EMAT10007 – Introduction to Computer Programming

Assignment 1 – Mark Scheme 2019

There are a total of 100 points available. This project is a weighted 20% of the overall mark. While this is not a huge amount, the feedback will be very useful for them moving forwards with the larger second project.

Note that we assess if they have understood the principles and can employ them to solve problems.

If you are in doubt, please let me know. Also, if you see plagiarism please let me know. (Either in person at the marking session or by email and highlighting on the online spreadsheet.)

Part 1 - Basic Implementation (15 marks)

- Correctly working implementation
  - encryption/decryption (3 marks each)
  - upper case (1 mark)
  - non-letters preserved (2 marks)
  - multi-line preserved (1 mark)
- Correctly working and robust user input (5 marks):
  - cipher mode,
  - rotation value,
  - message.

Part 2 - Analysing messages (10 marks)

- Correctly working and uses the right message (i.e. the readable version) and provides the correct statistics. (2 marks each)
  - Total number of words.
  - Number of unique words.
  - Minimum, maximum, and average word length.
  - Most common letter. If there are two letters which have the same frequency, accept either of them or both of them as valid.
  - (Up to) The 10 most common words and their frequency, e.g., the : 4. (N.B. The output stats should be formatted, not just a printing of a data structure like [“the”, 4] or (“the”, 4).) We are checking here that the output includes the words which clearly have the highest frequencies, that all the words and frequency pairings are correct and that they only print ten words. If there are words which have the same frequency but including all of them will surpass the limit of ten, then the output is accepted for any subset of those words.
- Global penalty of one mark for incorrect message order.
- N.B. Do not re-penalise for encryption/decryption missing.

Part 3 - File Reading (5 marks)

- Program can read in messages using a valid file name. (2 marks)
- Correctly working user input for file usage. (1 mark)
- Robustness to incorrect file names. (2 marks)

Part 4 - Automated decryption (20 marks)

- Correctly working automated decryption.
- Can give partial marks for attempting, using the common words file, getting auto-decryption of some lines, etc.

Part 5 - Enhancements (10 marks)

Look for these specified in the code with `#!#`.

- Fully working extra feature (5 marks each up to two)

General Implementation (30 marks)

- Use of CamelCase (only give 1 mark if they've consistently used another convention such as underscores) and sensible variable names (deduct half a mark here if constants not capitalized). (5 marks - 2.5 each)
- Using the right data structures, e.g. sets for unique words, dictionary for word counts, or some other suitable structure. (5 marks)
- Useful and sensible comments in the code. (5 marks)
- Efficient/taut code implementation. Avoids repeating/redundant code as much as possible, e.g., statistics calculated right at the end, once, or moved to functions for reuse. Also award marks here for code structure. (5 marks)
- Correct usage of functions, i.e. use of arguments and return statements. (5 marks)
- Extra points for an elegant implementation, e.g. using the collections module for the most common words or using `string.ascii_uppercase` to get the alphabet. (5 marks)

Report (10 marks)

The report is here to help us to see if they understand their code and to check for plagiarism:

- Analysis – what worked, what could have changed.
- Marks should be penalised if they are significantly over/under the page limit (< half a page, more than 2 when ignoring 'waffle')
- Waffle should not be given any marks – for example: 'I really enjoyed writing the code...', 'I worked a lot on the code....'
- They should provide an example of a design decision.
- They should reference any external code or ideas that they may have used, marks may be penalised if they do not. Check for this in the comments as well. If they are using, for example, ASCII functions (ord and char) and there's no explanation, then remove points.
- Use the report to check for plagiarism!

Feedback

It is crucial to give useful feedback to the students. This feedback should help them to significantly improve their code for the second assignment. Here are couple of points that to consider:

- Start with positives about the project, such as the progress they made or highlight anything they did well.
- **Be constructive in your criticism.** This feedback should be mostly based on what is taught; limit your expectations a little, but provide additional suggestions if you think they would help.
  E.g. 'I would strongly caution against using global variables in functions, as their use does not scale well for larger projects as it means you can overwrite important variables. Instead a better approach would be to return the variables from functions.'
- Avoid using **you** when talking about negatives, only positives.
- Comment on overall structure.
- Is the code readable? What could be improved.
- Where did they score extra marks, or lose marks and why?