**EMAT10007 – Introduction to Computer Programming**

# Assignment 2021 – Encrypted Information

## Overview

- The objective of the assignment is to submit an interactive Python program that allows the user to:
  - Input a message to be encrypted/decrypted using a Caesar cipher.
  - Specify the rotation used by the Caesar cipher.
  - Automate the identification of the rotation needed to decipher an encrypted message.
  - Extract statistics about the messages, such as letter frequency, maximum word length, etc.
- Your Python program should be accompanied by a **short, 1-2 page** report, submitted as a .pdf file called report.pdf. This should discuss your implementation and any design choices. Your report should contain the following sections:
  - Introduction: An overview/summary of what your program does to show that you have attempted to implement all the required components.
  - Analysis: Discuss what programming techniques you chose to meet the objectives of the assignment and discuss why you made these decisions. For example, why you chose to use a specific data structure or how you improved the re-usability of your code.
  - Conclusion: Discuss anything in your program that you think could be improved. If possible, make suggestions for what you could try to do to improve it if you had more time.

  You must reference any external code or ideas used, including code snippets you may have found online.
- **The deadline is 13:00 on Friday 10th December** You must upload your assignment via **Blackboard**. Your submission should include only the program (.py file(s)), report (.pdf file) and any output files generated by the program (.csv, .pdf etc) .
- 60 marks available:
  - Exercise 1-5 [45 marks]
  - Report [10 marks]
  - **readability** and **re-usability** of your code (sensible naming, use of comments and documentation strings, tidy code, avoiding repetition). [5 marks]
- 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 if you haven't already - they have been designed to prepare you for this assignment.
- Plan 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.
- Start your project early. Be aware that as the deadline approaches, the weekly drop-in session are likely to get busier.
- Read this information sheet carefully one more time before you submit your assignment to check you have understood and answered all questions.
- Finally, don't forget to ask for help! You will be able to ask lecturers and TAs for help and advice during the weekly drop-in sessions.

## 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 "N" because "N" is indexed 13 positions to the right of "A".
  Because there are 26 letters in the alphabet, ROT13 replaces each letter by its partner 13 characters further along the alphabet. For example, HELLO becomes URYYB (and URYYB becomes HELLO again).
- The Caesar cipher is only defined for the letters of the alphabet, meaning that punctuation, spaces, and numbers are left unchanged.
- Wikipedia has some more information about the Caesar cipher:
  `https://en.wikipedia.org/wiki/Caesar_cipher`

**Caeser Cipher Examples**

Input: **hello world**
Encrypt, rotation 13
Output: **uryyb jbeyq**

Input: **hello world**
Encrypt, rotation 3
Output: **khoor zruog**

Input: **khoor zruog**
Decrypt, rotation 3
Output: **hello world**

Input: **hello world**
Encrypt, rotation -3
Output: **ebiil tloia**

## Background: ASCII characters

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()`).

### Part 1 - Encryption and Decryption

**Total: 10 marks**

1. The file that is run to execute your program should be named **main.py**
2. When run, the program should ask the user for:
   - The **cipher mode** : User should enter `E` to encrypt a message or `D` the decrypt a message
   - A **rotation mode** : User should enter `M` to choose a numerical value or `R` for the program to select a random integer value.
     - If `M` is entered, the user should be prompted to enter an integer value indicating how many places the cipher should shift each character and this number should be assigned to a variable `rotation` as a numerical value.
     - If `R` is selected a random integer value should be generated and assigned to a variable `rotation` as a numerical value.
   - A **message** : User should enter a message to be encrypted or decrypted.

   **[6 marks]**
3. If no input is given, or incorrect input is given (e.g. no rotation value is entered, or the user enters a value other than `E` or `D` when asked if they want to encrypt or decrypt a message), the program should print an error message and prompt the user for the input again. The program should then continue to work as before.
   **[1 mark]**
4. The program should print the encrypted message if the cipher mode is `encrypt` and the decrypted message if the cipher mode is `decrypt`.
   The message should be printed in **UPPER CASE** letters.
   Numbers, punctuation and spaces should be unchanged.
   **[3 marks]**

### Part 2 - Analysing Messages

**Total: 12 marks**

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

The program should do the following:

1. Compute the following data on the plaintext (**un-encrypted**) message (i.e. whether `encrypt` or `decrypt` is used, the un-encrypted message should be used to analyse words):
   (a) Total number of words
   (b) Number of unique words
   (c) Minimum word length
   (d) Maximum word length
   (e) Most common letter

