Assignment 2

General Instructions:

1. Deadline: Sunday, October 13 at 11:59 pm. (Submissions will be accepted until Tuesday, 15 midnight with no late penalty – no submissions accepted afterwards)
2. This assignment is worth 10 pts of your course grade.
3. Download the file: "template_A2.py" and rename it to "solution_A2.py". This should be strictly observed for the testing file to work.
4. At the top of the file: edit the following header (Any submission without this header will be rejected):

```
#-------------------------
# Your Name and ID
# CP460 (Fall 2019)
# Assignment 2
#-------------------------
```

5. There are two other files: "utilities_A2.py" and "test_A2.py". Download the two files into the same folder as "solution_A2.py".
6. Download the ciphertext, plaintext and dictionary files into the same folder
7. You may only edit the solution_A2.py. The "utilities_A2.py" and "test_A2.py" files should not be changed. If you need to add any other utility function, add it into the solution_A2.py file.
8. Make sure to go through the utilities file. You do not need to create a function that is already provided to you.
9. Every new function you create should have a header outlining input parameters, return values and a description. similar to the following:

```
#---------------------------------------------------
# Parameters:   ciphertext(string)
#               key (none)
# Return:       plaintext (string)
# Description:  Decryption using Polybius Square
#--------------------------------------------------
```

10. At the end, submit ONLY the solution_A2.py file.

Q1: Vigenere Cipher Implementation (1.6 pts)

In class, the code for the Vigenere cipher (version 1) using a key word of a single character was provided. In this task, you will modify the encryption and decryption schemes such that the autokey is a given phrase, i.e. of length two or more characters. We will call this Vigenere Cipher (version 2).

Write the encryption function `e_vigenere2(plaintext, key)` that would encrypt any given plaintext using the Vigenere Cipher through an autokey which has two or more characters. Also, write the decryption function `d_vigenere2(ciphertext, key)` that would perform the reverse process to restore the original plaintext.

A valid ***key*** should be a non-empty string that contains only alpha characters.

It would be easier to take the implementations of `e_vigenere1(plaintext, key)` and `d_vigenere1(plaintext, key)` and edit them according to the new requirements.

Remember that both `e_vigenere1` and `e_vigenere2` are called by `e_vigenere` which calls the proper function depending on the key length. Similarly, `d_vigenere1` and `d_vigenere2` are called by `d_vigenere`.

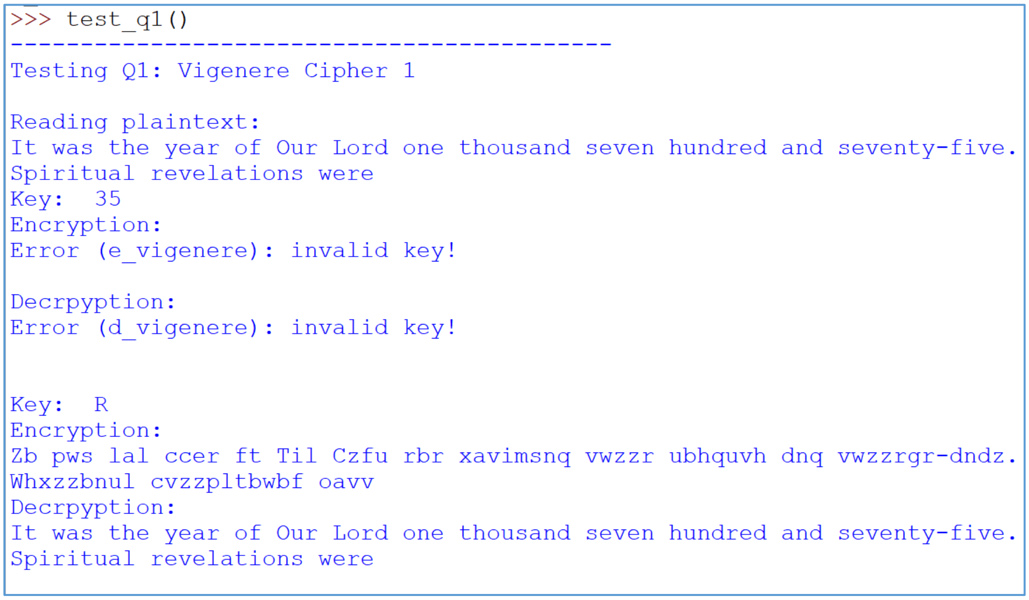
The `e_vigenere` and `d_vigenere` functions are provided to you. You would need to copy from your class notes `e_vigenere1` and `d_vigenere1` functions, and write your own `e_vigenere2` and `d_vigenere2`.

