Lab Test 2 - Specification

Lab Test 2 - 15/03/2022
Class Group: TU259

Worth: 10% of the overall mark for the module	**Deadline:** 22/03/2022, 2pm. No late submission accepted.

Home assignment

This is a home assignment, but you **cannot use the internet to find solutions** for similar problems or reuse someone else's code.

Submission

Please submit your solution (Python file and report) via Brightspace. It is your responsibility to make sure you've uploaded the correct file as changes after the test has finished won't be accepted.

Marking Scheme

- Questions will receive full marks if the output is correct and it has appropriate layout and use of comments.
- Indentation and white space have to be used appropriately.
- Code should be easy to read, and naming conventions should be adopted.
- Ensure that variable names are informative while appropriate and meaningful comments are placed.
- The code should work without issue when run on the lecturer's machine.

Plagiarism

Plagiarism will result in a zero mark (0%). You should make yourself familiar with the plagiarism policy of Technological University Dublin. **You might be contacted by the lecturer to demo your code if the submission is believed to contain suspected plagiarism.**

Problem Description

Document retrieval is the task of finding documents that meet the search criteria input by a user. The most well-known example is web search, where a user types in a set of key words and the search engine finds web pages that are relevant to their search query. True document retrieval can be quite difficult, as it needs to consider many different factors. In this project you will implement a very simple document retrieval engine.

Program Specifications

The file ap_docs.txt contains several old newswire articles. We will use this as our document collection; when a user gives us a set of keywords, we will find the documents in this collection that match their search terms. Each article in the collection is separated by a line that contains only the "<NEW DOCUMENT>" token.

Your program will read in the documents from the file and number each document starting with 1 (the first document in the file is document 1, the second is document 2, etc.). In order to look up search terms, we will need to know which words appear in each document. We will use a dictionary for this purpose. *Each entry in your dictionary should have a word as the key and the word's value as the set of document numbers that this word appears in.* This arrangement allows you to look up a keyword in the dictionary and immediately get all the documents that it appears in, making it easy to figure out documents that might meet a search query.

Once your program has read the file, it will prompt the user to do one of three things: 1) search for documents that match the search words input by the user, 2) display a document, or 3) quit the program. If the user chooses to search, your program should prompt for a string of search words and find documents that contain *all* those keywords. It will then print out the document number of all the relevant documents. If no documents in the collection contain every keyword input by the user, your program should print a message that says that no relevant documents were found. If the user chooses to display a document, your program should prompt for a document number and print out the entire document that corresponds to that number. Your program should continue to prompt until the user chooses to quit.

Example output:

```
What would you like to do?
1. Search for documents
2. Read Document
3. Quit Program
>1
Enter search words: stock prices

Documents fitting search:
17 3 222 223

What would you like to do?
```

1. Search for documents
2. Read Document
3. Quit Program
>**2**
Enter document number:**17**
Document #17

Stock prices declined sharply for the third
straight day in Tokyo Saturday, following an overnight tumble on Wall Street.
On the Tokyo Stock Exchange, the Nikkei Average of 225 selected issues, which lost 154.57 points Friday, shed 305.99 more points, or 1.2 percent, during Saturday's half-day session to finish the week at 25,320.72.
``Prices were down almost across the board prompted by profit-taking selling,'' said Hiromi Yoneyama of Wako Securities.
He said investors sidestepped the market, following the second straight tumble on Wall Street, and before the close of Japan's 1987 fiscal year, which ends on March 31.
Yoneyama, however, added investors appeared to be optimistic about the prospects for the new fiscal year.
In New York Friday, the Dow Jones average of 30 industrials fell 44.92 points to 1,978.95 at the close of the market's worst week this year.
On the first section in Tokyo trading, major losers included large-capital issues, such as steels, heavy electricals and shipbuildings. A light 400 million shares were traded during the session.
The foreign exchange market is closed on Saturdays.

What would you like to do?
1. Search for documents
2. Read Document
3. Quit Program
>**3**

Process finished with exit code 0

Assignment Notes

1. You will likely need to store two types of data in two different data structures. In one, you will have a dictionary that stores single **words as the key** with **the value as the set of documents** (numbers) that the word appears in. For example, this dictionary with two entries show the structure you should have:

```
D = {"stock":{224,17,3,138,222,223},
"prices":{17,226,3,222,46,223}}
```

In the other data structure, you will store the actual text of the documents, so that you can display it for the user when they ask. You can use a list or a dictionary for the second data structure. For example:

```
L = ["first document text", "second document text", ..., "last
document text"]
```

If you try to print `L[16]` it should show the text contained in document #17 listed above.