   **[6 marks]**
2. Save each metric in a .txt file called metrics.txt in the same directory as the program.
   - total number of words
   - number of unique words
   - minimum word length
   - maximum word length
   - most common letter

   Each metric should appear on a new line with the following format:

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

   (i.e. total number of words is 57)
   **[3 marks]**
3. Sort all unique words by the frequency that they appear in the message, in descending order, and print (up to) the five most frequently occurring words.
   If there are five or fewer unique words in the message, include all unique words.
   **Input:** `This one that one`
   **Output:** `[one, This, that]` or `[one, that, This]`
   If there are more than five unique words, include only the five most common words.
   If there are more than five unique words, and multiple unique words with the same frequency, such that the sorted list of unique words exceeds length 5, then exclude these words, so that the list length does not exceed 5.
   **Input:** `Here you see a long message for you to encrypt, it may take you a while`
   **Output:** `[you, a ]`
   **[2 marks]**
4. Print the (up to) five most common words sorted in descending order with the following format:

   ```
   the: 4
   ```

   (i.e. 'the' has been found 4 times)
   **[1 mark]**

## Part 3 - Messages from a file

**Total: 5 marks**

1. Modify your program so that **before** prompting the user to enter the message to encrypt/decrypt the user is prompted to select a **message entry mode** by entering:
   (a) `m` : to indicate the user will type in a message to encrypt/decrypt
   (b) `f` : to indicate the user will enter the name of a text file, the contents of which will be encrypted/decrypted

   **[1 mark]**
2. If the user enters `m` when prompted, the user should be asked to enter the message directly, as in Part 1. If the user instead chooses `f`, the user should alternatively be asked to enter a filename (or file path if the file is located in another directory).
   **[2 marks]**
3. In either case, if no input is given, or incorrect input is given (e.g. if no message is entered is given, or the filename provided cannot be found, the program should print an error and prompt the user for the input again. The program should then continue to work as before.
   **[1 mark]**
4. If the user chooses `f`, the program should read the contents of the file, assisgn the contents to a variable, then encrypt/or decrypt message in the way defined in Parts 1 and 2 of the assignment.
   **[1 mark]**

## Part 4 - Automated decryption

**Total: 8 marks**

1. Modify your program so that when run, the program will accept a new option when it prompts the user to enter the cipher mode: The user should enter `E` to encrypt a message or `D` the decrypt a message as in Part 1, but should now additionally accept `A` indicating that the user wants to auto-decrypt the message.
   **[1 mark]**
2. The program should then automate the decryption process by implementing the following algorithm:
   (a) Read in `words.txt`, a file of common English words (this is provided and can be downloaded from BlackBoard).
   (b) Iterate through all possible rotations, applying the rotation to the first 10 words only (or all words if the message contains 10 words or less).
   (c) During each iteration:
      (i) attempt to match words in the rotated first 10 words with words found in the common words file, `words.txt`.
      (ii) If one or more matches are discovered, then present the (up to) 10 words to the reader, and ask if the 10 words have been successfully decrypted.
      (iii) If the user answers "yes", then apply the successful rotation to decrypt the rest of the file.

If the user answers "no", then continue to iterate until the first line is successfully decrypted.

(iv) Print the decrypted message.

(v) 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.

[**7 marks**]

## Part 5 - Enhancing your Program

**Total: Up to 10 marks**

Implement your own idea to enhance your Caesar cipher program further.
Please submit your answer to part five as a **separate file** named **enhanced.py**.
This is to prevent you from losing marks in the case that your enhancements cause the code you wrote for Parts 1 - 4 to behave differently.
You **must** comment all enhancements using the following format to enable the person marking your project to easily locate your enhancements.:

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

An example comment:

```
#!EXTRA# Generate bar chart showing frequency of the five most common words
```

Suggested enhancements:

- Produce a bar chart showing (up to) the five most common words (found in Part 2) on the horizontal axis and the number of times they appear in the message on the vertical axis. You can modify the appearance of the bar chat by adding a title and/or axis labels, and changing the colour, and you can save the plot as a .pdf file, for example.
- Create a Caesar cipher Python module that is imported to perform encryption and decryption operations within a main program.
- Write a Caesar cipher class containing encryption and decryption methods.
- Implementing a different type of cipher in Python
  e.g. `https://www.tutorialspoint.com/cryptography/traditional_ciphers.htm`

---