Assignment 3

General Instructions:

1. Create an Eclipse Python project called: "Assignment3".
2. Copy all the files included in the Assignment zip file which include:
    - A3_template.py (You need to rename to A2_solution.py)
    - A3_test.py (testing file)
    - A3_output.txt (sample output)
    - utilities_tempate.py (rename to utilities.py)
    - dictionary file (engmix.txt)
    - 10 text files (input, plaintext and ciphertext files)
3. Enter your credentials on top of the file: A3_solution.py. No submission will be accepted without credentials.
4. Your need to submit ONLY ONE file: called: "A3_solution.py". Do not export from eclipse or submit any additional files.
5. Deadline: Sunday, November 8 at 11:55 pm.
6. You may not include any packages/libraries other than the ones already included.
7. You may create your own utility functions, as long you write them in the file: A3_solution.py. Add the function under the relevant task section. Every new function should have a docstring outlining input parameters, return values and a description.
8. Make sure to test your solution using the given test file: "A3_test.py". You may edit your local version of the test file. However, since you will not be submitting the testing file, such changes will be discarded.
9. Your output should exactly match the output provided in A2_output.txt. You will lose points for any mismatch including missing a space or a dot. Use a text comparison tool https://text-compare.com/ to verify that you have a matching output.

Grading Rubric:

Task	Points	Comments
Task 1: Utilities	0.8	4 functions each is 0.2 pts
Task 2: Vigenere Cipher	1.5	Square (0.1), encryption (0.7), decryption (0.7)
Task 3: Shift Cipher	1.3	Encryption (0.5), decryption (0.5), cryptanalysis (0.3)
Task 4: Cryptanalysis Vigenere Key	1.8	blocks_to_basket(0.2) Friedman (0.4), cipher_shifting (0.4) cryptanalysis key length (0.2) cryptanalysis of key (0.6)

Preparation

Utilities:

The file utilities.py contains only one function:

```
1- get_base(base_type)
```

You need to add the following functions. Use the latest version if multiple versions exist:

```
2- file_to_text(filename)
3- text_to_file(text,filename)
4- text_to_block(text,b_size,padding=0,pad=PAD)
5- def get_positions(text,base)
6- def clean_text(text,base)
7- def insert_positions(text, positions)
8- def shift_string(s,n,d='l')
9- def new_matrix(r,c,fill)
10- def print_matrix(matrix)
11- def is_plaintext(text, dict_list, threshold=0.9)
12- def analyze_text(text, dict_list)
13- def text_to_words(text)
14- def load_dictionary(dict_file)
15- def get_language_freq(language='English')
```

Note how the utilities file contain three constants, which should be used in the solution file whenever necessary:

```
DICT_FILE = 'engmix.txt'
PAD = 'q'
BLOCK_MAX_SIZE = 20
```

A3_solution.py

The columnar transposition cipher will be used in the solution of the permutation and ADFGVX ciphers. Therefore, include the following three functions at the bottom of the file:

```
1- def _get_order_ct(key)
2- def e_ct(plaintext,key)
3- def d_ct(ciphertext,key)
```

Task 1: Utility Functions (0.8 pts)

Provided Utilities:

Create the following utility functions inside the A3_solution.py file:

[1] Compare Texts:

Implement the function: `compare_texts(text1,text2)`. The function compares two strings, character by character, and returns the number of matches.

[2] Letter Frequency

Implement the function: `get_freq(text,base=None)`. The function finds the frequencies (count) of characters in the given text.

If base is `None`, the function returns the frequencies of the English alphabet, where both upper and lower case characters are considered, e.g. upper and lower case '`a`' is counted as one character.

If a base is given, then the function should count the frequency of only the characters defined in the base. In this scenario, the case of the character will be defined based on the base.

The result is a list of integers pointing to the frequencies of characters based on the character order of the base.

[3] Index of Coincidence:

Implement the function: `index_of_coin(text,base_type = None)`. The function computes and returns the index of coincidence of a given text.

The function invokes the `get_freq` function to get the character frequencies using the given `base_type`.

Study the sample output and watch for the scenarios when the value of I is 0.

[4] Chi Squared Statistics:

Implement the function: `chi_squared(text,language='English')`. The function computes the chi squared value for a given text against a specific language. The default language is English.

The function invokes the utility function `get_language_freq`. If the language is supported, the frequency table is returned and is used in the calculation. Otherwise, the function will return an empty list which forces the chi squared to print an error message and return -1.

The function assumes that the frequency of characters is computed over alphabetical characters, i.e. `base = None` in `get_freq` function.

Task 2: Vigenere Cipher (1.5 pts)

In this task you will implement the encryption and decryption functions for the Vigenere Cipher.

