proj12: Ceasar Cipher

Encryption is the process of obscuring information to make it unreadable without special knowledge. For centuries, people have devised schemes to encrypt messages — some better than others — but the advent of the computer and the Internet revolutionized the field. These days, it's hard not to encounter some sort of encryption, whether you are buying something online or logging into google accounts.

A cipher is an algorithm for performing encryption (and the reverse, decryption). The original information is called plaintext. After it is encrypted, it is called ciphertext. The ciphertext message contains all the information of the plaintext message, but it's not in a format readable by a human or computer without the proper mechanism to decrypt it; it should resemble random gibberish to those not intended to read it.

A cipher usually depends on a piece of auxiliary information, called a key. The key is incorporated into the encryption process; the same plaintext encrypted with two different keys should have two different ciphertexts. Without the key, it should be difficult to decrypt the resulting ciphertext into readable plaintext.

This project will deal with a well-known (though not very secure) encryption method called the Caesar cipher. In this problem set you will need to devise your own algorithms and will practice using recursion to solve a non-trivial problem.

Caesar Cipher
In this project, we will examine the Caesar cipher. The basic idea in this cipher is that you pick an integer for a key, and shift every letter of your message by the key. For example, if your message was "hello" and your key was 2, "h" becomes "j", "e" becomes "g", and so on.

In this project, we will use a variant of the standard Caesar cipher where the space character is included in the shifts: space is treated as the letter after "z", so with a key of 2, "y" would become " ", "z" would become "a", and " " would become "b".

Specification

Problem 1. Encryption and Decryption
Write a program to encrypt plaintext into ciphertext using the Caesar cipher. Once you've written this function, you should be able to use it to encode strings.

Problem 2. Code-breaking
Your friend is really excited about the program she wrote for Problem 1 of this problem set. She sends you emails, but they're all encrypted with the Caesar cipher!

The problem is, you don't know which shift key she is using. The good news is, you know your friend only speaks and writes English words. So if you can write a program to

find the decoding that produces the maximum number of words, you can probably find the right decoding (There's always a chance that the shift may not be unique. Accounting for this would probably use statistical methods that we won't require of you.)

Part a: Pseudocode
Write out an algorithm you could use to solve this problem.

PSEUDOCODE

Pseudocode is writing out the algorithm/solution in a form that is like code, but not quite code. Pseudocode is language independent, uses plain English (or your native language), and is readily understandable. Algorithm related articles in wikipedia often use pseudocode to explain the algorithm.

Think of writing pseudocode like you would explain it to another person – it doesn't generally have to conform to any particular syntax as long as what's happening is clear.

Pseudocode is a compact and informal high-level description of a computer programming algorithm that uses the structural conventions of a programming language, but is intended for human reading rather than machine reading. Pseudocode typically omits details that are not essential for human understanding of the algorithm, such as variable declarations, system specific code and subroutines. The purpose of using pseudocode is that it is easier for humans to understand than conventional programming language code, and that it is a compact and environment-independent description of the key principles of an algorithm. No standard for pseudocode syntax exists, as a program in pseudocode is not an executable program.

Part b: Python code
Implement find_best_shift. This function takes a wordlist and a bit of encrypted text and attempts to find the shift that encoded the text. A simple indication of whether or not the correct shift has been found is if all the words obtained after a shift are valid words. Note that this only means that all the words obtained are actual words. It is possible to have a message that can be decoded by two separate shifts into different sets words. While there are various strategies for deciding between ambiguous decryptions, for this problem we are only looking for a simple solution.

To assist you in solving this problem, we have provided a helper function: is_word(wordlist, word). This simply determines if word is a valid word according to wordlist.

This function ignores capitalization and punctuation.
Hint: You may find the function string.split to be useful for dividing the text up into words.

Once you've written this function you can decode your friend's emails!

Extension

Problem 3. Multi-level Encryption & Decryption

Clearly the basic Caesar cipher is not terribly secure. To make things a little harder to crack, you will now implement a multi-level Caesar cipher. Instead of shifting the entire string by a single value, you will perform additional shifts at specified locations throughout the string. This function takes a string text and a list of tuples shifts. The tuples in shifts represent the location of the shift, and the shift itself. For example a tuple of (0,2) means that the shift starts are position 0 in the string and is a Caesar shift of 2. Additionally, the shifts are layered. This means that a set of shifts [(0,2), (5, 3)] will first apply a Caesar shift of 2 to the entire string, and then apply a Caesar shift of 3 starting at the 6th letter in the string.

Problem 4. Multi-level Code-breaking

Your friend has sent you another message, but this one can't be decrypted by your solution to Problem 2 — it must be using a multi-layer shift.
To keep things from getting too complicated, we will add the restriction that a shift can begin only at the start of a word. This means that once you have found the correct shift at one location, it is guaranteed to remain correct at least until the next occurrence of a space character.

Part a: Pseudocode

As in Problem 2, Part b, we want you to sketch out a high level step-by-step algorithm for solving this problem. *HINT*: Use recursion.

Part b: Python code

To solve this problem successfully, we highly recommend that you use recursion (did we say use recursion again?). The non-recursive version of this function is much more difficult to understand and code. The key to getting the recursion correct is in understanding the seemingly unnecessary parameter "start". As always with recursion, you should begin by thinking about your base case, the simplest possible sub-problem you will need to solve. What value of start would make a good base case? (Hint: the answer is NOT zero.)

To help you test your code, we've given you two simple helper functions:

random_string(wordlist, n) generates n random words from wordlist and returns them in a string.

random_scrambled(wordlist, n) generates n random words from wordlist and returns them in a string after encrypting them with a random multi-level Caesar shift. You can start by making sure your code decrypts a single word correctly, then move up to 2 and higher.

NOTE: This function depends on your implementation of apply_shifts, so it will not work correctly until you have completed Problem 3.

Problem 5. The Moral of the Story

Now that you have all the pieces to the puzzle, please use them to decode the file, fable.txt. At the bottom of the skeleton file, you will see a method get_fable_string() that will return the encrypted version of the fable.