The function descriptions are as follows:

```
#-----------------------------------------------------
# Parameters:   plaintext (string)
#               key (string)
# Return:       ciphertext (string)
# Description:  Encryption using Vigenere cipher
#               Autokey is of length 2 or more alpha characters
#                Non-alpha characters → no substitution
#                Preservers the case of the plaintext characters
#----------------------------------------------------
```

```
def e_vigenere2(plaintext, key):
    # your code here
    return ciphertext

#-----------------------------------------------------
# Parameters:   ciphertext (string)
#               key (string)
# Return:       plaintext (string)
# Description:  Decryption using Vigenere cipher
#               Autokey is of length 2 or more alpha characters
#               Non-alpha characters → no substitution
#               Preservers the case of the plaintext characters
#-----------------------------------------------------
def d_vigenere2(ciphertext, key):
    # your code here
    return plaintext
```

Below are the results of executing the testing module:



```
>>> test_q1()
----------------------------------------
Testing Q1: Vigenere Cipher 1

Reading plaintext:
It was the year of Our Lord one thousand seven hundred and seventy-five.
Spiritual revelations were
Key:  35
Encryption:
Error (e_vigenere): invalid key!

Decrpyption:
Error (d_vigenere): invalid key!


Key:  R
Encryption:
Zb pws lal ccer ft Til Czfu rbr xavimsnq vwzzr ubhquvh dnq vwzzrgr-dndz.
Whxzzbnul cvzzpltbwbf oavv
Decrpyption:
It was the year of Our Lord one thousand seven hundred and seventy-five.
Spiritual revelations were
```

```
Key:  ON
Encryption:
Wg kng gvr mroe cs Chf Ycer bbr huchgnbq grjrb uiaresq oar fsisahl-tvjr.
Gcwewginz esisyogwbbf krfr
Decrpyption:
It was the year of Our Lord one thousand seven hundred and seventy-five.
Spiritual revelations were

Key:  SIT
Encryption:
Ab psa mzm rwik gn Hmz Egzw gvx lphmatfl lwdxf pnflkwl tfl lwdxfbr-xqow.
Aiazblctd zxnmesbbgvl omkw
Decrpyption:
It was the year of Our Lord one thousand seven hundred and seventy-five.
Spiritual revelations were

Key:  FORD
Encryption:
Nh ndx hyh dsru tt Fxw Zfui ceh yvfxxoeg xsmhs vlqifvg fbu vjjvqym-wlas.
Jsnfzwzoc ujjvofhzrsg nhws
Decrpyption:
It was the year of Our Lord one thousand seven hundred and seventy-five.
Spiritual revelations were
```

```
Key:  PEACE
Encryption:
Xx wcw ile aipv oh Sjv Lqvs sng xwsuuech sgztr hwrsvef ech sgztrta-jxze.
Utxvivypp rgztpavmdrs yigi
Decrpyption:
It was the year of Our Lord one thousand seven hundred and seventy-five.
Spiritual revelations were

Key:  Jellyfish
Encryption:
Rx hlq ypw fnec zd Tcj Sxvo zlj bzvdwlyb xmnlw lfybwmv hwh dptjvlf-omgp.
Quqjpcylw pjdwsjxtzlx ewyn
Decrpyption:
It was the year of Our Lord one thousand seven hundred and seventy-five.
Spiritual revelations were

Key:  ENVIORNMENT
Encryption:
Mg rig kuq crtv ba Wii Yavq hrr opclfmrq liizv vlapvrw eay asmrzxl-ymiz.
Adzeuxhtp ezdscnfmbgw jzzs
Decrpyption:
It was the year of Our Lord one thousand seven hundred and seventy-five.
Spiritual revelations were

-------------------------------------------
```

Q2: Vigenere Cryptanalysis Utilities (2.4 pts)

In order to execute a successful Vigenere cryptanalysis, a cryptanalysis would need several automated tools (functions) at hand. In this task, you will develop six of these tools:

```
#----------------------------------------------------------------------------
# Parameters:   text (string)
#               size (int)
# Return:       blocks: list of strings
# Description:  Break a given string into blocks of strings of given size
#               Result is provided in a list
#----------------------------------------------------------------------------
def text_to_blocks(text,size):
    # your code here
    return blocks
```

```
#-----------------------------------
# Parameters:   text (string)
# Return:       modifiedText (string)
# Description:  Removes all non-alpha characters from the given string
#               Returns a string of only alpha characters
#-----------------------------------
def remove_nonalpha(text):
    # your code here
    return modifiedText
```

```
#----------------------------------------------------------------------------
# Parameters:   blocks: list of strings
# Return:       baskets: list of strings
# Description:  Assume all blocks have same size = n (other than last block)
#               Create n baskets
#               In basket[i] put character #i from each block
#----------------------------------------------------------------------------
def blocks_to_baskets(blocks):
    # your code here
    return baskets
```

