Lab 3: Intro to Objects

INFSCI 0201
Intermediate Programming (Spring 2025)

This week's lab is an attempt to pull together some concepts we've explored with Strings and solve a problem in an object-oriented way. You will have to write a class for a CaesarCipher object, which will perform two unique operations (encrypt and decrypt) that make up a modified Caesar cipher.

Submission Details

You are to write a Python program that meets the requirements outlined in the Lab 5 Tasks section.

- As with all programming assignments in this course, you must submit Lab 5 through GitHub in the git repository the instructor created for you. You must place your Lab 5 eclipse project in the folder **/labs/lab03/**
- If you did this all correctly, this project will be in the file **/labs/lab03/caesar.py**

Testing

There will be four main tests for this lab, which cover the following operations:

1. Your encrypt method shifts correctly
2. Your encrypt method changes the case correctly
3. Your encrypt method handles non-letter input correctly
4. Your decrypt method is correct in all cases
5. **Your class follows proper naming conventions and object design requirements**.

Note: This is the first lab where style guide violations will count as test failures! Be sure you're strictly adhering to the Python coding style guide on Canvas.

Caesar Ciphers and our Modified Caesar Cipher

A Caesar Cipher (or “shift cipher”) is one of the earliest, most well-known encryption methods commonly learned by undergraduates studying mathematics and technology. One needs a (plaintext) message and a (right-shift) key to encrypt. The key indicates the number of letters we must ‘shift’ the characters of the message to reach a ciphertext.

For example, let’s say we had a message “ABC” and a key of 2. We would iterate through each letter of the message and shift each letter to the right by 2. So, ‘A’ would become ‘C’, ‘B’ would become ‘D’, and ‘C’ would become ‘E’, so we would end with a ciphertext of “CDE”. This should loop around (like a modulo), so for example, if one wanted to shift the character ‘Z’ by 2, it would become ‘B’.

To decrypt, we simply need to do the same thing in the opposite direction (i.e. shift to the left instead of the right). So, to decrypt “CDE” with a key of 2, we see that two characters to the left of ‘C’ becomes ‘A’, ‘D’ becomes ‘B’, and ‘E’ becomes ‘C’, resulting once again in the plaintext “ABC”. Decryption also must worry about ‘looping around’ with a modulo operation (i.e. shifting ‘B’ to the left by 2 results in ‘Z’).

Caesar Ciphers are not always consistent with how they handle special characters, white-space characters, and the issue of capitalization. So, here’s how we will handle that:

- In our modified Caesar Cipher, white space will remain unchanged between plaintext and ciphertext.
- For regular English characters, the shift will loop around if the shifted value is beyond 26.
    - For instance, “y” with a shift value of 5 will result in “d”.
- Special characters (like @#$%^&*(), etc.) will shift their ASCII values between the plaintext and ciphertext using the same logic as regular English characters.
    - Python has built-in functions ord() and chr() to convert a character between its integer representation of its Unicode value (can proxy as ASCII values) and its string representation based on the Unicode value.
    - https://docs.python.org/3/library/functions.html#ord
    - https://docs.python.org/3/library/functions.html#chr
- All characters will become lowercase before they are shifted, and stay as lowercase characters
    - https://docs.python.org/3/library/stdtypes.html#str.lower

Lab 3 Tasks

- You will implement a “Caesar” class.
- When instantiated as a “Caesar” object, it can perform encryption and decryption in our modified Caesar Cipher operation.
- This Caesar object should have exactly one attribute: an integer representing the key to the cipher. **This field should be considered private, and you should have a corresponding getter and setter** (You can use either traditional getter and setter, or use Python property)
- The **key** attribute should be considered as an instance attribute, not a class attribute
- You must implement an ‘encrypt’ method, which performs encryption based on the ***modified*** Caesar Cipher described above with the following signature:
    - **def encrypt(plaintext):**
        - plaintext is the message to be encrypted
        - The method should return the ciphertext, encrypted using this object’s key
- You must implement a ‘decrypt’ method, which performs decryption based on the ***modified*** Caesar Cipher described above.
    - **def decrypt(ciphertext):**
        - ciphertext is the encrypted message to be decrypted
        - The method should return the plaintext message, decrypted with this object’s key.
- You are encouraged to write a test script to validate the encrypt and decrypt methods, but the AutoTester will not evaluate your test script.
- Example main method with output

```
cipher = Caesar()
cipher.set_key(3)
# cipher.key = 3 # if you are using Python property
print(cipher.encrypt(“hello WORLD!”)); # prints “khoor zruog$”
print(cipher.decrypt(“KHOOR zruog$”)); #prints “hello world!”

cipher.set_key(6);
print(cipher.encrypt(“zzz”)); //prints “fff”
print(cipher.decrypt(“FFF”)); //prints “zzz”

cipher.set_key(-6); // Negative keys should be supported!
print(cipher.encrypt(“FFF”)); //prints “zzz”
```