Midterm

Due Time and General Instructions:

This is a take-home exam. It is available from Sunday, October 20 at 9:00 am and is due on Tuesday, October 22 at noon.

It is an open-book exam. Seeking any external assistance is not allowed. This includes collaboration with classmates, using online forums and assistance from any other source. Any detected act of plagiarism will mean immediate failure in the course.

Downloading Your files:

To start your midterm, you need the following files:

File name	Description	Action(s)	Restrictions
template.py	File containing your solution	rename this file to solution.py	you can edit the provided functions but you cannot add any new functions
testing.py	File used for testing your solution	None	Do not edit this file
utilities.py	File hosting utility functions	Add your credentials on top. Add your utility functions. Include only those that are used in your code	You can edit and add functions to your file. You may not include external libraries other than the ones included
Ciphertext files	Four ciphertext files to be decrypted	Make sure to select the files with your name	Do not change these files
Supporting Files	4 sample files + a dictionary file	None	Do not change these files

Midterm Structure:

This midterm is composed of five questions. The first question asks you to identify the cipher types, while the other four functions are ciphers that you need to break. The ciphers are related to four encryption schemes (not listed in order):

1- Vigenere Cipher
2- Simple Substitution Cipher
3- Xshift Cipher
4- Xcrypt Cipher

What to submit:

You need to submit ONE zip file that contains the following 11 files:

1- solution.py
2- utilities.py
3- Four ciphertext files (the ones given to you)
4- Four plaintext files (your outputs)
5- yourname_sub_log.txt file (substitution cipher log)

You do not need to include the testing file or any of the supporting files.

It is critically important that running the testing file should not cause your program to crash. This means that even if you do not provide a solution, you still need to make sure that the testing file, when main() is run, does not crash. You can do this by including empty plaintext files, printing an output on the console that shows what you have tried. Do not edit the testing file.

If your program crashes for any reason, you immediately lose 2 points. If the instructor fails to fix the bug quickly, you will lose 3 more points. You might be asked to come to the instructor's office to demonstrate how to make your program run.

Task 1: Matching your files (2 pts)

Your first task is to perform your analysis to match the given cipher files with their encryption schemes.

In `solution.py`, edit the function: `match_files( )` to reflect your results. Also, provide how you reached such conclusion through editing the print statements. If you need to add any utility function, then you have to do it at the `utilities.py` file.

Running the testing module would give:

```
>>> main()
------------- File Matching -------------
The Vigenere ciphertext file is:ciphertext_Qutaiba_Albluwi_q6.txt
I found that the above file is a vigenere cipher by ...

The Substitution ciphertext file is: ciphertext_Qutaiba_Albluwi_q13.txt
I found that the above file is a substitution cipher by ...

The xshift ciphertext file is:ciphertext_Qutaiba_Albluwi_q11.txt
I found that the above file is an xshift cipher by ...

The xcrypt ciphertext file is: ciphertext_Qutaiba_Albluwi_q8.txt
I found that the above file is an xcrypt cipher by ...
```

Note that the file numbers above are incorrect.

Task 2: Vigenere Cipher
(3 pts)

One of the four ciphertext files was produced by a Vigenere Cipher. Provide the `e_vigenere`, `d_vigenere` and `cryptanalysis_vigenere` functions in your solution.py file. All other utility functions should be placed at utilities.py.

The cryptanalysis function should return the key and recovered plaintext.

In the comments_q1() function you need to provide detailed explanation of how did you guess the key length, and how you guessed the keyword.

It reasonable to say that generating a generic vigenere cryptanalysis tool is very lengthy. Therefore, your cryptanalysis function need to only decrypt properly your file in addition to the sample file.

A sample output for running the testing function:

```
------------- Vigenere Cipher -------------
Testing Encryption/decryption against sample file:
        Encryption Successful
        Decryption Successful

Cryptanalysis for Sample File:
found key =  disposition

Cryptanalysis of ciphertext_Qutaiba_Albluwi_[redacted].txt:
Key found:  disappear

Plaintext:
little boys," sneered Liza.

"There were several adults to-day," he assured her with a faint flush;
but when she tossed her head he had not a word of

Verifying cryptanalysis results: OK

Comments:
Explain here how you found the key length then how you found the keyword
```

Task 3: Substitution Cipher
(5 pts)

One of the four ciphertext files was produced using a Simple Substitution Cipher. The cipher has the following features:

Base String:

The base string is the following:

```
abcdefghijklmnopqrstuvwxyz.,; #"?'!:-
```

The base string consists of 26 characters of the lower case alphabet characters, in addition to the following 11 punctuations: [period, comma, semicolon, space, new line, double quote, question mark, single quote, exclamation mark, colon, hyphen].