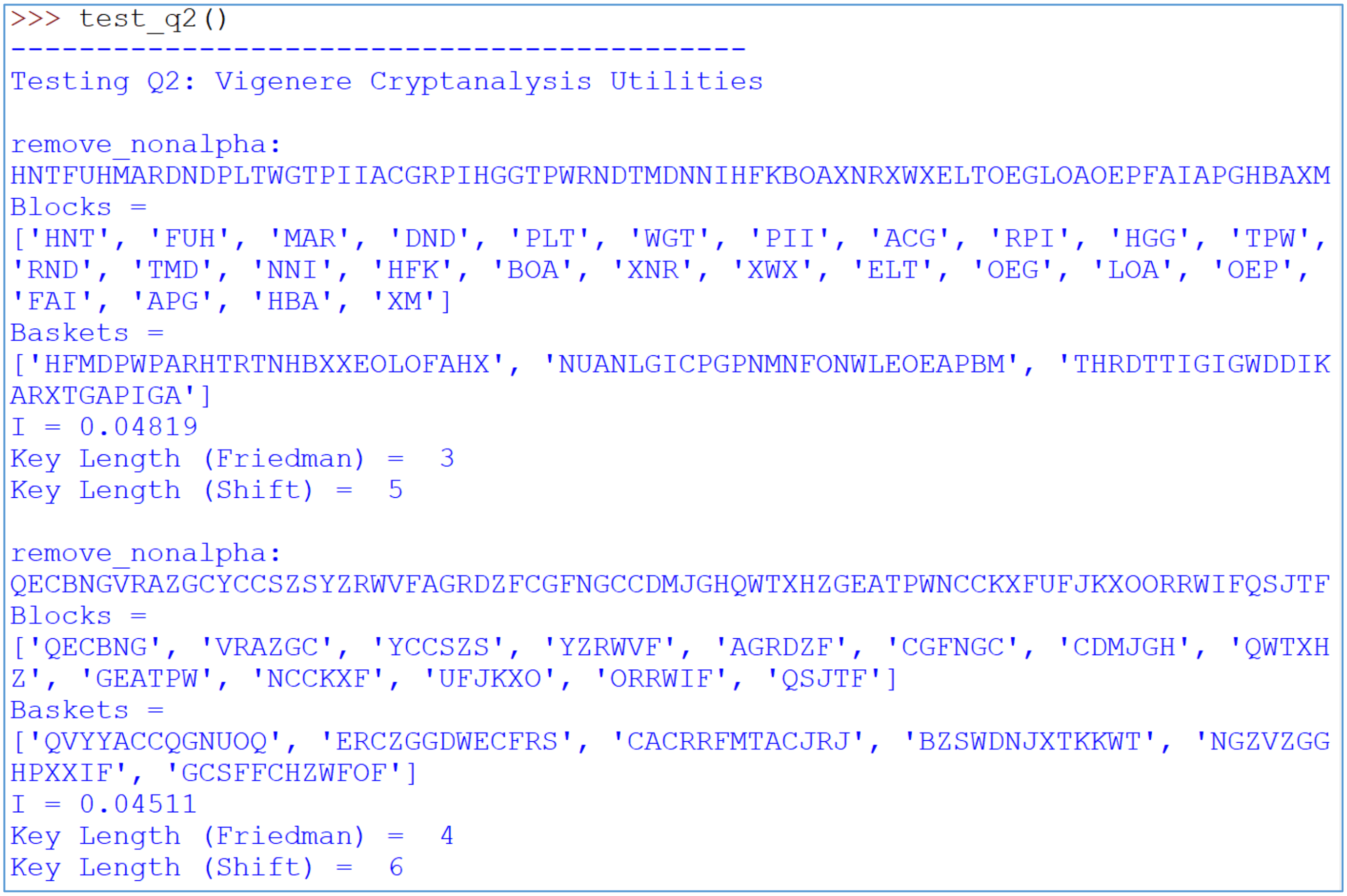
```
#----------------------------------------------------------------------------
# Parameters:   ciphertext(string)
# Return:       I (float): Index of Coincidence
# Description:  Computes and returns the index of coincidence
#               for a given text
#----------------------------------------------------------------------------
def get_indexOfCoin(ciphertext):
    #your code here
    return I
```

```
#----------------------------------------------------------------
# Parameters:   ciphertext(string)
# Return:       k (int) key length
# Description:  Uses Friedman's test to compute key length
#               returns key length rounded to nearest integer
#---------------------------------------------------------------
def getKeyL_friedman(ciphertext):
    # your code here
    return k
```

