

# Project 3: Caesar Substitution Cipher

**Collaboration Policy:** This assignment must be completed individually. You may discuss high-level ideas with other students, but you should not share code.

## Description

The goal of this homework assignment is to create a tool for encrypting and decrypting text. First, the user picks whether to encrypt or decrypt. The user then enters a key that will be used to encrypt (or decrypt) the text. The user then enters a phrase, which is then encrypted or decrypted, and the result is displayed to the user.

You will need to read the first three parts of this spec very carefully (and probably more than once) to understand exactly what you're building. Do not start writing code until you are sure you understand everything in this spec. Make sure to complete the "check your understanding" exercises to make sure that you've understood the fundamental ideas.

## Encrypting and Decrypting Text Using a Ciphertext Alphabet

For the purposes of this assignment, we will call unencrypted text "plaintext", and we will call encrypted text "ciphertext".

The encryption scheme for this assignment is a simple substitution cipher. The scheme works by replacing each character in the plaintext with the corresponding character in a "ciphertext alphabet", which is always the same length as the plaintext alphabet.

For example:

```
plaintext alphabet:  ABCDEFGHIJKLMNOPQRSTUVWXYZ
ciphertext alphabet: KLMNOPQRSTUVWXYZABCDEFGHIJ
```

Given the ciphertext alphabet, if we encrypted the string "writing code is cool" it would become "gbsdsxq myno sc myyv". This is because we replace "w" with "g", "r" with "b" and so forth. Decryption works exactly the same way, but in reverse. Thus, "myyv" using the above alphabets becomes "cool".

## Check Your Understanding

What would the output from our encryption algorithm be if we encrypted the plain text "hello"? To check your answer, highlight the following invisible text:

## Generating a Ciphertext Alphabet from a Key

In your program, rather than having the user specify a ciphertext alphabet directly, you will instead have the user enter a secret key that will be used to generate the ciphertext alphabet. We will start by setting our ciphertext alphabet equal to the plaintext alphabet. Then, we will move each character to the right by the number of letters specified by the key. When we get to the end, the letters wrap around.

So for example, if our key is 3, we will start with:

ciphertext alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Then move each letter to the left 3 positions:

ciphertext alphabet: XYZABCDEFGHIJKLMNOPQRSTUVW

### Check Your Understanding

If the encryption key is 13, what is the corresponding ciphertext alphabet? To check your answer, highlight the following invisible text:

## Overview of Program Behavior

At the beginning of the program:

The computer will encrypt if the user's input starts with an "e"

The computer will decrypt if the user's input starts with a "d"

To handle text entered by the user that needs to be encrypted or decrypted, you should use the following rules for each character:

If it is a letter, replace as described above with a letter of the same case (uppercase or lowercase).

If it is a space, keep the space.

If it is anything else, skip it. It should not appear in the output.

So for example, if we had the alphabets below:

plaintext alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ

ciphertext alphabet: KLMNOPQRSTUVWXYZABCDEFGHIJ

And they entered "Writing code is,,,,,,5555 Cool", we'd get "Gbsdsxq myno sc Myyv" as output.

## Example Output

Here is an example of a user running the program:

**Computer:** Do you want to Encrypt or Decrypt?

**User:** Horse

**Computer:** Invalid input. Your input must start with either E for encrypt or D for decrypt.

**Computer:** Do you want to encrypt or decrypt?

**User:** encrriipppppqttt

**Computer:** Enter an encryption key. Key must be numeric.

**User:** pancake 1000

**Computer:** Error, key must be numeric.

**Computer:** Enter an encryption key. Key must be numeric.

**User:** 29

**Computer:** What text would you like to encrypt?

**User:** Programming is my true destiny.

**Computer:** Modoxjjfkd fp jv qorb abpqfkv

[program ends]

[user restarts program]

**Computer:** Do you want to Encrypt or Decrypt?

**User:** d

**Computer:** Enter an encryption key. Key must contain only letters.

**User:** 3

**Computer:** What text would you like to decrypt?

**User:** Modoxjjfkd fp jv qorb abpqfkv!!!

**Computer:** Programming is my true destiny

## Check Your Understanding

Read the transcript carefully and make sure you agree with all of the things the computer says.

## Starting Materials

Load this [starter file](#) which includes a few blank, but well-documented blocks that will hopefully help facilitate and start the process of understanding how to break down this problem into smaller pieces.

Once you add the code for each of the starter blocks, you'll still need to write the code to pull them all together so that you can run the program and get output as illustrated by the example above.

## Bonus

Modify the substitution encryption scheme above so that it is able to handle any unicode character between 32 and 126, not just alphabetical characters. With this updated scheme, you should not ignore a unicode character unless it is outside the valid range (32 through 126). (Note that *Snap!* includes blocks to convert between a character and its numeric unicode value.)

## Submission Guidelines

The following must be placed in the D2L dropbox. Add a comment to the dropbox if you completed the extra credit.

### Project file



This is the file with all your *Snap!* code. Name this file as Proj3 with your first and last name, such as: Proj3\_MaryMosman.xml

### Readme file

The readme file is a separate file that can be shared along side the code so to give context and extra information about the project. Write a file that has 2 sections:

- The first section - Summary, should describe what the program is and does. In this case, it should describe the game (in your own words, not mine), and how to play it.
- The second section - Learning, should describe what you found challenging (or not) about the assignment and how you worked through the challenges and solved the problem. Perhaps you tried something and it didn't work, then you tried a different approach. Perhaps you didn't know where to start and talked it through with a friend. Perhaps the problem was easy because you've done something similar before. The only "wrong" answer here is not talk about how you solved the problem.

## Tips

- Remember the saving and testing tips from Project 1! Save often and back it up in multiple locations.
- Start by putting thoughts down on paper. Break the problem into smaller parts before you start writing the code.
- Code a little bit at a time (block by block) and test as you go.
- In Snap!, the equals predicate returns true for any same letter, regardless of whether or not they are upper or lower case. Example:  is true! However the unicode of “A” is 65 whereas the unicode of “a” is 97.
- Don’t forget to make sure you’re using lists and text as appropriate. Use blocks like  to convert between the two types.
- Remember that lists are mutable! **Avoid changing the original alphabet variable.** Make copies of that list if need be by setting different variables to “original alphabet”.