**CMPUT 331, Winter 2025, Assignment 1**

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

## 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") to improve 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** or **README.pdf**)

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 "AaBbCcDd...", where each letter corresponds to its shift amount 0, 1, 2, 3, 4, 5, ..., 51

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 "AaBbCcDd...", your implementation should seamlessly accommodate others, such as "ZzYyXxWw...". 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: one of the 52 uppercase/lowercase letters corresponding to the shift amounts, for instance, A = 0, a = 1, B = 2, b = 3, C = 4, c = 5... 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:

```
python3 -i  a1p1.py
>>> test(); encrypt("AaBbCcDd...", "x")
'xYyZzAaB...'
>>> test(); decrypt("xYyZzAaB...", "x")
'AaBbCcDd...'

>>> test(); encrypt("Welcome to 2025 Winter CMPUT 331!", "x")
'tCJAMKC RM 2025 tGLRCP zjmrq 331!'
>>> test(); decrypt("tCJAMKC RM 2025 tGLRCP zjmrq 331!", "x")
'Welcome to 2025 Winter CMPUT 331!'

>>> test(); encrypt("This is Problem 1 of Assignment 1.", "y")
'rGHR HR nQNAKDL 1 NE yRRHFMLDMS 1.'
>>> test(); decrypt("rGHR HR nQNAKDL 1 NE yRRHFMLDMS 1.", "y")
'This is Problem 1 of Assignment 1.'
```

**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 (plaintext) as a key. If the previous character is punctuation, use the letter before it.

```
python3 -i  a1p2.py
>>> test(); encrypt("AaBbCcDd...", "y")
'yabcdefg...'
>>> test(); decrypt("yabcdefg...", "y")
'AaBbCcDd...'
```

```
>>> test(); encrypt("Welcome to 2025 Winter CMPUT 331!", "y")
'uaQORBR YI 2025 keWHYW tOBJN 331!'
>>> test(); decrypt("uaQORBR YI 2025 keWHYW tOBJN 331!", "y")
'Welcome to 2025 Winter CMPUT 331!'

>>> test(); encrypt("This is Problem 2 of Assignment 1.", "x")
'qaQB BB hgGQNQR 2 BU fsLBPUARSH 1.'
>>> test(); decrypt("qaQB BB hgGQNQR 2 BU fsLBPUARSH 1.", "x")
'This is Problem 2 of Assignment 1.'
```

**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.

Note that non-letter characters (such as spaces, punctuation, and numbers) will not be encrypted. When encountering these characters, the encryption process will simply skip them and resume with the next letter. Here is an example:

```
encrypt("Welcome to 2025 Winter CMPUT 331!", "XxYyZz")
```

It should work as follows:

```
XxYyZzX xY      yZzXxY yZzXx
Welcome to 2025 Winter CMPUT 331!
TCjBnMb Rm 2025 uhNqCp aLoRq 331!
```

In this example, the letters above the plaintext represent the key used to encrypt the plaintext into the ciphertext. Spaces indicate that no key is applied. Specifically, "X" is the key used to encrypt the first letter "W". Numbers, such as "2025", and punctuation marks, such as "!", are not encrypted, so no key is applied to them.

```
python3 -i a1p3.py
>>> test(); encrypt("AaBbCcDd...", "XxYyZz")
'XYZABCAB...'
```

```
>>> test(); decrypt("XYZABCAB...", "XxYyZz")
‘AaBbCcDd...’

>>> test(); encrypt("Welcome to 2025 Winter CMPUT 331!", "XxYyZz")
‘TCjBnMb Rm 2025 uhNqCp aLoRq 331!’
>>> test(); decrypt("TCjBnMb Rm 2025 uhNqCp aLoRq 331!", "XxYyZz")
’Welcome to 2025 Winter CMPUT 331!’

>>> test(); encrypt("This is Problem 3 of Assignment 1.", "ZYXzyx")
‘SffS HQ OplBKCl 3 mc zRQhekMDLs 1.’
>>> test(); decrypt("SffS HQ OplBKCl 3 mc zRQhekMDLs 1.", "ZYXzyx")
‘This is Problem 3 of Assignment 1.’
```

**Problem 4** (4 Marks)
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. 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 four ciphers have?

4b: Is the problem 2 cipher stronger than the original caesar cipher?

4c: Will encrypting the space help address the weaknesses of the Caesar cipher?

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