EMAT10007 – Introduction to Computer Programming

Assignment 1 – Building Programs

Overview

- The objective of Assignment 1 is to submit a Python program that provides the user with some basic **cryptographic** utilities, such as:
  - The ability to encrypt/decrypt messages using a Caesar cipher;
  - To read in the message either directly via user input, or by reading in the contents of a file;
  - To specify the rotation used by the Caesar cipher;
  - To produce a list of statistics about the messages, such as letter frequency, average word length, etc.;
  - Write a short, 1-2 page report, discussing your implementation.
- **The deadline is 15:00 on Friday 30$^{\mathbf{th}}$ November.** You must upload your assignment including the program (.py) and report (.PDF) to **Blackboard**.
- Along with your marks you will receive feedback written by your assigned TA. This feedback is important going forward with the next assignment, so you should ask your TA any questions you have regarding the comments you receive.
- For this assignment it is required that you submit individual work. You may discuss the creative approaches to solve your assignment with each other, but you must work individually on all of the programming and the accompanying report.
- For this assignment it is required that you submit individual work. You may discuss the creative approaches to solve your assignment with each other, but you must work individually on all of the programming. **We will be checking for plagiarism!**

Background: The basics of cryptography

- A Caesar cipher is a simple, well known cipher used in the encryption of strings. It is a 'substitution cipher', meaning that each letter is *substituted* with a corresponding letter in the alphabet at a predefined offset from the input letter' position.
- The table below shows a Caesar cipher with a rotation value of 13; a popular special case of the Caesar cipher known as ROT13.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
Input	A	B	C	D	E	F	G	H	I	J	K	L	M	N	...
Output	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	...

- In this example, the letter "A" is replaced by the letter indexed 13 positions to the right of "A"; the letter "N".
- This particular cipher is only defined for the letters of the alphabet, meaning that punctuation, spaces, and numbers are left unchanged.

Part 1 — Encryption and decryption

1. Your program should accept, as the **first** and **second** arguments, one of the following pairs of arguments:

   (a) `--encrypt [rotation]`

   (b) `--decrypt [rotation]`

   where `[rotation]` is a positive integer – you should check for this. If it is not, then the program should provide usage instructions for before stopping.

   If no arguments are provided, then the program should provide usage instructions for before stopping.

2. The **third** and **fourth** arguments are optional, and should be of the following format:

   (a) `--file [filename]`

   where `[filename]` should be replaced by the name of an input file provided for this assignment.

3. When the first argument is `--encrypt`, your program should:

   (a) Check whether `--file` has been passed in as an argument. If no `--file` argument is passed to the program, then the user should be prompted to enter a message via `input()`. Each non-empty line of input is stored as a separate message, until an empty line is entered (the user just presses `enter/return` twice), such that `input()` will return `""` - an empty string.

   (b) If `--file` has been passed in as an argument, your program should attempt to open a file with the name given as the **fourth** argument, and then read the contents of the file into a list. Each line should be considered a separate message.
   If a file by the provided name cannot be found, then print an error and stop the program.

   (c) Encrypt each message by the provided `rotation`, by passing the message and rotation to a **function** as arguments.

4. When the first argument is `--decrypt`, your program should follow the same steps as above, except your program should call a **decryption** function on each message.

   (a) Decryption follows the same process as encryption, only with the rotation being reversed.

5. All messages (either encrypted or decrypted) should be converted to **UPPER CASE** only.

Part 2 — Analysing messages

1. During the execution of your program, you should collect the following data on the plaintext (unencrypted English) messages:
   (a) Total number of words;
   (b) Number of unique words;
   (c) Minimum, maximum, and average word length;
   (d) Most common letter;
   (e) (Up to) The 10 most common words and their frequency, e.g., `the : 4`.
2. After all of the messages have been encrypted/decrypted by your program, you should print out the above statistics.
3. The statistics should be based on the full list of messages; you do not need to print these for each message.

Part 3 — Automated decryption

1. Modify your program so that, when no `rotation` argument is provided after the `--decrypt` argument is passed to your program, you read in a file of common English words (to be provided).
2. You should then attempt to automate the decryption process by implementing the following:
   (a) Iteratively apply the decryption function to the first message, incrementing the rotation by 1 each time.
   (b) During every iteration, apply the decrypt function and attempt to match words in the decrypted message with words found in the `common words` list.
   (c) If matches are discovered, then present the message to the reader, and ask if the message has been successfully decrypted:
      i. If the user answers "no", then continue to increment the rotation until the message is successfully decrypted.
      ii. If the user answers "yes", then print the rotation for this message, and start again at (a) with the next message, until all messages have been successfully decrypted (they may not share the same rotation).
3. This list of words should be used to incrementally attempt to decrypt the message by increasing the `rotation` used, until the message is found to contain the English words.

