1. [60%] Frequency analysis

[5%] Calculate their (plaintext) letter frequency distribution.

python3 num_freq.py <input_file> <output_file> for counting double digit numbers frequencies

python3 text_freq.py <input_file_path> <output_csv_path> for counting latin characters frequencies

The files with the plain text (books) are under **ask1/plain_txt**
csv files (Frequency) saved under **ask1/freq_txt**
ALL files have the frequencies with alphabetical order from a to z the files
are not like this :
a,281
b,3022
c,107
d,12744
e,2167
f,4046
g,6862
h,19769
i,3743
j,3507
k,8836
l,11302
m,186
n,1231
o,6648
p,4362
q,11005
r,11148
s,2865
t,106
u,8889
v,9302
w,15086
x,4322
y,1434
z,3519
they are like this :
281
3022
107
12744
2167
4046

6862
19769
3743
3507
8836
11302
186
1231
6648
4362
11005
11148
2865
106
8889
9302
15086
4322
1434
3519
Because it will help me later with the copy and pasting

[20%] Encrypt these texts using the classical substitution algorithms listed below. You do not have to implement the algorithms. Use 3 rd party tools or libraries to perform all the encryption operations

For this part cipher texts are under **ask1/cipher_txt**

Also under **ask1/ciphers** there are :
CrypTool workshops (.cwm) of **caesar, book, playfair, vigenere**
Python scripts of **polybius, atbash(monoalphabetic)**

[5%] For each algorithm, you must record and discuss the choices you made. These choices depend on the algorithm, for example:

- Encryption key, keyword, key text
- Encryption key size and complexity
- Ciphertext alphabet
- Symbols mapping, etc.

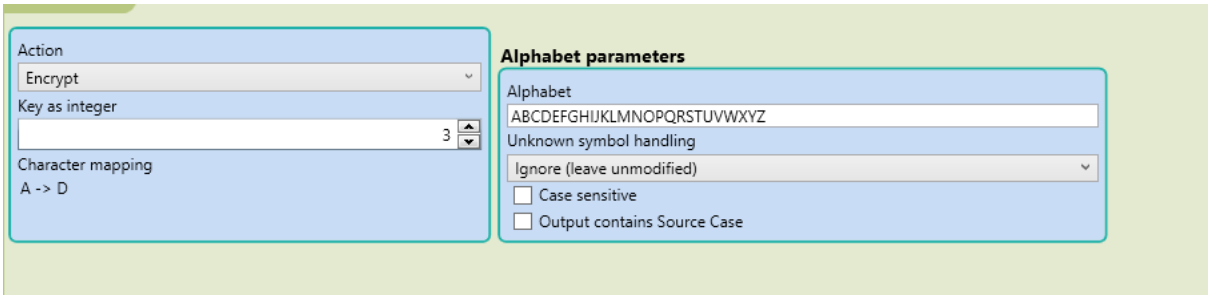
- Polybius square

python3 polybius.py <input_path> <output_file>
I wrote my own script : first I lower every latin letter in the text and then I find its 2 number representative from the polybius square.
if the char is a literate character then i leave it as is in the ciphertext.

```
for char in file_content:
    if char in latin_characters:
        # finding row of the table
        row = int((ord(char) - ord('a')) / 5) + 1

        # finding column of the table
        col = ((ord(char) - ord('a')) % 5) + 1

        # if character is 'k'
        if char == 'k':
                row = row - 1
                col = 5 - col + 1

        # if character is greater than 'j'
        elif ord(char) >= ord('j'):
                if col == 1 :
                    col = 6
                    row = row - 1

                col = col - 1
        s += str(row) + str(col)
    else:
        s += char
```

● Caesar (ROT13 is one case)

in cryptool settings :



we choose key int(3) this means that each letter of the latin alphabet in the plain text is shifted 3 times to the left in the cipher test so e.g A ----encode---> D

● Monoalphabetic (Atbash is one case)

Usage: python3 atbash.py <input_path> <output_file>

here we convert everything to upper case and then we map every letter to its opposite.

```
def atbash_cipher(message):
    lookup_table = {'A' : 'Z', 'B' : 'Y', 'C' : 'X', 'D' : 'W', 'E' : 'V',
        'F' : 'U', 'G' : 'T', 'H' : 'S', 'I' : 'R', 'J' : 'Q',
        'K' : 'P', 'L' : 'O', 'M' : 'N', 'N' : 'M', 'O' : 'L',
        'P' : 'K', 'Q' : 'J', 'R' : 'I', 'S' : 'H', 'T' : 'G',
        'U' : 'F', 'V' : 'E', 'W' : 'D', 'X' : 'C', 'Y' : 'B', 'Z' : 'A'}
```