Note, how the new line character is represented above as a '#' symbol. This is because it is inconvenient to print the key in two lines. A utility function called: *adjust_key* is provided to you which exchanges the '#' symbol with a new line character and vice versa.

Also note that when printing the above in python console there will be a \ before the single quote (why? Review the basics of python formatting output).

If you would like to save the above in a variable then use triple quotes between the above string. This is because the string contains both single and double quotes characters.

You can hard-code the above in your *get_baseString()* function which should be added to your utilities files.

Substitution String:

The substitution string is a random arrangements of the above string, with one important (and convenient) exception. An alphabet character is always exchanged for an alphabet character. Also, a punctuation character is always

exchange for a punctuation characters. However, there is no way that an alphabet would be exchange for a punctuation character and vice versa.

(Indeed, this is a treasure to know).

Note that the substitution string is random in nature and does not correspond to any English word or phrase.

Key:

Since the base string is known, and we are not using any password (passphrase) in the substring, the key is simply the substitution string.

Encryption and Decryption

This is a mono-alphabetic simple substitution. Both encryption and decryption get the substitution string as a key. Both schemes take consideration of the case of the alphabet characters. This means, an upper case in plaintext should correspond to an upper case in ciphertext, and vice versa. The same is also true for lower case characters.

You would need to implement both *e_substitution* and *d_substitution* in your solution file. You can add utility functions as you like at the utilities.py file. Before proceeding to cryptanalysis, make sure that your encryption and decryption scheme work for the given sample file.

Cryptanalysis:

You are not required to create an automated tool for cryptanalysis of the above substitution cipher. Instead you need to do that manually while documenting all your steps.

Your *cryptanalysis_substitution* function should simply hard-code the substitution string which you have manually found, decrypt the ciphertext and return both the key (substitution string) and the recovered plaintext.

Cryptanalysis Log:

Use the debug tool given to you to produce a detailed description of your cryptanalysis steps. Initially, you will have a lot of predicted guesses, elimination, and try-and-error. However, once you have done, you need to provide detailed replay of all of your steps. This means running the debug tool, enter your exchange (replace) commands, provide a description of why made that replacement, then move to the next command.

Overall, there will be 37 exchanges before entering the 'end' command. Once done, copy the contents of your console to a file called: "yourname_sub_log.txt'.

To effectively do the above, you need to document all your steps in a separate text file and insert it to the console, sequentially. A sample log output is given to you. Note that the log output is just a sample, with no proper descriptions provided.

To execute the debug function use the following commands:

```
>>> baseString = utilities.get_baseString()
>>> baseString = utilities.adjust_key(baseString)
>>> ciphertext = utilities.file_to_text('ciphertext_subcipher_sample.txt')
>>> utilities.debug_substitution(ciphertext,baseString)
```

A sample output of running the above debug program on the sample files is available for you at "sample_sub_log.txt" file.

Testing:

Running the testing module should give you something like:

```
---------- Substitution Cipher ------------
Testing Encryption/decryption against sample file:
        Encryption Successful
        Decryption Successful

Cryptanalysis for Sample File:
found key =  poejiqsrbltxwaznfcdhmvgkuy:'" !.?,-#;

Cryptanalysis of ciphertext_Qutaiba_Albluwi_[redacted].txt:
Key found:  yhribekafszmpxwojductnvlgq!-#;". '?,:

Plaintext:
Princess Mary, who had
come up to him.

Princess Mary looked at her brother in surprise. She did not understand
what he was laughing at. Everything he

Verifying cryptanalysis results: OK

Comments:
See Qutaiba_Albluwi_sub_log.txt file
```

Task 4: XShift Cipher
(5 pts)

One of the four ciphertext files was produced using a Special Shift Cipher, which will be referred to as: xshift cipher.