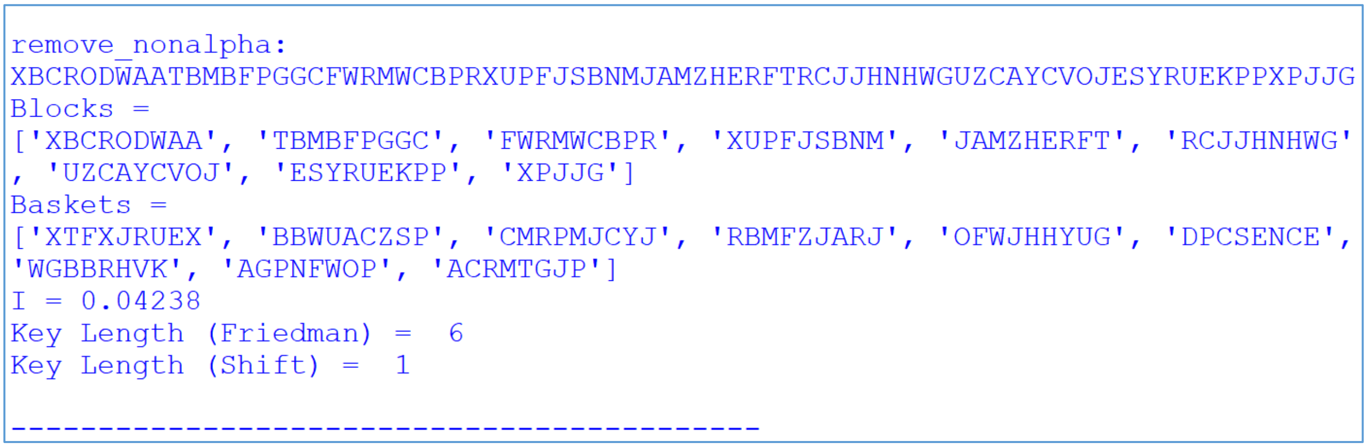
```
#----------------------------------------------------------------
# Parameters:   ciphertext(string)
# Return:       key (int)
# Description:  Uses the Ciphertext Shift method to compute key length
#               Attempts key lengths 1 to 20
#---------------------------------------------------------------
def getKeyL_shift(ciphertext):
   # your code here
   return k
```

Below are the results of executing the testing module:

Note: Since the given ciphertext is short, it is no surprise that the key estimation tools produced inaccurate results.



```
>>> test_q2()
-------------------------------------------
Testing Q2: Vigenere Cryptanalysis Utilities

remove_nonalpha:
HNTFUHMARDNDPLTWGTPIIACGRPIHGGTPWRNDTMDNNIHFKBOAXNRXWXELTOEGLOAOEPFAIAPGHBAXM
Blocks =
['HNT', 'FUH', 'MAR', 'DND', 'PLT', 'WGT', 'PII', 'ACG', 'RPI', 'HGG', 'TPW',
'RND', 'TMD', 'NNI', 'HFK', 'BOA', 'XNR', 'XWX', 'ELT', 'OEG', 'LOA', 'OEP',
'FAI', 'APG', 'HBA', 'XM']
Baskets =
['HFMDPWPARHTRTNHBXXEOLOFAHX', 'NUANLGICPGPNMNFONWLEOEAPBM', 'THRDTTIGIGWDDIK
ARXTGAPIGA']
I = 0.04819
Key Length (Friedman) =  3
Key Length (Shift) =  5

remove_nonalpha:
QECBNGVRAZGCYCCSZSYZRWVFAGRDZFCGFNGCCDMJGHQWTXHZGEATPWNCCKXFUFJKXOORRWIFQSJTF
Blocks =
['QECBNG', 'VRAZGC', 'YCCSZS', 'YZRWVF', 'AGRDZF', 'CGFNGC', 'CDMJGH', 'QWTXH
Z', 'GEATPW', 'NCCKXF', 'UFJKXO', 'ORRWIF', 'QSJTF']
Baskets =
['QVYYACCQGNUOQ', 'ERCZGGDWECFRS', 'CACRRFMTACJRJ', 'BZSWDNJXTKKWT', 'NGZVZGG
HPXXIF', 'GCSFFCHZWFOF']
I = 0.04511
Key Length (Friedman) =  4
Key Length (Shift) =  6
```



```
remove_nonalpha:
XBCRODWAATBMBFPGGCFWRMWCBPRXUPFJSBNMJAMZHERFTRCJJHNHWGUZCAYCVOJESYRUEKPPXPJJG
Blocks =
['XBCRODWAA', 'TBMBFPGGC', 'FWRMWCBPR', 'XUPFJSBNM', 'JAMZHERFT', 'RCJJHNHWG'
, 'UZCAYCVOJ', 'ESYRUEKPP', 'XPJJG']
Baskets =
['XTFXJRUEX', 'BBWUACZSP', 'CMRPMJCYJ', 'RBMFZJARJ', 'OFWJHHYUG', 'DPCSENCE',
'WGBBRHVK', 'AGPNFWOP', 'ACRMTGJP']
I = 0.04238
Key Length (Friedman) =  6
Key Length (Shift) =  1

-------------------------------------------
```

