", corrected\_code)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

OUTPUT:-





**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):

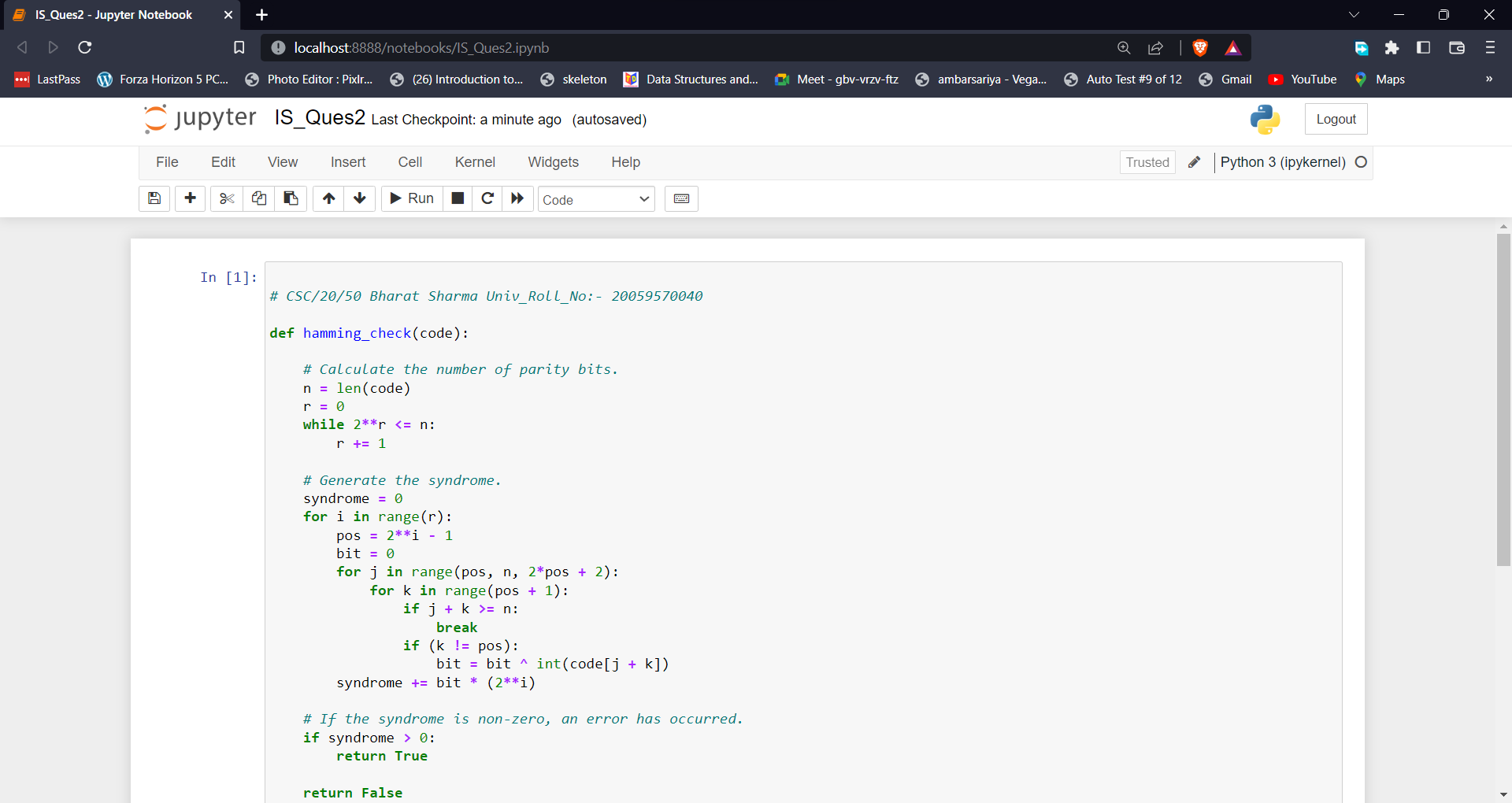
print("Errors detected!")

