

Q1 Commands

5 Points

List the commands was used in this level?

The commands used in this level are:

go
enter
pick
c
c
back
give
back
back
thrnxtzy
read
the_magic_of_wand

We have to apply the above commands in sequence to complete this level

Q2 Cryptosystem

10 Points

What cryptosystem was used in the game to reach the password?

The cryptosystem used in this level is Monoalphabetic Substitution Cipher along with Permutation(Transposition) Cipher with the block size of length 5.

In Monoalphabetic substitution cipher what happens is a letter is always substituted by a particular letter only. For example: Suppose 'a' can be enciphered as 'd' everywhere then this is monoalphabetic cipher.

In Polyalphabetic substitution cipher what happens is a letter is substituted by different different letter. For example: For example, 'a' can be enciphered as 'd' in the starting of the text, but as 'n' at the middle.

In Permutation cipher, we just permute/rearrange the characters present in a block. The permutation is given by a key.

In this level the Permutation key is:

CipherText: 12345

Permutation : 45213

Decryption Key

This means that 1st letter of CipherText goes to the 4th letter of plaintext and 2nd letter of CipherText goes to 5th letter of plaintext and so on..

In this Level we have used Substitution Cipher along with the Permutation Cipher and here the block size is 5.

Q3 Analysis

30 Points

What tools and observations were used to figure out the cryptosystem and the password? (Explain in less than 1000 lines)

Tools Used Are:

1. Firstly with the help of python code we firstly checked if the given ciphertext is encrypted with CAESAR CIPHER or not.
2. We then with the help of python code (attached in answer 5) tried finding the Index of coincidence. With the help of which we got to know that the given text is a monoalphabetic substitution cipher.
3. We then applied the frequency analysis and then substituted the letters based on the frequency as per english language corpus. But Failed.
4. Then we used Kasiki examination to figure out the block size.
5. After this we guessed that this ciphertext "nqg_vfusr_ec_wawy" is our required password and as per our observation from previous assignment that the word before it would be the word "the password". Therefore we guessed that the word sequence "vml_lhvqpawr" is the word "the password".
6. With the help of above observation we figured out the permutation key. Then after that we applied the permutation key to decrypt the ciphertext so that only monoalphabetic substitution is left and permutation is removed.

7. We then with the help of frequency analysis and decrypted words we broke the complete ciphertext to get our password.

Observations And Analysis:

1. The first thing that came to our mind when we saw the ciphertext was that it could be Caesar cipher. Then we tried all possible shifts with the as in Caesar cipher (shift cipher) we can have utmost 26 shifts we tried out all of them. But unfortunately none of the shifts gave us any sensible text. (Code and output attached in answer 5)

2. We then with the help of python code (attached in answer 5) tried finding the Index of coincidence.

3. We got the index of coincidence as: IOC: 0.05728363111531379. We know that if the index of coincidence is around 0.0686 (source: William F. Friedman: The incidence of coincidence and its application in cryptography book) then the encryption technique which is used is monoalphabetic substitution and not polyalphabetic substitution. Therefore from here we got to know that the encryption is monoalphabetic substitution.

4. Then we did the frequency analysis. From frequency analysis we got the following results:

[['q', 0.1056338028169014], ['v', 0.10211267605633803], ['a', 0.08098591549295775], ['c', 0.07746478873239436], ['w', 0.06690140845070422], ['f', 0.06690140845070422], ['l', 0.05985915492957746], ['t', 0.045774647887323945], ['y', 0.045774647887323945], ['s', 0.03873239436619718], ['p', 0.03873239436619718], ['n', 0.035211267605633804], ['r', 0.03169014084507042], ['x', 0.028169014084507043], ['g',

0.028169014084507043], ['e', 0.02464788732394366],
['d', 0.02464788732394366], ['j', 0.02112676056338028],
['u', 0.02112676056338028], ['m',
0.017605633802816902], ['h', 0.017605633802816902],
['b', 0.01056338028169014], ['k',
0.007042253521126761], ['i', 0.0035211267605633804]]

5. We then compared this result with the english letter and then did the substitution. What we actually did to check that the given ciphertext is pure monoalphabetic substitution cipher or not we guessed that this ciphertext "nqg_vfusr_ec_wawy" is our required password and as per our observation from previous assignment that the word before it would be the word "the password". Therefore we guessed that the word sequence "vml lhvqpawr" is the word "the password". We were also optimistic for this word to be the word "the password" because in this word also we have 11 letters and in our guessed word also we have 11 letters therefore we were quite sure of our guess.

