CMPUT 331, Fall 2020, Assignment 1

Before beginning work on this assignment, carefully read the Assignment Submission Specifications posted on eClass.

The Caesar Cipher does not offer much security as the number of keys is small enough to allow a brute-force approach to cryptanalysis. However, with some modifications, it can be greatly improved. In this assignment, you will modify the Caesar Cipher implemented by the textbook ("caesarCipher.py") to improve its encryption strength.

You will produce four files for this assignment:

- **a1p1.py, a1p2.py, a1p3.py**: Python solutions to problems 1, 2, and 3 respectively
- **a1.pdf**: Written responses to problem 4.

Submit your files in a zip file named 331-as-1.zip.

Problem 1 (2 Marks): For this problem, modify the provided skeleton file, 'a1p1.py". You may use the provided "caesarCipher.py" file as reference to implement two functions named "encrypt" and "decrypt" each taking two arguments:

- message: plaintext string to encrypt
- key: one of the 52 uppercase/lowercase letters corresponding to the shift amounts: A = 0, a = 1, B = 2, b = 3, C = 4, c = 5... etc.

Your encrypt function should return the message string such that each letter has been caesar-shifted by the key amount. Non-letters characters (such as spaces or punctuation) are to be appended to the resulting string unmodified. Your decrypt function should reverse the shift performed by encrypt and return the original plaintext. The following demonstrates invocation sequences that should run without error and produce identical output:

```
python3 -i  a1p1.py
>>> encrypt("encrypted with a shift of thirteen", "g")
'LUJYFWALK DPAO H ZOPMA VM AOPYALLU'
>>> decrypt("LUJYFWALK DPAO H ZOPMA VM AOPYALLU", "g")
'encrypted with a shift of thirteen'
>>> encrypt("AAA", "a")
'aaa'
>>> decrypt("AAA", "a")
'zzz'
```

Problem 2 (2 Marks): One weakness of the Caesar Cipher comes from the fact that every letter is enciphered the same way. If the first letter of the message is shifted forward by three positions, *every* letter is shifted forward by three positions. Let's start by fixing this flaw. Make a copy of your "a1p1.py" code named "a1p2.py" and modify it such that:

1. The first letter of the message is shifted according to the chosen key, exactly as before (i.e. for encryption a key of 'b' shifts the first letter up by 3).
2. All remaining letters are shifted using the previous letter of the message as a key. If the previous character is punctuation, use the letter before it.

```
python3 -i  a1p2.py
>>> encrypt("!gold", "b")
'!IVAP'
>>> decrypt("!IVAP", "b")
'!gold'
```

Problem 3 (2 Marks): Now make another copy of your "a1p1.py" code and name it "a1p3.py" (Note: copy a1p1.py and not a1p2.py). Modify your encrypt and decrypt functions so that the key is a string instead of a single letter. We will call this string the *keyword*. Modify your code so that it enciphers a message as follows:

1. If the keyword is n characters long, then the 1st, 2nd, 3rd..., and nth letters of the message are enciphered using the 1st, 2nd, 3rd..., and nth letters of keyword respectively. You may assume that the keyword only contains upper and lowercase letters. If the message is shorter than the keyword then the extra characters in the keyword are to be ignored.
2. The rest of the string is enciphered using the above method except as if the (n+1)th character is the beginning of the string.

```
python3 -i a1p3.py
>>> encrypt("AAA!AAA", "AaBb")
'AaB!bAa'
>>> decrypt("AaB!bAa", "AaBb")
'AAA!AAA'
```

Problem 4 (4 Marks): Short answer: For this problem, consider the encryption strength of the original caesar cipher and each of your modified ciphers. Some of the ciphers, multiple valid keys that result in plaintext being enciphered in the same way. How many distinct keys, keys which each produce different ciphertexts for the same message, do each of the four ciphers have? Is the problem 2 cipher stronger than the original caesar cipher? Can you think of any weaknesses your Caesar Cipher modifications have that do not pertain to the number of keys and that may allow an attacker to easily guess your key? (Hint think about lowercase and uppercase letters). Discuss how this weakness could be addressed. Include your responses in your a1.pdf.