Part 1: Overall Process & Decision Making

Initial thoughts:

Entering the project, I was expecting to have to deal with building a very complicated project from scratch, but that wasn't the case. After reading the instructions, I realized there was no need to overcomplicate this assignment as it is straightforward on the surface with subtle intricacies that can be easily overlooked.

My assumptions about the problem:

A number of these assumptions were made upon iterating over my solution

- Efficiency/optimization, while important, are not the main focus of the problem.
    - Why? Because the solution of iteration via nested for loops does not leave much room for optimization and will not result in performance issues at this scale.
    - I instead focused on an understandable implementation that considers and tackles edge cases.
- The search term will only ever be a single word without spaces.
    - Why? Because in the instructions, the search term is only referred to as a singular object and a term is by definition a word.
- The search term will never contain special characters unless they are part of the word being searched for.
    - Why? A term is by definition a word, and words only contain letters except special cases (apostrophes `'` and hyphens `-`)
        - Examples: (can't, Andrew's, co-founder, well-known)
- Numbers can be included in search term words.
    - Why? In special cases, numbers are used in words such as the names of inventions. Example: The "Freeze-Ray-9000"

Breaking down the steps of the problem:

After reading and understanding the assignment, I identified the steps needing to be completed:

1. Be able to iterate through each scanned text object (each word of each book).
    a. For each book, iterate through each line.
        i. For each line, iterate through each word.
            1. If the word matches the search term, save the line to output.
2. Add special cases

   Special cases to be added:

    a. Removing special characters that do not belong in words (i.e. not apostrophe or hyphens.)
        i. Characters to be removed: [. , / # ! $ % ^ & * ; : { } = ` ~ ( )]
    b. Removing hyphens if and only if they do not form compound words (such as "well-known")
    c. Handling word that got cut off of the line (such as darkness being cut off and separated into "dark-" and "ness" on the first line of the example inputObject.
3. <u>Write unit tests that test each special case</u>
    a. I wrote unit tests for each special case as I was implementing each case. To show examples of certain cases in action, I needed to add additional books with specific lines of text to the input object.

<u>My approach:</u>

1. Put into words, my initial solution was the simplest solution. To find a specific word in the book, I searched through each word in each page in each book for a group of letters/symbols that matched the search term. It is a complete single scan of the given texts using nested for loops. This approach was able to pass the provided unit tests. But to be able to iterate through every word in the given texts was only the first step.
2. Next, now that the framework for my solution was set up, I implemented special cases that I had identified and listed above under "Add special cases". I implemented new test cases after each new special case that I had implemented.

Part 2: Testing and Iteration

1. **Strategy for writing tests**

    I wanted to keep my tests simple and to the point, while not forgetting to test key details of the problem. As I wrote code for an edge case, I implemented a test case for the edge case and would not move on until the test was passed.

    If given more time, and if it were allowed, I would want to use external testing libraries or frameworks to write tests to make unit tests more readable and maintainable. I would also want to use more accurate names than "test#result", but I followed the naming convention given in example tests 1 and 2.

2. **What about the solution are you most proud of?**

I’m most proud of how I was able to identify edge cases that were not clearly stated or obvious as I iterated through my solution. Edge cases that I identified/tested for were:

- **Unit test 3:** Handle when results found
- **Unit test 4:** Be case sensitive in searching for the search term
- **Unit test 5:** Results only include if the exact search term is found, not if it is found as a sub-word
    - For example, “inside” would not be a match when searching for “in”
- **Unit test 6:** Searching for compound words correctly (words with hyphens)
- **Unit test 7:** Removing hyphens from words only if the hyphen is not used in a compound word.
    - For example, the ‘wait-’ in, “But wait- there’s more!”, would be treated as ‘wait’ instead of ‘wait-’
- **Unit test 8:** Handling (removing) other special characters found in texts

3. **Most difficult part of the problem**

    In my case, the most difficult part of this project was making the judgment calls on cases that the instructions did not specifically cover. For example, my decision to treat the search term as a single English word was based on the wording of the problem, but that is not explicitly stated.

    There are several cases that I could potentially address still, but I chose not to since I would require more clarifying information to handle them correctly. For example, when words get cut off on new lines, should that word that is then split into “dark-” and “ness” be treated as a single word, “darkness”? If so, would searching for “darkness” result in one or two lines that it is split across? Does it even make sense to identify a single word as being on multiple pages? These are the questions that arose in one case that I was unsure of how to handle and in a real situation, would ask clarifying questions on.

4. **If given more time…?**

If I was given more time to work on the problem and the ability to ask clarifying questions, I would want to address further very specific edge cases that I thought of and research common natural language processing techniques and standards. For example, given my implementation, the phrase “ad’v.e/ntur!e” would be read as “adventure” in my search implementation. Should this be considered correct?