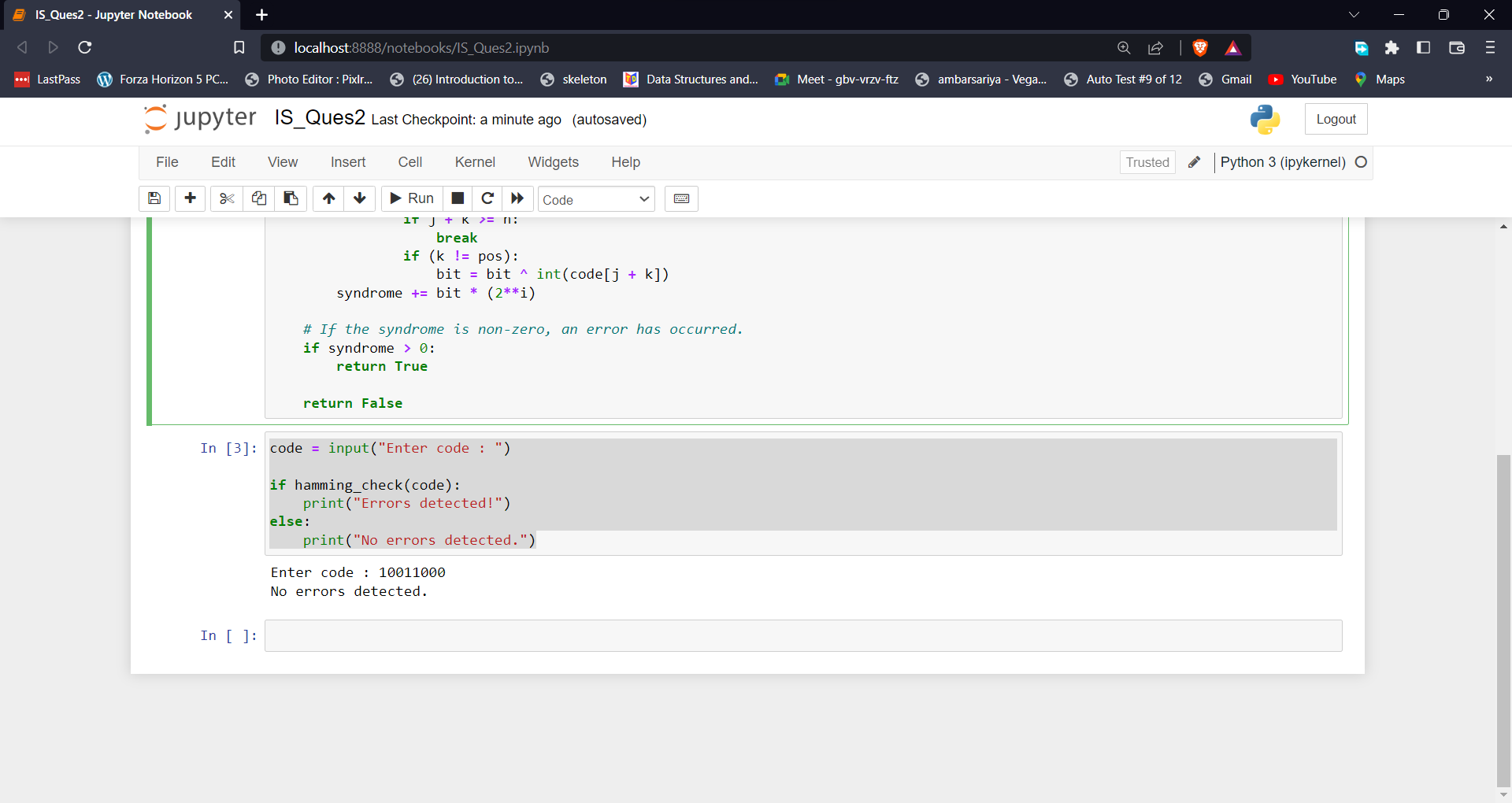
else:

print("No errors detected.")

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

OUTPUT:-





**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

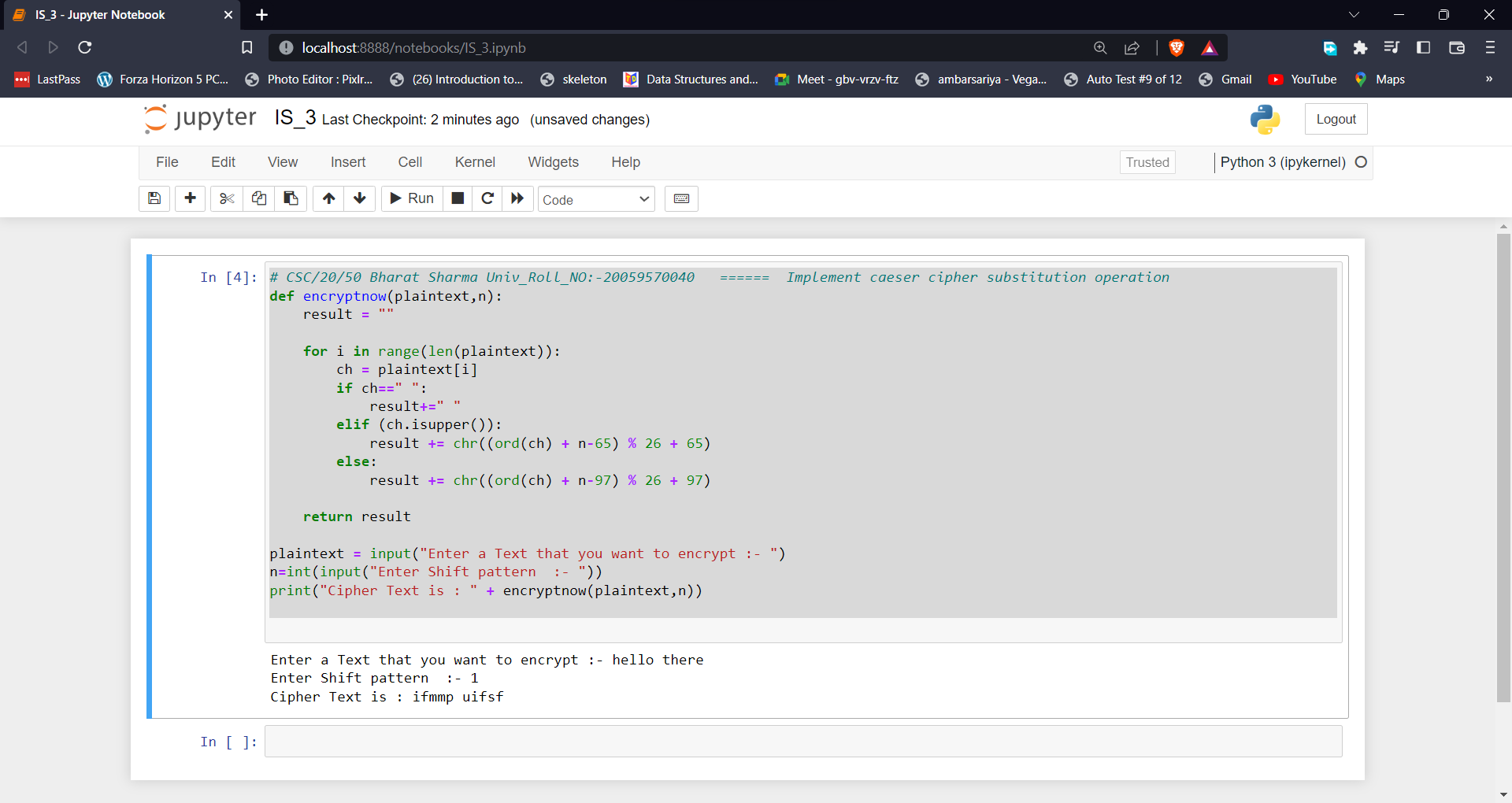
plaintext = input("Enter a Text that you want to encrypt :- ")

