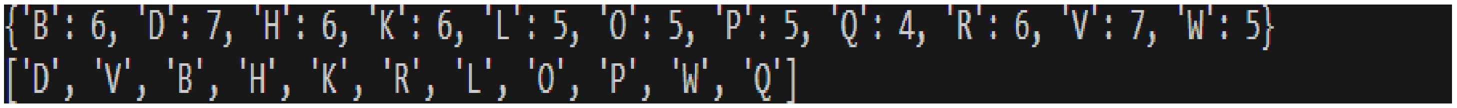
Computer System Security (COMP 424)
Professor M.Boctor

Assignment -1: You intercepted a single ciphertext. Decipher it as much as you can. To receive full or partial credit you must show all your work. Attach any code you have implemented (you can use any programming language) or any code you have found anywhere that is publicly online (but you must include citations of all sources you used in the report).
‘KUHPVIBQKVOSHWHXBPOFUXHRPVLLDDWVOSKWPREDDVVIDWQRBHBGLLBBPKQUNRVOH QEIRLWOKKRDD’
You may assume you already know:
1. The encryption/decryption algorithm is a combination of columnar transposition and simple shift substitution
2. The key length is less than or equal to 10 letters long
3. The original message is in English
4. The original message contains only letters (i.e., no punctuation marks, numbers, etc).

Solution : The encryption algorithms used were Column Transposition and Simple Shift Substitution. Hence, to decrypt the need to apply the algorithms in decryption is necessary. I used python as my coding language due to the simple tools and libraries it offers. The important point to take into consideration is key length which is less than 10. Therefore there are an overwhelming amount of possible keys.

The foremost thing to do was to learn about the cipher text. I started with extracting frequent characters present in the cipher text and sorted them in a list of sorted characters. Among English alphabets the most common letters used are E, T, A, O, I, N, S, R, H, and L. Thereby, I started mapping the most frequent one with the most common ones and determined what the possible number of shifts could be. But out of all the characters the selection is done only when the characters are repeated more than 3 times.

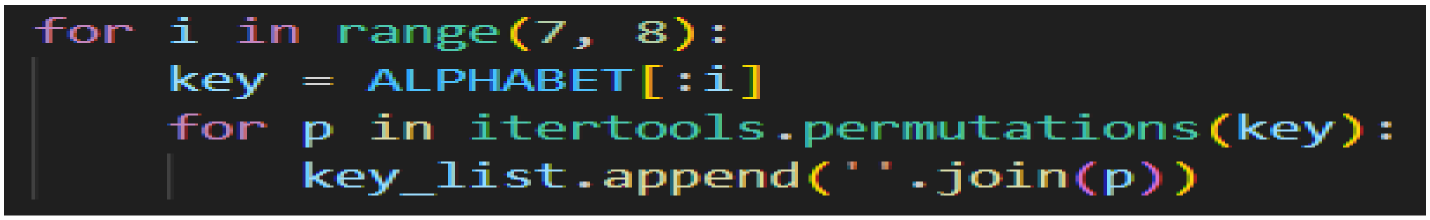


```
{'B': 6, 'D': 7, 'H': 6, 'K': 6, 'L': 5, 'O': 5, 'P': 5, 'Q': 4, 'R': 6, 'V': 7, 'W': 5}
['D', 'V', 'B', 'H', 'K', 'R', 'L', 'O', 'P', 'W', 'Q']
```

Figure 1: Frequent characters in cipher text and their frequency (line 1) and sorted list of most common characters in cipher text (line 2)

As it is visible D and V are the most frequent characters which appear 7 times followed by B, H ,K, R ... .Q.

There was a need to use simple substitution cipher recursively and hence also to store all the output from substitution as a list of secondary ciphers. All these secondary ciphers are now needed to be passed through a recursive column transposition decipher with a key value along with it. The important part here is to identify a key value which is important for the decryption algorithm. Hence, before applying a column transposition we need to generate keys with all the letters of the English alphabet. It is given that the key length is less than or equal to 10 and hence by applying a permutation of various key lengths on the ALPHABET set I extracted a list of all the keys.



