Module One:

Creating & Attacking Caesar Ciphers

# Grading

Total points: 45

**[5 points]** Your submission is labeled as “cyb404\_module01\_lab00\_[nau\_id]\_[lastname]\_[firstname].zip.” For example, if I were to submit a file, it would be labeled as cyb404\_module01\_lab00\_mv668\_vigil-hayes\_morgan.zip. **FAILURE TO COMPLY WITH THIS STEP CAN LEAD TO A ZERO GRADE.**

**[5 points]** Your submission files are correctly formatted.

* [1 points] The Caesar cipher code is all contained in a file caesar\_cipher.py
* [1.5 points] There is a comment for every functional unit of code that details the purpose of the cipher code.
* [1 points] The Caesar cipher brute force attack code is all contained in a file caesar\_cipher\_cracker.py
* [1.5 points] There is a comment for every functional unit of code that details the purpose of the attack code.

**[15 points]** Your Caesar cipher correctly encrypts and decrypts plaintext alphabet messages given positive or negative integers

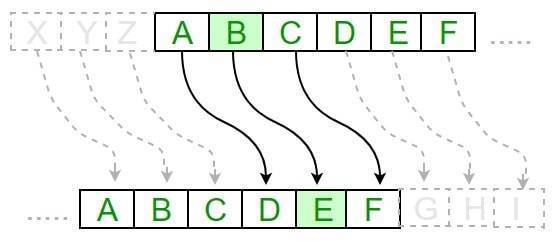
* [5 points] Correct response to Q00
* [5 points] Correct response to Q01
* [5 points] Correct response to Q02

**[10 points]** Your Caesar cipher cracker correctly attacks and identifies the correct plaintext message given just an encrypted message as demonstrated by a correct response to Q03.

# Part 1: Implementing a Caesar cipher

The Caesar Cipher is an encryption algorithm that takes in a ***key*** (integer) and **text** (string). It encrypts the **text**by moving every letter of the **text** “forward” in the alphabet a total of ***key*** places. This ***key***acts as the password that will be required to decrypt the encrypted **text**.

Today, we will use this algorithm for **text** containing only the following: uppercase alphabet (A-Z), lowercase alphabet (a-z) and the empty space character. For example, here is an implementation of Caesar Cipher using ***key***= 3.

Implementation of Caesar Cipher with ***key***= 3

In the example, A → D, B → E, C → F etc. The letters move three places “forward.” Notice how this “wraps around” when you get to X? X → A, Y → B and Z → C. You can think of the alphabet as a clock that starts at A and goes clockwise up to Z and then back to A again. *What do you think happens if****key****= -3? Hint: look at the arrows.*

**In this lab, you will be implementing a Caesar cipher based on the starter code I provide in caesar\_cipher\_starter.py**

The places where you need to focus your implementation efforts are highlighted with the #TODO comment.

You will use ASCII codes to manipulate the characters in our text. Let’s link this to the Caesar Cipher algorithm. When ***key***= 3, A (065) → D (068) and X (088) → A (065). Notice how we don’t always just add ***key***to the ASCII code? This is because of the “wrap around” rule. X (088) has to go back to A, which is 065 not 091. Remember, A-Z is 065-090 and a-z is 097-122 in ASCII. We will need to take this into account when we write our code.

Python has two special functions that let us manipulate characters and their ASCII codes. They are **chr** and **ord**. We will use them in our code. Here is how they work:

|  |  |
| --- | --- |
| 1  2  3  4 | >>> ord("a")  # What is the ASCII code representation of alphabet letter "a"?  97  >>> chr(97)   # What character does the ASCII code 97 represent?  "a" |

Note, when using ASCII, your code should be able to treat uppercase and lowercase letters differently. Your code should be able to correctly encode alphanumeric inputs and preserve spacing. It should be able to handle both negative and positive integers as key inputs.

Your code should run with the following:

python3 caesar\_cipher\_starter.py -f <decrypt | encrypt> -m <message> -k <key>

**Q00.** To test the correctness of your Caesar cipher, provide the encrypted version of the message “Hello World” using a key of 34.

**Q01.** To test the correctness of your Caesar cipher, provide the encrypted version of the message “Network cybersecurity” using a key of -5.

**Q02.** To test the correctness of your Caesar cipher, provide the decrypted version of the message “O gs voiqrk Xoiq” assuming a key of 32.

# Part 2: Cracking the Caesar cipher

As you can probably already tell (and based on your reading), the Caesar cipher is relatively easy to hack.

**Write a script called caesar\_cipher\_cracker.py that takes an encrypted message as input and attempts to identify the plaintext message without a key being provided.** *Hint: There are only 25 possible unique messages based on your message clock.*

**Q03.** Demonstrate your attack script’s ability to attack the Caesar cipher by decrypting the following message:

Yrzc Trvjri

Plaintext: