Assignment Report EMAT10007 - Vandam Dinh

Introduction

When my program is executed it asks the user to choose a 'Cipher Mode', either 'Encrypt', 'Decrypt' or 'Auto-decrypt'. The user selects by inputting the appropriate letter.

- If the user chooses to encrypt or decrypt, the program asks the user for the 'Rotation Mode', manual or random.
  - The program asks for the shift value if the user chose manual in the previous step. The program will output the random shift value if the user chose random in the previous step.
  - The program then asks the user to choose a 'Message Entry Mode', the ability to type a message or choose a file.
  - The program asks for the message you'd like to encrypt or decrypt, depending on the Cipher mode the user chose and if they chose to type a message.
  - The program then encrypts or decrypts the message the user specified, outputs it and all the metrics such as the most common words in descending order and their frequencies.
  - A file in the same directory as `main.py` called metrics.txt has metrics written into it. The metrics include:
    - Total number of words
    - Number of unique words
    - Minimum word length
    - Maximum word length
    - Most common letter
  - The program always analyses the unencrypted message.
- If the user chooses Auto-decrypt, the program asks the user to choose a 'Message Entry Mode', the ability to type a message or choose a file.
  - The program then asks, either, to enter a message or enter a filename.
  - Once the message is accepted, the program automatically applies the Caesar Cipher and compares the results to a words list called words.txt in the same directory as main.py.
  - If a match is found, it is outputted for the user to confirm if it's correct.
    - If correct, the program will output the decrypted message, analyse it and output all the metrics.
    - If incorrect, the program continues shifting until it finds another match.

Analysis

Data Types and Data Structures:

- In Part 1.2, the program asks the user to choose an integer value for how much the cipher should shift by. Initially when the function takes in the input, the input variable `rInt` is a string, the program then tries to convert it into an integer, if successful `rInt` is assigned to a different variable called `rotation`. If the user doesn't enter an integer value, then the function sees the error `ValueError` and asks the user to input again. The value of the rotation needs to be an integer, so it can be used to manipulate characters.
- In Part 2.1, my function to analyse called `analyse()` applies onto the variable `infoMessage`, `infoMessage` is a string where all numbers and punctuation are removed. If a user inputted this, (assigned as a variable called `message`):

  ```
  Hello, my name is Vandam Dinh and I like to code! :)
  ```

  `infoMessage` would look like this:

  ```
  Hello my name is Vandam Dinh and I like to code
  ```

  In my `analyse()` function, I turn `infoMessage` into a list, so that I can find the number of elements in the list, which returns the total number of words in the inputted message. As a list, I can also find the minimum and maximum word lengths. I also turn `infoMessage` into a set to remove any duplicates, this allows me to find how many unique words there are.

  In order to find the most common letter, I have to remove all the spaces from `infoMessage` first. Then I use `collections.Counter()` and `.most_common()` to find the most common letter in the string.
- In Part 2.3 I needed to sort all the unique words by frequency in descending order. Firstly, I realised that I would need `infoMessage` as a list again, so I assigned it to a variable called `msgList`. I was then able to arrange `msgList` in order of descending frequency, I called this `sortedList`. I also needed to be able to find out what the frequency of a particular word was, so I created a variable called `wordFreq`,

where I used `collections.Counter(sortedList)` to turn it into a container that stores each word as dictionary keys and their values are their frequencies. I then turned it into a dictionary called `wordDict`.

If the number of unique words was less than or equal to five, the program would print the words in descending frequency by turning `sortedList` into a set to remove any duplicates and outputting it.

If the number of unique words was greater than 5, the program would compare the frequency of the 5th and 6th most frequent words using `wordDict`. If they were matching, then the program would only return 4 of the most frequent words. It would do the same with the 4th and compare it with the 6th, if they matched, then it would only return 3 of the most frequent words. It compares all the way up to the first word, if all the words have the same frequency no list is outputted, if a list was outputted it could mislead users to think that those words are the most frequent where in reality they're the same as all the others. The output is assigned to a variable called `reducedList`.

- In Part 2.4, I need to print up to the five most common words in descending order. I am able to create a for loop where the range is the 'length' of `reducedList` (i.e. the number of terms in the list). It follows the same format as 2.3 where the sorted list does not exceed the length 5. The function pairs the word with its frequency by grabbing the word's frequency from `dictFreq`. We can do this because we know the key of the value, it's the word itself.
- In Part 4.2, I read `words.txt` and assign it to a variable called `wordsFile`. This variable is a string, where all the line breaks are converted into spaces. (I also create a version where the first character of each word is capitalised, so the program has a greater chance of matching sentences, I assign it to a variable called `c_wordsFile`, this is EXTRA). After applying the Caesar cipher, the result is a string called `decrypted`, in order for me to use the function `any()` I need to convert it into a list, I do this and assign it to a variable called `decryptedList`, I do the same for `wordsFile` and `c_wordsFile` and assign them to `wordsFileList` and `c_wordsFileList` respectively.