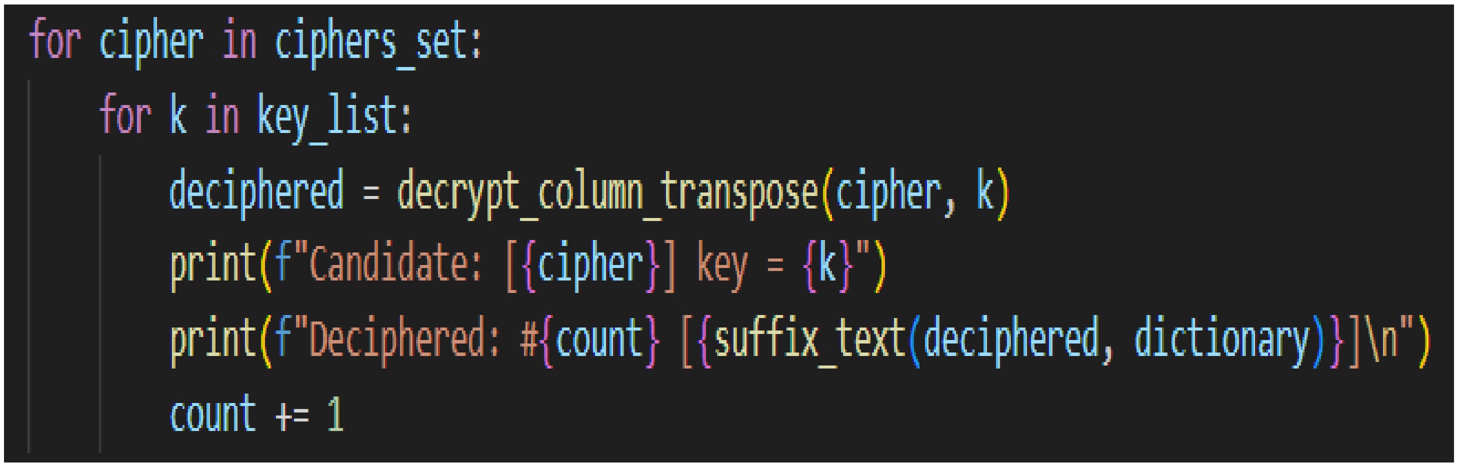
```
for i in range(7, 8):
    key = ALPHABET[:i]
    for p in itertools.permutations(key):
        key_list.append(''.join(p))
```

Figure 2: Python code to extract keys from English letters (ALPHABETS) in the range 7 to 8. Used python library itertools.permutations to obtain possible permutation of keys.

Here, the range is the length of the key being generated and key_list is the list of keys that is generated.

Now, there are following two lists with us:
1. cipher_set: A list of all the secondary ciphers generated from recursive implementation of simple shift substitution.
2. key_list: A permutation of all the keys generated from ALPHABETS in the range less than or equal to 10.

Therefore, combining these two we need to utilize and create a transposition function which provides us all the output as required. Thus, a two dimensional recursive for loop is implemented which uses cipher_set in the outer loop and a key_list in the inner loop which allow us to test all the keys for all secondary ciphers one by one. Then these ciphers and keys are passed into column transposition decrypt function.

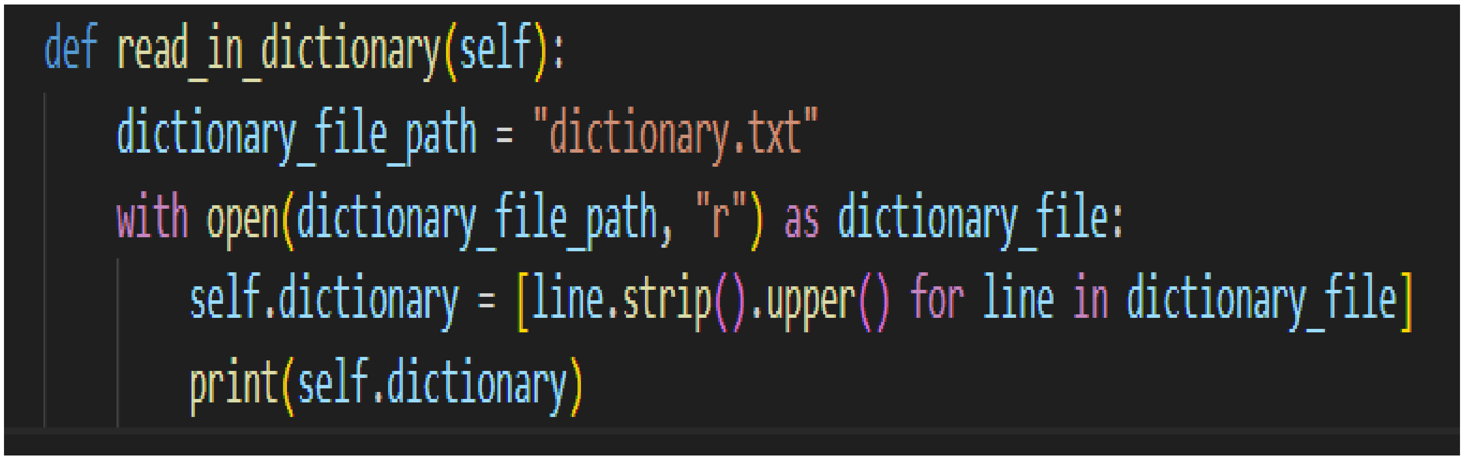


```python
for cipher in ciphers_set:
    for k in key_list:
        deciphered = decrypt_column_transpose(cipher, k)
        print(f"Candidate: [{cipher}] key = {k}")
        print(f"Deciphered: #{count} [{suffix_text(deciphered, dictionary)}]\n")
        count += 1
```

