Assessment 9: Caesar Cipher

CSCI 128

Introduction

In this assessment, you will implement (1) a Caesar cipher encoder and (2) a brute force Caesar cipher breaker.

Problem

A Caesar cipher is a simple substitution cipher in which each upper or lowercase letter is replaced by the letter x characters further down the alphabet, where x is a chosen shift in the range 1-25. For example, given the string 'Quizzes', a shift of 1 would result in 'Rvjaaft', where each letter is shifted to be one later position in the alphabet, wrapping around to the beginning of the alphabet if necessary.

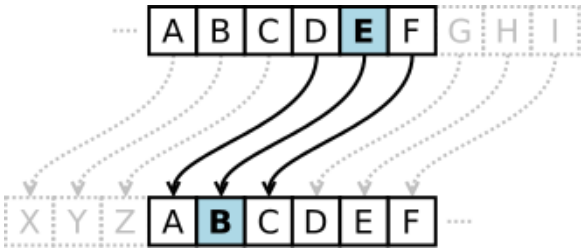

Figure 1: Source: wikipedia (cc license).

The Caesar cipher is a very insecure cipher, as there are only 25 possible shifts. One could easily 'brute force' the solution by trying all of the possible shifts and seeing which one produces sensible text (which is what you will do in this assessment).

For this assignment you will create and implement two primary functions: an encryption function and a decryption function. You will also create a main program that acts as a menu so that the user can select if they are encrypting, decrypting, or testing.

The encryption function must be named `encrypt` and take two arguments in this order: the message to encrypt (a string) and the shift to encrypt it by (an integer). It must return one string, the encrypted text.

A couple of notes for encryption:

- Only letters should be shifted; do not shift (but still retain) symbols, numbers, spaces, or anything except letters.
- Capitalization should be retained.
- Your program should be able to handle both positive and negative shifts.

- Your program should be able handle shifts larger than 25 and smaller than -25.

We encourage you to test your solution to encryption before moving on to decryption.

For the decryption part of this assessment, implement a brute force Caesar cipher breaker. A brute force method is one that checks **every possible option** and chooses the decrypted text that contains the supplied keyword. In other words, try all 26 possible shifts (0 - 25) and select the one that causes the keyword to appear in the decrypted text.

The decryption function must be named `decrypt` and must take 2 parameters in the following order: the text to decrypt (a string) and the keyword (also a string). It must return the decrypted text (a string) and the shift used (an integer). The return value must be a list with two values just mentioned in order.

A couple of notes for decryption:

- Capitalization should be retained.
- There might be punctuation in the encoded message. However, keys will be provided without that punctuation. For example, you should be able to decrypt, "Mjqqt Btwqi", using keyword "World" and find "Hello World!".
- You are NOT given the shift. Instead, try every possible shift until you detect the keyword in your decrypted text.
- When determining the shift, output the smallest possible **positive** shift that could have been used to encode the original message (e.g., 2 instead of 28).
- Remember that you should provide the shift of the encrypted message relative to the original text. For instance, if your encrypted message is 'Rvjaaft', and 'Quizzes' is the original word, a shift of 1 would be the output (rather than -1 or 25).
- As we are trying to crack a code, our key may not be 100% correct. Your program must handle the case when the key can't be found in any of the possible shifts. In that case, your function should return the single string: "ERROR".

Main Program

Once your two functions are implemented and working, you will also need to create a main program and provide a simple menu for the user to select an option, provide input, and see the resulting output.

If the user chooses to encrypt the message (option 1) take as inputs the message to encrypt and the shift. Then the program should output the encrypted message. The input 'Quizzes' with a shift of 1 would output 'Rvjaaft'.

If the user chooses option 2, your program should take as inputs a message that has been encrypted using a Caesar cipher and one keyword that may exist in the original message.

Then the program should output the original message and the shift used to encode it on separate lines.

In cases where the user selects option 3, please read the next section.

Testing

It is common in software engineering to practice what is known as test driven development. Thus far in CSCI 128, we have written tests for you in the Autograder. With this assessment, you will start writing your own tests for your code.

The core of any test is an assertion that a statement is true. In Python, this can be accomplished with the `assert` keyword.
For instance:

```
# assert that 2 + 2 is 4
assert 2 + 2 == 4
# assert that Hello is in "Hello World"
assert "Hello" in "Hello World"
```

`assert` is designed to raise an error if the assertion is not true. This may sound extreme, but it is appropriate since we just showed our code is not working as intended.
For instance:

```
# assert that 2 + 3 is 4
assert 2 + 3 == 4  # AssertionError
```

For this assignment, we will be writing two test functions, one for each of your encrypt function and decrypt functions.

