Python Marking Scheme 2018

Total amount of points is 100. This project is weighted 20% of the overall mark. So, this is not a huge amount, but the feedback will be very useful for them.
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 (email and highlight in the online Excel sheet)

Also, when you see plagiarism please let me know (email and highlight in the online Excel sheet)

Part 1 – Basic Implementation (total 20)

Fundamental Implementation (10)

This includes implementation of encryption and decryption

- Fully working encryption counts also if input is provided by user only (i.e., no parameters are implemented)

Using input parameters (10)

- Fully working implementation with all input parameters (8) - 2 each
- --[encrypt/decrypt] [rotation]
    - --[encrypt/decrypt] [rotation] –-file [filename]
    - --decrypt
- --decrypt –-file [filename]
- Properly working file reading (2)

Part 2 – Analysing message (Total 10)

- Fully working code (provides the right statistics)
- Each is 2 points:
    - Total number of words;
    - Number of unique words;
    - Minimum, maximum, and average word length;
    - Most common letter;
    - (Up to) The 10 most common words and their frequency, e.g., **the : 4**
      The output stats should be formatted, not just a printing of a data structure like [“the”, 4] or (“the”, 4).

Part 3 – Automated decryption (Total 20)

- Fully working automated decryption

Part 4 – Enhancements (10)

- Full 10 points for implementation of 2 or more extra features
- 5 Points for one extra feature
- Look for these specified in the report or in comments starting with #!#

General Implementation (30)

- CamelCase and sensible variable names (5) – 2.5 each
- Using the right data structures (5)
    - E.g., Sets for unique words, dictionary for word counts, or some other suitable structure
- Useful and sensible comments in the code (5)
- Extra points for an elegant implementation (5)
- Efficient/taut code implementation (5)
    - Avoids repeating/redundant code as much as possible, e.g., statistics calculated right at the end, once, or moved to functions for reuse
- Robust implementation, i.e., catches wrong input from users (5)
    - Assignment specified "If a file by the provided name cannot be found, then print an error and stop the program.", as an example
    - Negative rotations should also be checked for

Report (Total 10)

The report is here to help us to see if they really understood what they were doing 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 their 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 significantly to write excellent code for their second submission. Here are couple of points that you should consider:

- Start with positives about the project if you can, such as the progress they made
- Be constructive in your criticism, e.g., the code could have been written more tautly by using a function for reading data from a file
    - 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
- Comment on overall structure
- Is the code readable? What could be improved
- Where did they score extra marks, or lose marks and why?