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))

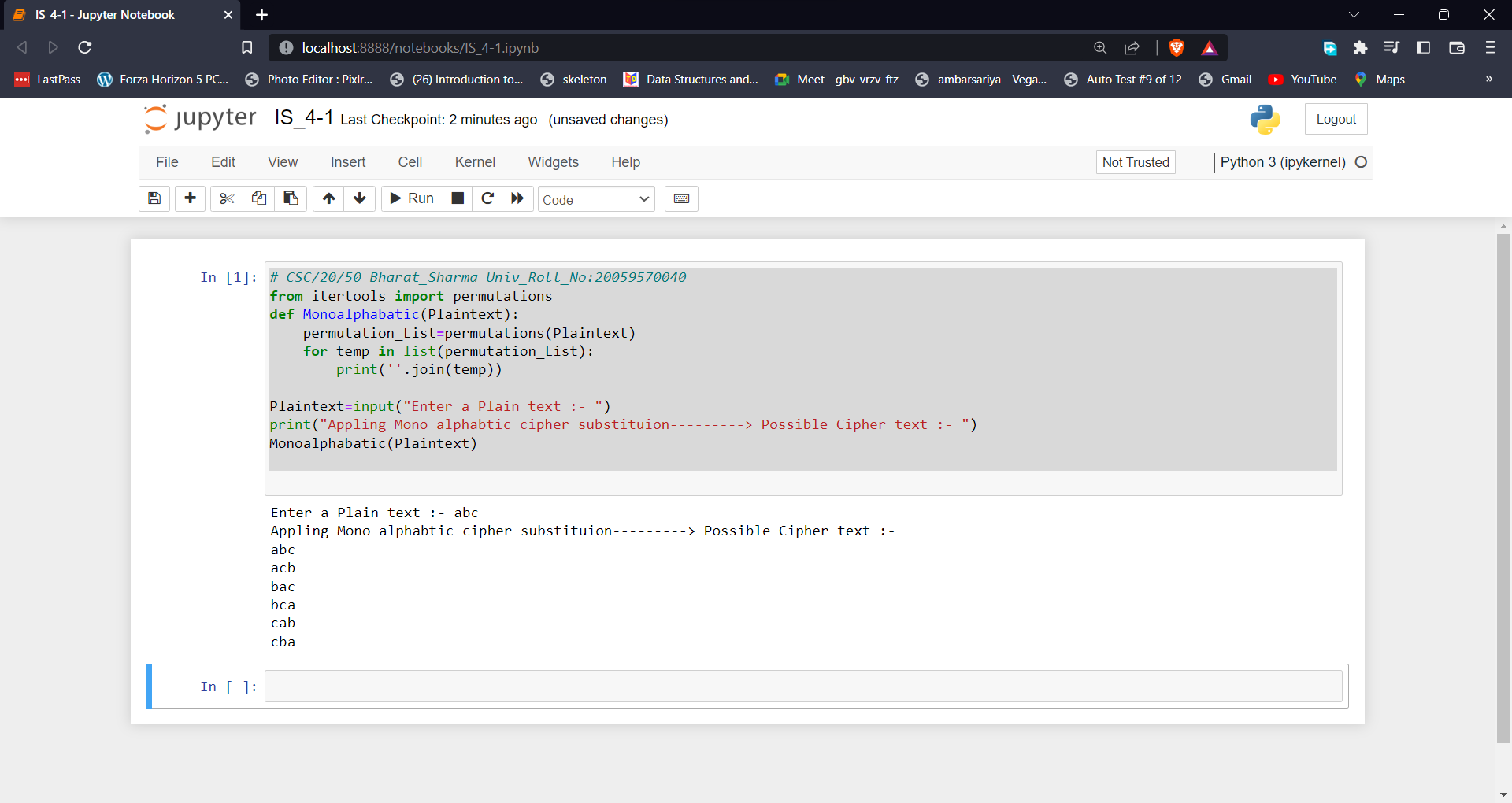
Plaintext=input("Enter a Plain text :- ")

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 :- ")