6. But when we did the substitution as per the given frequency analysis in this word we can clearly see that in the word "lhvqpawr" for this word to be the word "password" in "password" we have a double "ss" but in this word there is no repeating character so that its substitution could be "ss" therefore we concluded that it is not only monoalphabetic substitution cipher but also combination of something else.

7. We can see that if we combine the two guessed words ciphertexts we get "vml lhvgpawr". From here we saw that there is double "l" which act as a candidate for double "ss" in the word "password". Therefore from here we come to know there is permutation (transposition) cipher used in the given cipher.

8. For performing the inverse permutation cipher we

wanted to get the key of permutation cipher. But before that also we wanted to have the block length size used in the permutation cipher.

9. For finding the block length size we made use of the Kasiski Examination. Kasiski examination is a method used in cryptanalysis to break ciphers that repeat portions of plaintext. The basic idea is to find repeated sequences of characters in the ciphertext, which might correspond to repeated sequences of characters in the plaintext, thereby revealing information about the key length. The results which we got were (Code and Output attached in answer 5):

The repeated Sequences are: {'qmn': [0, 85], 'mnj': [1, 86], 'njv': [2, 87], 'flc': [13, 199], 'lvi': [27, 232], 'fvx': [30, 110], 'vxj': [31, 111], 'xja': [32, 112], 'fw': [62, 65], 'pqq': [72, 213], 'qrx': [82, 152], 'cwt': [101, 224], 'wty': [102, 225], 'uqf': [108, 165], 'fav': [127, 167], 'vea': [135, 195], 'eas': [136, 196], 'asf': [137, 197], 'bqv': [142, 202], 'tra': [181, 206]}

Distances between repeated sequences:

```
defaultdict(<class 'list'>, {85: [(0, 85), (1, 86), (2, 87)], 186: [(13, 199)], 205: [(27, 232)], 80: [(30, 110), (31, 111), (32, 112)], 3: [(62, 65)], 141: [(72, 213)], 70: [(82, 152)], 123: [(101, 224), (102, 225)], 57: [(108, 165)], 40: [(127, 167)], 60: [(135, 195), (136, 196), (137, 197), (142, 202)], 25: [(181, 206)]})
```

GCDs for repeated distances: {85: 85, 80: 80, 123: 123, 60: 60}

10. To determine the size of the key using Kasiski examination, you need to analyze the GCDs (greatest common divisors) of the repeated distances between sequences in the ciphertext. The most common factors among the distances between repeated patterns can

indicate the possible key lengths.

11. From the repeated distances GCDs and repeated distances value the most common factors are: 1, 2, 4, 5, But actually 2 is the factor for only 80 & 60. Similarly 4 is a factor of only 80 & 60. Although 5 is a factor of 85, 80 & 60. Similarly, 1 is a factor of many but taking block length as 1 is not good as then we won't be able to do the permutation.

12. Therefore since 5 is the factor of majority of the values therefore it is best to use 5 as our block size.

13. Now we divide the given ciphertext in 5-5 letters blocs also we remove all the punctuation marks for now. The result which we get is:

"qmnjv sanvw ewcfl ctvpr jtjtv vplvl fvxja vqild hcxm
nvcna cyclp afcgy tvfw fwwgq yppqq pqcsy wsqrx qmnjv
afycg vtlvh fcwty laeuq fvxja tkbvc qnsqs lhfav awncc
veasf uqbqv qtcyl lrqrx xwacf ypsdc uqfav rqcge fppya
ttrac xwvta awwdd veasf lcbqv dtraw mvupq quwxd
ecgqc wtyqy aflvl qsyqk lhqsn afqvm llhvq pawrn qgvfu
srecw awyqp fnwga wdgf"

14. Now once we have decided our block size now we have to find out our key used in permutation. For determining the key of the cipher text we take the ciphertext of our guessed word (the password) which is:
afqvm llhvq pawrn
***th epass word*

-> From here we can see that here we have "ll" these should correspond to ""ss" therefore we get the key of decryption key of permutation cipher mapping as:

CipherText : 12345

Plaintext: 45***

Or

Plaintext: 54***

->Therefore we also have a mapping of l in ciphertext correspond to s in plaintext. therefore we have the mapping l->s. Also since "pawrn" correspond to "word*". Therefore we have a mapping of either p maps to d or a maps to d.