Q3: Block Rotation Cipher
(2.5 pts)

In this task, you will implement the encryption, decryption and cryptanalysis modules for the Block Rotation Cipher.

In Block Rotation Cipher, the plaintext is divided into blocks of size ***b***. Then each block is rotated to the left ***r*** times. The key is tuple ***(b,r)***.

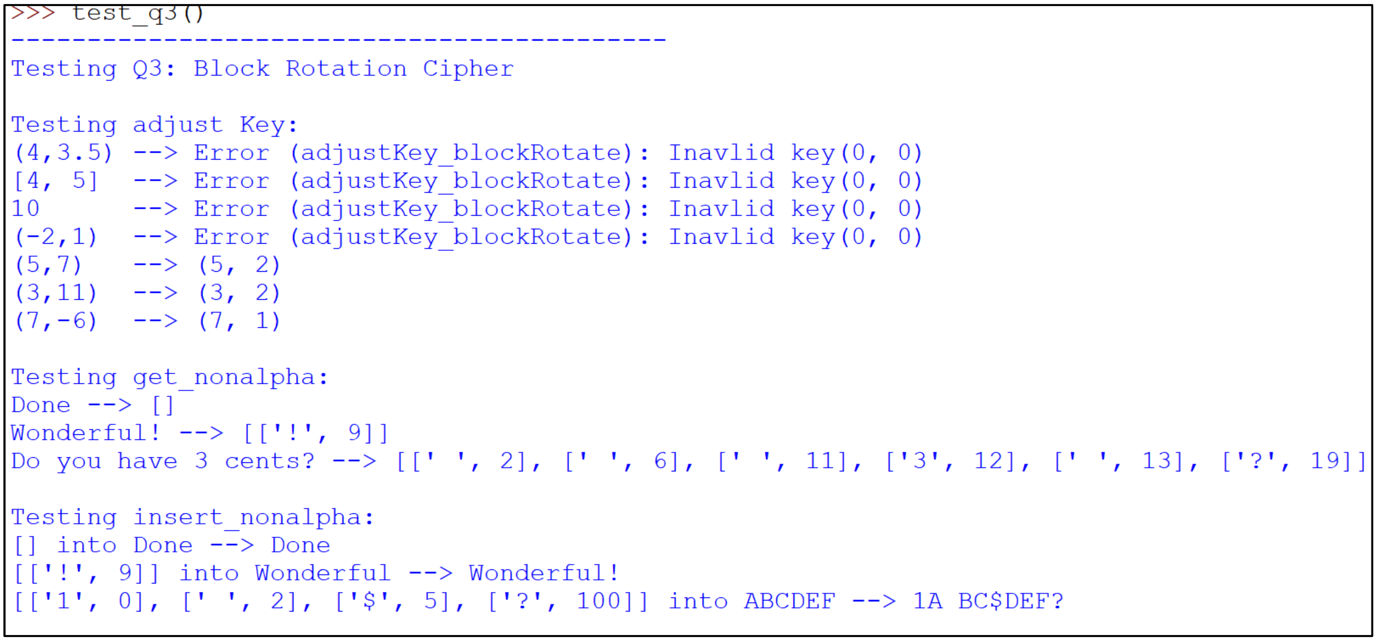
Start by implementing the three given utility functions (to be inserted in solution_A2.py).

```
#-----------------------------------------------------------
# Parameters:   key (b,r)
# Return:       updatedKey (b,r)
# Description:  Assumes given key is in the format of (b(int),r(int))
#               Updates the key in two scenarios:
#               1- The key is too big (use modulo)
#               2- The key is negative
#               if an invalid key is given →
#                     print error message and return (0,0)
#-----------------------------------------------------------
def adjustKey_blockRotate(key):
    # your code here
    return updatedKey
```

```
#-----------------------------------
# Parameters:   text (string)
# Return:       nonalphaList (2D List)
# Description:  Analyzes a given string
#               Returns a 2D list of non-alpha characters along with their
#                positions in the original text
#               Format: [[char1, pos1],[char2,post2],...]
#               Example: get_nonalpha('I have 3 cents.') -->
#                   [[' ', 1], [' ', 6], ['3', 7], [' ', 8], ['.', 14]]
#-----------------------------------
def get_nonalpha(text):
    # your code here
    return nonalphaList
```

```
#-----------------------------------
# Parameters:   text (str)
#               2D list: [[char1,pos1], [char2,pos2],...]
# Return:       modifiedText (string)
# Description:  inserts a list of nonalpha characters in the positions
#-----------------------------------
def insert_nonalpha(text, nonAlpha):
    # your code here
    return modifiedText
```