Functions and Imported Functions from Python Packages

- For the majority of this program, I defined functions and called them in `main.py`. The main purpose of this was that I was able to call them again easily without having to copy and paste lines and lines of code. Encrypt, decrypt and auto-decrypt are almost the same, but the steps are just arranged differently, so using functions made it easy to format the program.
- In Part 1.2, I imported the function `.randint(,)` from the package `random` so that I could generate a random integer between 1 and 25.
- In Part 2.1, I need to analyse a string of text, I can only do this if all punctuation and numbers are removed from the string, so I imported the function `.sub()` from the package `re`. This is what helps me define `infoMessage`.
- In Parts 2.1, 2.2 and 2.3 I imported the function `.Counter()` from the package `Collections`, this allowed me to create a container that stores each word as dictionary keys and their values are their frequencies. This was useful if I wanted to find and use a particular word's frequency later on, such as in Part 2.3
- In Part 3.3, I needed to find out if a file path existed or not. I did this by importing the function `.isfile()` from the package `os.path`.

More Advanced Programming Techniques

- The program was arranged over the files `main.py`, `functions.py` and `config.py`. `main.py` is used to execute the code and where the user sees all the outputs. `functions.py` is used to contain all the functions which `main.py` uses. This made it so that it was easier to alter the functions without accidentally breaking something in `main.py`, also I think it looks more readable than if it was in one file. `config.py` contains all the variables that `main.py` and `functions.py` use, I couldn't get it all to work when all the variables were mixed up between the files so I found this way to be the best.
- In Part 4.2 I mentioned previously that I create a version of `words.txt` string where the first character of each word is capitalised. I realised that if a user inputted a grammatically correct sentence where they use capital letters, sometimes those particular words wouldn't match with the words in `words.txt`. This is because the program doesn't think an uppercase and lowercase letter match with each other. By creating this capitalised version, `c_wordsFile`, this increases the chances of the program matching a word.
- Also in Part 4.2, at one point, the program asks the user to input whether it correctly decrypted the message. I improved the program by creating an error message and looping it back to the question if 'y' or 'n' wasn't taken as an answer. Before, if 'y' or 'n' wasn't entered, for me the cipher would continue.
- I created the functions `encrypt()` and `decrpyt()` where they perform the Caesar cipher and reverse respectively. These functions are then called in `main.py`.
- I created a function called `chart()`. In Part 2.3 where I have the `sort()` function, I created two variables called `listFreq` and `listWord`. `listFreq` is a list which contains the frequency of a word in descending order, and `listWord` is a list which contains the words in descending frequency. I then call those variables later in the function `chart()`, where I define the x-axis as `listWord` and the y-axis as `listFreq`. I only use the first five terms in those lists. Then a bar chart is outputted to the user. The x and y-axis are labelled, and the chart has a title.

Conclusion

Overall, I am quite happy with my program. I believe I was able to implement all the steps that were given, but there are a few changes I would've liked to make if I had more time. I feel like I could've utilised classes, but I wasn't quite sure what the best way I could have implemented them, or maybe I didn't even need them at all. I would've also like to see if I was able to create a GUI, but I just didn't have time to research into all the different frameworks that are available.

References

Part 1 - Encryption and Decryption

- 2. I couldn't change the value of cError to False inside the function so I had to look up how to do so using: https://www.programiz.com/python-programming/global-keyword https://stackoverflow.com/questions/18791882/how-to-make-program-go-back-to-the-top-of-the-code-instead-of-closing
- 2. To generate a random integer: https://stackoverflow.com/questions/3996904/generate-random-integers-between-0-and-9
- 3. To check if a string contains a number: https://docs.python.org/3/library/re.html

Part 2 - Analysing Messages

- 1. To remove integers from a string: https://www.delftstack.com/howto/python/remove-numbers-from-string-python/
- 1. To find the minimum and maximum length of a word I used: https://stackoverflow.com/questions/47113916/find-the-length-shortest-word-in-a-string-in-python/47113942
- 1. To find the most common letter I used: https://stackoverflow.com/questions/4131123/finding-the-most-frequent-character-in-a-string
- 1. But I ran into a problem were the most common was ' ', so I removed the spaces with: https://www.journaldev.com/23763/python-remove-spaces-from-string
- 3. To sort a list by frequency: https://www.geeksforgeeks.org/python-sort-list-elements-by-frequency/
- 3. To keep and order of a list when converting to a set: https://blog.finxter.com/python-list-remove-duplicates-and-keep-the-order/
- 3. To find the frequency of elements: https://stackoverflow.com/questions/2161752/how-to-count-the-frequency-of-the-elements-in-an-unordered-list

Part 3 - Messages from a file

- 3. To find out if a file exists: https://stackoverflow.com/questions/82831/how-do-i-check-whether-a-file-exists-without-exceptions

Part 4 - Automated Decryption

- Resource I used to help my create the Caesar Cipher: https://www.pythonpool.com/caesar-cipher-python/
- Creating my own ciphers to test with: https://cryptii.com/pipes/caesar-cipher
- 2. To join a list together to form a string: https://stackoverflow.com/questions/12453580/how-to-concatenate-items-in-a-list-to-a-single-string
- 2. To see if any element in a list matches with an element in another: https://www.techbeamers.com/program-python-list-contains-elements/#any-method

Part 5 - Extras

- To have integer values on the y-axis: https://stackoverflow.com/questions/12050393/how-to-force-the-y-axis-to-only-use-integers-in-matplotlib