2. Search queries should not be case sensitive, i.e. searching for `"Stocks"` should give all documents that contain `"stocks"`, `"STOCKS"`, etc.
3. You should remove punctuation from the start and end of a word as well. If the string `"stock,"` appears in the document, this should be counted as an instance of the word `"stock"` (without the comma). You might find the string module's `string.punctuation` useful.
4. Don't forget to convert strings to numbers (and vice versa) where appropriate.
5. You may find sets to be useful. You can add elements to a set with the `add` method. You can also make a set out of an existing list. If you want to find the values that are common between two sets, you can use set intersection. Think how you can use that to keep a list of non-duplicated documents in which a word appears, or to find the list of common documents between multiple words.

```
>>> set1 = set()   # initialize to an empty set
>>> set1.add(2)
>>> set1.add(3)
>>> set1
{2, 3}
>>> my_lst = [3,3,1,3,5]
>>> set2 = set(my_lst)
>>> set2
{1, 3, 5}
>>> set3 = set1 & set2   # intersection of set1 and set2
>>> set3
{3}
```

6. Begin small. A tiny test file for development: `ap_docs2.txt` is provided. This file has three two-line documents.
7. Develop in pieces.
    a. Read and print the whole file - just to do something to get started.
    b. Read the file, divide the file into documents, and put the documents in a list. Simply make a string out of each document. The main idea here is to collect lines of a file into a string until you encounter the token that indicates the start of the next document. At that point, append your string to your list and start the next document with an empty string.
    c. Make step (b) into a function that returns the list.
    d. Starting with an empty dictionary, go through each document in your list one word at a time. If a word is in your dictionary, add the current document number to the word's set. If a word isn't yet in the dictionary, add it to the dictionary with its value as a set with the current document number as its only element.

e. Make step (d) into a function that returns the dictionary.
f. Prompt for one search word and use your dictionary to find the documents (numbers) that contain that word.
g. Next enter two words at the prompt, use the dictionary to find each word's documents, and then use the set intersection operation to find the common documents.
h. Make step (g) a function that returns a set of documents. Also, make it general enough to handle any number of search words.
i. Now work on the main part of the program that is a loop that prompts for user input.
j. Thoroughly test on `ap_docs2.txt`, and then test on the full `ap_docs.txt` file

Marking Scheme

Correct functionality (65%)

- Ability to handle files and deal with exception handling.
- Functions/methods are used correctly to break down the problem. They are all used for a single purpose or for a single task.
- Nice use of data structures such as dictionaries and sets.
- Command line menu is well implemented, and all its options work as expected

Layout and use of comments (20%)

Indentation and white space have been used appropriately. Code is easy to read, and naming conventions have been adopted. Code is well documented, and it is easy to understand.

Report (15%)

Understanding of the code will be evaluated based on a report written by you to explain your system. It should be no longer than 1-page. Please detail what you did, what were the difficulties, how did you solve it, which data structures have you used, etc. Add as many details as necessary. The goal is to demonstrate that you have a clear understanding of your submission.

Rubric

	Expectations Far Surpassed (100%)	Expectations Exceeded (75%)	Expectations Met (50%)	Expectations Not Met (25%)	Not Done (0%)
Functionality	All requirements are met. All deliverables included in submission without deviation from requirements.	Requirements met. Deliverables included without major and few minor omissions or deviations from requirements.	Basic requirements met. Deliverables included without major omissions or deviations from requirements	Basic requirements not met or deliverables deviate substantially from requirements	Not convincingly attempted
Layout and use of comments	Layout and comments are flawless. Docstrings have been used throughout the whole code. Naming conventions have been adopted. Comments are meaningful and make the reasoning more clear.	Layout and comments are well implemented. Docstrings used throughout most parts of the code.	Layout and comments are ok. The code is well organised and not difficult to understand.	Code is not well organised. The main scope is not linear and it is hard to follow. Few or no comments. The code is hard to understand and requires a lot of thinking to follow.	Not convincingly attempted
Report	Report is well written and organised. It follows the appropriate length without being repetitive. Difficulties and challenges are well described and related to the solution delivered. It demonstrated in-depth knowledge of the code submitted.	Report is well written and organised. It follows the appropriate length without being repetitive. Some difficulties and challenges are described. It demonstrated good knowledge of the code submitted.	Report follows the appropriate length without being too repetitive. Some difficulties and challenges are described. It demonstrated some knowledge of the code submitted but not fully convincing.	Report is incomplete. Significant parts are missing. It does not demonstrate clear knowledge of the code submitted. Student likely did not understand the requirements and/or was not able to implement them correctly.	Not convincingly attempted