Assignment 3

General Instructions:

1. Deadline: Sunday, November 3rd at 11:59 pm. (No late submissions accepted)
2. This assignment is worth 10 pts of your course grade.
3. Download the file: "template_A3.py" and rename it to "solution_A3.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 3
#-------------------------
```

5. There are two other files: "utilities_A3.py" and "test_A3.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_A3.py. The "utilities_A3.py" and "test_A3.py" files should not be changed. If you need to add any other utility function, add it into the solution_A3.py file. Also, you can not import any library other than those provided to you.
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_A3.py file.

Q1: Columnar Transposition Cipher (2 pts)

Your task in this question is to write the encryption and decryption schemes for the columnar transposition. This requires writing three functions:

1- Utility function: get_keyOrder_columnarTrans(key)

2- Encryption function `e_columnarTrans(plaintext, key)`

3- Decryption function `d_colunmnarTrans(ciphertext,key)`

The ***key*** is any English string. The utility function get_keyOrder performs two main tasks. The first is to validate the input key. Invalid keys include empty strings and non-string objects. In that case, the function prints an error message and returns the list [0].

The second objective is to return the order of characters in the key. This is represented as a list in which its elements represent the expected order.

For instance, if key = 'cave' would return [1,0,3,2]. This is explained as:

- since 'a' is the smallest character, then it should be placed in the first column → [1]
- Second smallest character in key is 'c', which should be put in the second column. Since the position of 'c' is 0 → [1,0].
- The third character is 'e' which is in position 3. This should be put in the third column → [1,0,3]
- The last character is 'v' which is in position 2 → [1,0,3,2]

The function description is as follows:

```
#-------------------------------------------------------------
# Parameters:   key (string)
# Return:       keyOrder (list)
# Description:  checks if given key is a valid columnar transposition key
#               Returns key order, e.g. [face] --> [1,2,3,0]
#               Removes repititions and non-alpha characters from key
#               If empty string or not a string -->
#                   print an error msg and return [0] (which is a)
#               Upper 'A' and lower 'a' are the same order
#-------------------------------------------------------------
```

```
def get_keyOrder_columnarTrans(key):
    # your code here
    return keyOrder

#-------------------------------------------------------------
# Parameters:   plaintext (str)
#               kye (str)
# Return:       ciphertext (list)
# Description:  Encryption using Columnar Transposition Cipher
#-------------------------------------------------------------
def e_columnarTrans(plaintext,key):
    # your code here
    return ciphertext

#-------------------------------------------------------------
# Parameters:   ciphertext (str)
#               kye (str)
# Return:       plaintext (list)
# Description:  Decryption using Columnar Transposition Cipher
#-------------------------------------------------------------
def d_columnarTrans(ciphertext,key):
    # your code here
    return plaintext
```

Below are the results of executing the testing module:

```
>>> test_q1()
-------------------------------------------
Testing Q1: Columnar Transposition

Testing get_keyOrder_columnarTrans:
Key order for 34 = Error: Invalid Columnar Transposition Key [0]
Key order for ['O', 'N'] = Error: Invalid Columnar Transposition Key [0]
Key order for  = Error: Invalid Columnar Transposition Key [0]
Key order for 034 = Error: Invalid Columnar Transposition Key [0]
Key order for ?=! = Error: Invalid Columnar Transposition Key [0]
Key order for r = [0]
Key order for K = [0]
Key order for Dad = [1, 0]
Key order for Face = [1, 2, 3, 0]
Key order for apple = [0, 3, 2, 1]
Key order for good day = [3, 2, 0, 1, 4]
Key order for German = [4, 1, 0, 3, 5, 2]
```