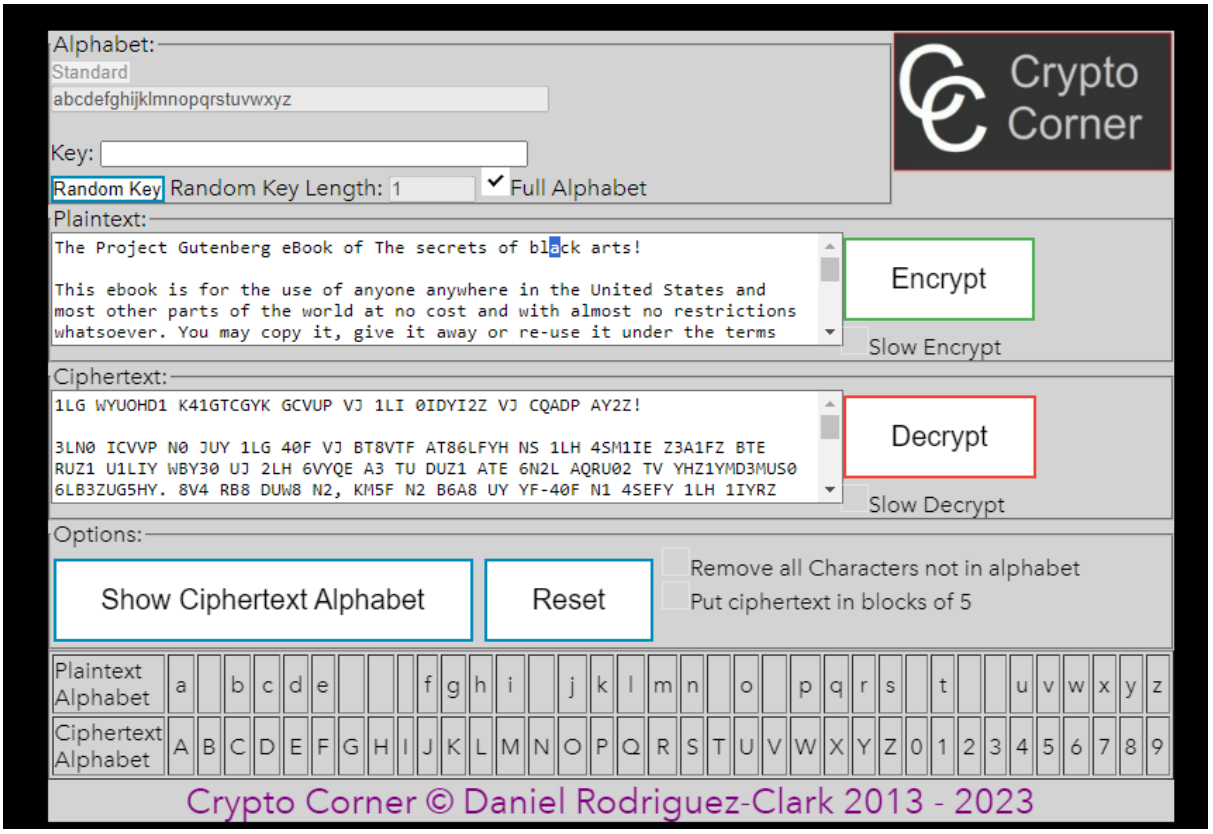
A->Z
B->Y
…
Z->A

if it is a non latin char we print it as it is in the ciphertext.

● Homophonic

I will not use the tool from the lectures; it only accepts a limited amount of characters.

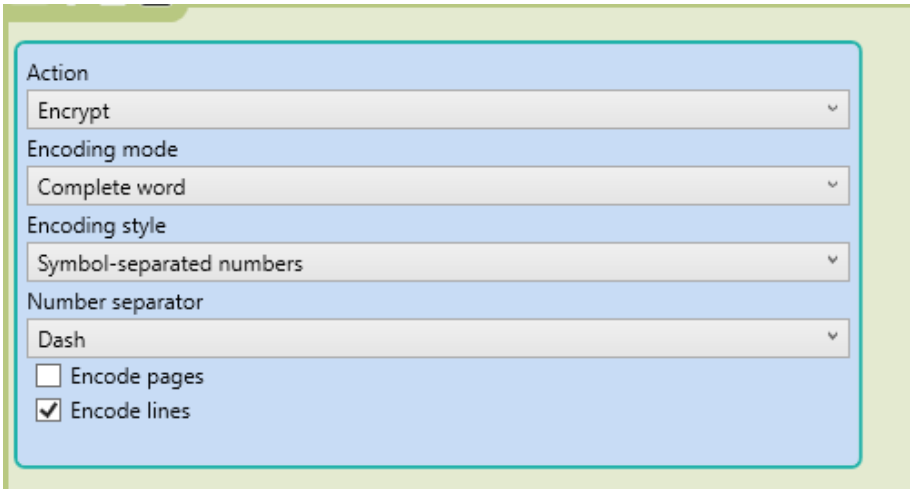
Instead i used this tool :
https://crypto.interactive-maths.com/homophonic-substitution.html



we can see the alphabet bellow e.g:
letter ‘e’ haves 4 different ciphertext substitutes “F G H I”
all the cipher texts are under ask1/cipher_txt/homophonic

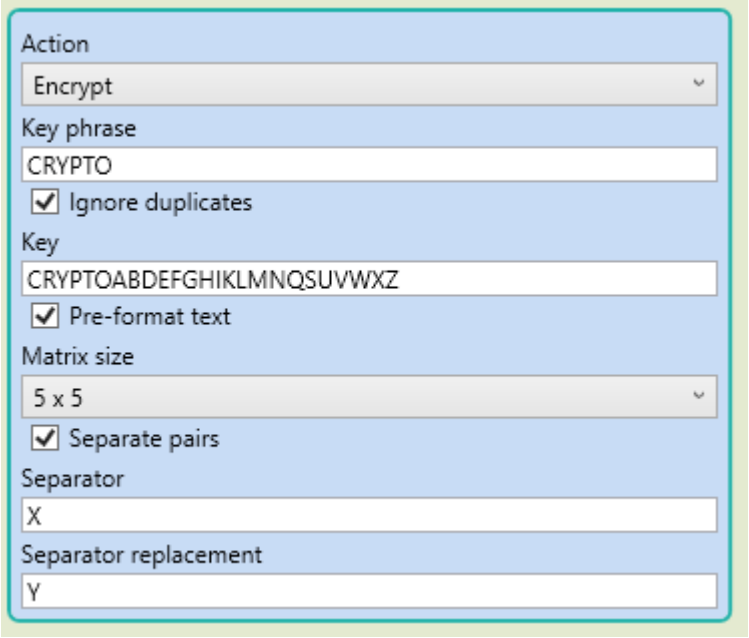
● Book

in cryptool i will use as key the iliad.txt file under the ask1/book folder with settings:



- Playfair

settings:



In cryptool i used as key the word CRYPTO so the alphabet created was : CRYPTOABDEFGHIKLMNQSUVWXZ