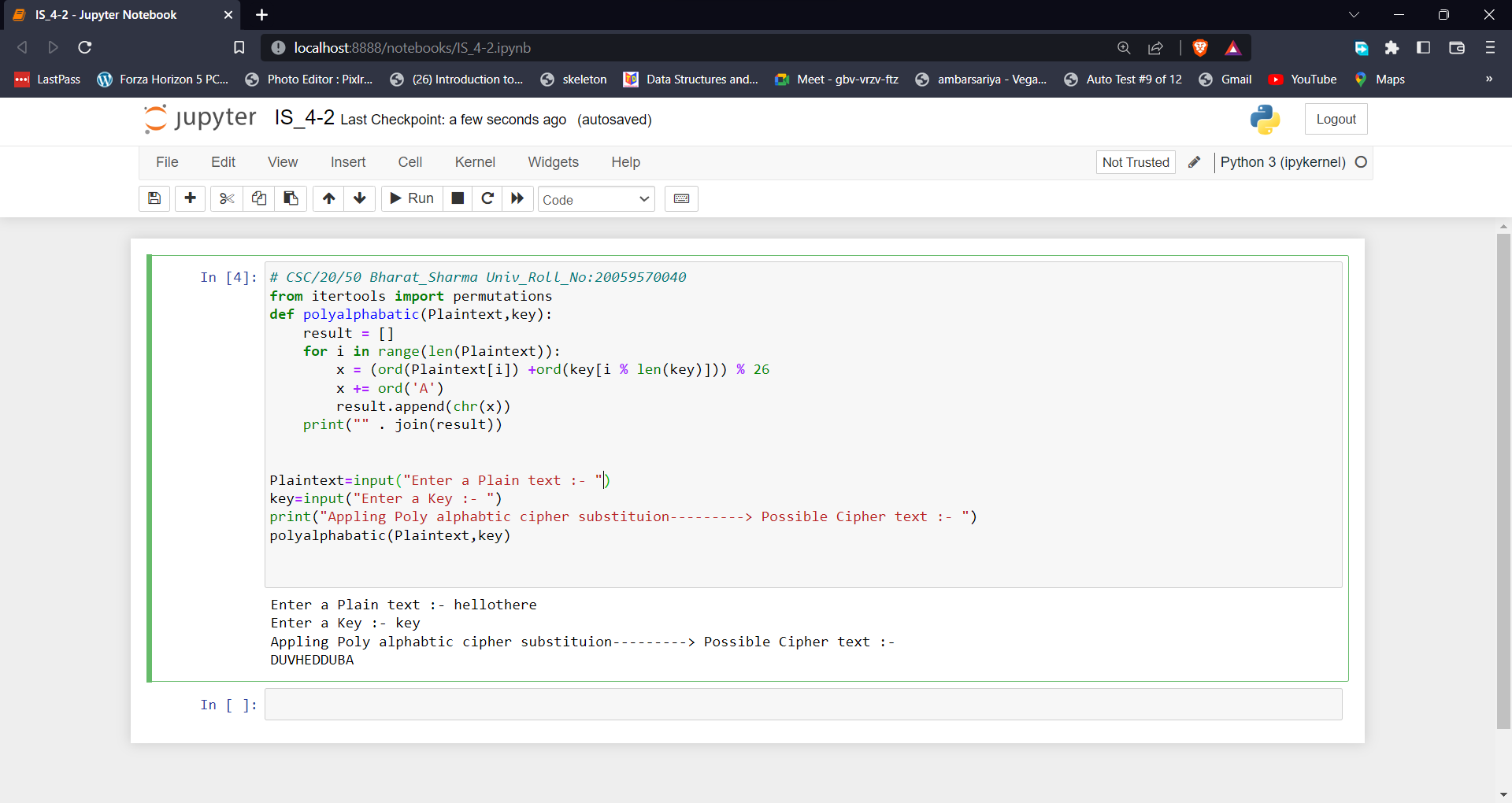
key=input("Enter a Key :- ")

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])