Start by implementing the `_viginere_square()` function which constructs a list with the vigenere square. Each element in the list is a shift of the lowercase alphabet. The output will look like the following:

```
-------------- Testing Vigenere Square:
Vigenere Square =
abcdefghijklmnopqrstuvwxyz
bcdefghijklmnopqrstuvwxyza
cdefghijklmnopqrstuvwxyzab
defghijklmnopqrstuvwxyzabc
efghijklmnopqrstuvwxyzabcd
fghijklmnopqrstuvwxyzabcde
ghijklmnopqrstuvwxyzabcdef
hijklmnopqrstuvwxyzabcdefg
ijklmnopqrstuvwxyzabcdefgh
jklmnopqrstuvwxyzabcdefghi
klmnopqrstuvwxyzabcdefghij
lmnopqrstuvwxyzabcdefghijk
mnopqrstuvwxyzabcdefghijkl
nopqrstuvwxyzabcdefghijklm
opqrstuvwxyzabcdefghijklmn
pqrstuvwxyzabcdefghijklmno
qrstuvwxyzabcdefghijklmnop
rstuvwxyzabcdefghijklmnopq
stuvwxyzabcdefghijklmnopqr
tuvwxyzabcdefghijklmnopqrs
uvwxyzabcdefghijklmnopqrst
vwxyzabcdefghijklmnopqrstu
wxyzabcdefghijklmnopqrstuv
```

```
xyzabcdefghijklmnopqrstuvw
yzabcdefghijklmnopqrstuvwx
zabcdefghijklmnopqrstuvwxy
```

Then implement the encryption and decryption functions. Both function invoke the _viginere_square function provided above.

The two functions should handle three scenarios:

1- If the key is a single character, the function uses autokey method for encryption and decryption
2- If the key is a multi-character string, the function uses running key method for encryption and decryption
3- For all other cases, the function prints an error message and returns an empty string.

You may write one big encryption and decryption functions `e_vigenere(plaintext,key)` and `d_viginere(ciphertext,key)` which handle the above cases.

Even better, you can create the following helper functions to be invoked by the generic encryption and decryption functions:

```
1- _e_viginere_auto(plaintext,key)
2- _e_viginere_run(plaintext,key)
3- _d_viginere_auto(ciphertext,key)
4- _d_viginere_run(ciphertext,key)
```

Based on the defined Viginere Square, encryption and decryption will only be applied to alphabetical characters. The case of characters should be preserved.

Task 3: Shift Cipher
(1.2 pts)

In this task you will implement the encryption, decryption and cryptanalysis functions for the Shift Cipher (generic form of Caesar Cipher).

The encryption and decryption functions receive a key represented as a tuple containing number of shifts (int) and a base string (str). Shifts are applied to the base string. Characters not defined in the base string are not encrypted/decrypted.

There are three scenarios:

1- If base is a non-empty string, encryption and decryption are performed on characters defined in the base.
2- If base is None, encryption and decryption are performed on the English alphabets while preserving the base of the characters
3- Otherwise, an error message is printed and an empty string is returned.

The cryptanalysis function: `cryptanalysis_shift(ciphertext, base_type= None)` receives an arbitrary ciphertext and finds the key and plaintext. By default, the ciphertext is retrieved using a key of `(int,None)`. If a base type is provided, then a base string of that type is generated and used in the key.

You need to use the chi squared method while brute forcing through the key space. You may not use the `is_plaintext` function in this function.

Task 4: Vigenere Cryptanalysis (1.9 pts)

In this task you will you will develop some functions used in cryptanalysis of Vigenere Cipher.

The first function is `_blocks_to_baskets(blocks)` which receives a list of blocks, each containing a string. Assume that all blocks are of similar size. The function creates a list of baskets, such that basket[i] contains the i^{th} character from each block in their respective order.

The second function is `friedman(ciphertext)` which uses the Friedman's method to guess the key length of a vigenere ciphertext.

The function computes the Friedman value, and translates it into two keys: the floor and ceiling of that number. For instance, if the value is 6.2 the function produces [6,7] because 6 is floor of 6.2 and 7 is ceiling of 6.2. Since 6.2 is closer to 6 than 7, then it is listed before 7. To generalize, if the freidman value is `#.5` or more then the ceiling value should be listed first.

The third function is `cipher_shifting(ciphertext,max_key = MAX_KEY_L)`. The function uses the cipher shifting method to guess the key length of a vigenere ciphertext. The function shifts the ciphertext and compares the shifted version against the

original ciphertext. The function performs 1 to CIPHER_SHIFT_FACTOR shifts before stopping.

Upon each shift, the chi squared value is computed. The two keys that produce the lowest two chi squared values are to be returned in their respective order.

Note that the number of shifts should to be mapped to the key length. For example, if the key length is 5, then shifts 1 and 6 would mean the same thing. Similarly, if key length is 11, then shifts 1 and 12 would mean the same key. In this last example, if 12 shifts produce the maximum chi squared value, then a key length of 1 should be returned as the first key.

The fourth function: `_cryptanalysis_vigenere_key_length(ciphertext)` combines results of both friedman and cipher shifting methods to produce a list of candidate key lengths. Duplicate copies should be removed. The keys are to be ordered in the following order:

1- Keys that appear in both Friedman and Cipher Shifting
2- Keys that appear in Friedman
3- Keys that appear in Cipher Shifting

The result will be a list of integers representing two to four key lengths.

The last function is `cryptanalysis_vigenere(ciphertext)`. The function first finds the key length candidates using the above functions. Then it brute force through these key lengths (there are 2 to four key lengths) and attempts to decipher the ciphertext. As soon as an English text (use `is_plaintext`) is detected, the key and plaintext should be returned.

For the cryptanalysis of the key, use the method outlined in the class notes (use of baskets and chi_squared).

Make sure that your function is as efficient as possible, or otherwise, your function will take forever in the cryptanalysis.

If the function fails to retrieve the plaintext, it should return two empty strings.

---------- End of Part One ----------