```
Testing Encryption/Decryption:
key =  German
plaintext =   DEFENDEASTERNWALLOFTHECASTLE
ciphertext =  NELCqEAWTTDENFSETLEEDROAqFSAHL
plaintext2 =  DEFENDEASTERNWALLOFTHECASTLE

key =  Truth Seeker
Key order = [5, 3, 6, 1, 4, 0, 2]
plaintext1 =
The story was very pretty and interesting, especially at the point where the
rivals suddenly recognized
ciphertext =
tspaegcaphesno oa   ea"Avw inttn"ni ti nniraof tacne t o t wyynipye tadeea;h
eeaCgdPaailtir.mwelpskeehk
plaintext2 =
The story was very pretty and interesting, especially at the point where the
rivals suddenly recognized
-----------------------------------------
```

Q2: Permutation Cipher
(2 pts)

Your task in this question is to write the encryption and decryption schemes for the permutation cipher. This requires writing two functions:

1- Encryption scheme: *e_permutation(plaintext,key)*

2- Decryption scheme: *d_permutation(ciphertext,key)*

There are two modes of encryption and decryption:

- Mode 0: Stream Mode
- Mode 1: Block Mode

The key is tuple of (key,mode), where the valid modes are 0 and 1. If an invalid mode is passed, the encryption and decryption schemes should print an error message and return an empty string. The "key" part of the tuple represent a number composing of some permutation of numbers from 1 to n (sequential unique characters). If an invalid key is passed, an error message should be printed and an empty string is returned.

The stream mode is similar to columnar transposition. Therefore, you should call the functions you created in Q1. However, you should find a way to transfer the key into a form that can be accepted by columnar transposition encryption and decryption functions.

The block mode breaks the text into blocks, each of size equal to number of characters in the key. Each block is transposed through the order presented in the key. The output is the concatenation of the blocks.

In both modes, the encryption should add the character 'q' for padding and the decryption should remove the padded q's.

Below are the function definitions:

```
#------------------------------------------------------------
# Parameters:   plaintext (str)
#               key(key,mode)
# Return:       ciphertext (str)
# Description:  Encryption using permutation cipher
```

```
#               mode 0: stream cipher --> columnar transposition
#               mode 1: block cipher --> block permutation
#               a padding of 'q' is to be used whenever necessary
#-------------------------------------------------------------
def e_permutation(plaintext,key):
    # your code here
    return ciphertext

#-------------------------------------------------------------
# Parameters:   ciphertext (str)
#               key(key,mode)
# Return:       plaintext (str)
# Description:  Decryption using permutation cipher
#               mode 0: stream cipher --> columnar transposition
#               mode 1: block cipher --> block permutation
#               a padding of 'q' is to be removed
#-------------------------------------------------------------
def d_permutation(ciphertext,key):
    # your code here
    return plaintext
```

Running the testing module will yield:

```
>>> test_q2()
-----------------------------------------
Testing Q2: Permutation Cipher

key =  7356412
plaintext =   Love is blind. Friendship closes its eyes

Testing Mode 0 (Stream Cipher):
ciphertext =  idnciss.dltqobFhs  ne  evlrieeeiipsyL  sos
plaintext2 =  Love is blind. Friendship closes its eyes

Testing Mode 1 (Block Cipher):
ciphertext =  sv ieLo.lndi bdreni Fli cpshte isosqeesys
plaintext2 =  Love is blind. Friendship closes its eyes

Testing Mode 2 (Invalid Mode):
Error (e_permutation): invalid mode
ciphertext =

Testing Invalid key
Error (e_permutation): Invalid key
ciphertext =
-----------------------------------------
```

Q3: ADFGVX Cipher
(1.5 pts)

Your task in this question is to write the encryption and decryption schemes for the ADFGVX cipher. This requires writing two functions:

1- Encryption scheme: *e_adfgvx(plaintext,key)*

2- Decryption scheme: *d_adfgvx(ciphertext,key)*

The ADFGVX is a composite cipher of both (a modified version of) Polybius cipher and columnar transposition.

The Polybius cipher uses the following square:

	A	D	F	G	V	X
A	F	L	1	A	O	2
D	J	D	W	3	G	U
F	C	I	Y	B	4	P
G	R	5	Q	8	V	E
V	6	K	7	Z	M	X
X	S	N	H	0	T	9

The utility function `get_adfgvx_square()` is provided to you in the utilities_A3.py file.

The columnar transposition part of the cipher is just as developed in Q1.

Below are the function definitions:

