Lab 7: Caesar Cipher

Labs are graded for participation rather than correctness. Keep all your lab code in your course GitHub repo to receive credit for your work. We'll be looking to see that you have at least partially completed all problems in each lab.

Professional Skills: Collaborative programming

Programming is not always a solo activity. For example, pair programming is a software development technique in which two people work on a coding task together. Why is pair programming used? Research has shown that, although pair programming takes a little longer, it typically results in higher quality code.

For this lab, you may choose to work with other students to develop your collaborative coding skills and earn professional skills points in addition to lab credit. We're calling it collaborative coding rather than pair programming because you may work in groups with more than two members. You only need to engage in collaborative programming once to get professional skills credit. There will be more opportunities in future labs, so don't worry if you're not ready to do this now.

To participate tonight:

- Make sure you have the latest version of the Zoom desktop application.
    - If you are in China, follow the instructions for Zoom under the "Currently Accessible" heading here: https://service.northeastern.edu/tech?id=kb_article&sys_id=c9758e2e1b4b101035c3628a2d4bcb1c
- Log into Zoom using your Northeastern credentials. This step may not be essential. If you are in China, consult the link above for help.
- Rejoin the lab meeting and self-select a breakout room. Ideally, there should be 2-4 students in a room.
- Alternatively, you can arrange to get together with other students outside of the official lab Zoom. You can always pop back in to get help from the course staff.

How to code collaboratively:

- As a group, discuss how you might approach the lab assignment (see below). What programming concepts will you need to use?
- One group member should be responsible for actually typing the group's code. That person should share their screen. That typer should take directions from other group members and share their own ideas—remember, this is a team effort.
- At the end of the lab, push whatever you have completed to one team member's GitHub repo. Include the names of all group members in the file comment so everyone can get

credit. Share the lab solution with all group members—this should be possible in Zoom chat or Canvas messaging. Each group member can then decide if they want to push the code to their own repos.

- You may continue to work on the lab individually outside of lab time if you like but it's not required.

The Assignment

The process of encryption encodes messages to prevent unauthorized (or nefarious) access to our data. Computer scientists employ encryption algorithms all the time; they are used to protect everything from credit card numbers to medical information to email messages.

The concept of encryption has been around for thousands of years. Among the earliest known encryption methods is the Caesar cipher. The Caesar cipher applies a *shift* to each letter in the original message, substituting it with another letter further down the alphabet. If the shift hits the end of the alphabet, we loop back around to the letter *a*. For example, if the shift is 3 and the message is pizza, then we shift as follows:

p + 3 = s
i + 3 = l
z + 3 = c
z + 3 = c
a + 3 = d

Encrypted message: **slccd**

For this lab, we'll shift only lowercase letters. If the message to be encrypted/decrypted contains a capital letter, convert it to lowercase. You may use Python's string methods to convert case. For all non-letter characters that appear in the string (digits, spaces, and punctuation), do not apply any shift.

A valid shift must be between 0-25, inclusive. If there's an attempt to shift more or less than that, don't report an error but use the modulo operator to bring the shift amount within range.

Part One: Encrypt
Write a function to encrypt a given string. Your function takes two inputs:

- The original message, a string.
- The shift amount, an int.

And it returns a string, the encrypted message.

Write helper functions to modularize your code if appropriate.

Helpful hints:

- Python converts a character to its ascii value with the function **ord**. Try it out in interactive mode with the statement: **ord('a')**
- Python converts an ascii value to its corresponding letter with the function **chr**. Try it out in the VS Code Terminal with the statement: **chr(97)**
- The lowest valid ascii value for a lowercase letter is 97 ('a'). The highest valid ascii value for a lowercase letter is 122 ('z').

Part Two: Decrypt

Now write the corresponding decryption function. Similar to encryption, decryption shifts the input string backwards in the alphabet instead of forwards.

This function takes two inputs:

- The encrypted message, a string.
- The shift amount, an int.

And returns a string, the decrypted message.

(Or, if you'd prefer, modify your encryption function so it can do both encrypt and decrypt, depending on what the caller asks for. As long as your functions are concise and clear, we're good.)

Your `main()` function should:

- Prompt the user for a string and a shift amount
- Encrypt the string and print the encrypted value
- Decrypt the string and give the user back their original input

Example of running the program

```
Enter a message to encript:
Beware the Ides of March
Enter a shift amount:
10
Your super secret message is: logkbo dro snoc yp wkbmr
...decrypting encoded message now...
It was: beware the ides of march
```

Part Three: Decrypt an Unknown Message

What if we are given an encrypted message and we don't know the shift? We want to figure out what the original message was.

Many encryption algorithms are considered strong because the only way they can be deciphered is with *brute force*: The adversary must try every possible way of decrypting until

they stumble upon the original message, and there are so many possibilities that they can't finish within their lifetime, or yours.

Your goal for the rest of the lab is to decrypt the strings below using a brute-force method. Run your decryption function with every possible shift value (1-5 for this exercise, instead of 0-25) until it becomes a word or phrase that you, the human, understand. BUT WAIT THERE'S MORE....

The strings below have had **two** encryption functions applied: (1) shift and (2) slide. We already know what *shift* does. A *slide* of length *n* moves each character forward by *n* positions within the string, wrapping around when needed to the beginning of the string.

For example, if my message is: **bcd**
Then if I apply a slide of 1, I get: **dbc**
Or a slide of 2, and I get: **cdb**

If I apply both a shift and a slide, then the message gets harder to decipher. For example, if my shift is 1 and my slide is 1, then:
eat my shorts
becomes
tfbu nz tipsu

The 5 strings below have each had unknown shifts and slides applied to an original, sensible English message. All of the shifts and slides applied are between 1 and 5, inclusive. What are the original messages?

> ***.glygo rsvvmw'w gshi avmxiw gsqqirxw efsyx *lmq**
>
> **aqwuvwem, kp cp kphkpkvg nqqr, ykvj**
>
> **.sjajw ywzxy f htruzyjw dtz hfs'y ymwtb tzy f bnsitb..**
>
> **yko.vjg swguvkqp qh yjgvjgt eqorwvgtu ecp vjkpm ku nkmg vjg swguvkqp qh yjgvjgt uwdoctkpgu ecp u**
>
> **.ejwem pqttku ytkvgu dqqngcp gzrtguukqpu vjcv ctg dqvj vtwg cpf hcnug**

Part 4: Encrypt More, For Fun
If you're done with the other parts and it's still early, try encrypting your own strings with a shift + slide, and give the encoded messages to another pair to decrypt.