In class, the shift cipher always used a shiftString equal to the lower case alphabet. However, the xshift cipher uses a shiftString that has the following form [Set 1][Set 2], i.e. the concatenation of two sets. The two sets satisfy the following characteristics:

1- Both sets contain only alphabetical characters
2- One set is upper case and one set is lower case.
3- Each set could be either regular alphabet or an Atbash alphabet (the alphabet reversed).

The above criteria will always produce different two sets. The result is a unique shiftString of length 52 characters (26+26). Examples include, but not limited to, [Atbash upper][regular lower], [regular lower][regular upper] and [atbash lower][atbash upper].

Shifting is always done to the left. Number of shifts can be any positive integer. The key is the tuple (shiftString, shifts).

Start by implementing *e_xshift* and *d_xshift*. Then implement *cryptanalysis_xshift*.

Your brute-force should be "smart". If you use an exhaustive brute-force with all possible combinations and possible shifts, and you successfully break the cipher, you will get 3 out 5. A smart brute-force would require analyzing the ciphertext for finding out the possible combination(s) and then brute-forcing through a pre-defined range. In the comments function, calculate the brute-force total space, then provide a description of your brute-force design highlighting how much it is less than the total space.

Running the testing function will give:

```
------------- Xshift Cipher -------------
Testing Encryption/decryption against sample file:
        Encryption Successful
        Decryption Successful

Cryptanalysis for Sample File:
found key =  ('abcdefghijklmnopqrstuvwxyzZYXWVUTSRQPONMLKJIHGFEDCBA', 22)

Cryptanalysis of ciphertext_Qutaiba_Albluwi_[redacted].txt:
Key found:  ('abcdefghijklmnopqrstuvwxyzZYXWVUTSRQPONMLKJIHGFEDCBA', 36)

Plaintext:
their nests.  Leaning over
the battlements and looking far down, I surveyed the grounds laid out
like a map: the bright and velvet lawn closely girdling the grey base of
the mansion; the field, wide as a park, dotted with its ancient timber;
the wood, dun and sere, divided by a path visibly overgrown, greener with
moss than the trees were with foliage; the church at the gates, the road,
the tranquil hills, all reposing in the autumn day's sun; the horizon
bounded by a propitious sky, azure, marb

Cryptanalysis Successful? Yes

Comments:
Explain here your brute-force design
```

Task 5: Xcrypt Cipher
(5 pts)

One of the four ciphertext files was produced using an encryption scheme which we will call Xcrypt.

The Xcrypt encryption works through writing a plaintext in a table, character by character in a vertical manner. The key represents the number of rows.

Here is an example:

Plaintext: THISMIDTERMISEASY

Key = 4

T	M	E	S	Y
H	I	R	E	q
I	D	M	A	q
S	T	I	S	q

Note how a padding of ‘q is used to make sure that the table is complete.

A ciphertext will be produced through reading the above table horizontally.

Using the above example, the ciphertext is: TMESYHIREqIDMAqSTISq.

The scheme considers all characters in the plaintext, including alphabets (both cases), numerical characters, punctuations, special symbols, new line and space.

Encrypting the file: plaintext_xcrypt_sample.txt using a key of 82 will produce the file: ciphertext_xcrypt_sample.txt.

You would need to edit the three functions: `e_xcrypt`, `d_xcrypt` and `cryptanalysis_xcrypt`.

Start by writing your encryption and decryption schemes and test it against the given sample files. The testing module does that for you.

Then write your cryptanalysis function. The function should use brute-force to produce the correct plaintext file, which should be called: `plaintext_yourname_qX.txt`. where the X character should match the corresponding character in the file name of `ciphertext_yourname_qX.txt`.

The key can be any value from 1 to 500.

The testing function will also produce something similar to the following:

```
------------- Xcrypt Cipher -------------
Testing encryption over sample file:
Encryption Successful

Testing decryption over sample file:
Decryption Successful

Cryptanalysis of ciphertext_Qutaiba_Albluwi_[redacted].txt:
Key found:  109

Plaintext:
I'll go home with
you. I wouldn't frighten Kate-poor little Kate. Give me your hand! Have
you a cab?"

"Yes, I have one waiting."

"Then I shall go in

Cryptanalysis Successful? Yes

My Comments:
Used threshold is:  0.97
Cryptanalysis Method: blablalba
```