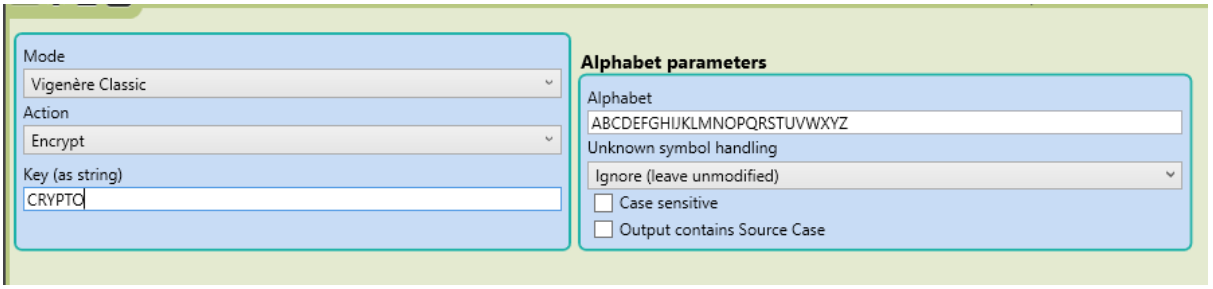
so we have the array :

C R Y P T
O A B D E
F G H I K
L M N Q S
U V W X Z

so the word “pr oj ec t” haves pairs : pr oi ec tz and is encoded like TY DF OT ET with the above array

● Polyalphabetic: Vigenère

cryptool settings :



Here we chose CRYPTO as the key (in each iteration every letter of the key corresponds to the column of the vigenere matrix and every plain text letter to the row -> then next iteration we hop to the next letters). This key will be repeated until the plain text is completely decoded every time the key reaches the last letter.

	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
A	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
B	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	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

so the word THE with key CRYPTO will be row,col : C,T-> **V** | R,H -> **Y** | Y,E -> **C** = **VYC**

● Running Key

I used this tool

http://practicalcryptography.com/ciphers/running-key-cipher/

For the key I used the book **iliad.txt** under **ask1/book.**

Like vigenere we take the plain text letter as column to the tabula recta and the key letter (from iliad.txt) as the row and find the cipher letter. The difference with the vigenere is that the key does not repeat; instead it goes on the whole iliad letter by letter until the end of the plain text and decrypts.

settings in site:

	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
A	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
B	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	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

Plaintext

The Project Gutenberg eBook of A visit to the Roman catacombs

Key Stream

This ebook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions

v Encrypt v | ^ Decrypt ^

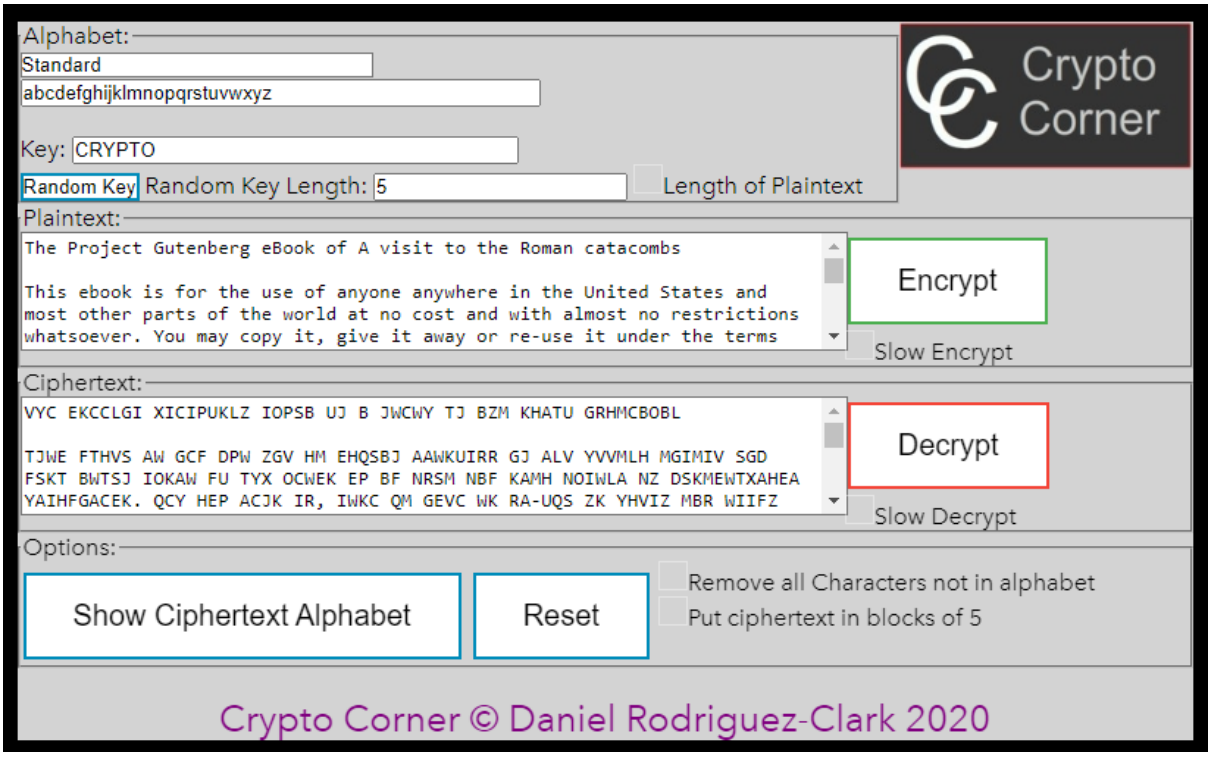
Ciphertext

momhvpxsmbyzhvgiilyiptoxmtnzifgpasklmehtehpimefgfblxzifhncgdwlmsiihvnkstyhrucepdnrjvgfwbngkacgptppglt
gsjefwdwumwhuwnwaklgsaxyckidlbchcmkhybvrlpiphyzdfwnfszxmgumtmpsgwntshxhlztimxyotewhtljxxogmkxnoe
gbtcyxwhebnuwlzllfhsbajcseprtrhfayzanxrofvxucokjlgyiegyiwpryimlwpvllfibshvrfeigisrkqsofelnwikuqykkprvylrafcy

so lets see the first 4 letters :
T,T -> M
H,H -> O
E,I -> M
P,S -> H
its correct!!
All the ciphertexts are under **ask1/cipher_txt/running_key**

