CSCI 1133, Spring 2015 — Assigned: Friday March 13, 2015
Programming Assignment 7 — Due: 11:55pm Thursday March 26, 2015

This is ***not*** a collaborative assignment; you must design, implement and test the solution(s) on your own. You may not consult or work with anyone other than the course instructor or TAs. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class. Obtaining or *providing* solutions to any homework problems for this class is considered academic misconduct. If you are not sure what this means, consult the class syllabus or discuss it with the course instructor.

This assignment requires writing a single Python script that must be submitted online *prior* to the due date/time. Late submissions will not be accepted. Name your source code: `hw7.py` Submit your source code file using the appropriate homework submission link on the Moodle website.

The total point value for programming assignments will be awarded for solutions that are *complete*, *correct*, and *well constructed*. A *"well constructed"* program entails good design, appropriate comments and general readability (descriptive names for variables and procedures, appropriate use of blank space, etc.). The following will result in a score reduction equal to a percentage of the total possible points:

- Incorrectly named/submitted source file (10%)
- Constraints not followed (40%)
- Failure to execute due to syntax errors (30%)

Note that your work will be graded using, and must function correctly with, the current version of Python 3 on CSE Labs UNIX machines. If you complete your programming assignment using a different system, it is your responsibility to ensure your programs work on CSELabs machines *prior* to submitting them.

A. (40 *points*) **Substitution Cipher**

The study of *cryptography* is concerned with methods for converting, or *encrypting*, readable messages (called *plaintext*) into unreadable messages (called *ciphertext*). This is generally done to keep the contents of the original plaintext message "hidden" from other people. There are many encryption schemes, and modern methods are quite sophisticated and secure.

The *Simple Substitution Cipher* is an ancient method for encryption that involves scrambling the message by "swapping" individual letters in the plaintext message with letters from a secret *key* alphabet. You may recognize it in the form of the "Cryptoquip" that generally appears in the daily newspaper. It works like this (where '_' actually represents the "blank space" character, " "):

For example, given the plaintext alphabet and secret key below, if you want to encrypt the message "hi mom", you would begin by finding the ordinal of the letter 'h' and replace 'h' with the letter from the secret key with the same ordinal: 'n'. Then, you would find the ordinal of the letter 'i' and replace it with the secret key letter at the same ordinal: 'o', then the blank which corresponds to the secret key value 'f' and so on. Completing for all the letters in the plaintext, "hi mom" is encrypted as: "nofsus"!

```
Plaintext alphabet:  a b c d e f g h i j k l m n o p q r s t u v w x y z . , ! _
                     | | | | | | | | | | | | | | | | | | | | | | | | | | | |
Secret key:          g h i j k l m n o p q r s t u v w x y z . , ! _ a b c d e f
```

Write a Python module consisting of three functions that will encrypt and decrypt simple substitution ciphers as follows:

1. `encryptMsg(plaintext,key,alphabet)`

   Takes a plaintext string, an alphabet string and a secret key string as arguments and returns an encrypted cipher string. Note, within this function, you must first convert the plaintext string to all lower case and remove any punctuation/characters that do not appear in the alphabet string!

2. `decryptMsg(ciphertext,key,alphabet)`

   Will take a ciphertext string, an alphabet string and a secret key string and return the plaintext string.

3. `makeKey(alphabet)`

   Generate and return a secret-key string by randomly shuffling the characters in the alphabet string argument. Hint: this involves turning the string into a list, using the `random.shuffle()` method, then turning the list back into a string.

Test your functions by doing the following (see examples below):

1. Call makeKey() to construct a random secret-key string for some alphabet (you may use the alphabet described above if you wish)
2. Print the alphabet string and the generated secret-key string
3. Input some plaintext message that you would like to encrypt
4. Encrypt the plaintext message by calling encryptMsg() and print out the returned ciphertext
5. Decrypt the ciphertext by calling decryptMsg() and print out the returned plaintext

When you are convinced your functions are working correctly, remove the test code and submit the module with only the function definitions as described.

Constraints:

- You may use any string or list method that is appropriate to solving this problem
- Write the functions using basic string and list functions/methods. You may only import the random module
- You must name the function definitions as described
- You must use function arguments as described above
- You must remove all scaffolding prior to submission

Examples:

```
Alphabet: 'abcdefghijklmnopqrstuvwxyz.,! '
Key:       'nu.t!iyvxqfl,bcjrodhkaew spzgm'
Input plaintext: Hey, this is really fun!
Cipher text:    'v! zmhvxdmxdmo!nll mikbg'
Decrypted text: 'hey, this is really fun!'
```