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, maximum word length.
  - Automate the identification of the rotation needed to decipher an encrypted message.
- Your Python program should be accompanied by a **short, 1-2 page** report, submitted as a .pdf file. 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 re-usability of your code.
- 100 marks in total are available:
  - 85 marks for the Python program (includes 5 marks for general Python implementation)
  - 15 marks for the report
- **The deadline is 13:00 on Friday 11th December (GMT)**. You must upload your assignment including the program (.py), report (.pdf) and any additional files generated by or needed to run your program (e.g. .txt files) to **Blackboard**. You can find the submission point under Assessment, Submission and Feedback.
- 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.
  Some blocks of the code may include the answer multiple questions. Make sure you indicate this in your comments where applicable. Your comments should tell the reader what your code does, in addition to indicating which question you are answering.
- 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 questions 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: Appropriate use of data types and data structures.
  - Comments: Concise, clear and useful comments.
  - Naming: Appropriate variable and function names
  - 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).
  - Concise, efficient code e.g. use of functions, list comprehensions, imported functions from Python packages.
  - More advanced programming techniques such as use of classes and user-defined libraries stored as separate .py files.
  - 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.
- 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.

Programming Tips and Hints

- ASCII (American Standard Code for Information Interchange), is a character encoding used to represent text (letters and other characters) as numbers https://www.ascii-code.com/. Upper and lower case alphabet letters, for example, are represented by numbers in the range 65-90 and 97-122 respectively. Punctuation marks, spaces etc also have ASCII encodings.
  Python `ord()` and `chr()` are built-in functions that can be used to convert from a string character to the number used to represent it in ASCII (`ord()`) and from a number from the set used in ASCII the character it represents (`chr()`).

- The built-in Python function `sorted()`, returns a sorted list of items in an iterable (e.g. a list). If each item is a tuple, `sorted()` will sort the tuples using the first element in each tuple.
  So to sort **multiple data structures**, using the order of **one** of them, first re-organise the data-structures into a data structure, where each element is a tuple.
  https://docs.python.org/3/howto/sorting.html

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 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 [Total: 20 marks]

1. Your program should ask the user for the following information:
   - The `cipher mode` : encrypt or decrypt the message
   - A `rotation value` : number of places (positive or negative) the cipher should shift each character.
   - A `message` : the text to be encrypted/decrypted.

   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 output should be to `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 left unchanged.
   All messages (either encrypted or decrypted) should be returned as **UPPER CASE** only.
6. Modify your program so that the user may select to use a random number as the `rotation value`. If the user selects to use a random number, the cipher will shift the text by a number of places equal to a random number generated by the program. Note, the option for the user to specify the `rotation value` explicitly must remain in the program.

Part 2 - Analysing Messages [Total: 25 marks]

1. In this exercise, the definition of a **word** is a collection of characters with a space at the leading and trailing end of the characters, excluding any numbers and punctuation marks.
   - `python` is a word
   - `123` is not a word
   - `py8thon` should be interpreted as the word **python**
   - `python!` should be interpreted as the word **python**

   During the execution of your program, you should collect the following metrics on the plain-text (**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. In other words, if there are **ten or fewer** words in total in the message, sort all words in the message, but if there are **greater than ten**

words in total, select only the ten most common words to sort.
(Note: If multiple words appear in the message the same number of times, this may result if two or more words being the tenth most common. In this case your program should limit the set of words to show only ten of the most common words.)

Hint: The built-in Python function `sorted()` returns a sorted list of items in an iterable (e.g. a list). If each item in the list is a tuple, `sorted()` will sort the tuples by comparing the the first element of each tuple, and if it's equal the second, and so on. So to sort **multiple data structures**, using the order of **one** of them, first reorganise the data-structures into a data structure, where each element is a tuple. https://docs.python.org/3/howto/sorting.html

(d) Minimum and maximum word length.

(e) Most common letter (Remember to exclude spaces and punctuation marks!).

Hint: Be careful of punctuation when checking for unique words (e.g. `'hello'`, and `'hello,'`, may be incorrectly interpreted as two different words due to the comma in the second word). You may need to disregard punctuation to correctly count the number of unique words.

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

```
the : 4
```

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

3. The program should save the metrics:
   - total number of words
   - number of unique words
   - minimum word length
   - maximum word length

   as a .txt file. Each metric should appear on a new line. The name of the metric followed by the value should appear in the file e.g. `total number of words` followed by the value:

```
total number of words:  57
```

Part 3 - Messages from a File [Total: 5 marks]

1. Modify your program so that **before** asking the user to input the `message` the user is prompted to select 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` (including file

path) 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 [Total: 15 marks]

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.

   (c) In the case that no successful decryption is found, the program will be unable to collect the metrics (Part 2) on the plaintext message. You should account for this in your code and ensure that, if no match is found, the program skips collecting the metrics and printing the decrypted message, and continues to work.

Part 5 - (* Optional) Enhancements [Up to 15 marks]

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, a bar chart is generated ...`

If you implement any enhancements to your program, that modify your answers to Parts 1-4 you must make a copy of your original answer so that it can be marked (for example, comment out your original answer so that it can still be seen or submit the modified program as a separate Python file).
In the report, discussions related to your enhancements should go in a separate section.

1. Suggestions for using metrics collected on the data:
   - Produce a bar chart showing (up to) the ten most common words (Part 2) on the horizontal axis and the number of times they appear in the message on the vertical axis.
   - Save the plot as a .pdf file.
2. Suggestions for improving the `encrypt` process:
   - Modify the `encrypt` process so that for each line in the message, a new rotation value is selected at random, then applied to the line (update `decrypt` to decrypt line by line).
3. Suggestions for improving the `auto-decrypt` process:
   - Modify the program to try every possible rotation value and sort the solutions according to the number of words matched with the list of common words in `words.txt`.
     The program should select the solution with the greatest number of matched words as the decrypted solution. If there is a tie between more than one solution regarding the number of words matched, 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).