Figure 3: Two dimensional for loop using cipher_set and key_list to decipher using column transposition function.

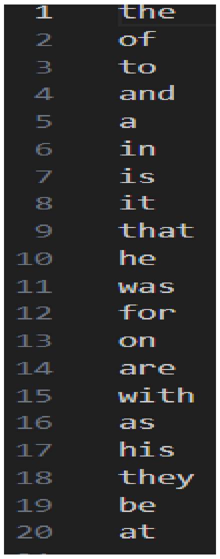
Here, there are possible outcomes in the range of thousands and hence to manage and reduce the load in the terminal a function suffix_text is introduced which outputs only the deciphered text having a word which exists in our dictionary. In a way this function filters our results and outputs the most relevant ones and returns none for others.

But to implement the suffix_text functionality a class Dictionary is needed which will read the text file and wherein we can use the words from the dictionary to match with our output. Hence a class dictionary is used which will read the file and will be initiated on the system start. Another thing to take into consideration is the format of the dictionary as the text file I found has all the text in a columnar direction. Hence, we need to format the output as such that we can use it in our program.



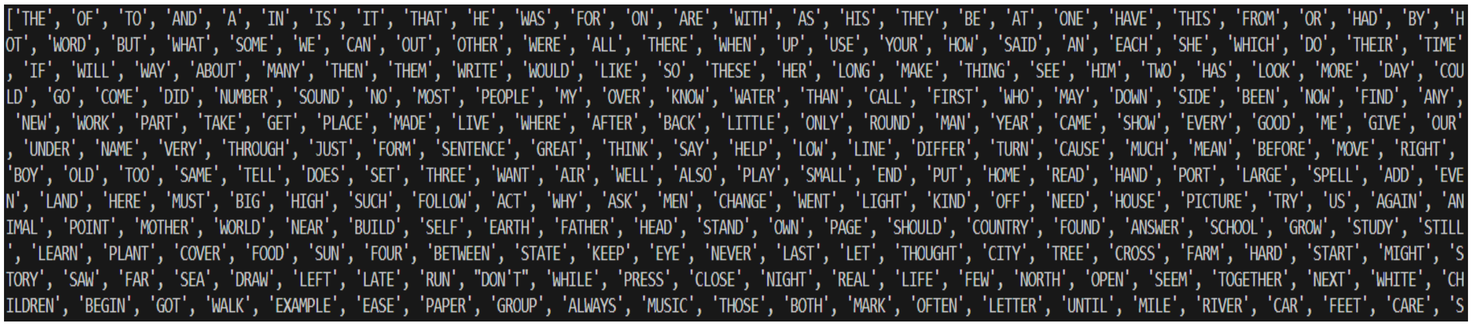
```python
def read_in_dictionary(self):
    dictionary_file_path = "dictionary.txt"
    with open(dictionary_file_path, "r") as dictionary_file:
        self.dictionary = [line.strip().upper() for line in dictionary_file]
        print(self.dictionary)
```

Figure 4: read_in_dictionary function which initiates the dictionary and creates a list.



```
the
of
to
and
a
in
is
it
that
he
was
for
on
are
with
as
his
they
be
at
```