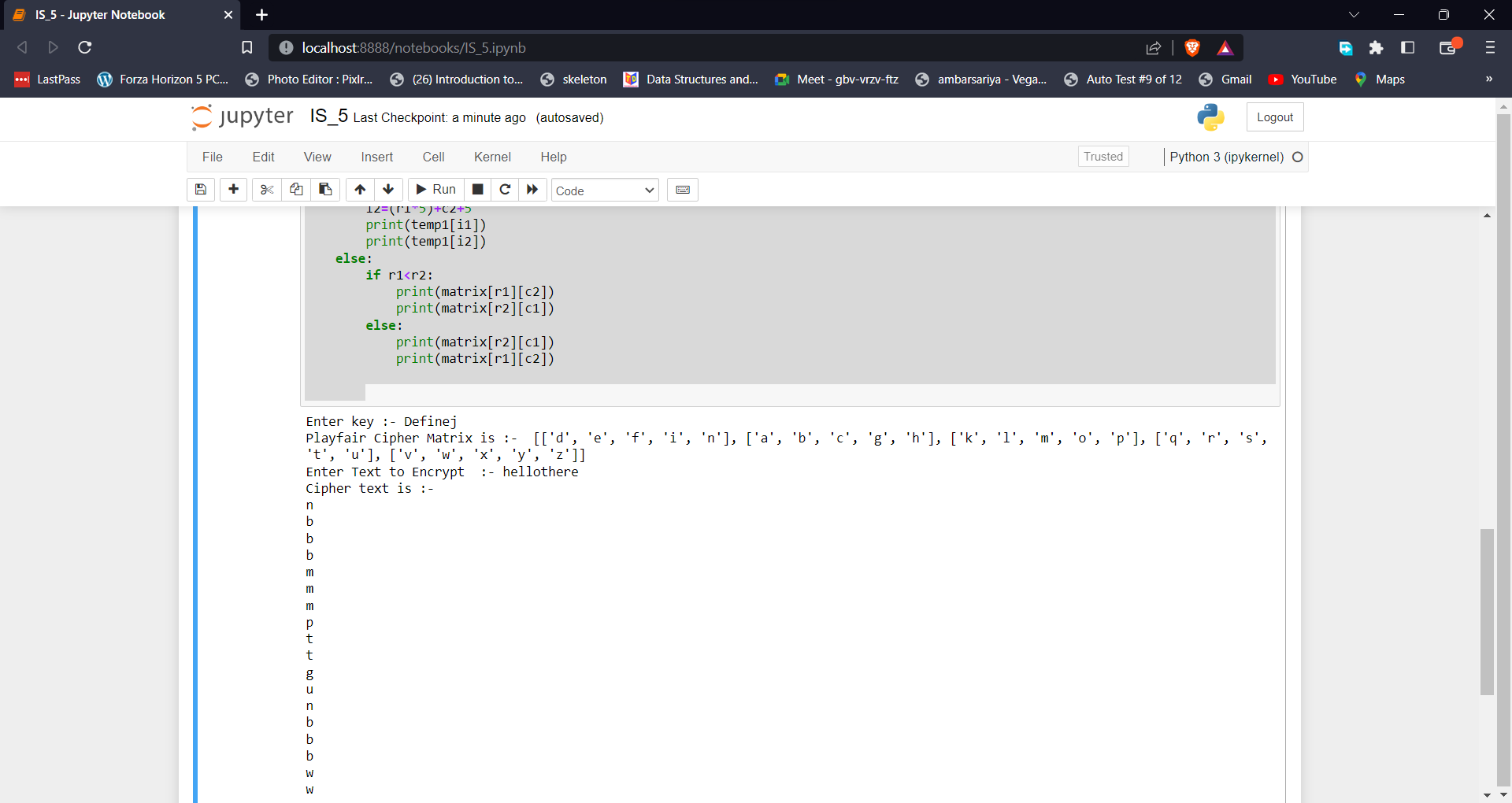
else:

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)

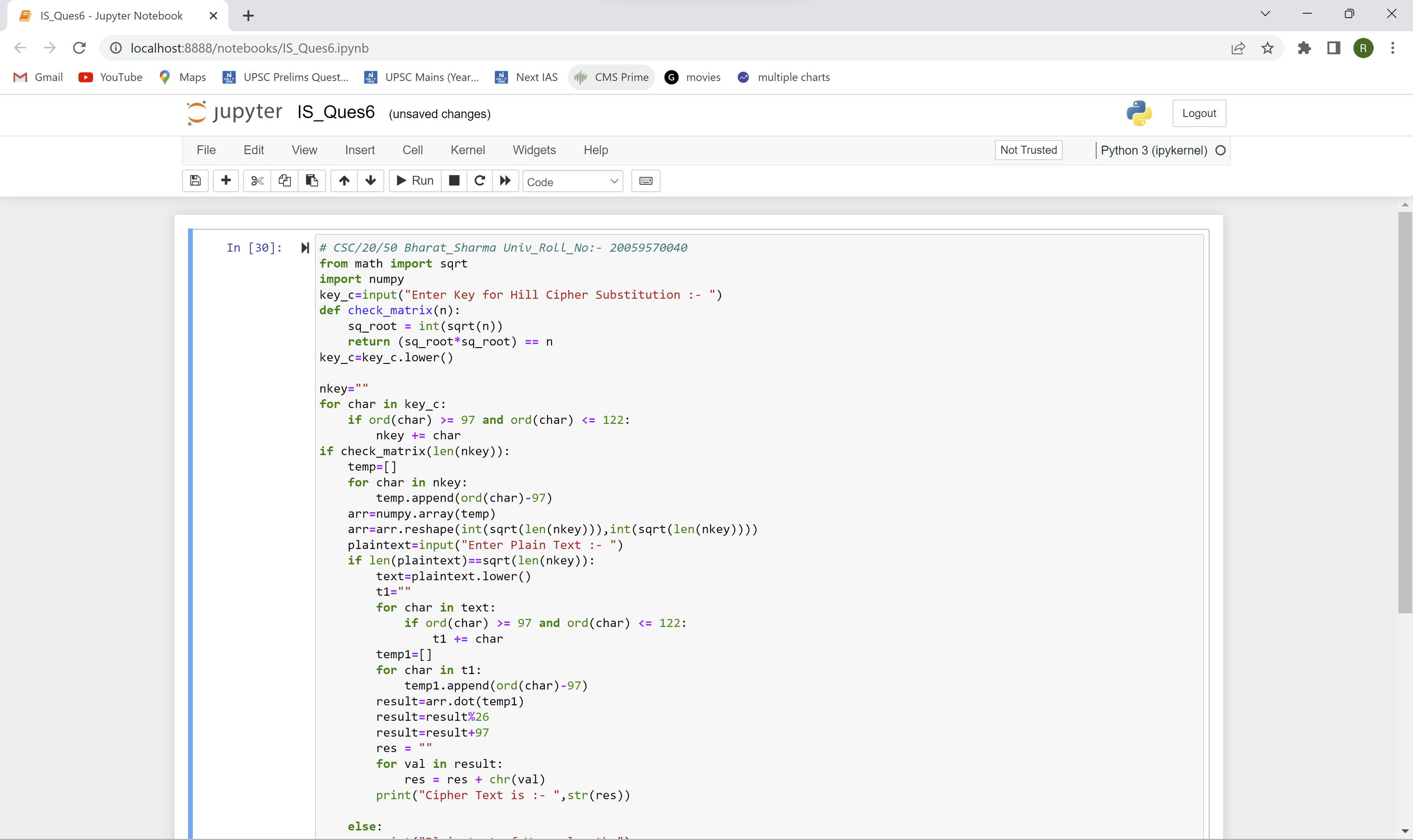
print("Cipher Text is :- ",str(res))

else:

print("Plain text of Wrong length ")

else:

print("Key is not valid ") \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