● Auto Key

I used the tool
https://crypto.interactive-maths.com/autokey-cipher.html
with key: **CRYPTO**



This cipher also uses the tabula recta like the vigenere but the difference lies in the key : Auto key cipher creates the key like so : it takes the key (CRYPTO here) and appends it to the top of the plain text and this is used as the key

(here it will be CRYPTOTheProjectGutenberg….. until the end where the 6 last letters of the plain text will not be used as the key because CRYPTO went on top)

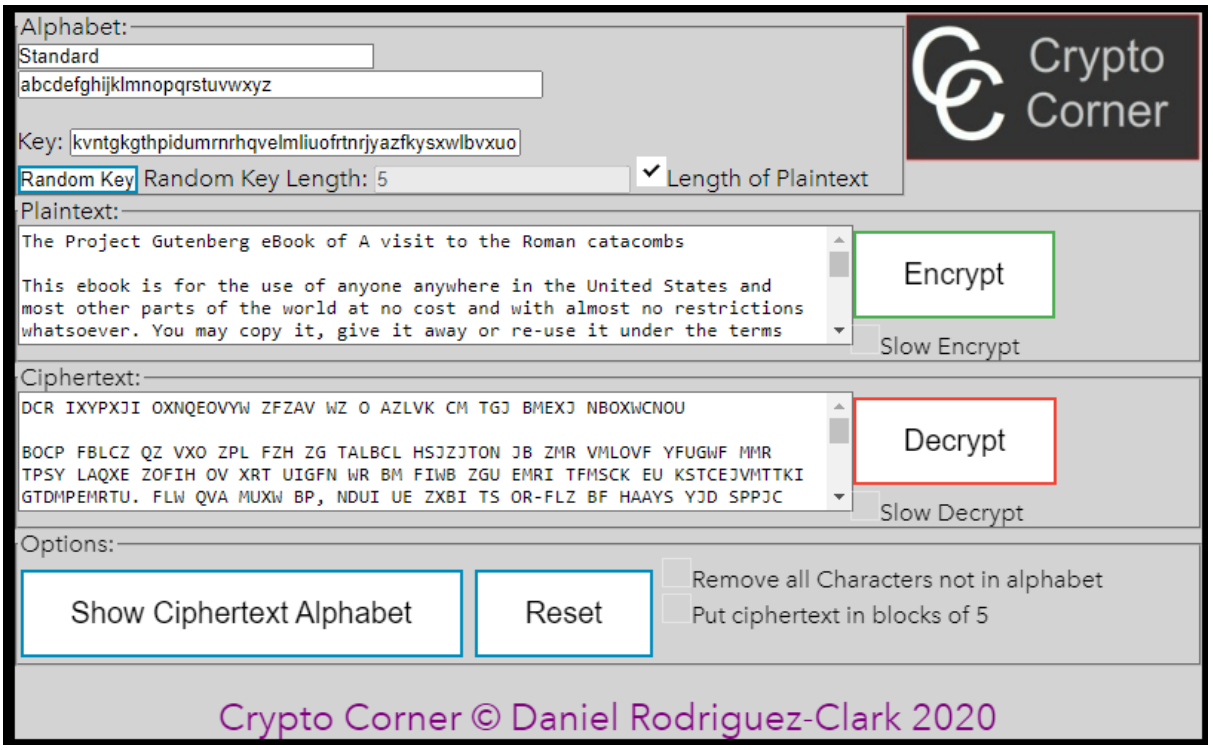
to find the **row,column** of the tabula recta for every letter of the plain text and eventually create the cipher text.

cipher texts are under **ask1/cipher_txt/auto_key**

● OTP

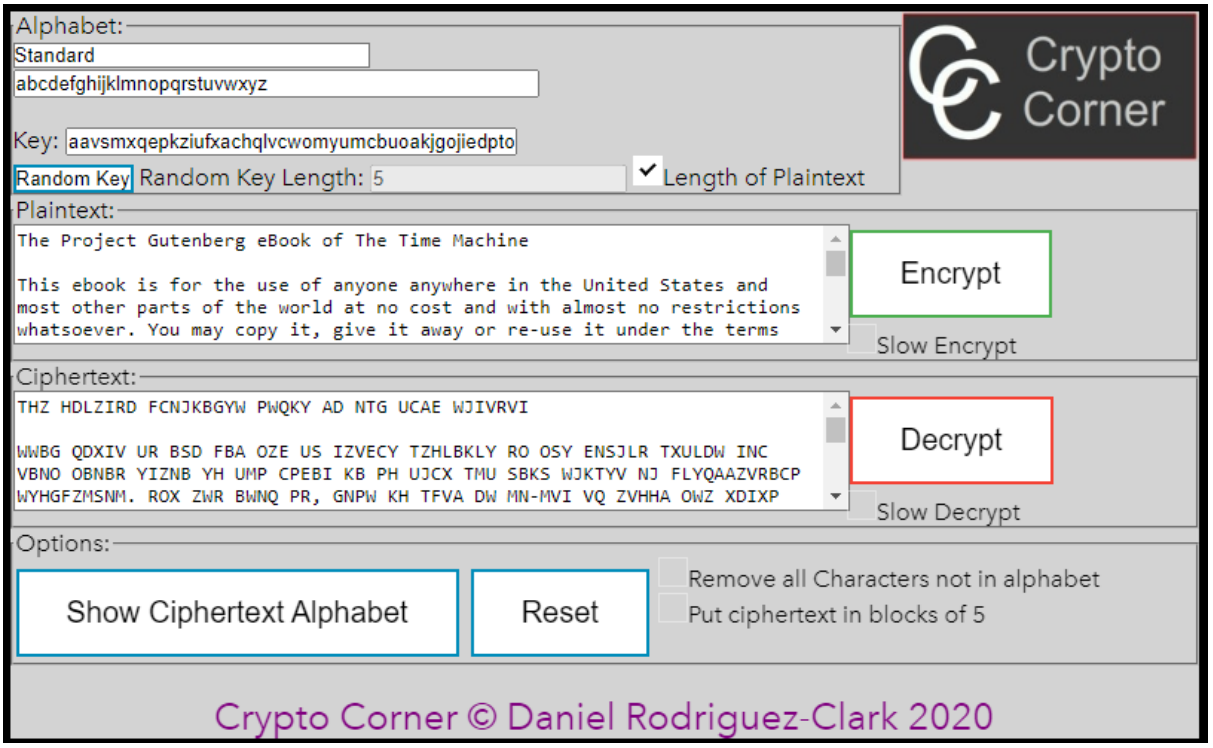
Similar to the running key cipher but the key will not be from existing words it will be truly random sequence of characters as big as the plain text to do this i will use this online tool : https://crypto.interactive-maths.com/autokey-cipher.html