```
#-------------------------------------------------------------
# Parameters:   plaintext(string)
#               key (string)
# Return:       ciphertext (string)
# Description:  Encryption using ADFGVX cipher
#-------------------------------------------------------------
def e_adfgvx(plaintext, key):
    # your code here
    return ciphertext

#-------------------------------------------------------------
# Parameters:   ciphertext(string)
#               key (string)
```

```
# Return:       plaintext (string)
# Description:  Decryption using ADFGVX cipher
#-------------------------------------------------------------
def d_adfgvx(ciphertext, key):
    # your code here
    return plaintext
```

Running the testing module will give:

```
>>> test_q3()
-----------------------------------------
Testing Q3: ADFGVX Cipher

key       =  Berlin
plaintext =  This is the final warning. End the war now!
ciphertext = XxagddgxG xgvVa xx adXx adfdxagaf.dfadqffvaafdvdxfx!dxffagd
dggaqx x ddxdxvd f
plaintext2 = This is the final warning. End the war now!
-----------------------------------------
```

Q4: One Time Pad
(2 pts)

Your task in this question is to write a special implementation of the One Time Pad cipher This requires writing three functions:

1- Utility function: xor_otp(char1,char2)
2- Encryption function `e_otp(plaintext, key)`
3- Decryption function `d_otp(ciphertext,key)`

The first function takes two characters and performs a bit-wise xor operation. The function first converts each character to its ASCII value, then convert the decimal value into an 8-bit binary value. The xor is performed on the binary values. The result binary value is converted into a decimal value, which is translated into an ASCII character.

For example, to perform xor_otp('B','c'), the following steps are performed:

1- In ASCII code, B = 66 and c = 99
2- Converting to binary, B = 0100 0010 and c = 0110 0011
3- The result of xor is = 0010 0001
4- Translating result to decimal = 33
5- Translating result into ASCII = !

The encryption scheme first checks if key length and plaintext is equal. If not equal, an error message is printed and an empty string is returned. The same test is done in the decryption scheme.

Both encryption and decryption functions read all characters in the input, except \n (new line). The new line character is left without encryption/decryption. All other characters, including alphabets (upper and lower), numerical and punctuations are to be treated as input characters.

The encryption scheme xors every plaintext character with the corresponding key character. However, the result is shifted by 32. Using the above example, e_otp('B','c') → 'A' (which is '!' shifted by 32, i.e. 32+33 = 65 which is A).

Note that the encryption scheme might result in an ASCII character that is not visible when printing. To minimize this possibility, the shift by 32 was introduced.

The decryption scheme works in the reverse. However, you would need to de-shift by 32 first then do the xor operation. Remember, the XOR operation is reversible, meaning, Xor-ing twice will restore the original message.

Below are the function definitions.

```
#-------------------------------------------------------------
# Parameters:   plaintext(string)
#               key (string)
# Return:       ciphertext (string)
# Description:  Encryption using One Time Pad
#               Result is shifted by 32
#-------------------------------------------------------------
def e_otp(plaintext,key):
    # your code here
    return ciphertext

#-------------------------------------------------------------
# Parameters:   ciphertext(string)
#               key (string)
# Return:       plaintext (string)
# Description:  Decryption using One Time Pad
#               Input is shifted by 32
#-------------------------------------------------------------
def d_otp(ciphertext,key):
    # your code here
    return plaintext
#-------------------------------------------------------------
# Parameters:   char1 (str)
#               char2 (str)
# Return:       result (str)
# Description:  Takes two characters. Convert their corresponding ASCII value
#               into binary (8-bits), then performs xor operation. The result
#              is treated as an ASCII value which is converted to a character
#-------------------------------------------------------------
def xor_otp(char1,char2):
    # your code here
    return result
```

Running the testing modules will give:

```
>>> test_q4()
-----------------------------------------
Testing Q4: One Time Pad