Testing the above three functions will give:



```
>>> test_q3()
--------------------------------------------
Testing Q3: Block Rotation Cipher

Testing adjust Key:
(4,3.5) --> Error (adjustKey_blockRotate): Inavlid key(0, 0)
[4, 5]  --> Error (adjustKey_blockRotate): Inavlid key(0, 0)
10      --> Error (adjustKey_blockRotate): Inavlid key(0, 0)
(-2,1)  --> Error (adjustKey_blockRotate): Inavlid key(0, 0)
(5,7)   --> (5, 2)
(3,11)  --> (3, 2)
(7,-6)  --> (7, 1)

Testing get_nonalpha:
Done --> []
Wonderful! --> [['!', 9]]
Do you have 3 cents? --> [[' ', 2], [' ', 6], [' ', 11], ['3', 12], [' ', 13], ['?', 19]]

Testing insert_nonalpha:
[] into Done --> Done
[['!', 9]] into Wonderful --> Wonderful!
[['1', 0], [' ', 2], ['$', 5], ['?', 100]] into ABCDEF --> 1A BC$DEF?
```

Next, implement the encryption and decryption functions. The descriptions are provided.

One important observation, occasionally the last block will not be fully occupied by characters. In such scenarios, the rotation is ambiguous: does it apply only to the semi-full block, or to the entire block. One possible solution is to pad the empty spaces in the last block with some dummy character, which should be stripped after decryption. Note, this padding method weakens the security of the scheme (think why that is the case). We will select ‘q’ to be the padding character (think why ‘q’ was selected’).

```
#------------------------------------------------------------
# Parameters:   plaintext (string)
#               key (b,r): (int,int)
# Return:       ciphertext (string)
# Description:  break plaintext into blocks of size b
#               rotate each block r times to the left
#------------------------------------------------------------
def e_blockRotate(plaintext,key):
    # your code here
    return ciphertext
```

```
#------------------------------------------------------------
# Parameters:   ciphertext (string)
#               key (b,r): (int,int)
# Return:       plaintext (string)
# Description:  Decryption using Block Rotate Cipher
#------------------------------------------------------------
def d_blockRotate(ciphertext,key):
     # your code here
     Return plaintext
```

