Table of Contents

tl;dr

Specification

Walkthrough

<u>Usage</u>

Testing

Correctness

<u>Style</u>

Staff's Solution

How to Submit

<u>Hints</u>

Caesar

tl;dr

Implement a program that encrypts messages using Caesar's cipher, per the below.

\$ python caesar.py 13

plaintext: HELLO
ciphertext: URYYB

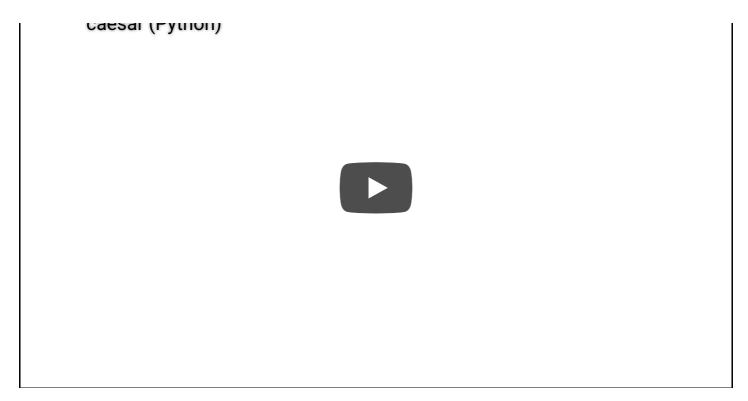
Specification

Design and implement a program, caesar, that encrypts messages using Caesar's cipher, exactly as you did in Problem Set 2 (https://lab.cs50.io/cs50/labs/2019/x/caesar/), except that your program this time should be written (a) in Python and (b) in CS50 IDE.

Implement your program in a file called caesar.py in your
 ~/workspace/pset6/caesar directory (if it doesn't already exist, create it now!).

- Your program must accept a single command-line argument, a non-negative integer. Let's call it *k* for the sake of discussion.
- If your program is executed without any command-line arguments or with more than one command-line argument, your program should print an error message of your choice (with print) and exit (https://docs.python.org/3/library/sys.html#sys.exit) immediately with a status code of 1.
- You can assume that, if a user does provide a command-line argument, it will be a non-negative integer (e.g., 1). No need to check that it's indeed numeric.
- Do not assume that k will be less than or equal to 26. Your program should work for all non-negative integral values of k less than 2^{31} 26. In other words, you don't need to worry if your program eventually breaks if the user chooses a value for k that's too big or almost too big to fit in an int. (Recall that an int can overflow.) But, even if k is greater than 26, alphabetical characters in your program's input should remain alphabetical characters in your program's output. For instance, if k is 27, k should not become k even though k is 27 positions away from k in ASCII, per asciichart.com (http://www.asciichart.com/); k should become k since k is 27 positions away from k novided you wrap around from k to k.
- Your program must output plaintext: (without a newline) and then prompt the user for a string of plaintext (using get string).
- Your program must output ciphertext: (without a newline) followed by the plaintext's corresponding ciphertext, with each alphabetical character in the plaintext "rotated" by *k* positions; non-alphabetical characters should be outputted unchanged.
- Your program must preserve case: capitalized letters, though rotated, must remain capitalized letters; lowercase letters, though rotated, must remain lowercase letters.
- After outputting ciphertext, you should print a newline.

Walkthrough



Usage

Your program should behave per the examples below. Assume that the underlined text is what some user has typed.

```
$ python caesar.py 1
plaintext: HELLO
ciphertext: IFMMP

$ python caesar.py 13
plaintext: hello, world
ciphertext: uryyb, jbeyq

$ python caesar.py 13
plaintext: be sure to drink your Ovaltine
ciphertext: or fher gb qevax lbhe Binygvar

$ python caesar.py
Usage: python caesar.py k
```

\$ python caesar.py 1 2 3 4 5
Usage: python caesar.py k

Testing

Correctness

check50 cs50/problems/2019/x/sentimental/caesar

Style

style50 caesar.py

Staff's Solution

~cs50/2019/x/pset6/caesar

How to Submit

Execute the below, logging in with your GitHub username and password when prompted. For security, you'll see asterisks (*) instead of the actual characters in your password.

submit50 cs50/problems/2019/x/sentimental/caesar

You can then go to https://cs50.me/cs50x (https://cs50.me/cs50x) to view your current scores!

Hints

Recall that <code>argv</code> is a list of strings representing the command line arguments. Recall that we can use <code>len(argv)</code> in order to figure out how many strings exist in that list; this is the equivalent idea to <code>argc</code>, from C.

And so you can access k with code like

```
k = argv[1]
```

assuming it's actually there! And assuming you've imported [argv], as by:

```
from sys import argv
```

Once you have both k and some plaintext, p, it's time to encrypt the latter with the former. Recall that you can iterate over the characters in a string, printing each one at a time, with code like the below:

```
for c in p:
    print(c, end="")
```

That end="" line just overrides Python's default behavior when printing which, unlike C, tacks on a newline by default!

You may also wish to have a look at Python's ord() and chr() functions!