Home Work #1

Write a program in Python 3 (3.9 recommended) without using any libraries to analyze specified portions of a large text file so as to respond to a potentially unending sequence of queries. The text file is known to be so large that it is not feasible to load its contents into memory. However, the number of unique words in the file is manageable.

§1. The text consists of printable ASCII characters with a newline character (`\n`) ending each line and either a space or a newline ending each word. (No word spans multiple lines.) For simplicity, you can ignore punctuation and case. (So, Ph.D. is the same as phd; i.e. is the same as ie, etc.) Numbers must be treated as words. (So, 123 and 123. are equivalent to the word 123; 123.45, and 1.2345E02 are equivalent to 123.45.) Paragraphs are separated by one or more empty lines (`\n\n`, or `\n\n\n`, or ...). Paragraphs are numbered 0, 1, 2, ...

§2. First, the user executes the function `main()` to start the program and, in response to a prompt, enters the name of the input text file.

§3. Next, the user is prompted for, and enters in response, paragraph numbers $k_1, \ldots, k_M$ in one or more lines separated by spaces and terminated by an empty line. (No number spans multiple lines.)

§4. Henceforth, paragraph numbers will not be accepted. Instead, your program should then enter into a loop accepting word or paragraph queries until it receives a termination command.

§5. Word queries are inputs of the form
`$ w`
in a single line where `w` is the word being queried. Your program should respond by printing the number of times that word `w` has been queried (including the current query), and if the word occurs in one of the given paragraphs $< k_1, \ldots, k_M >$, then the paragraph number in which it occurs along with the offset within that para (the first word in the para has zero offset, the second has offset one, etc.)

§6. Paragraph queries are inputs of the form
`# j`
in a single line where `j` is a non-negative integer representing a paragraph number. Your program should print out for each of the M specified paragraphs:
$n,\ \ 1 : w_1,\ \ n : w_n,\ \ r : w_r,$
where n is the number of words in the para, w_i is the i^{th} word in the para, and r is a random integer between 1 and n exclusive.

§7. A termination command is an input of the form
`/`
in a single line; in response, your program should terminate.

§8. Develop the following functions. Please do not hesitate to deploy additional functions and data structures.

8.1. `gen_read_para(filename)`: a generator function designed to return the content of the *next* paragraph from the file specified in its input. Its output must be a list of words and `None` if the end of file has been reached.

8.2. `get_words_in_para(para_stream, para_num)` will return a list of words in the specified paragraph in their order of appearance. It will call `next()` on the iterator `para_stream` (that gets created by invoking `gen_read_para()`).

8.3. `get_words_in_all_paras (para_stream, para_num_list)` will return all the words in all the paragraphs in the input list.

8.4. `analyze_paras(para_stream, para_num_list)` will create a global dictionary named DICTIONARY in which every word w in `para_num_list` will appear as a key; its value will be a dictionary with key p = paragraph number in which w appears and value $[\ldots s \ldots]$, a list of possibly several offsets of w in paragraph p.

8.5. `analyze(file_name, para_num_list)` will initiate the analysis and fill the global dictionary / dictionaries.

8.6. `main()` will prompt for and obtain the name of the text file as a character string in a single line; and starts the rest of the processing.

§9. The input text file is presumed to be in the current directory.

§10. The input text file should be traversed at most once and not overwritten.

§11. The paragraph list may contain duplicates.

§12. Your program should reject illegal queries.

§13. You may not make use of any libraries except `random`.

You can use string functions / methods like `strip(),lower()`.

§14. Your program must be readable and understandable with appropriate decomposition and documentation. Constants should be conspicuous, e.g., a 'Const_' prefix.

The program should start with your name and outline the goal of the program and what it achieves (and list requirements that are not fully operational).

Each function should start with a docstring describing it and its inputs and be robust.

§15. Submit a Jupyter Notebook whose cells I can run in order. (Control the number of cells.)

Its last cell should contain a call to `main()`.