Before beginning work on this assignment, carefully read the Assignment Submission Specifications posted on Canvas. Importantly, note that this assignment uses a modified 25-letter alphabet.

Introduction

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") and investigate how this affects its encryption strength.

You will produce five files for this assignment:

- **a1p1.py, a1p2.py, a1p3.py**: python solutions to problems 1, 2, and 3 respectively
- **a1.txt**: written responses to problem 4
- a README file (**README.txt**)

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 three functions named "get_map", "encrypt", and "decrypt".

The function "get_map" takes one argument:

- letters: a string of letters such as "ABCD...", where each letter corresponds to its shift amount A = 0, B = 1, C = 2, etc.

The "get_map" function plays a crucial role in establishing the connections between letters and their respective shift amounts. It should return two dictionaries: the first mapping each letter to its corresponding shift, and the second allowing the reverse mapping of shifts to their respective letters. These dictionaries are essential references for the other functions, for example, enabling efficient querying of shift amounts based on the given letter. While the default argument value is the 25-letter alphabet used in this assignment, "ABCD...", your implementation should seamlessly accommodate others, such as "ZYXW...". These variations will be tested during the assignment evaluation. Example output of "get_map":

```
char_to_index = {'A': 0, 'B': 1, 'C': 2, ...}

index_to_char = {0: 'A', 1: 'B', 2: 'C', ...}
```

The "encrypt" and "decrypt" functions take two arguments:

- message: plaintext string to encrypt
- key: a letter corresponding to a shift amount, for instance, A = 0, B = 1, C = 2... etc.

The "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. The "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:

```
>>> test(); encrypt("WELCOME TO 2026 WINTER CMPUT 331!", "X")
'TBHZLIB QL 2026 TFKQBO ZIMRQ 331!'
>>> test(); decrypt("TBHZLIB QL 2026 TFKQBO ZIMRQ 331!", "X")
'WELCOME TO 2026 WINTER CMPUT 331!'
```

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 "D" shifts the first letter up by 3).
2. All remaining letters are shifted using the previous letter of the message (plaintext) as a key. If the previous character is punctuation, use the letter before it.

```
>>> test(); encrypt("THIS IS PROBLEM 2 OF ASSIGNMENT 1.", "X")
"QAQA AA GFEPMPQ 2 ZT FSKAPTYQRF 1."
>>> test(); decrypt("QAQA AA GFEPMPQ 2 ZT FSKAPTYQRF 1.", "X")
"THIS IS PROBLEM 2 OF ASSIGNMENT 1."
```

Problem 3 (2 Marks):

In this problem, you will *crack* a Caesar cipher. Note that you will be working with the standard Caesar cipher from Problem 1, rather than the modified version from Problem 2. Your task is to write code that automatically identifies the key used to encrypt a message and recovers the original plaintext.

The Caesar Cipher has a limited number of possible keys: one for each letter of the alphabet. For this problem, you may assume that all ciphertexts use the same 25-letter alphabet. As a result, there are only 25 possible decryptions for any given ciphertext. Your program should try all possible keys and determine which decryption produces valid English text. To distinguish meaningful English from nonsense, you will construct a collection of valid English words and use it to evaluate candidate plaintexts.

Modify the provided skeleton file, "a1p3.py", to implement two functions: "form_dictionary" and "crack_caesar".

The function "form_dictionary" takes one argument, `text_address`, which specifies the location of a file containing English text, and returns a Python `set` containing as many valid English words as possible. By default, the file used will be `carroll-alice.txt`, containing the text of *Alice in Wonderland*.

When generating the dictionary, you should regularize all words to uppercase, remove all non-letter characters, and use spaces (as detected by Python's built-in `split()` method) to determine word boundaries. This process may result in some invalid words. However, you will find that it is sufficient to crack many Caesar ciphers.

The function "crack_caesar" takes two arguments:

- `ciphertext`: a string encrypted using the Caesar cipher
- `words`: a Python set of valid English words, as generated by **form_dictionary**

The function returns two values: the recovered plaintext, and the one-letter key that was used to encrypt it.

You may assume that test cases are constructed so that one clear correct answer exists. However, if your implementation finds multiple possible decryptions, you should return the one whose plaintext comes first in alphabetical order. The following example demonstrates correct behavior:

```
python3 -i a1p3.py
>>> test(); crack_caesar("TBHZLIB QL TLKABOHXKA", form_dictionary())
("WELCOME TO WONDERLAND", "X")
```

Problem 4 (4 Marks)

For this problem, consider the encryption strength of the original caesar cipher and each of your modified ciphers. Assume, when answering these questions, that the 25-letter alphabet is being employed. Please record your answers in the provided file "a1.txt" for the following questions.

4a: How many distinct keys (keys which each produce different ciphertexts for the same message) do each of the ciphers have?

4b: Assume that an attacker knows which cipher you are using, but is unaware of your key(s). In this case, is your Problem 2 cipher stronger than the original Caesar cipher?

4c: The Caesar cipher has several weaknesses. Will encrypting the space help address any of these weaknesses? Specify which weaknesses in your explanation.

Note: Please strictly follow the template provided in a1.txt. Any change in the template might result in a potential mark deduction.