EMAT10007 – Introduction to Computer Programming

Assignment 2020 – Encrypted Information

Overview

- The objective of the assignment is to submit an interactive Python program that allows the user to:
  - Encrypt/decrypt messages using a Caesar cipher.
  - Input the message to be encrypted/decrypted.
  - Specify the rotation used by the Caesar cipher.
  - Extract statistics about the messages, such as letter frequency, average word length, etc.
  - Extract, manipulate and create a visual representation of information contained in a decrypted message
- Your Python program should be accompanied by a **short, 1-2 page** report. This should discuss your implementation and any design choices. For example, why you chose to use a specific data structure or how you improved the reusability of your code.
- **The deadline is 15:00 on Friday 11th Decemeber** You must upload your assignment including the program (.py), report (.PDF) and any output files generated by the program (.csv, .pdf) to **Blackboard**.
- You must show which part of the assignment each section of your code answers by adding comments showing part and sub-section (e.g. Part 1.1) using the following format :

  ```
  # PART 1.1 Comment here ...
  ```

  Note that the order that the answers to the questions appear in the code may be different to the order they appear in this document so it is important to indicate to the marker where you have attempted to answer each question. You should also use comments to show what your code does.
- This is an individual project. 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!**

Helpful hints and suggestions

- Complete all of the exercise sheets first - they have been designed to prepare you for this assignment.
- You should spend plenty of time planning the flow of your program **before** you start programming. Look back at previous exercise sheets to see how to approach larger problems by breaking them down into smaller ones.

- Remember to think about the **readability** and **reusability** of your code. Some question you should ask yourself:
  - Have I named things sensibly? Could someone pick up my code and understand it?
  - Am I repeating lots of code? Can I reuse any?
  - Can I simplify the layout of my code to make it more readable?
- You will gain marks for:
  - Types: Proper use of data types and data structures.
  - Comments: Appropriate and useful comments.
  - Naming: Appropriate variable and function names (remember we want you to use CamelCase)
  - Report: Informative report, which explains your thought process and analyses the choices you made in your code design. You must reference any external code or ideas used.
  - Working code: Does the code do what it is supposed to? **READ THE QUESTIONS CAREFULLY to check**. Is the program robust, i.e., does it deal correctly with wrong inputs (user/program interaction).
- Finally, don't forget to ask for help! You will be able to ask your TA for help and advice during the weekly drop-in sessions and tutorials.

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’s position. The the size of the offset is called the *rotation*.

- 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 ask the user for the following information:
   - The `cipher mode` : encrypt or decrypt the message
   - A `rotation value` : how many places should the cipher shift each character.
   - A `message` : the text to be encrypted/decrypted. Your program should be able to process both single line and multi-line messages.

   The only actions required of the user to operate the cipher should be to:

   (i) run the python program

   (ii) provide these three inputs when prompted
2. If no inputs are provided, or the provided input is incorrect, the program should prompt the user for the inputs again.
3. When the `cipher mode` chosen is `encrypt` then your program should encrypt the message given by the user using the rotation given by the user.
4. When the `cipher mode` chosen is `decrypt` your program should decrypt the message given by the user using the rotation given by the user. (Note that decryption follows the same process as encryption, only the shift goes the opposite way.)
5. The program should then print the encrypted message if the cipher mode is `encrypt` and the decrypted message if the cipher mode is `decrypt`.
   Numbers, punctuation and spaces should be unchanged.
   All messages (either encrypted or decrypted) should be returned as **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) message:

   (a) Total number of words;

   (b) Number of unique words;

   (c) (Up to) The ten most common words sorted in descending order by the number of times they appear in the message.

   (d) Minimum, maximum, and average word length;

   (e) Most common letter;
2. **After** the whole message has been encrypted/decrypted by your program, the program should print out the above statistics.
   The most common words sorted in descending order (1c) should be printed with the following format:

   ```
   the : 4
   ```

   (i.e. 'the' has been found 4 times)

Part 3 - Messages from a file

1. Modify your program that **before** asking the user to input the `message` the user is prompted to a `message entry mode`:
   (a) `manual entry` : typing in a message to encrypt/decrypt;
   (b) `read from file` : specifying a text file, the contents of which will be encrypted/decrypted;
2. If the user chooses `manual entry`, the `message` should be typed in, as before.
   If the user chooses `read from file`, the user should provide a `filename` when prompted for the `message`.
   The prompt shown to the user when they are asked to input the message should indicate these requirements.
3. If the user chooses `read from file`, the program should attempt to open a file with the name given and read in the contents of the file as the `message`.
4. If the `filename` provided cannot be found, the program should print an error and ask again.
   The program should then continue to work as before.