OUTPUT:-

**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

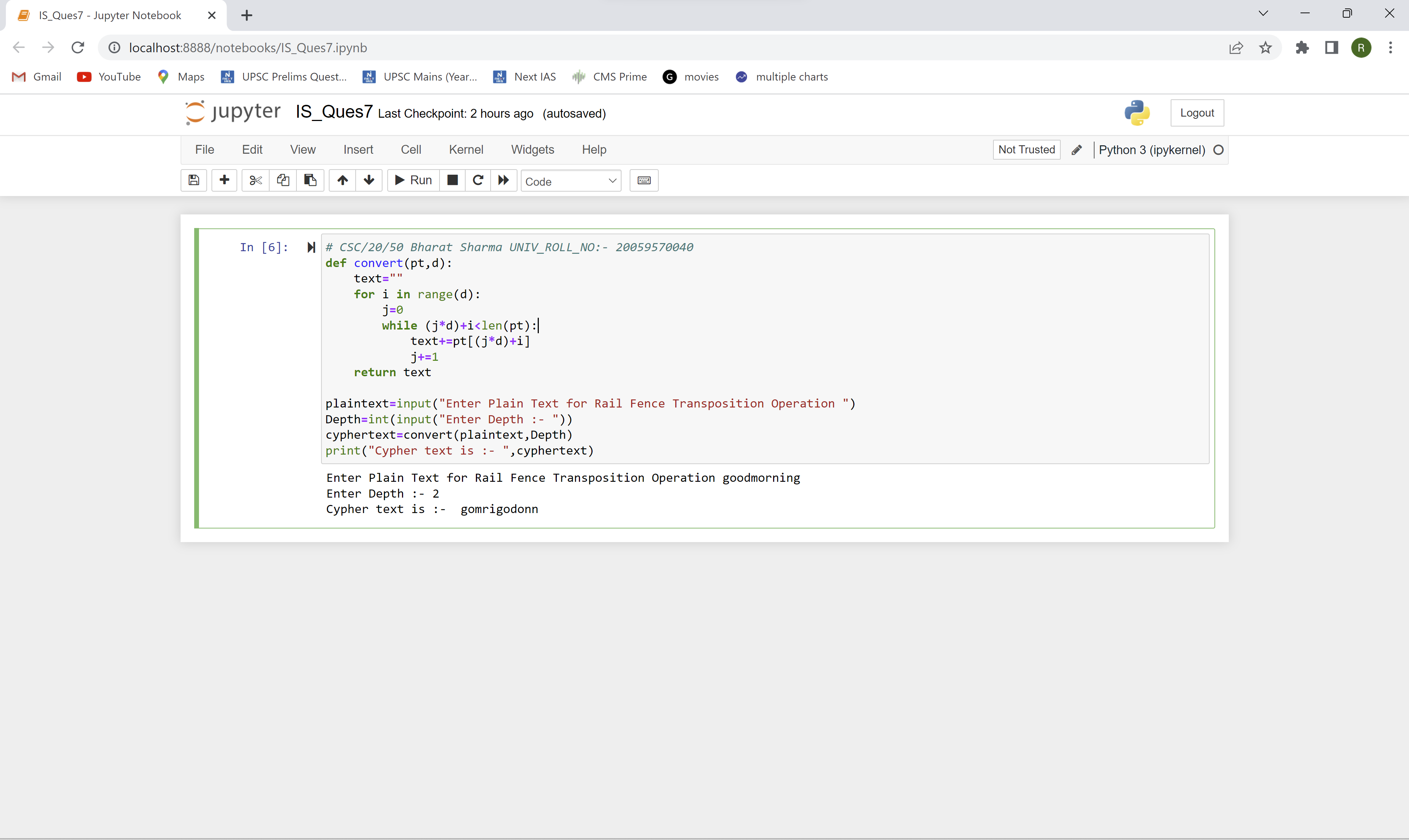
plaintext=input("Enter Plain Text for Rail Fence Transposition Operation ")

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 ")