but with different setting which will make it OTP :

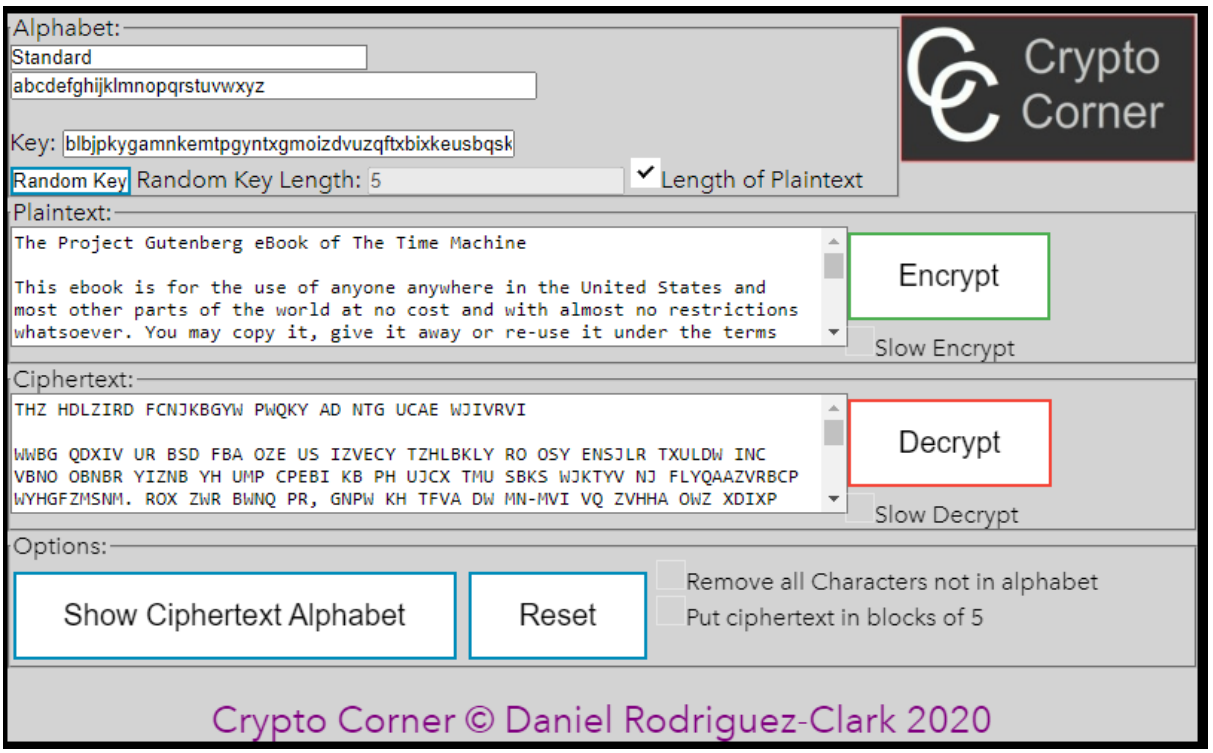


Here we have the Length of the Plain text checked and I also chose the random key button so it created an OTP key the same length as the plaintext. Then Running key cipher runs (row,column of tabula recta) but with a key of a random sequence of characters as big as the plain text.

different plain text creates different key :



now again with the same plaintext we have different key if we click the random key button:



cipher text under **ask1/cipher_txt/OTP**

[30%] For each text, draw ciphertext letter/symbol frequency distribution graphs based on the results obtained above.

- Use a suitable tool (e.g., excel) to visually compare the distributions in the ciphertext.
- See for example: Create an Interactive Chart with Checkboxes in Microsoft Excel
- https://www.youtube.com/watch?v=eMr1gFdLONU&ab_channel=Teacher%27sTech

Discuss the results. For example, elaborate on why some algorithms are better (i.e., more difficult to crack) than others.

NOTE : every letter frequency of every cipher-plain text is under: **ask1/freq_txt**

I made this part using Google spreadsheet here is the link : https://docs.google.com/spreadsheets/d/1pHixbvJnxHVfVBswH3srFrHpGBxAdelvgSi-PTVMk_I/edit?usp=sharing

IN CASE THIS LINK DOESN'T WORK I HAVE THE .xlsx file under ask1/Letter_Frequencies.xlsx but the interactive charts will not work if you want to use the interactivity put checkboxes in the cells that say TRUE over the cipher names.