Part 4 - Automated decryption

1. Modify your program to include an additional `auto-decrypt` option for `cipher mode`.
2. If `auto-decrypt` is chosen, the program should read in `words.txt`, a file of common English words (this is provided and can be downloaded from BlackBoard).
3. The program should then attempt to automate the decryption process by implementing the following algorithm:
   (a) Iterate through all possible rotations, applying the rotation to the first line of the message.
   (b) During each iteration:
      (i) attempt to match words in the rotated first line with words found in the common words list.
      (ii) If one or more matches are discovered, then present the line to the reader, and ask if the line has been successfully decrypted:
      (iii) • If the user answers “no”, then continue to iterate until the first line is successfully decrypted.
         • If the user answers “yes”, then apply the successful rotation to decrypt the rest of the file.

Part 5 - Using Data From imported files

1. Use your program to decrypt the file `douglas_data_encrypted.txt`, a data set about wooden beams.
2. Plot the **knot ratio** against the **bending strength** of the beams.
   Fit a trend-line to the data and display this on the same plot along with the equation of the fitted line.

3. The root mean square error (RMSE) is metric often used to judge how well a trend line fits the data. The RMSE is the square root of the mean of the sum of the error, ε_i, squared, for all data points. A smaller RMSE indicates a better fit between raw and fitted data.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2}$$

The error ε_i is the difference between the raw data point, a, and the fitted data point, y, for some input value x_i:

$$\varepsilon_i = a(x_i) - y(x_i)$$

Find the RMSE for the raw data and fitted data on the knot ratio vs. beam strength plot and display the value on the plot.

4. Save the plot as a .pdf file.

5. The bending stiffness k of a cantilever beam can be found by:

$$k = \frac{3EI}{L^3}$$

where E is the Young's modulus, L is the distance from the beam support, $I = \frac{b^4}{12}$ is the area moment of inertia of a beam with a square cross section with sides of length b (Figure 1).

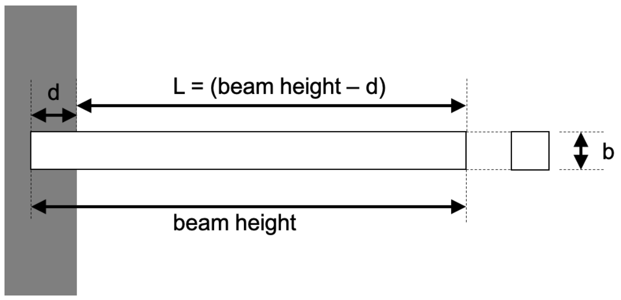

Figure 1: Beam configured as a cantilever showing parameters to calculate bending stiffness

Modify your program to:

(a) Use parameters `E` and `BEAMHEIGHT` from the decrypted file to find the bending stiffness in Nmm^{-1} for each beam. Assume the beam is configured as a cantilever where L = (`beam height -` d), d = 10 cm. **Take care to check the units used in the decrypted file!**

(b) Save the data-set as a .csv file:

- Include the bending stiffness as an extra column.
- Exclude the `SAMPLE` column from the original data-set.

Part 6 - (* Optional) Enhancements

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 optional!**
Important: If you implement any enhancements to your program, you **must** comment them using the following format:

```
#!EXTRA# Comment here ...
```

as this will allow the TAs marking your project to easily locate your enhancements. An example comment might start like this:

```
#!EXTRA# 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.

1. Suggestions for improving the `encrypt` process:
   - Modify your program so that the user may select `random` instead of a `rotation value`. The cipher will shift the text by a number of places equal to a random number generated by the program.
   - Modify the `encryt` process so that for each line in the message, a new rotation value is selected at random, then applied to the line (Update the `decrypt` to decrypt line by line).
2. Suggestions for improving the `auto-decrypt` process:
   - Modify the program to try every possible rotation value and sort them according to number of words matched with the list of common words. The program should select the solution with the greatest number of instances as the decrypted solution.
     If the greatest number of instances applies to more than one word, the program should present the solutions to the user to identify which is correct. (This can help reduce how many times you need to ask the user if the rotation is correct.)