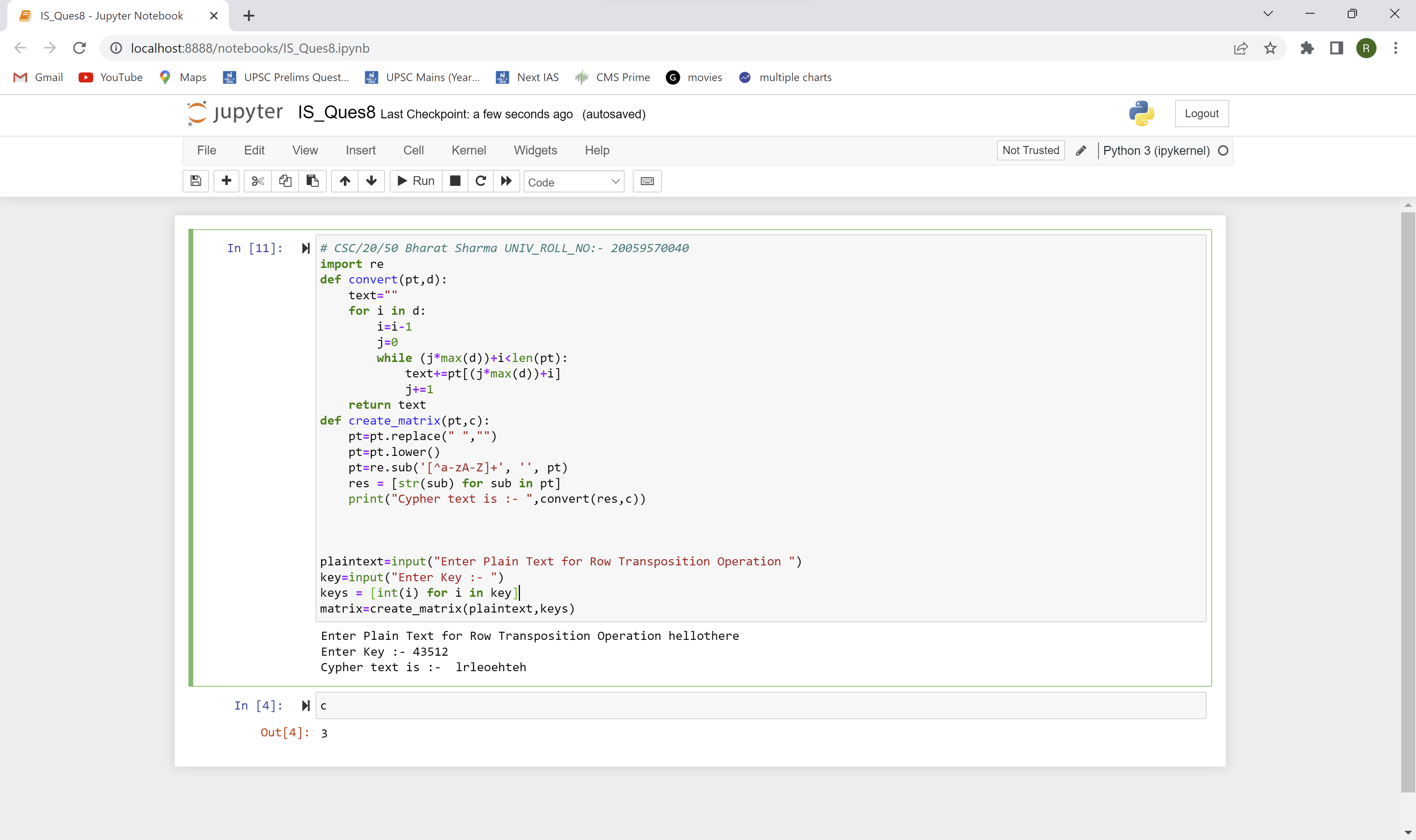
key=input("Enter Key :- ")

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 : ")

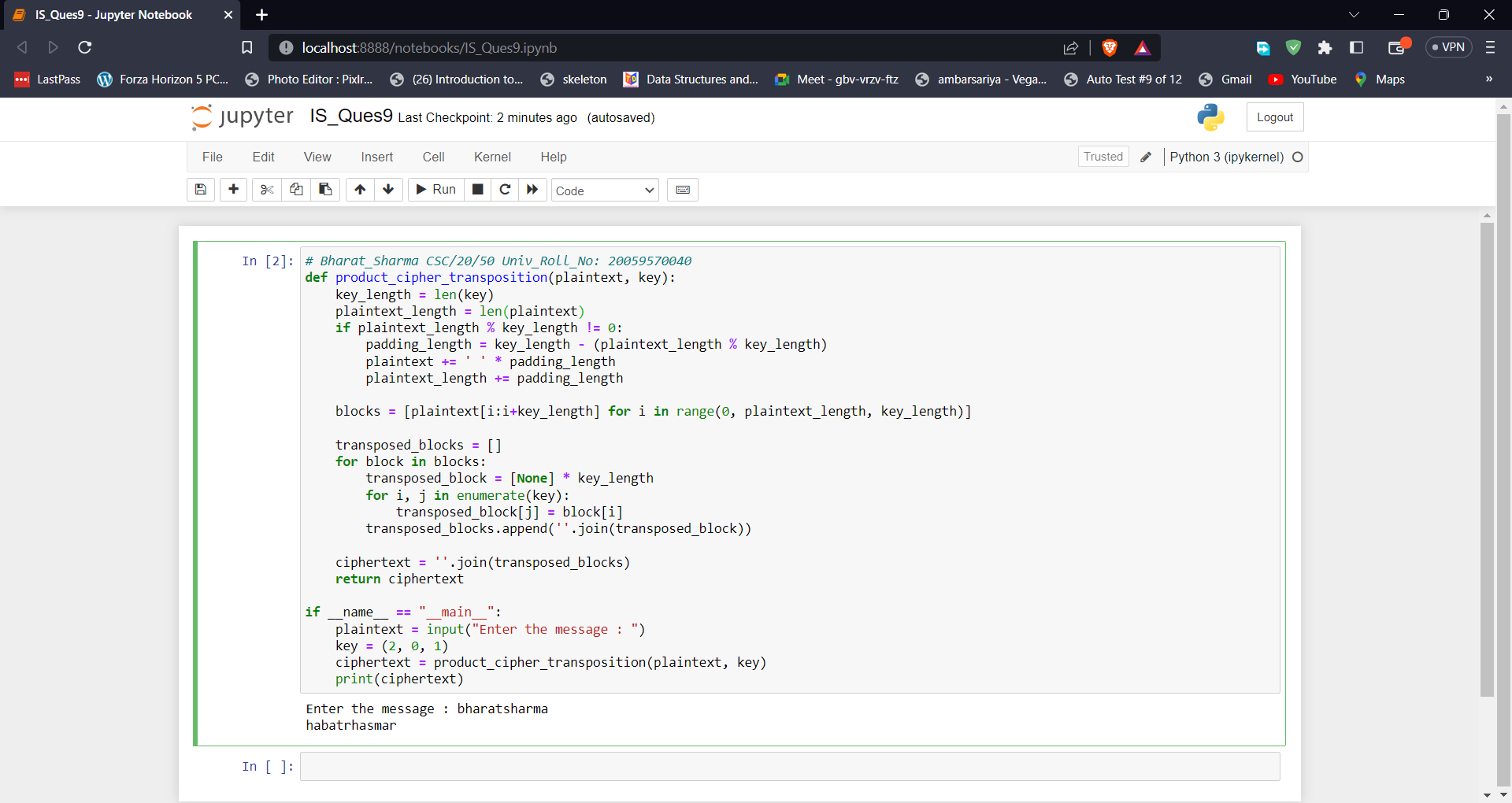
key = (2, 0, 1)