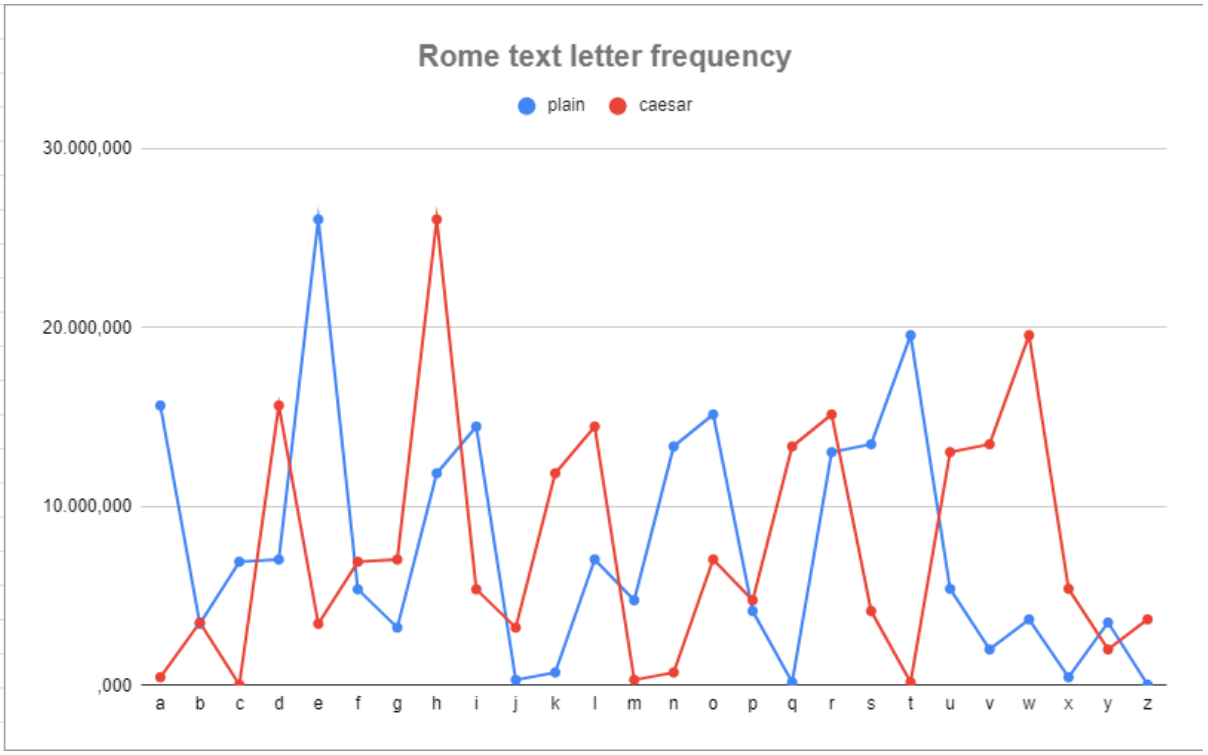
All the interactive graphs are on the same sheet.

If you are looking for them, scroll down to it.

To use the interactive graphs click on the check boxes above each cipher and the distribution line will appear on the graph to the right.