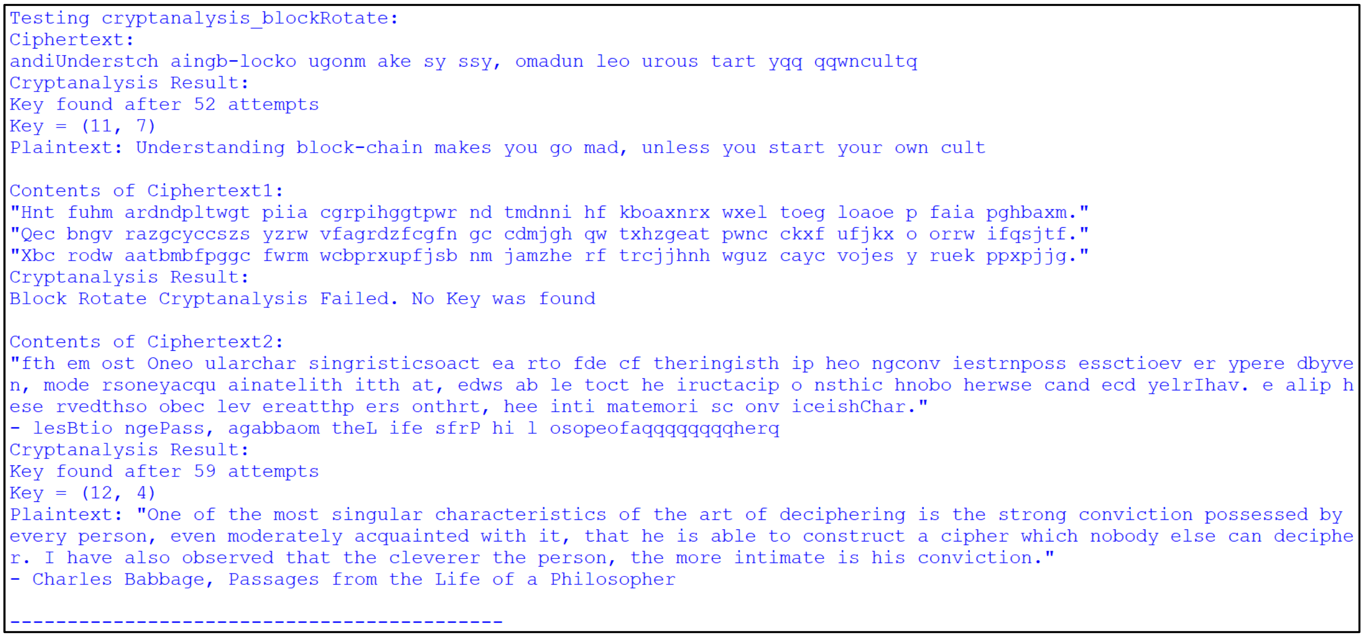
Testing the above two functions yields:

```
Testing e_blockRotate and d_blockRotate
Key =  (4, 3)
plaintext =             abcdefghijklmnopqrstuvwxyz
After encryption:       dabchefglijkpmnotqrsxuvwqyzq
After Decryption:       abcdefghijklmnopqrstuvwxyz

Key =  (10, 6)
plaintext =
The internet, our greatest tool of emancipation, has been transformed into the most
dangerous facilitator of totalitarianism we have ever seen
After encryption:
ern eTheinte, ate tourgrof emst to oltionancipa, ntr ahas beeedinnsfo rmos tdt othe
musfaange rotorocilit ai tarftotalwehaia ni smse enve ever
After Decryption:
The internet, our greatest tool of emancipation, has been transformed into the most
dangerous facilitator of totalitarianism we have ever seen

Key =  (8, 5)
plaintext =
One must acknowledge with cryptography no amount of violence will ever solve a math
problem.
After encryption:
sta Onem uledcknowhc rgew itrapyptogmo uh ynoaio ln tofville ncew olve versh p ream
atqqqob.lem
After Decryption:
One must acknowledge with cryptography no amount of violence will ever solve a math
problem.
```

Finally, implement a cryptanalysis module. The module attempts to break a ciphertext to reveal both b and r. The module attempts block sizes from ***b1*** to ***b2***. The function prints an error message if cryptanalysis fails and returns empty strings.

```
#-----------------------------------------------------------
# Parameters:   ciphertext (string)
#               b1 (int): starting block size
#               b2 (int): end block size
# Return:       plaintext,key
# Description:  Cryptanalysis of Block Rotate Cipher
#               Returns plaintext and key (r,b)
#               Attempts block sizes from b1 to b2 (inclusive)
#               Prints number of attempts
#-----------------------------------------------------------
def cryptanalysis_blockRotate(ciphertext,b1,b2):
     # your code here
     return plaintext,key
```

Running the testing function, will give the following:



```
Testing cryptanalysis_blockRotate:
Ciphertext:
andiUnderstch aingb-locko ugonm ake sy ssy, omadun leo urous tart yqq qqwncultq
Cryptanalysis Result:
Key found after 52 attempts
Key = (11, 7)
Plaintext: Understanding block-chain makes you go mad, unless you start your own cult

Contents of Ciphertext1:
"Hnt fuhm ardndpltwgt piia cgrpihggtpwr nd tmdnni hf kboaxnrx wxel toeg loaoe p faia pghbaxm."
"Qec bngv razgcyccszs yzrw vfagrdzfcgfn gc cdmjgh qw txhzgeat pwnc ckxf ufjkx o orrw ifqsjtf."
"Xbc rodw aatbmbfpggc fwrm wcbprxupfjsb nm jamzhe rf trcjjhnh wguz cayc vojes y ruek ppxpjjg."
Cryptanalysis Result:
Block Rotate Cryptanalysis Failed. No Key was found

Contents of Ciphertext2:
"fth em ost Oneo ularchar singristicsoact ea rto fde cf theringisth ip heo ngconv iestrnposs essctioev er ypere dbyve
n, mode rsoneyacqu ainatelith itth at, edws ab le toct he iructacip o nsthic hnobo herwse cand ecd yelrIhav. e alip h
ese rvedthso obec lev ereatthp ers onthrt, hee inti matemori sc onv iceishChar."
- lesBtio ngePass, agabbaom theL ife sfrP hi l osopeofaqqqqqqqqherq
Cryptanalysis Result:
Key found after 59 attempts
Key = (12, 4)
Plaintext: "One of the most singular characteristics of the art of deciphering is the strong conviction possessed by
every person, even moderately acquainted with it, that he is able to construct a cipher which nobody else can deciphe
r. I have also observed that the cleverer the person, the more intimate is his conviction."
- Charles Babbage, Passages from the Life of a Philosopher

-------------------------------------------
```