To test your encrypt function, create a function called `test_encrypt`, that takes, in order: 1) a word to encrypt, 2) the shift to use, and 3) the expected output.
For example:

```
def test_encrypt(word, shift, expected):
```

This function should call your encrypt function and assert that its output matches the expected value. This function should not return or print any values. The assertion is enough.

To test your decrypt function, create a function called `test_decrypt` that takes 1) a word to decrypt, 2) the expected word, and 3) the expected shift.

For example:

```
def test_decrypt(word, keyword, expected_word, expected_shift):
```

This function should call your decrypt function and assert that its output matches both the expected word and the expected shift.

If the user selects option 3, your program should run your two test functions, `test_encrypt` and `test_decrypt`. You should have at least 3 test cases for the `test_encrypt` function and at least 3 test cases for the `test_decrypt` function.

Additionally, there is *no* output needed for this option; if the tests fail they will automatically raise an `AssertionError` and will do nothing on a success.

So for instance, this program's `if __name__ == '__main__':` might look something like:

```
if __name__ == '__main__':
    ...
    # code to get operation
    if ... == 1:
        # code to perform operation 1
    elif ... == 2:
        # code to perform operation 2
    elif ... == 3:
        test_encrypt(quizzes, 1, rvjaaft)
        test_decrypt(rvjaaft, quizzes, quizzes, 1)
        # remaining tests...
```

Input

The first line of input will be either 1, 2, or 3. 1 = encrypt and 2 = decrypt and 3 = test.

- If a 1 is given, then a message and a shift will be provided on the next two lines.
- If a 2 is given, then an encrypted message and the keyword will be provided on the next two lines instead.
- If a 3 is given, then all of the tests written will run.

Output

- If a 1 is given, output the encrypted message using the given shift.
- If a 2 is given, output the decrypted message on the first line and the shift in the second. If you fail to decrypt the message, output one single line as follows:
  "OUTPUT ERROR"
- If a 3 is given, then in the case that no tests fail, nothing will be outputted. In the case that a test does fail, an `AssertionError` will be raised by the the `assert` keyword.

Strategy

There are many ways to approach this problem. Your solution does not have to use all the following advice, but you may find it helpful.

- Python has a number of built-in functions that can help here. Consider looking at the string methods: `isalpha()` and `isupper()`/`islower()`.
- It is not an accident that we advised you to create functions for encryption and decryption. You will likely also need to use the encryption function when you decrypt, since it still just shifting letters by a certain number of places.
- There are several ways to convert between letters and numbers. Consider using a list of letters in the alphabet and the `.index()` function, or investigating the Python `ord()` and `chr()` functions.
- The modulo operator (%) will likely be helpful in dealing with the variety of different shift values that may be given as input. If you did not know, modulo also works on negative numbers.

Reflection

Cybersecurity attacks happen daily and are a threat to data at all levels, from personal privacy to national security. Your data or the data of someone you know likely has been compromised at some point. Once data is public, algorithms and automated systems using that data can perpetuate existing biases or create new ones, leading to unfair treatment in areas such as employment, lending, and criminal justice. Write a short paragraph in relation to one of the following questions. Be sure to cite and reference any facts you provide as evidence.

- What is more important, personal data privacy or the right of an organization in which a person participates to provide digital surveillance and oversight?
- How unbiased should algorithms used in the criminal justice system be?
- Is any amount of algorithmic bias acceptable? If so, describe such a situation or context.

Sample Executions

Function-specific call to encrypt - Example 1

```
print(encrypt("My secret message", 10))

# should print:
Wi combod wocckqo
```

Function-specific call to encrypt - Example 2

```
print(encrypt("N0t numb3r5", 7))

# should print:
U0a ubti3y5
```

Function-specific call to encrypt - Example 3

```
print(encrypt("Large negative shift", -82))

# should print:
Hwnca jacwpera odebp
```

Function-specific call to decrypt - Example 1

```
print(decrypt("Ger csy vieh xlmw?", "read"))

# should print:
['Can you read this?', 4]
```

Function-specific call to decrypt - Example 2

```
print(decrypt("Ujqhlgyjshzq ak fwsl!", "is"))

# should print:
['Cryptography is neat!', 18]
```

Function-specific call to decrypt - Example 3

```
print(decrypt("Ujqhlgyjshzq ak fwsl!", "message"))

# should print:
ERROR
```

Example 1

```
OPTION> 1
MESSAGE> Hello World!
SHIFT> 31
OUTPUT Mjqqt Btwqi!
```

Example 2

```
OPTION> 2
MESSAGE> Mjqqt Btwqi!
KEY> World
OUTPUT Hello World!
OUTPUT 5
```

Example 3

```
OPERATION> 2
MESSAGE> Mjqqt Btwqi!
KEY> Mines
OUTPUT ERROR
```

Example 4

```
OPERATION> 3
# All tests passed so no output.
```