15. In the English language frequency distribution table we see that d is at 10 in English vocabulary and as per our frequency analysis a is at 3rd whereas p is at 11th therefore p should correspond to d. Therefore we have p->d.

16. Now since p->d now our key become 45***. Now since we know that afqvm=>***th. Therefore now since the key is 45***. Therefore we get a mapping a->t and f->h.

17. Now from frequency analysis we see that in ciphertext v,q occur very very high times therefore they can't be p of the English as p occur very less frequently in English there we get a mapping of h->p (we are currently observing the word llhvq->epass)

18. Now from frequency analysis we see that w occurs at 6th position and in the English vocabulary o occurs at 4th also a is already moved to last position and r,n are very less to be considered for o as their map therefore w maps to o. (We are observing pawrn->word*) Because of this our permutation key also become => 452**

This means that 1st letter of CipherText goes to the 4th letter of Plaintext and so on..

19. Now for permutation Decryption key we have 2 options:

Key: 45231

Or

Key: 45213

20. Now if we follow 45213 as our permutation key we have $e \rightarrow v$ and $q \rightarrow s$

From here we get mapping as $v \rightarrow e$ and $q \rightarrow a$

Also then we get the mapping

$R \rightarrow w$

$N \rightarrow r$

From word \rightarrow rondt

If we follow same approach for the permutation key 45231 we will get a mapping of $p \rightarrow p$ from the decrypted cipher text which is not quite likely therefore we discard the mapping 45231 and our correct permutation decryption mapping is 45213.

21. Therefore in this way our Permutation Decryption key comes out to be 45213.

CipherText: 12345

Permutation: 45213

Decryption Key

This means 1st letter of CipherText goes to 4th letter of PlainText, 2nd letter of CipherText goes to 5th letter of PlainText, 3rd letter of CipherText goes to 2nd letter of PlainText, 4th letter of CipherText goes to 1st letter of PlainText and 5th letter of CipherText goes to 3rd letter of PlainText. Similarly the Permutation Encryption Key is (43512)

22. If we apply the currently found mappings and we also apply the permutation key then the result which we get is:

"jREAmER Os THcS eODE Wctt jE jtESSED jx THE SidEAmx

SPcRcT REScDcyg cy THE HOtE. gO AHEAD, AyD scyD A
 WAX Os jREAMcyg THE SPett Oy Hcu eAST jx THE Ebct
 kAssAR. THE SPcRcT Os THE eAbE uAy cS AtWAXS WcTH
 xOd. scyD THE uAgce WAYD THAT Wctt tET xOd OdT Os
 THE eAbES. cT WOdtd uAmE xOd A uAgcecy, yO tESS
 THAy kAssAR! SPEAm THE PASSWORD
 THE_uAgce_Os_WAYD TO gO THROdgH."

Here capital letters have been decrypted and small letters are still to be decrypted

23. Now since we have de-permuted the ciphertext now the ciphertext has just become a monochromatic substitution cipher text which can be solved by frequency analysis.

24. Now we see various words which are almost decrypted. Lets see "THcS" here we can see that the word is "THIS" therefore c maps to i, c->i. Similarly, here "THE SPIRIT Os" s should map to f, s->f.

25. Similarly, here "RESIDIyg Iy" the word should be "RESIDING IN" therefore y maps to n, y->n and also g maps to g, g->g

26. Similarly, here "WOdtD" the word is "WOULD" therefore we get the mapping as d maps to u implying d->u and t maps to l which implies t->l.

27. Similarly, here "jE jLESSED" the word is "BE BLESSED" therefore j maps to b implying j->b. Similarly, here "WILL BE BLESSED Bx" is "WILL BE BLESSED BY" therefore x maps to y implying x->y.

28. Similarly, here "BREAmING THE SPELL ON HIu" is "BREAKING THE SPELL ON HIM" therefore we get the mapping as m maps to k implying m->k and also we get the mapping as u maps to m implying u->m.

29. Similarly, here "MAGIe" is "MAGIC" therefore e maps to c implying e->c. Similarly, here "CAbES" is "CAVES" therefore b maps to v implying c->v. Similarly, here "SiUEAKY" is "SQUEAKY" therefore i maps to q implying i->q.