Q4: Cipher Detector
(1.5 pts)

Write a function that will detect the cipher type of a given ciphertext. The function does not decrypt the ciphertext or attempt to find the key. The function simply tells the type of the cipher.

The function classifies ciphertexts into one of the following types:

1- Atbash Cipher
2- Spartan Scytale
3- Shift Cipher
4- Polybius Cipher
5- Vigenere Cipher
6- Unknown

The function searches for characteristics of the ciphertext. If these characteristics are found, then based on the choices given, there is a high chance that the ciphertext belongs to one of the given categories.

You are not allowed to create any utility functions for this task. Just create one single function. If you understand how these ciphers work, along with the English language frequencies, your writing of the function would be compact.

Use the Index of Coincidence and the Chi-Squared calculations in your function.

Remember, the value of I for the English language is 0.065 and for a random text is 0.0385. For this task, use a margin of error equal to 0.003. This means, 0.068 and 0.062 would be considered English texts. Similarly, 0.0415 would be considered a random text.

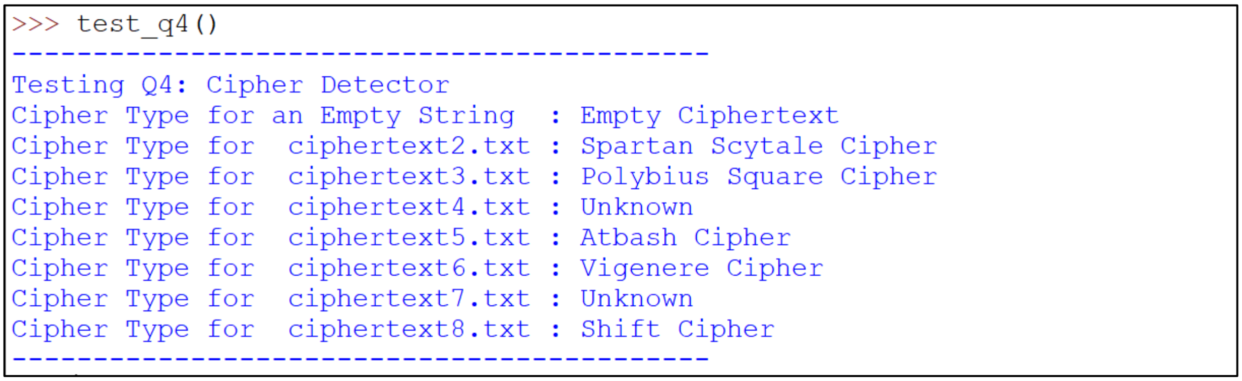
With regards to the Chi-squared statistics, an English text should produce a value that is less than 150.

Note that an English text is a shift cipher with key equal to 0. For the purposes of this task, do not classify such scenario as a shift cipher.

The function description is as follows:

```
#-----------------------------------------------------------
# Parameters:   ciphertext (string)
# Return:       cipherType (string)
# Description:  Detects the type of a given ciphertext
#               Categories: "Atbash Cipher, Spartan Scytale Cipher,
#                   Polybius Square Cipher, Shfit Cipher, Vigenere Cipher
#                   All other ciphers are classified as Unknown.
#               If the given ciphertext is empty return 'Empty Ciphertext'
#-----------------------------------------------------------
def get_cipherType(ciphertext):
     # your code here
     return cipherType
```

Running the testing module should yield the following:



```
>>> test_q4()
-------------------------------------------
Testing Q4: Cipher Detector
Cipher Type for an Empty String  : Empty Ciphertext
Cipher Type for  ciphertext2.txt : Spartan Scytale Cipher
Cipher Type for  ciphertext3.txt : Polybius Square Cipher
Cipher Type for  ciphertext4.txt : Unknown
Cipher Type for  ciphertext5.txt : Atbash Cipher
Cipher Type for  ciphertext6.txt : Vigenere Cipher
Cipher Type for  ciphertext7.txt : Unknown
Cipher Type for  ciphertext8.txt : Shift Cipher
-------------------------------------------
```