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:

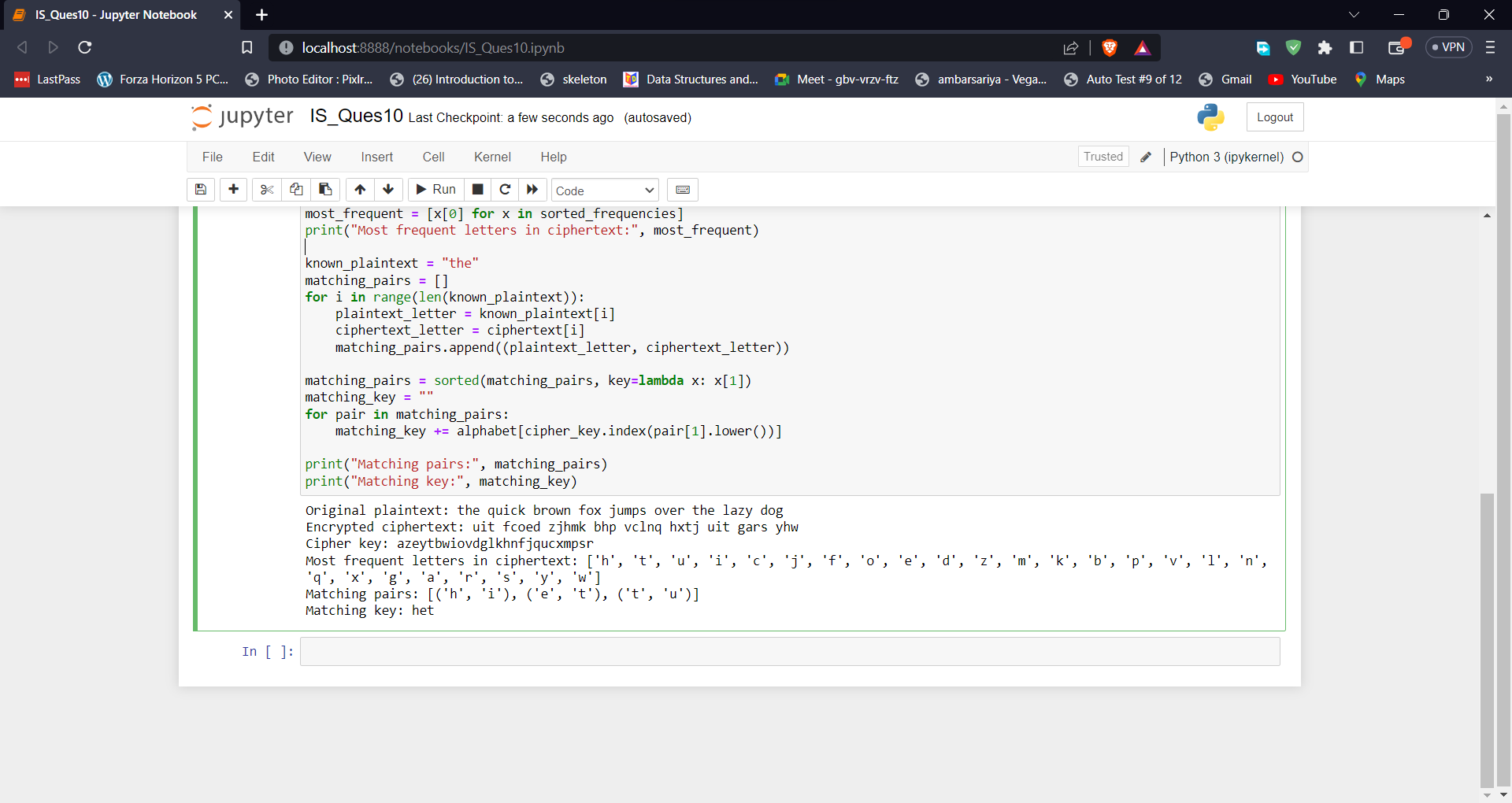
matching\_key += alphabet[cipher\_key.index(pair[1].lower())]

print("Matching pairs:", matching\_pairs)

print("Matching key:", matching\_key)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

OUTPUT:-



**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:-

