Summer 2021

Assignment #2

Assignment Overview

In this assignment you will practice with strings, functions, conditionals and loops (for, while) to perform specialized string manipulation operations, namely **encrypting and decrypting plain-text messages**.

Background

Cryptography is "is the practice and study of techniques for secure communication in the presence of third parties called adversaries." [Wikipedia]

The companion field of **cryptanalysis** focuses on "finding some weakness or insecurity in a cryptographic scheme, thus permitting its subversion or evasion." [Wikipedia]

The main classical cryptography methods are "**transposition** ciphers, which rearrange the order of letters in a message and **substitution** ciphers, which systematically replace letters or groups of letters with other letters or groups of letters. An early substitution cipher was **the Caesar cipher**, in which each letter in the plaintext was replaced by a letter some fixed number of positions further down the alphabet. Suetonius reports that Julius Caesar used it with a shift of three to communicate with his generals." [Wikipedia]

Project Specification

In this assignment you will implement one of the earliest (and simplest) encryption/decryption algorithms: the Caesar cypher, where each letter in the plaintext is replaced by a letter some fixed number of positions further down the alphabet. The number of positions is the 'key' and can be chosen by the user (rather than hardcoded into your program).

Your program ("**baseline solution**") will:

1. Print a brief (2-5 lines) message explaining the purpose of this "app".
2. Prompt for:
   - The input phrase (upper and lower case letters only, no spaces, punctuation symbols, or other characters);
   - The key;
   - The choice of operation (E/D, for encrypt/decrypt).
3. Process the phrase and key according to the encryption/decryption algorithm.
4. Print the output phrase and ask if the user wants to submit another phrase/key/option or quit.

Deliverables

You must submit:

- The file **a2_username.py**
  - This is your source code solution; be sure to include your name, date, assignment number and comments describing your code.
- A **README.md** file with "project notes" (describing what my TA and I cannot see by looking at your source code and/or running your program).
  - Examples: design decisions, documented limitations, future improvements, etc.
- A **screenshot of the results** produced by your code (make sure to show input and output for at least one run).

Notes and Hints:

- Follow the "cardinal rules" of programming in Python (as per the textbook).
- Start by breaking the program down into parts and solve smaller problems before producing the final solution.
- Try to handle special cases and prevent runtime errors to the best of your knowledge.
- Don't overdo it! You might want to try some of the bonus points options below but should refrain from going beyond the scope of the course so far.
    - Things to avoid (as cool as they might be) include: reading from / writing to files; making the solution web-accessible; implementing complex error-checking schemes; using classes and other OOP aspects (polymorphism, inheritance, etc.).

Bonus opportunities:

If you successfully implement one or more of the bonus options below **without breaking the baseline solution** (*) you will earn extra points (max 10% each):

1. **Brute force attack**
    - As you can tell, the baseline solution is very easy to break via brute force.
    - If you implement an **elegant** solution for breaking the code using brute force, you will earn up to extra 10 points.
2. **Encoding spaces, other characters, and punctuation symbols**
    - One of the tell-tale signs of simple encryption schemes is that they preserve spaces, punctuation symbols, etc.
    - If you implement an improved solution that takes those into account and encrypt/decrypt them as well, you will earn up to extra 10 points.
3. **More complex key**
    - An elegant way to increase the robustness of the baseline algorithm to brute force attacks is to allow an alphabetical phrase to be used as a key, traverse the key in a circular fashion, and take each character in the string as the numerical key for the shift operation.
    - If you implement an improved solution that allows the user to choose a second phrase as key, you will earn up to extra 10 points.

(*) A common mistake students make is to attempt bonus items before producing a "perfect 100" baseline solution.
Make no mistake: if your baseline solution doesn't meet **all** grading criteria (see rubric on Canvas), you are not eligible for bonus points. Period.