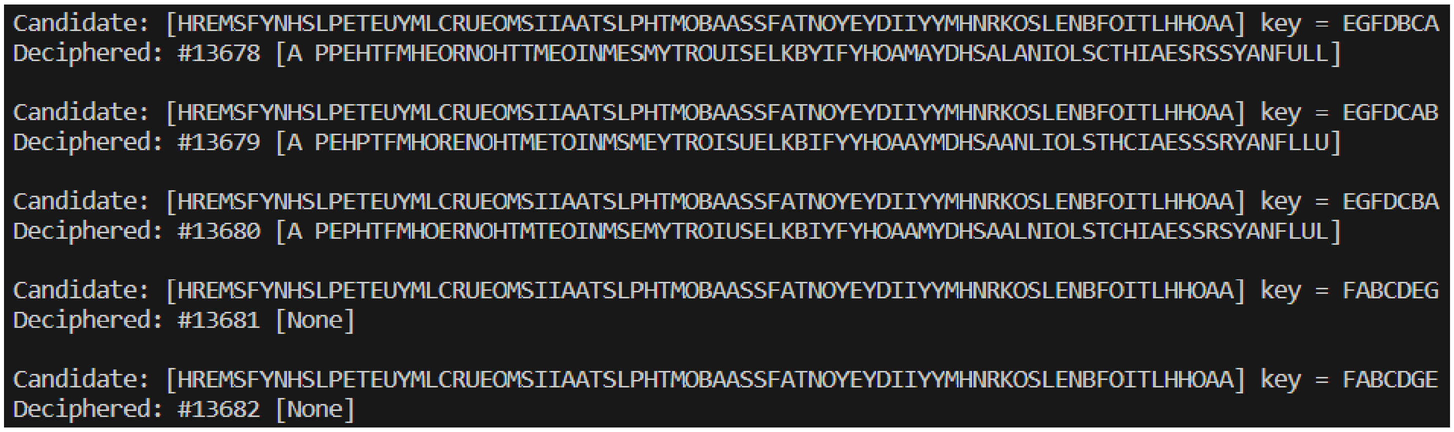
Figure 5: Dictionary words in Dictionary.txt file



```
['THE', 'OF', 'TO', 'AND', 'A', 'IN', 'IS', 'IT', 'THAT', 'HE', 'WAS', 'FOR', 'ON', 'ARE', 'WITH', 'AS', 'HIS', 'THEY', 'BE', 'AT', 'ONE', 'HAVE', 'THIS', 'FROM', 'OR', 'HAD', 'BY', 'H
OT', 'WORD', 'BUT', 'WHAT', 'SOME', 'WE', 'CAN', 'OUT', 'OTHER', 'WERE', 'ALL', 'THERE', 'WHEN', 'UP', 'USE', 'YOUR', 'HOW', 'SAID', 'AN', 'EACH', 'SHE', 'WHICH', 'DO', 'THEIR', 'TIME'
, 'IF', 'WILL', 'WAY', 'ABOUT', 'MANY', 'THEN', 'THEM', 'WRITE', 'WOULD', 'LIKE', 'SO', 'THESE', 'HER', 'LONG', 'MAKE', 'THING', 'SEE', 'HIM', 'TWO', 'HAS', 'LOOK', 'MORE', 'DAY', 'COU
LD', 'GO', 'COME', 'DID', 'NUMBER', 'SOUND', 'NO', 'MOST', 'PEOPLE', 'MY', 'OVER', 'KNOW', 'WATER', 'THAN', 'CALL', 'FIRST', 'WHO', 'MAY', 'DOWN', 'SIDE', 'BEEN', 'NOW', 'FIND', 'ANY',
 'NEW', 'WORK', 'PART', 'TAKE', 'GET', 'PLACE', 'MADE', 'LIVE', 'WHERE', 'AFTER', 'BACK', 'LITTLE', 'ONLY', 'ROUND', 'MAN', 'YEAR', 'CAME', 'SHOW', 'EVERY', 'GOOD', 'ME', 'GIVE', 'OUR'
, 'UNDER', 'NAME', 'VERY', 'THROUGH', 'JUST', 'FORM', 'SENTENCE', 'GREAT', 'THINK', 'SAY', 'HELP', 'LOW', 'LINE', 'DIFFER', 'TURN', 'CAUSE', 'MUCH', 'MEAN', 'BEFORE', 'MOVE', 'RIGHT',
'BOY', 'OLD', 'TOO', 'SAME', 'TELL', 'DOES', 'SET', 'THREE', 'WANT', 'AIR', 'WELL', 'ALSO', 'PLAY', 'SMALL', 'END', 'PUT', 'HOME', 'READ', 'HAND', 'PORT', 'LARGE', 'SPELL', 'ADD', 'EVE
N', 'LAND', 'HERE', 'MUST', 'BIG', 'HIGH', 'SUCH', 'FOLLOW', 'ACT', 'WHY', 'ASK', 'MEN', 'CHANGE', 'WENT', 'LIGHT', 'KIND', 'OFF', 'NEED', 'HOUSE', 'PICTURE', 'TRY', 'US', 'AGAIN', 'AN
IMAL', 'POINT', 'MOTHER', 'WORLD', 'NEAR', 'BUILD', 'SELF', 'EARTH', 'FATHER', 'HEAD', 'STAND', 'OWN', 'PAGE', 'SHOULD', 'COUNTRY', 'FOUND', 'ANSWER', 'SCHOOL', 'GROW', 'STUDY', 'STILL
', 'LEARN', 'PLANT', 'COVER', 'FOOD', 'SUN', 'FOUR', 'BETWEEN', 'STATE', 'KEEP', 'EYE', 'NEVER', 'LAST', 'LET', 'THOUGHT', 'CITY', 'TREE', 'CROSS', 'FARM', 'HARD', 'START', 'MIGHT', 'S
TORY', 'SAW', 'FAR', 'SEA', 'DRAW', 'LEFT', 'LATE', 'RUN', "DON'T", 'WHILE', 'PRESS', 'CLOSE', 'NIGHT', 'REAL', 'LIFE', 'FEW', 'NORTH', 'OPEN', 'SEEM', 'TOGETHER', 'NEXT', 'WHITE', 'CH
ILDREN', 'BEGIN', 'GOT', 'WALK', 'EXAMPLE', 'EASE', 'PAPER', 'GROUP', 'ALWAYS', 'MUSIC', 'THOSE', 'BOTH', 'MARK', 'OFTEN', 'LETTER', 'UNTIL', 'MILE', 'RIVER', 'CAR', 'FEET', 'CARE', 'S
```