30. Similarly, here "IT WOULD MAKE YOU A MAGICIAN, NO LESS THAN kAFFAR!" is "IT WOULD MAKE YOU A MAGICIAN, NO LESS THAN JAFFAR!" Therefore k maps to j implying k->j. Jaffar is actually a fictional character in Walt Disney Pictures' animated film Aladdin (1992). Jafar was a evil magician the Alladin picture.

31. Therefore our final plaintext is:

"breaker of this code will be blessed by the squeaky spirit residing in the hole. go ahead, and find a way of breaking the spell on him cast by the evil jaffar. the spirit of the cave man is always with you. find the magic wand that will let you out of the caves. it would make you a magician, no less than jaffar! speak the password the_magic_of_wand to go through."

32. Our Mapping obtained is:

```
{  
'a':>'t'  
'b':>'v'  
'c':>'i'  
'd':>'u'  
'e':>'c'  
'f':>'h'  
'g':>'g'  
'h':>'p'  
'i':>'q'  
'j':>'b'  
'k':>'j'  
'l':>'s'
```

'm': 'k'
'n': 'r'
'o': (No Mapping Found)
'p': 'd'
'q': 'a'
'r': 'w'
's': 'f'
't': 'l'
'u': 'm'
'v': 'e'
'w': 'o'
'x': 'y'
'y': 'n'
'z': (No Mapping Found)
'A': 'T'
'B': 'V'
'C': 'I'
'D': 'U'
'E': 'C'
'F': 'H'
'G': 'G'
'H': 'P'
'I': 'Q'
'J': 'B'
'K': 'J'
'L': 'S'
'M': 'K'
'N': 'R'
'O': (No Mapping Found)
'P': 'D'
'Q': 'A'
'R': 'W'
'S': 'F'
'T': 'L'
'U': 'M'
'V': 'E'
'W': 'O'
'X': 'Y'

```
'Y': 'N'  
'Z': (No Mapping Found)  
0->0  
1->1  
2->2  
3->3  
4->4  
5->5  
6->6  
7->7  
8->8  
9->9  
space->space  
' '->' '  
'!'->'!  
'.'->'.'  
'_'->'_  
'?'->'?  
'*'->'*'  
'.'->'.'  
}
```

33. The required password is "the_magic_of_wand" without the quotes.

Q4 Password

5 Points

What was the final command used to clear this level?

the_magic_of_wand

Q5 Codes

0 Points

Upload any code that you have used to solve this level.



Download

SecureSavants_ModernCrypto_Assignment3.ipynb

In []:

```
#Index Of Coincidence
alph =
"abcdefghijklmnopqrstuvwxyz"

def isLetter(char):
    return (char in alph)

def countLetters(text):
    count = 0
    for i in text:
        if(isLetter(i)):
            count += 1
    return count

def getIOC(text):
    letterCounts = []

    # Loop through each
    letter in the alphabet - count
    number of times it appears
    for i in
    range(len(alph)):
        count = 0
        for j in text:
            if j ==
            alph[i]:
                count += 1
        letterCounts.append(count)

    # Loop through all
    letter counts, applying the
    calculation (the sigma part)
    total = 0
```

```
        for i in
range(len(letterCounts)):
            ni =
letterCounts[i]
            total += ni *
(ni - 1)

        N = countLetters(text)
        c = 26.0 # Number of
letters in the alphabet
        total = float(total) /
((N * (N - 1)))
        return total

t = "qmnjvsa nv wewc flct vprj
tj tvvplvl fv xja vqildhc xmlnvc
nacyclpa fc gyt vfvw. fv wgqyp,
pqg pqcs y wsq rx qmnjvafy cg
tlvhf cw tyl aeug fv xja tkbv
cqnsqs. lhf avawnc cv eas fuqb
qvq tc yllrqr xxwa cfy. psdc uqf
avrqc gefq pyat trac xwv taa wwd
dv eas flcbq. vd trawm vupq quw
x decgqwt, yq yafl vlqs yqklhq!
snafq vml lhvqpawr
nqg_vfusr_ec_wawy qp fn
wgawdgf."
text = t.lower()
total = getIOC(text)
print("IOC: " + str(total))
print("Normalised IOC: " +
str(total * 26.0))
```