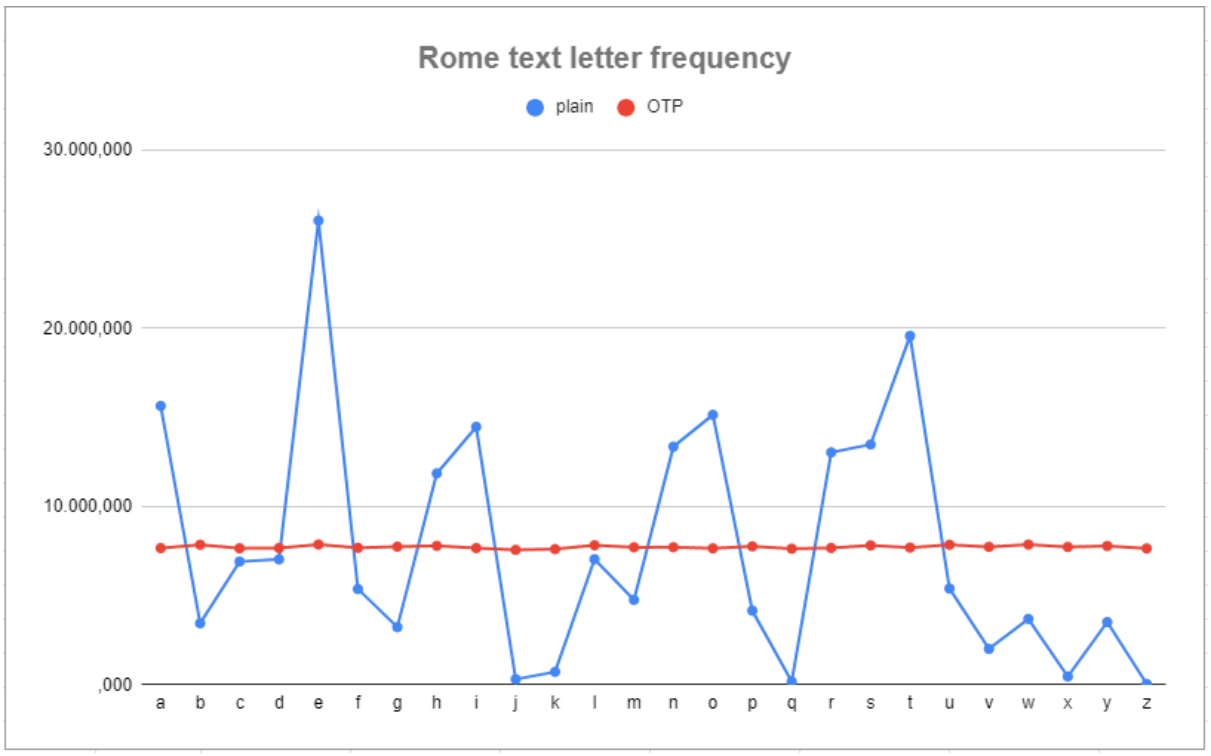
let's discuss about the distributions

lets compare the **plain text** and the **caesar** of **Rome.txt**



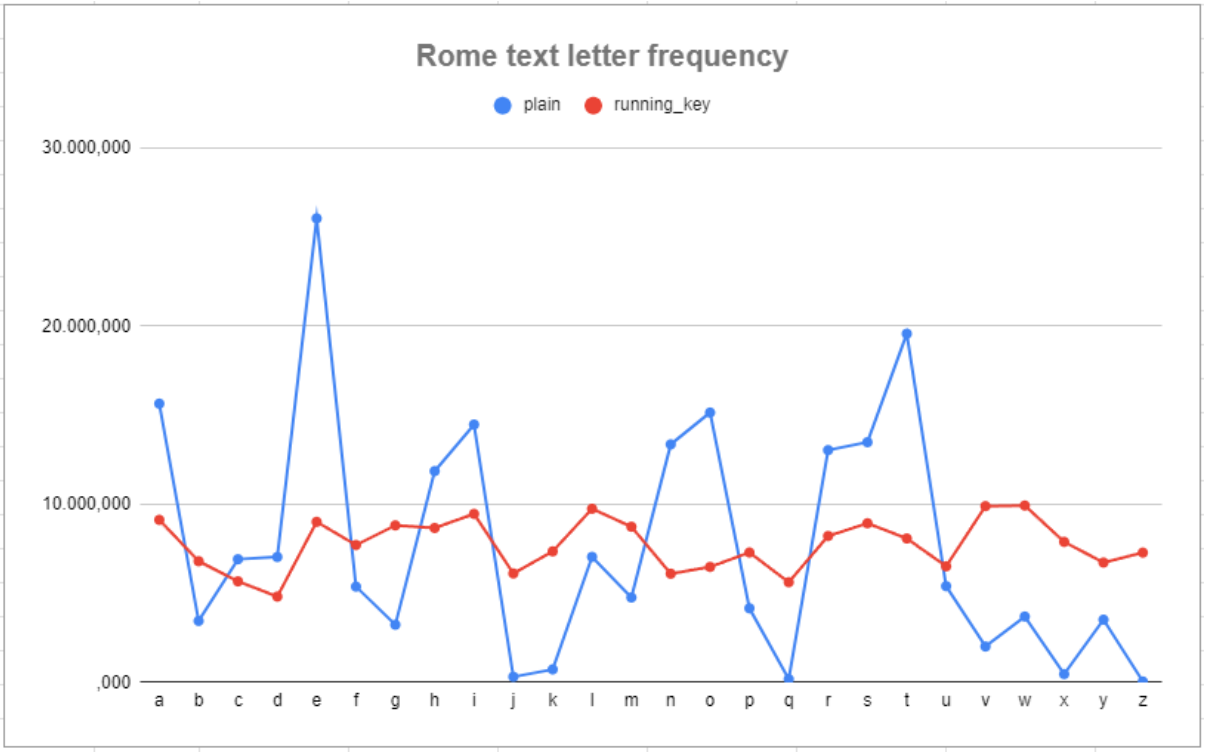
We can see that each letter of the plain text has the same number of frequency as the one that is 3 places in front of it in the caesar cipher text. That happens because caesar replaces the plain text letter with the one 3 times in front of it

Lets see the **OTP** cipher



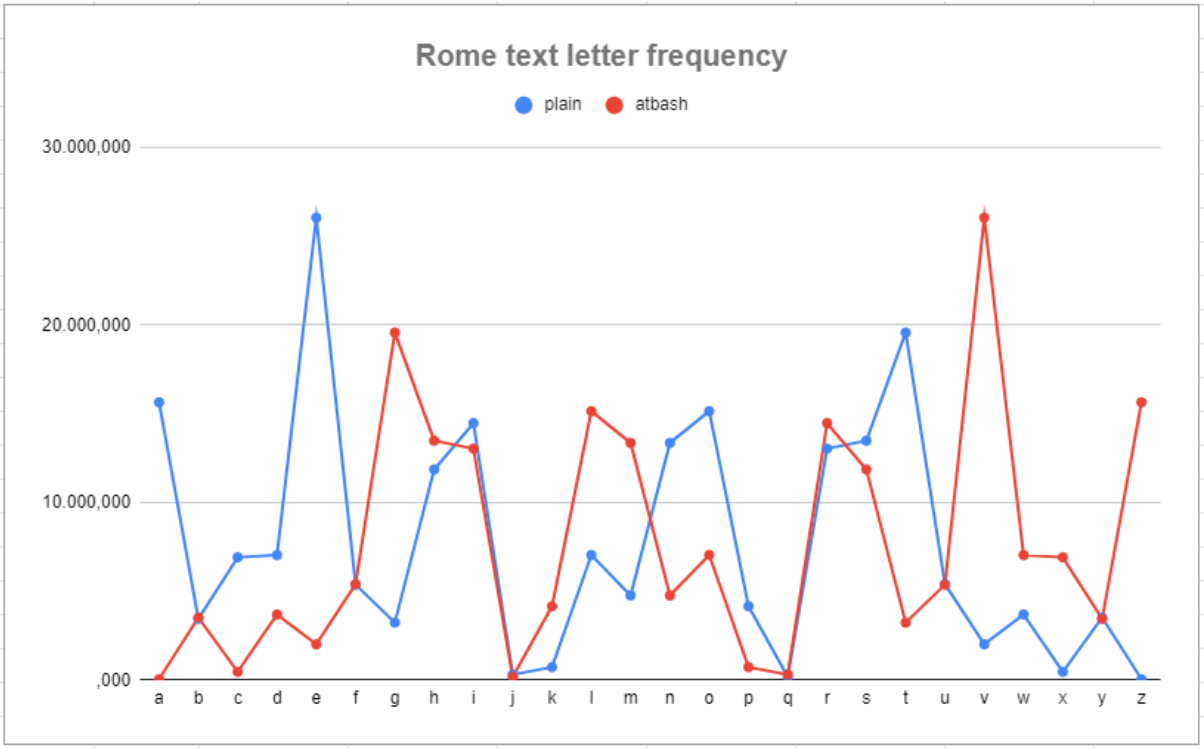
we can see that because the OTP uses a random key the same length as the plain text. So it is not vulnerable to frequency analysis.

Lets see the **Running key** cipher



As we can see it somehow manages to hide the frequencies quite well but because the key is not completely random but it's made by the iliad book it has some bumps.