(Optional) Part 4 — Enhancements

Instruction: If you implement any enhancements to your program, you **must** comment them using the following format: `#!# Comment here ...` as this will allow the TAs marking your project to easily locate your enhancements. An example comment might start like this:

```
#!# Here I create a set of common patterns in English to ...
```

In the report, discussions related to your enhancements should go in a separate section.

The following section outlines some ideas to extend your program for those students confident with programming and looking to achieve additional marks. **This section is strictly optional**.

1. You may try to improve on the automated decryption process in a number of ways, all of which are left open for you to research and implement:

   (a) Try out every possible rotation value, and sort them according to which rotation matches the most words matched to the rotation value resulting in the least words matched. This should help reduce how many times you need to ask the user if the rotation is correct.

   (b) Research and identify common patterns in the English language. Develop a function which will attempt to match these in the encrypted input message in order to identify the rotation *without* using the common words dictionary.

2. Add an alternative argument after `--encrypt` and `--decrypt`, with the value `random`, which can be used like so:
   `--encrypt random`
   which calls a completely different set of encryption/decryption functionality described below.

   Encryption:

   (a) For each word in the message, you should select a rotation value at random, and then apply the rotation to the word.

   (b) You should then append a random letter to the beginning of the encrypted word, and the letter `rotation` positions down the alphabet to the end of the word. Example program input and output might be:
   `Please enter a message:  "Hello world!"`
   `Output:  AURYYBN EASVPH!I`
   You may be able to work out that the first word uses a rotation of 13 (A $\rightarrow$ N) and the second word uses a rotation of 4 (E $\rightarrow$ I).

   Decryption: Given that we now know the new encryption scheme, you should implement the decryption functionality for input encrypted using this new scheme.

Helpful hints and suggestions

- **Part 1** of this assignment involves implementing encryption and decryption functionality based on the Caesar cipher, with `arguments` passed to the program – typically via the `command line` – which is similar to how you pass arguments, or `parameters`, to functions.

  In order to pass arguments to your program, you will need to add the arguments via Pycharm's `run` menu.

  1. Select `Run` ¬ `Edit Configurations...`

2. With your program selected on the left, you should see a text box labelled `Parameters`. You place your program arguments here like so:
   `--encrypt 13`
3. Parameters are separated by spaces.
4. You should start building your program by targeting a specific set of arguments, and making sure it works properly for that set of arguments. Then continue to expand the program to work for different arguments.
5. Ensure that your program exits when the arguments are not recognised as being in the correct sequence/format.

Then, to use the arguments passed to your program, you can add the following to your program:

```
import sys
Arguments = sys.argv[1:]
```

where `sys.argv[1:]` ignores the first argument passed to the program, which is always the `program name`, and creates a list of your program arguments, simply called `Arguments`.

- **Comment your code!** — Comments are something we haven't discussed much up to this point, however they are crucial for many reasons:

  1. They help other readers to understand what your code is doing;
  2. They help *you* remember what your code is doing, when you come back to it weeks/months/years later. The whole point of writing code is that it can be reused many times, so to make it reusable, add comments;
  3. To help your TA assess your understanding of the problem you are trying to solve, and to assess how well your solution solves it. If you use a data structure to implement some functionality, explain it in the comments.

  – Short comments are great to summarise the next several lines of code by giving a high-level overview.

  – For longer, more detailed comments (likely useful in functions and for Part 4), by typing `"""` and pressing enter, PyCharm will automatically produce a formatted multi-line comment for you to describe your function or large section of code.

- You should spend plenty of time planning out how the flow of your program might look *before* you start programming. For example, you might consider:

  – Should I implement separate functions for encryption and decryption?

  – If I use a single function, does that simplify the flow of the main section of my program? Or is it easier for me to work with two separate functions?

  – The input may come from either the `input()` function, or from a file, but the result is the same: a list of messages – Should I handle the data from the input sources using the same code where possible?

- You can halt the execution of your program at any point by calling the function `sys.exit()`. You will need to `import sys` in order to use this function. You should make calls to this function when:
  - You have printed out the message statistics, after all of the messages have been processed;
  - When the `--file` argument has been passed, but the file cannot be read.

  and any other time you feel is appropriate, e.g., if the user gives a negative rotation value. This may be useful to you.

Alterations to be made for 2019/20

- **Part 1** Parameters are to be removed - auto-marking is no longer being used
- **Part 3** more clarity on different rotation for each message.