Figure 6: List of words formatted from Dictionary.txt to a list of strings.

Finally, after applying and filtering our output the terminal shows us all the possible deciphered text along with its key.

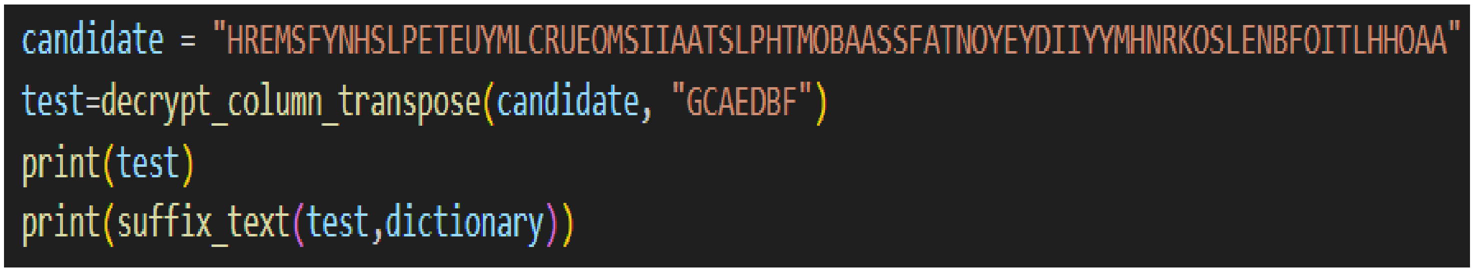


```
Candidate: [HREMSFYNHSLPETEUYMLCRUEOMSIIAATSLPHTMOBAASSFATNOYEYDIIYYMHNRKOSLENBFOITLHHOAA] key = EGFDBCA
Deciphered: #13678 [A PPEHTFMHEORNOHTTMEOINMESMYTROUISELKBYIFYHOAMAYDHSALANIOLSCTHIAESRSSYANFULL]

Candidate: [HREMSFYNHSLPETEUYMLCRUEOMSIIAATSLPHTMOBAASSFATNOYEYDIIYYMHNRKOSLENBFOITLHHOAA] key = EGFDCAB
Deciphered: #13679 [A PEHPTFMHORENOHTMETOINMSMEYTROISUELKBIFYYHOAAYMDHSAANLIOLSTHCIAESSSRYANFLLU]

Candidate: [HREMSFYNHSLPETEUYMLCRUEOMSIIAATSLPHTMOBAASSFATNOYEYDIIYYMHNRKOSLENBFOITLHHOAA] key = EGFDCBA
Deciphered: #13680 [A PEPHTFMHOERNOHTMTEOINMSEMYTROIUSELKBIYFYHOAAMYDHSAALNIOLSTCHIAESSRSYANFLUL]

Candidate: [HREMSFYNHSLPETEUYMLCRUEOMSIIAATSLPHTMOBAASSFATNOYEYDIIYYMHNRKOSLENBFOITLHHOAA] key = FABCDEG
Deciphered: #13681 [None]

Candidate: [HREMSFYNHSLPETEUYMLCRUEOMSIIAATSLPHTMOBAASSFATNOYEYDIIYYMHNRKOSLENBFOITLHHOAA] key = FABCDGE
Deciphered: #13682 [None]
```

