Anita Pal

Faculty of Arts and Humanities
Linnaeus University
ap224pu@student.lnu.se

Introduction

The following Python implementation, hosted on GitHub[1] and Google Collab[2], contains functions for simple string manipulation, such as counting words in a given text, determining the number of sentences, and sorting sentences or words by length.

Problem Definition

As per the assignment's guidelines, the implementation above uses built-in Python methods (e.g. *split*) and regular expressions. The rationale for using a regular expression is that it enables a developer to match text input against specific rules (Google). The reasons for using Python methods include, but are not limited to, their efficiency, which does not require any imports or additional code (Obregon, 2023).

One helpful method that the implementation (i.e., line 44) uses is the *f-string*, which evaluates expressions with curly braces at runtime before formatting and embedding them in another *string* (DataCamp). Another interesting Python method is *sorted*, which sorts a list and returns a new one; it is useful when a new sorted list is the desired output (Yi, 2020).

In terms of error handling, the implementation uses *if-else* statements to inform the user that they have chosen to pass an empty string as an argument, for example. According to Andreas Kagoshima (2023), this type of error handling falls under the "Look Before You Leap" (if-else) pattern. This type is concerned with handling known conditions and simple checks.

Lastly, the implementation aims to follow the *DRY* principle—Do Not Repeat Yourself—as much as possible, not only to enable code reuse but also to reduce errors, as maintainability is easier when code is reused (Alfonso, 2023).

Methodology Approach

The first function, *count_words_in_sentence* (GeeksforGeeks), takes in a sentence and, before it carries out any operations on it, checks that it does not contain a length of anything less than zero. If the text's length is greater than 0, the function uses Python's built-in *split* function to split that sentence into a list of items (W3Schools).

Consequently, using the *len* method within that same line (i.e. line 42), the total number of items in said list can be retrieved (cf. Real Python). To print the result as a string, the word count is converted to one, allowing it to be incorporated into another string using f-string formatting (cf. DataCamp). An alternative function returns the result as an integer (i.e., *return_word_count*). It helps with later text manipulations.

The second function, *count_number_of_sentences*, takes a text input and uses a regular expression to find all full stops within a sentence (Tyler, 2015); this allows the retrieval of all sentences within that text, which are then returned as an integer containing the count. Similarly to *count_words_in_sentence*, the result is converted to a string before being incorporated into the output using an f-string (cf. DataCamp). As before, a helper function *(count_sentences_and_return_int*) returns an integer and serves a similar role in later text manipulations.

The function *get_word_count_of_each_sentence_in_a_text* first splits a given text via the full stop punctuation marker (i.e. line 81). Then, after initialising an empty list that will store each sentence as a list item (i.e. line 82), the *return_word_count* helper function is used within a for loop to retrieve the number of each sentence. If the value equals the equivalent of null[3], the result is added to the number_count list (i.e., line 86).

---

[1] The lines used throughout this report refer to the py file within that repository.
[2] The author is very delete-happy; therefore, the decision was made to host the code on GitHub as well, for future reuse purposes.
[3] https://www.w3schools.com/python/ref_keyword_none.asp

The *sorted_number_of_words_in_each_sentence* stores an empty integer and an empty list (i.e. lines 91 and 92) to append the values of a stored list that was passed into the function as an argument into the empty one, according to their position (e.g. the second position requires some different grammatical phrasings). Doing this enables printing outputs in a given string format (see *determine_length_of_sentences*).

Both previously described functions serve as the building blocks of the *determine_length_of_sentences* function, which calculates the word count for each sentence (i.e., line 102) of a given text input before storing it in a list. The said list is then sorted using Python's *sorted* method (cf. Yi, 2020) before being passed to the *sorted_number_of_words_in_each_sentence* function, which creates a list ready for printing (i.e., line 104). However, to get the full output, a for loop must be engaged to print out the word length of each sentence, line by line (i.e, line 106).

