My decision making towards completing this assessment began first by understanding all specifications surrounding the function at the center of the assessment. This entailed fully grasping the functionality of *findSearchTermInBooks()*, what its inputs and outputs were intended to be, and comprehending the different variations of the function's inputs and outputs. I then proceeded to write unit tests for the various scenarios that the function would potentially have to address before embarking on writing the function itself. By writing the tests before implementing the function, I found myself thinking more carefully about the design of the function and its additional considerations that I would have to make when addressing how the function would handle traversing through the JSON input and its properties. Since the function would always return a single JSON object containing the *SearchTerm* string, I addressed this point early in my code by assigning the value of the search term input to the corresponding value within the *result* variable. Next, I handled how I would traverse through each book object in the JSON input as well as how I would traverse through each book object's scanned text pieces. Finally, if I encountered a case where the *searchTerm* input did exist in a book's scanned text piece, I would proceed to updating the ISBN, page number and line number of the return variable with those three corresponding values from the JSON input so the return variable would reflect this finding.

My strategy for writing tests began with designing unit tests for the positive and negative scenarios outlined in the assessment's second task's instructions along with the scenario where case-sensitivity would have to be factored in. I also envisioned that the function would have to deal with the various forms that the JSON object may take,

such as when it contained no books and when it had multiple books. Doing so guided me towards writing code that addressed a variety of scenarios that the function would be required to handle. If given more time to work on this problem, I would seek to make the test suite more robust by creating tests to consider additional special characters, including those not used in the English language. The parts of my solution that I am proudest of are the unit tests that I created that consider a scenario in which there were no books to search through and a scenario where the JSON input contained books but those books had *Content* values without any text to examine. The part of the problem that was most difficult to solve was figuring out how to traverse through each JSON book object while also keeping track of the index of the current book object I was examining while using a for-each loop, since that type of loop typically does not require the tracking of indices. Fortunately, I was able to find out that Javascript allows for the tracking of this kind of information in its for-each loops unlike in a language such as Python, which allowed me to track both the book object's indices and the indices of each line of *Text* found in a specific JSON book object. The two primary edge cases that I addressed in my code were the scenarios where a JSON object without any books was passed as an input and the scenario where a JSON book object without any *Content* was passed as an input to the function. I tested each of these scenarios by creating one variable with nothing within the square brackets and by creating another variable which had two book titles and their ISBN numbers but that were missing values in their respective *Content*'s. If I were given more time to work on this problem, I would have also thought about edge cases where words contained a hyphen, such as "check-in" or "warm-up", since my code's current iteration splits the *Text* property of the

JSON book object on whitespace (‘ ‘) and I would have to incorporate logic that would detect individual words that make up a hyphenated term (such as “check” or “up”, in the examples I provided).