IOC: 0.05728363111531379

Normalised IOC: 1.4893744089981584

In []:

```

#Substitution Cipher

def
substitution_cipher(text,subs_val):
    decrypted_text = ""
    for char in text:
        if char.isalpha():
            # Check if the
character is lowercase
            if char.islower():
                decrypted_text +=
chr((ord(char) - ord('a') +
subs_val) % 26 + ord('a'))
            # Check if the
character is uppercase
            elif char.isupper():
                decrypted_text +=
chr((ord(char) - ord('A') +
subs_val) % 26 + ord('A'))
            else:
                decrypted_text += char
    return decrypted_text

for i in range(1,26):
    print("Substitution Value: ",i)
    Ciphertext = "qmnjvsa nv wewc
flct vprj tj tvvplvl fv xja vqildhc
xmlnvc nacyclpa fc gyt vfvw. fv
wgqyp, pqg pqcs y wsq rx qmnjvafy
cgv tlvhf cw tyl aeug fv xja tkbv
cqnsqs. lhf avawnc cv eas fuqb qvq
tc yllrqr xxwa cfy. psdc uqf avrqc
gefq pyat trac xwv taa wwd dv eas
flcbq. vd trawm vupq quw x
decgqcwt, yq yafl vlqs yqklhq!
snafq vml lhvqpawr
nqg_vfusr_ec_wawy qp fn wgawdgf."
    print("CipherText: ", Ciphertext)
    decrypted_text =
substitution_cipher(Ciphertext,i)
    print("Decrypted:
",decrypted_text)

```


Substitution Value: 1
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: rnokwtb ow xfxd gmdu wqs
Substitution Value: 2
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: soplxuc px ygge hnev xrt
Substitution Value: 3
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: tpqmyvd qy zhzf io fw ysu
Substitution Value: 4
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: ugrnzwe rz aiag jpgx ztv
Substitution Value: 5
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: vrsoaxf sa bjbh kqhy auw
Substitution Value: 6
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: wstpbyg tb ckci lriz bvx
Substitution Value: 7
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: xtugczh uc dldj msja cwy
Substitution Value: 8
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: yuvrdai vd emek ntkb dxz
Substitution Value: 9
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: zvwsebj we fnfl oulc eya
Substitution Value: 10
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: awxtfck xf gogm pvmd fzb
Substitution Value: 11
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: bxyugd l yg hphn qwne gac
Substitution Value: 12
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: cyzvhem zh iqio rxof hbc
Substitution Value: 13
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: dzawifn ai jrjp sypg ice
Substitution Value: 14
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: eabxjgo bj kskq tzqh jdf
Substitution Value: 15
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: fbcykhp ck ltlr uari keg

Substitution Value: 16
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: gcdzliq dl mums vbsj lfh
Substitution Value: 17
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: hdeamjr em nvnt wctk mgi
Substitution Value: 18
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: iefbnks fn owou xdul nhj
Substitution Value: 19
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: jfgcolt go pxpv yevm oik
Substitution Value: 20
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: kghdpmu hp qyqw zfwn pjl
Substitution Value: 21
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: lhieqnv iq rzrx agxo qkn
Substitution Value: 22
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: mijfrow jr sasy bhyp rln
Substitution Value: 23
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: njkgspk ks tbtz cizq smc
Substitution Value: 24
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: oklhtqy lt ucuu djar tnp
Substitution Value: 25
CipherText: qmnjvsa nv wewc flct vpr
Decrypted: plmiurz mu vdvb ekbs uoc

In []:

```

#Program to remve whitespaces and
punctuation marks from a given string

import string

def
remove_spaces_and_punctuation(input_s
    # Create a translation table to r
punctuation
    translation_table = str.maketrans
'', string.punctuation)

    # Remove spaces
    input_string = input_string.repla
    "")

    # Remove punctuation using transla
table
    input_string =
input_string.translate(translation_ta

    return input_string

# Test the function
input_string = "qmnjvsa nv wewc flct
tvvplvl fv xja vqildhc xmlnvc nacyclp
gyt vfvw. fv wgqyp, pqg pqcs y wsq rx
qmnjvafy cg v tlvhf cw tyl aeug fv xja
cqnsqs. lhf avawnc cv eas fuqb qvq t
yllrqr xxwa cfy. psdc uqf avrqc gefq
trac xwv taa wwd dv eas flcbq. vd tra
quw x decgqwt, yq yafl vlqs yqklhq!
vml lhvqpawr nqg_vfusr_ec_wawy qp fn
wgawdgf."
result =
remove_spaces_and_punctuation(input_s
print("Input string:", input_string)
print("Result:", result)