Now let's compare **plain text** with **atbash.**



We can see that if we rotate the Atbash distribution 180 around it will fall exactly over the plain text distribution that is because Atbash maps every letter to its opposite in the alphabet

2. [20%] Reverse columnar transposition cipher.

Everything is under **ask1/transposition_cipher/**
encrypt.py
decrypt.py

For the Encryption my script:
python3 encrypt.py "Password" "FAR OUT IN THE UNCHARTED BACKWATERS"

```
proteopro@calpino:/mnt/c/Users/prote/Desktop/hy458/ask1/transposition_cipher$ python3 encrypt.py "Password" "FAR OUT IN THE UNCHARTED BACKWATERS"
Before sorting
80 FTRW
65 AHTA
83 REET
83 OUDE
87 UNBR
79 TCAS
82 IHC#
68 NAK#
After sorting
65 AHTA
68 NAK#
79 TCAS
80 FTRW
82 IHC#
83 REET
83 OUDE
87 UNBR
UNBROUDEREETIHC#FTRWTCASNAK#AHTA
```

UNBROUDEREETIHC#FTRWTCASNAK#AHTA

For the Decryption my script:

python3 decrypt.py "Password" "UNBROUDEREETIHC#FTRWTCASNAK#AHTA"

```
proteopro@calpino:/mnt/c/Users/prote/Desktop/hy458/ask1/transposition_cipher$ python3 decrypt.py "Password" "UNBROUDEREETIHC#FTRWTCASNAK#AHTA"
FAROUTINTHEUNCHARTEDBACKWATERS
```

FAROUTINTHEUNCHARTEDBACKWATERS

```python
def rearrange_list(plain, sorting_steps):
    #reverse the sorting steps to get the original order
    sorting_steps.reverse()
    for i in range(len(sorting_steps)):
        plain[sorting_steps[i][0]], plain[sorting_steps[i][1]] = plain[sorting_steps[i][1]], plain[sorting_steps[i][0]]
    return plain

def keyword_sort_map(int_keyword, keyword_len):
    sorting_steps = []
    for i in range(keyword_len):
        for j in range(keyword_len):
            if int_keyword[i] < int_keyword[j]:
                int_keyword[i], int_keyword[j] = int_keyword[j], int_keyword[i]
                sorting_steps.append([i, j])
    return sorting_steps
```

keyword_sort_map sorts the keyword and keeps the sorting steps in **sorting_steps** list then when I want to decrypt I just apply the same steps to the cipher text columns but in reverse and then I get the original.

3. [20%] Polybius Square (variant)

Everything is under **ask1/polybius/**
encrypt.py
decrypt.py

For the encryption :

Usage: python3 polybius.py <plain text>

For **“HELLOWORLD”**

```
proteopro@calpino:/mnt/c/Users/prote/Desktop/hy458/ask1/polybius$ python3 encrypt.py "HELLOWORLD"
polybius Square:
['X', 'Y', 'A', 'B', 'C']
['D', 'E', 'F', 'G', 'H']
['I', 'Z', 'K', 'L', 'M']
['N', 'O', 'P', 'Q', 'R']
['S', 'T', 'U', 'V', 'W']
Plain Text:  HELLOWORLD
char:  H i:  2 j:  5
char:  E i:  2 j:  2
char:  L i:  3 j:  4
char:  L i:  3 j:  4
char:  O i:  4 j:  2
char:  W i:  5 j:  5
char:  O i:  4 j:  2
char:  R i:  4 j:  5
char:  L i:  3 j:  4
char:  D i:  2 j:  1
Cipher Text:  25223434425542453421
```

For **“JOIN US”**

```
proteopro@calpino:/mnt/c/Users/prote/Desktop/hy458/ask1/polybius$ python3 encrypt.py "JOIN US"
polybius Square:
['X', 'Y', 'A', 'B', 'C']
['D', 'E', 'F', 'G', 'H']
['I', 'Z', 'K', 'L', 'M']
['N', 'O', 'P', 'Q', 'R']
['S', 'T', 'U', 'V', 'W']
Plain Text:  JOIN US
char:  J i:  3 j:  1
char:  O i:  4 j:  2
char:  I i:  3 j:  1
char:  N i:  4 j:  1
char:  U i:  5 j:  3
char:  S i:  5 j:  1
Cipher Text:  314231415351
```

For the decryption:

Usage: python3 polybius.py <plain text>

For **“252234344255424534 21”**

```
proteopro@calpino:/mnt/c/Users/prote/Desktop/hy458/ask1/polybius$ python3 decrypt.py "25223434425542453421"
polybius Square:
['X', 'Y', 'A', 'B', 'C']
['D', 'E', 'F', 'G', 'H']
['I', 'Z', 'K', 'L', 'M']
['N', 'O', 'P', 'Q', 'R']
['S', 'T', 'U', 'V', 'W']
Cipher Text:  25223434425542453421
row:  1 col:  4 char:  H
row:  1 col:  1 char:  E
row:  2 col:  3 char:  L
row:  2 col:  3 char:  L
row:  3 col:  1 char:  O
row:  4 col:  4 char:  W
row:  3 col:  1 char:  O
row:  3 col:  4 char:  R
row:  2 col:  3 char:  L
row:  1 col:  0 char:  D
Plain Text:  HELLOWORLD
```

For **“314231415351”**

```
proteopro@calpino:/mnt/c/Users/prote/Desktop/hy458/ask1/polybius$ python3 decrypt.py "314231415351"
polybius Square:
['X', 'Y', 'A', 'B', 'C']
['D', 'E', 'F', 'G', 'H']
['I', 'Z', 'K', 'L', 'M']
['N', 'O', 'P', 'Q', 'R']
['S', 'T', 'U', 'V', 'W']
Cipher Text:  314231415351
row:  2 col:  0 char:  I
row:  3 col:  1 char:  O
row:  2 col:  0 char:  I
row:  3 col:  0 char:  N
row:  4 col:  2 char:  U
row:  4 col:  0 char:  S
Plain Text:  IOINUS
```