ENGI 1006 HW3 (Chapters 4, 5 in 3rd edition or 4, 6 in 2nd edition)

Assignment Due on 11/03/2016 at 11:59PM

Academic Honesty Policy:
You are expected to submit **your own work**. Assignments are to be completed individually, NOT in groups. No collaboration is permitted. Please do not include any of your code snippets or algorithms in public Piazza posts. You cannot not use solutions from any external source. If you have any questions at all about this policy please ask the course staff before submitting an assignment.

For this, and all future homeworks, we'll have one main Piazza post for the homework overall, as well as more specific posts for each homework question.
General questions should be asked as follow-up discussions beneath the main Piazza post: https://piazza.com/class/isq8q3q8iev6dp?cid=90

Please note that we will be modifying this main Piazza post with clarifications on the homework where necessary, and you will be responsible for adhering to them.

If you have a question, please post a follow-up discussion *on the appropriate post.* Any relevant discussions will be incorporated as a clarification in the main post.

For all problems:

- **A skeleton of each function has been provided for you on Canvas. DO NOT modify the function signatures or return variables for any reason. Your job is to complete the body of each function. We've included test cases along with expected output so you can check your solutions. These test cases are under the line:**

  ```
  if __name__ == "__main__":
  ```

  You can modify these test cases within the skeleton code.

- **Use Python 3.5 for all Python parts. To check your Python version, enter "python --version" (no quotes) in a terminal.**

- **Make a comment at the top of each file explaining the high-level logic of your code. Make in-line comments detailing specific decisions and logic where necessary.**

- **You will be graded on your code style. Please refer to the Python style guide available on Courseworks in the resources folder.**

- **Download the skeletons from Canvas and keep names the same:**

- **make_histogram.py**
- **string_compression.py**
- **is_SNP.py**
- **file_split.py**
- **cryptography.py**

For submitting, put all files into a single folder named UNI_hw3. Compress your UNI_hw3 folder into a zip file named UNI_hw3.zip (eg. abc1234_hw3.zip). Make sure to use an underline, NOT a hyphen. Submit that zip file via courseworks.

1. make_histogram(a) (15 pts)

Write the function make_histogram(a) that takes a non-empty list of exam scores between 0 and 100, both inclusive, and returns a string representing a histogram of that data, with * representing one count in the histogram. Use string formatting to format the strings nicely and display them as shown. You are expected to follow the output string as shown in the skeleton test case (in terms of whitespace, newlines, alignment, width, etc.) Namely, all the leftmost * must be vertically aligned.

The bins should be: [0-9], [10-19], … [80-89], [90-99], [100], with all ranges [inclusive, inclusive].

For example: make_histogram([73, 65, 92, 74, 100, 70]) returns:

"""

00-09:

10-19:

20-29:

30-39:

40-49:

50-59:

60-69: *
70-79: ***
80-89:
90-99: *

100: *

"""

Make your code as efficient as possible.

2. String compression (20 pts)

In this question you will be asked to write a pair of functions, compress_string(s) and decompress_string(s), both within string_compression.py

a)

compress_string(s) takes a string s, composed only of lowercase letters, and returns a compressed version of s. The compression works like this: every time you encounter a consecutively repeated character in the string, the substring of that repetition is replaced in the compressed string with the number of times the letter was repeated, followed by the letter itself. For example:

>> my_string = “aaabccaaadaccbbb”

>> my_compressed_string = compress_string(my_string)

>> print(my_compressed_string)

>> 3a1b2c3a1d1a2c3b

Notice how singleton characters are compressed in the same way.

b)

decompress_string(compressed_s) takes a string that was properly compressed by compress_string() and returns decompressed version of it (which should be the string that was originally passed into compress_string()).

Picking up from the example above:

>> my_decompressed_string = decompress_string(my_compressed_string)

>> print (my_decompressed_string)

>> aaabccaaadaccbbb

Hint: Be careful about handling multi-digit counts

3. is_SNP(s1,s2) (20 pts)

DNA is a molecule that carries the genetic instructions used in the growth, development, functioning and reproduction of all known living organisms and many viruses. DNA is composed of simpler monomer units called **nucleotides**. There are four types of nucleotides, A (adenine), T (thymine), G (guanine) and C (cytosine). We can view DNA sequences as a string containing As, Ts, Gs and Cs, such as “AGATCTA” and “TGAATAGA”.

Single-nucleotide polymorphism, or SNP, is a variation in a **single** nucleotide of the DNA. The variation can be either insertion, deletion or mutation.

Insertion: you could turn string s1 into string s2 by inserting a certain character somewhere within s1, including before the beginning and after the end.

Deletion: you could turn string s1 into string s2 by removing a single character from s1

Mutation: you could turn string s1 into string s2 by changing a single character somewhere within s1

For example, these two sequences:

“AGATCTA”

“AGATCGA”

Compose a valid SNP, while

“AGATCTA”

“CCCCCCC”

do not.

Also note that going from “AGATCTA” to “AGATCTA” is not a valid SNP, as there is no variation.

With this in mind, write a function is_SNP(s1,s2), that takes two valid DNA strings (either or both possibly empty), and returns True if it is a valid SNP, and False otherwise.

4. File split (15 pts)

Write a function file_split(f) that splits a file f.txt into two new files: f_odd.txt and f_even.txt such that all odd-numbered lines go in f_odd.txt and all even-numbered lines go in f_even.txt.

For example, if we call file_split(“great1006”), then the function should split the file “great1006.txt” (which is in the same directory of this python file) into two files “great1006_odd.txt” and “great1006_even.txt” according to the rules above, and write those files within the same directory as great1006.txt

5. Cryptography (30 pts)

(a) Write a pair of functions encrypt_message(message, key) and decrypt_message(message, key) that allow one to encrypt and decrypt a message (which is a string) as follows:

To encrypt, take each alphabetical letter in the message and shift it by a certain number denoted by the *key.* For example, if key=1 and you encountered an A, you would shift the letter A by one position to the right and you would get a B as its encrypted value. If key=3 and you encountered a Z, you would shift letter Z by 3 positions, to the right, which would go Z → A → B → C you would wind up at C.

Decryption is the opposite of encryption, so to decrypt, take each encrypted letter in the message and shift it back by the *key* used to encrypt the message. For example, if you decrypt letter C by moving one position to the left (which means that key = 1) you get a B, if you shift letter C by 2 positions (key=2) you get an A.

>> encrypt_message(“Hello Zoe!”, 2)

“Jgnnq Bqg!”

Let *key* be an integer value between 1 and 26. Make sure you handle the upper and lower letters properly (upper wraps back around to upper, lower wraps to lower). Anything that is not a letter should not be encrypted.

(b) Suppose you receive an encrypted message but you didn’t receive a key to decrypt it. Write a search_key(message) function that would try all keys between 1 and 26 and returns a list consisting of [key, possible decrypted message] pairs. The return value would be a 26 x 2 dimensional list.

(c) Write a function encrypt_letter(file_name, key) that reads a file named file_name.txt and uses the functions defined in a) to encrypt it. Output the result to a new file named “file_name_encrypted.txt”. (The word “letter” in the function name refers to the item contained within an envelope, not an individual character).

For example, if we have “Hello Zoe!” in a file “great1006.txt”, then encrypt_letter(“great1006”, 2) will create a new file called “great1006_encrypted.txt” that contains “Jgnnq Bqg!”.

(d) Write a function decrypt_letter(file_name, key) that uses the functions defined in (c) to decrypt a text file named “file_name_encrypted.txt” and create a new file “file_name_decrypted.txt” composed of the decryption of file_name_encrypted.txt