```

Input string: qmnjvsa nv wewc flct vp
Result: qmnjvsanvwewcflctvprjtjttvvplv

In []:

```

#KasisKi Examination used for finding

```

```

block length of the permutation cipher

from collections import defaultdict
import math

def find_repeated_sequences(text,
min_length=3):
    repeated_sequences =
defaultdict(list)
    for i in range(len(text) - min_length
+ 1):
        sequence = text[i:i + min_length]
        if len(sequence) < min_length:
            continue

    repeated_sequences[sequence].append(i)
    return {seq: indices for seq, indices
in repeated_sequences.items() if
len(indices) > 1}

def calculate_distances(indices):
    distances = defaultdict(list)
    for indices_list in indices.values():
        for i in range(len(indices_list)
- 1):
            for j in range(i + 1,
len(indices_list)):
                distances[indices_list[i]
-
indices_list[j]].append((indices_list[i],
indices_list[j]))
    return distances

def find_gcd(distances):
    gcds = {}
    for distance, pairs in
distances.items():
        if len(pairs) > 1:
            gcd = math.gcd(pairs[0][0] -
pairs[0][1], pairs[1][0] - pairs[1][1])
            for i in range(2,
len(pairs)):
                gcd = math.gcd(gcd,
pairs[i][0] - pairs[i][1])
            gcds[distance] = gcd

```

```
        return gcds

def kasiski_examination(text):
    repeated_sequences =
find_repeated_sequences(text)
    print("The repeated Sequences are
",repeated_sequences)
    distances =
calculate_distances(repeated_sequences)
    print("Distances between repeated
sequences: ",distances)
    gcds = find_gcd(distances)
    return gcds

# Example usage:

result_kasiski =
kasiski_examination(result)
print("GCDs for repeated distances:",
result_kasiski)
```

```
The repeated Sequences are: {'qmn':
Distances between repeated sequences:
GCDs for repeated distances: {85: 85,
```

In []:

```
#FREQUENCY ANALYSIS

cipher_text=result
letter_list=[]
letter_freq_list=[]
count2=0
for l in cipher_text:
    count2=count2+1
for letter_cipher in cipher_text:
    if(letter_cipher in
letter_list):
        continue
    elif(letter_cipher=='.' or
letter_cipher=="'" or
letter_cipher==" " or
letter_cipher.isdigit() or
letter_cipher=="!" or
letter_cipher=="","):
        continue
    else:

letter_list.append(letter_cipher)
    temp=[]
    count= count_of_a =
cipher_text.count(letter_cipher)

temp.append(letter_cipher)
    temp.append(count/count2)

letter_freq_list.append(temp)

sorted_list =
sorted(letter_freq_list,
key=lambda y: y[1], reverse=True)
print(sorted_list)
print("\n")
```

```
[['q', 0.1056338028169014], ['v', 0.1
```

In []:

```
#Dividing the given string in blocks
letters each

def block_division_ciphertext(input_string):
    input_string = ' '.join(input_string)
    for i in range(0, len(input_string),
        return input_string

block_divided_string=block_division_c
print("String divided in form of block
block_divided_string)
print("\n")
```

String divided in form of blocks: qm

In []:

```
#Permutation Used 5,3,4,1,2

def
replace_characters(input_string):
    # Replace 'q' with 'e'
    input_string =
input_string.replace('l', 's')
    input_string =
input_string.replace('p', 'd')
    input_string =
input_string.replace('h', 'p')
    input_string =
input_string.replace('w', 'o')
    input_string =
input_string.replace('a', 't')
    input_string =
input_string.replace('f', 'h')
    input_string =
input_string.replace('v', 'a')
    input_string =
input_string.replace('q', 'e')
    # input_string =
input_string.replace('r', 'w')
    # input_string =
```

```
input_string.replace('n', 'r')
    return input_string

def permute_word(word):
    # Check if word length is not
    equal to 5
    if len(word) == 4:
        x=word[1]
        return
    word[0]+word[2]+x+word[3]

    # Permute the word according
    to the given permutation: 5, 3,
    4, 1, 2
    return word[4] + word[2] +
    word[3] + word[0] + word[1]

def permute_string(input_string):
    # Split the input string into
    words
    words = input_string.split()

    # Permute each word in the
    list of words
    permuted_words =
    [permute_word(word) for word in
    words]

    # Join the permuted words
    back into a string
    permuted_string = '
    '.join(permuted_words)

    return permuted_string
# Test the function
input_string =
block_divided_string
result =
replace_characters(input_string)
result_final =
permute_string(result)
print("Input string:",
input_string)
print("Result:", result_final)
```



```
print("\n")
```

