INFS2044 Assignment 2 Case Study

In this assignment you will be developing a system for extracting information from text files. The system will process text files, compute statistics about each file, and produce output in several different formats. Statistics computed by the system include the length of the file in words and the most frequent words in each file and their frequency.

Use Cases

The system supports a single use case:

UC1 Compute Summary Statistics:
The user specifies the files to be processed using a command line application, specifies the number of most frequent words to be identified, and specifies one or more output file names. The application reads each file, computes the summary statistics, and writes the result to the given output file(s) in the appropriate output format(s).

Summary statistics to be generated in this use case:

- Number of words in each file
- Most frequent N words in each file

The output format for each output file is determined by the file extension of each output file. Extensions and output files to be supported in this use case:

- .txt: Plain Text format.
- .csv: Comma-separated format.

The specification of these file formats is given in section “File Formats” later in this document.

Future variations:

- Additional output formats could be introduced.
- Additional summary statistics could be introduced.
- Filters for text processing could be introduced (such as different ways of identifying words in a file, ignoring selected words, etc)
- The functions may be eventually offered via a REST API in addition to the console application.

These variations are not in scope for your implementation in this assignment, but your design must be able to accommodate these extensions.

Example Command

The following command would find the top 10 most frequent words in each of the files a.txt, b.txt, and c.txt, and output the results in two formats: CSV format in out1.csv, and text format in out2.txt: (type the command all on one line)

```
$ python word_statistics_app.py --number=10 --output=out1.csv
--output=out2.txt a.txt b.txt c.txt
```

If the number of frequent words specified on the command line exceeds the total number of unique words in a file, then output the actual number of unique words and their frequencies.

File Formats

Text format (extension .txt):

Each line shows a short text containing the name of the file, the total length of the file in words, followed by the most frequent words and their frequencies in the file (in order of descending frequency; if there are multiple words with identical frequency, show them in ascending alphabetic order).

Example:

Suppose that file a.txt has 47 words in total (some of which may be occurrences of the same word), and that the most frequent words in that file are “the” (frequency 10), “cat” (frequency 8), “a” (frequency 8), apple (frequency 4).

The corresponding line in the output file would be (all on one line):

```
File a.txt contains 47 words. Frequent words are: the (10), a
(8), cat (8), apple (4).
```

CSV format (extension .csv):

The information shown is the same as for the text format, except that the file name and statistics are delimited by commas.

For the above example, the row in the file would be:

```
a.txt,47,the,10,a,8,cat,8,apple,4.
```

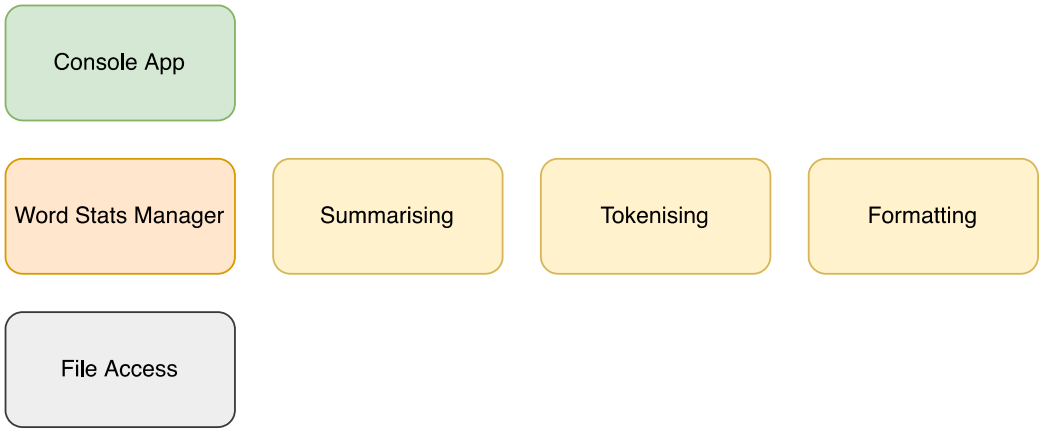
You can assume that the file name does not contain commas and quotation marks.

Input File Format

All input files are plain text files. Each file may contain one or more lines of text. Words are delimited by one or more whitespaces (that is, space, tab, or newline characters).

Decomposition

You must use the following component decomposition as the basis for your implementation design:



The responsibilities of the elements are as follows:

Elements	**Responsibilities**
Console App	Interact with the user (acquire user options)
Word Stats Manager	Orchestrates the use case process (reading, tokenising, summarising, formatting, outputting)
Summarising Engine	Computes the summary statistics
Tokenising Engine	Splits the input text into tokens (words)
Formatting Engine	Generates output from summaries
File Access	Interacts with the file system to read & write files

Scope

Your implementation must respect the boundaries defined by the decomposition and include classes for each of the elements in this decomposition.

The implementation must:

- run on python 3.10, and
- use only the python 3.10 standard libraries and the ‘click’ package (for the command line application),
- correctly implement the functions described in this document, and
- it must function correctly with any given plain text file (you can assume that the entire content of this file fits into main memory), and
- it must include a comprehensive unit test suite using pytest.

Focus your attention on the *quality* of your code.

It is not sufficient to merely create a program that is functionally correct to pass this assignment. The emphasis is on creating a well-structured, modular, object-oriented design that satisfies design principles and coding practices discussed in this course.

To give an indication about the amount of code that you will write: a comprehensive and largely self-documenting implementation of the system described here can be achieved in less than 200 lines of python code (including whitespace and the code that is given to you). This number does not include the code for the unit tests.