The *get_five_longest_words* function takes a text argument and cleans up the text using a regex (Dudhediya) before splitting it (i.e., lines 117-118). It then returns a sorted list containing the five largest words using list slicing to extract the desired items (Hunner, 2024). The *print_out_five_longest_words* function shares some similarities with the *sorted_number_of_words_in_each_sentence*, in that it enables an output of items formatted in a certain way (i.e., lines 104-106). However, unlike that function, it takes a text as input and uses the *get_five_longest_words* function to get that sorted list.

Analysis of Results

According to the tests carried out during and after the development process (i.e., lines 131-143), the code works, but remains plagued by limitations related to regexes not considering all possible scenarios (i.e. line 64) and some repetitive code (lines 93-99 and lines 124-129) that could have been consolidated into their own method, thus more closely aligning with the DRY principle (cf. Alfonso, 2023).

The duplication of the *count_words_in_sentence* and *count_words_in_sentence* functions to retrieve integers rather than strings is awkward; however, the alternative would have been even less elegant, as an *if/else* statement would have been required to handle cases where an integer was not needed, which might have created a large number of required arguments to pass into these functions.

Alternatively, the string outputs could have been handled outside of the functions, but this felt even less beneficial. In many ways, the awkwardness arose from the author's effort to follow the assignment guidelines as closely as possible.

Conclusions and Reflections

This assignment was a good exercise that got me thinking about how to write code more efficiently, especially since there are plenty of patterns and approaches for writing code more cleanly and efficiently (cf. Rahman, 2023). However, due to the time constraints, it was not possible to explore these options for this particular assignment.

(991 words).

AI Acknowledgements

The report did not use AI beyond quick sanity-checking of grammar and/or double-checking the references. The code was written by me, with credit being given where it is due via comments.

References

Alfonso, N. (2023, October 30). *DRY vs WET in Python: Navigating the code seas with clarity and efficiency*. Medium. https://medium.com/@Nelsonalfonso/dry-vs-wet-in-python-navigating-the-code-seas-with-clarity-and-efficiency-01ad672a820d

Datacamp. (n.d.). *Python f-string tutorial: Simplify string formatting in Python*. DataCamp. https://www.datacamp.com/tutorial/python-f-string

Dudhediya, B. (n.d.). *Answer to "Extract five longest words in the entire text, create a list with those items and order them alphabetically"* [Forum post]. Stack Overflow. https://stackoverflow.com/a (Retrieved November 23, 2025, under CC BY-SA 4.0 license)

GeeksforGeeks. (n.d.). *Python program to count words in a sentence*. GeeksforGeeks. https://www.geeksforgeeks.org/python/python-program-to-count-words-in-a-sentence

Google. (n.d.). *Regular expressions in Python*. Google for Developers. https://developers.google.com/edu/python/regular-expressions

Hunner, T. (2024, March 8). *List slicing in Python*. Python Morsels. https://www.pythonmorsels.com/slicing

Kagoshima, A. (2023, July 18). *Try-except handling vs if-else handling in Python*. Medium. https://medium.com/@a.kago1988/try-except-handling-vs-if-else-handling-in-python-d3a3e03ff008

Obregón, A. (2023, August 7). *Understanding Python's most common built-in functions*. Medium. https://medium.com/@AlexanderObregon/understanding-pythons-most-common-built-in-functions-f535b83ea397

Rahman, A. (2023, July 10). *10 Python patterns that will instantly level up your code*. Medium. https://medium.com/@abdur.rahman12/10-python-patterns-that-will-instantly-level-up-your-code-f329dea91f0f

Real Python. (n.d.). *Python len() function*. Real Python. https://realpython.com/len-python-function

Tyler. (2015, March 15). *Answer to "Count number of sentences in paragraph."* Stack Overflow. https://stackoverflow.com/a/29166831 (Retrieved November 23, 2025, under CC BY-SA 3.0 license)

W3Schools. (n.d.). *Python string split() method*. W3Schools. https://www.w3schools.com/python/ref_string_split.asp

Yi, T. (2020, September 14). *What is the difference between list.sort() and sorted() in Python?* DEV Community. https://dev.to/trinityyi/what-is-the-difference-between-listsort-and-sorted-in-python-5a6g