Testing xor_otp function:
xor(A,b)= #
xor(r,?)= M
xor(!,p)= Q
xor(f,Z)= <

Testing encryption/decryption:
key       =  This Is Not a Random Key
plaintext =  Cryptography is amazing!
ciphertext=  7:0#tF4rO?<yaiAa/).7iE"x
plaintext2=  Cryptography is amazing!
-----------------------------------------
```

Q5: Myszkowski Cipher
(2.5 pts)

Your task in this question is to write a special implementation of the One Time Pad cipher This requires writing three functions:

4- Utility function: `get_keyOrder_myszkowski(key)`

5- Encryption function `e_myszkowski(plaintext, key)`

6- Decryption function `d_myszkowski(ciphertext,key)`

The function definitions are as follows:

```
#-------------------------------------------------------------
# Parameters:   key (string)
# Return:       keyOrder (list)
# Description:  checks if given key is a valid Myszkowski key
#               Returns key order, e.g. [meeting] --> [3,0,0,5,2,4,1]
#               The key should have some characters that are repeated
#               and some characters that are non-repeated.
#               if invalid key --> return [1,1,0]
#               Upper and lower case characters are considered of same order
#               non-alpha characters sould be ignored
#-------------------------------------------------------------
def get_keyOrder_myszkowski(key):
    # your code here
    return keyOrder

#-------------------------------------------------------------
# Parameters:   plaintext(string)
#               key (string)
# Return:       ciphertext (string)
# Description:  Encryption using Myszkowsi Transposition
#-------------------------------------------------------------
def e_myszkowski(plaintext,key):
    # your code here
    return ciphertext

#-------------------------------------------------------------
# Parameters:   ciphertext(string)
#               key (string)
# Return:       plaintext (string)
# Description:  Decryption using Myszkowsi Transposition
#-------------------------------------------------------------
```

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

The first function takes the key as an input, validates it and then returns the order of its characters. In principle, the Myszkowski key should have some repeated characters and some unique characters. Therefore, all of the following are considered invalid keys:

1- A key that is not a string
2- An empty string
3- A string that contains no repeated alphabet characters
4- A string that contains no unique alphabet characters

If an invalid key is submitted, the function should print an error message and return [1,1,0] as the key order. Also, the function should discard any non-alpha characters.

Running the testing function would give:

```
>>> test_q5()
-----------------------------------------
Testing Q1: Myszkowski Cipher

Testing get_keyOrder_myszkowski:
Key order for 34 = Error: Invalid Myszkowski Key [1, 1, 0]
Key order for ['O', 'N', '0'] = Error: Invalid Myszkowski Key [1, 1, 0]
Key order for  = Error: Invalid Myszkowski Key [1, 1, 0]
Key order for 034 = Error: Invalid Myszkowski Key [1, 1, 0]
Key order for ?=! = Error: Invalid Myszkowski Key [1, 1, 0]
Key order for r = Error: Invalid Myszkowski Key [1, 1, 0]
Key order for cc = Error: Invalid Myszkowski Key [1, 1, 0]
Key order for Dad = [1, 0, 1]
Key order for Face = Error: Invalid Myszkowski Key [1, 1, 0]
Key order for apple = [0, 3, 3, 2, 1]
Key order for good day = [2, 3, 3, 1, 1, 0, 4]
Key order for B!AA?CCCD = [1, 0, 0, 2, 2, 2, 3]
```

The encryption scheme is similar to columnar transposition, except that for repeated characters, the plaintext is read in a row format, compared to the column format if the character is unique. Refer to the transposition guide for an example.

Running the testing module would give the following:

```
Testing encryption/decryption:
key       =  Swindon
key order =  [4, 5, 1, 2, 0, 3, 2]
plaintext =  AMIDSUMMERNIGHTSDREAM
ciphertext=  SIEIRDDMNHRMUGAAMTMES
plaintext2=  AMIDSUMMERNIGHTSDREAM

key       =  "Deemed"
key order =  [0, 1, 1, 2, 1, 0]
plaintext =  The Taming of the Shrew
ciphertext=  Tamof SqheTin  tehrw ghe
plaintext2=  The Taming of the Shrew
```