Input string: qmnjv sanvw ewcfl ctvpr
Result: anjem onast scheo radct ajtjt

In []:

```
# Permutation Used 4,5,2,1,3

def replace_characters(input_string):
    # Replace 'q' with 'e'
    input_string = input_string.replace(
'S')
    input_string = input_string.replace(
'D')
    input_string = input_string.replace(
'P')
    input_string = input_string.replace(
'O')
    input_string = input_string.replace(
'T')
    input_string = input_string.replace(
'H')
    input_string = input_string.replace(
'E')
    input_string = input_string.replace(
'A')
    input_string = input_string.replace(
'W')
    input_string = input_string.replace(
'R')
    input_string = input_string.replace(
'I')
    input_string = input_string.replace(
'F')
    input_string = input_string.replace(
'N')
    input_string = input_string.replace(
'G')
    input_string = input_string.replace(
'U')
    input_string = input_string.replace(
'L')
    input_string = input_string.replace
```

```

'B')
    input_string = input_string.replace('B', 'Y')
'Y')
    input_string = input_string.replace('Y', 'K')
'K')
    input_string = input_string.replace('K', 'M')
'M')
    input_string = input_string.replace('M', 'C')
'C')
    input_string = input_string.replace('C', 'V')
'V')
    input_string = input_string.replace('V', 'J')
'J')
    input_string = input_string.replace('J', 'Q')
'Q')
    return input_string

def permute_word(word):
    # Check if word length is not equal to 4
    if len(word) == 4:
        x=word[2]
        return word[0]+word[1]+x+word[3]

    # Permute the word according to the permutation: 4, 3, 5, 1, 2
    return word[3] + word[2] + word[4] + word[0] + word[1]

def permute_string(input_string):
    # Split the input string into words
    words = input_string.split()

    # Permute each word in the list of words
    permuted_words = [permute_word(word) for word in words]

    # Join the permuted words back into a string
    permuted_string = ''
    permuted_string += ' '.join(permuted_words)

    return permuted_string

# Test the function
input_string = block_divided_string

```

```

result = replace_characters(input_string)
result_final = permute_string(result)
print("Input string:", input_string)
print("Result:", result_final)
print("\n")

def remove_punctuation(input_string):
    # Create a translation table to remove punctuation
    translation_table = str.maketrans('', '', string.punctuation)

    # Remove punctuation using translation table
    input_string = input_string.translate(translation_table)

    return input_string

# Test the function
input_string = "qmnjvsa nv wewc flct tvvplvl fv xja vqildhc xmlnvc nacyclp vfvw. fv wgqyp, pqg pqcs y wsq rx qmr cgv tlvhf cw tyl aeuq fv xja tkbv cqr lhf avawnc cv eas fuqb qvq tc yllrqr cfy. psdc uqf avrqc gefq pyat trac xw wwd dv eas flcbq. vd trawm vupq quw x decgqwt, yq yafl vlqs yqklhq! snafq lhvqpawr ngg_vfusr_ec_wawy qp fn wgaw"
#
cleaned_cipher=remove_punctuation(input_string)
final_plaintext=list(input_string)
j=0
print(final_plaintext)
for i in range(len(final_plaintext)):
    if final_plaintext[i]==" " or
final_plaintext[i]=="_" or
final_plaintext[i]=="." or
final_plaintext[i]=="!" or
final_plaintext[i]==",":
        continue
    elif result_final[j]==" ":
        j=j+1

```


```
breaker of this code will be blessed
```

0 Points

SecureSavants

Assignment 3

● Graded


 Select each question to review feedback and grading details.

Group

VISHAL KUMAR

Souvik Mukherjee

Vikrant Chauhan

 [View or edit group](#)

Total Points

50 / 50 pts

Question 1

[Commands](#)

5 / 5 pts

Question 2

[Cryptosystem](#)

10 / 10 pts

Question 3

[Analysis](#)

30 / 30 pts

Question 4

[Password](#)

5 / 5 pts

Question 5

[Codes](#)

0 / 0 pts

Question 6

[Group name](#)

0 / 0 pts