Figure 7: Output in terminal showing Candidate(Secondary Cipher), Deciphered text and key.

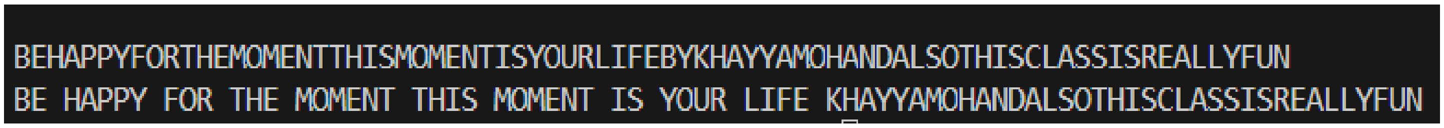
I had to manually scan through all the outputs to check if there was any relevant sentence or deciphered message in the list of all the deciphered texts. There were none in the range of 1 to 6 but with key length = 7 I started to find some relevant deciphered text. But with the key="GCAEDBF" I found the most relevant one! The deciphered text is "BE HAPPY FOR THE MOMENT THIS MOMENT IS YOUR LIFE KHAYYAMOHANDALSOTHISCLASSISREALLYFUN". This was the only string with more than 3 relevant words. Although the output is not as perfect as I expected it is enough to point me to the key.

I applied column transposition cipher without applying our filtering function suffix_text and found the perfect deciphered text "BEHAPPYFORTHEMOMENTTHISMOMENTISYOURLIFEBYKHAYYAMOHANDALSOTHISCLASSIS REALLYFUN".



```
candidate = "HREMSFYNHSLPETEUYMLCRUEOMSIIAATSLPHTMOBAASSFATNOYEYDIIYYMHNRKOSLENBFOITLHHOAA"
test=decrypt_column_transpose(candidate, "GCAEDBF")
print(test)
print(suffix_text(test,dictionary))
```

Figure 8: Validating the output with the obtained key and secondary cipher text.



```
BEHAPPYFORTHEMOMENTTHISMOMENTISYOURLIFEBYKHAYYAMOHANDALSOTHISCLASSISREALLYFUN
BE HAPPY FOR THE MOMENT THIS MOMENT IS YOUR LIFE KHAYYAMOHANDALSOTHISCLASSISREALLYFUN
```

Figure 9: Final Output

WORK CITED:

1. https://headfullofciphers.com/2020/08/27/cryptopy-caesar-cipher-aka-shift-cipher-in-python/
2. https://www.khanacademy.org/computing/computer-science/cryptography/ciphers/a/shift-cipher
3. https://github.com/HolzerSoahita/Cracking_code_python/tree/main/Simple_substitution
4. https://www.thesaurus.com/e/ways-to-say/most-common-letter/
5. https://codereview.stackexchange.com/questions/139075/a-simple-key-shift-cipher
6. https://www.geeksforgeeks.org/python-sort-python-dictionaries-by-key-or-value/
7. https://www.freecodecamp.org/news/sort-dictionary-by-value-in-python/#:~:text=reverse%20with%20a%20value%20of,sorted%20dictionary%20in%20descending%20order.&text=You%20can%20see%20the%20output,That's%20the%20default.
8. https://github.com/andrewdt23/Encryption-Decryption-Application/blob/e61f85711c6d3d5608ce21f829b207531b92b6fb/columnar.py#L49
9. https://github.com/hywhuangyuwei/columnar-transposition-cipher-brute-force-decipher/blob/master/decipher.py
10. https://www.geeksforgeeks.org/permutation-and-combination-in-python/