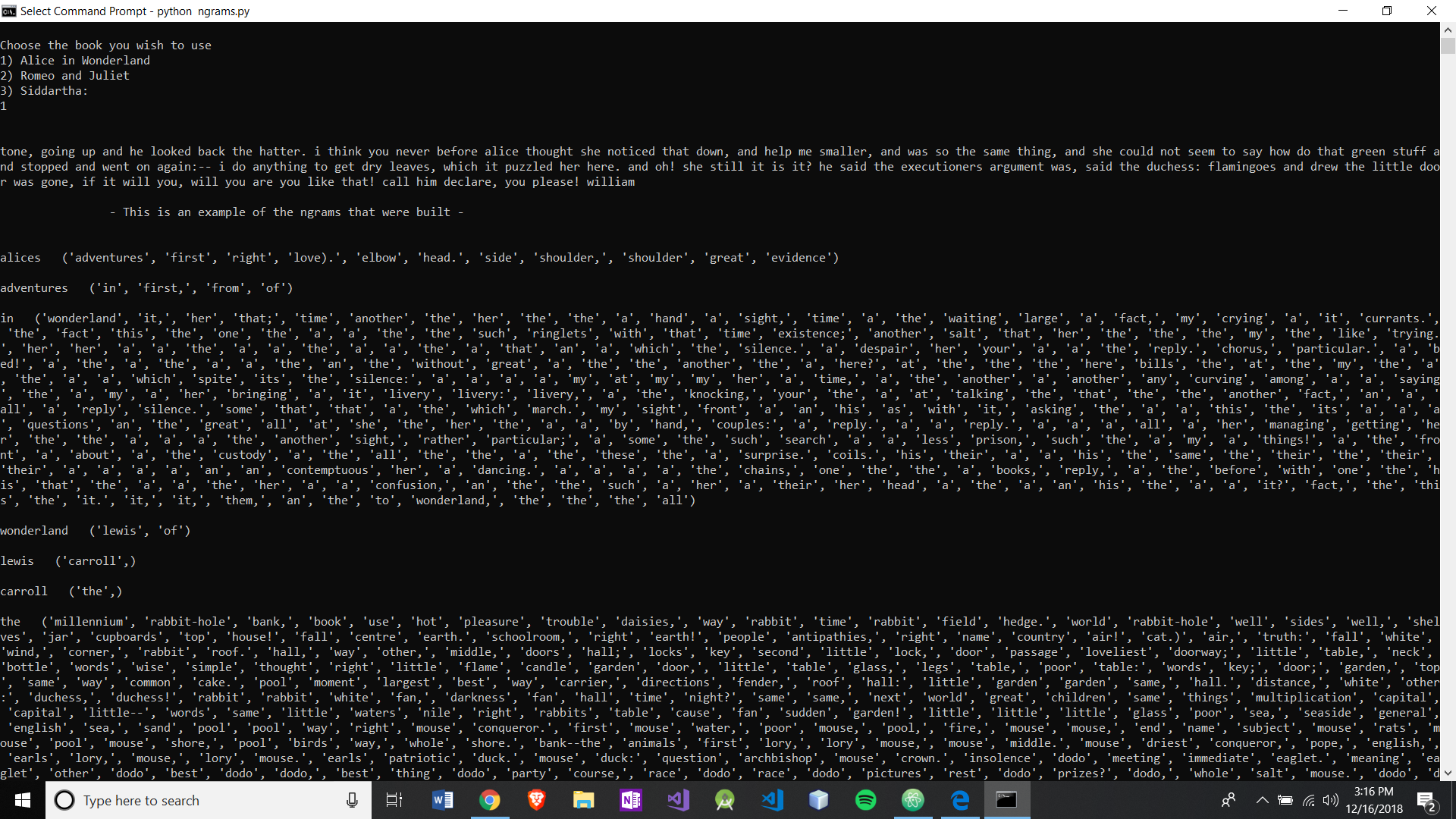
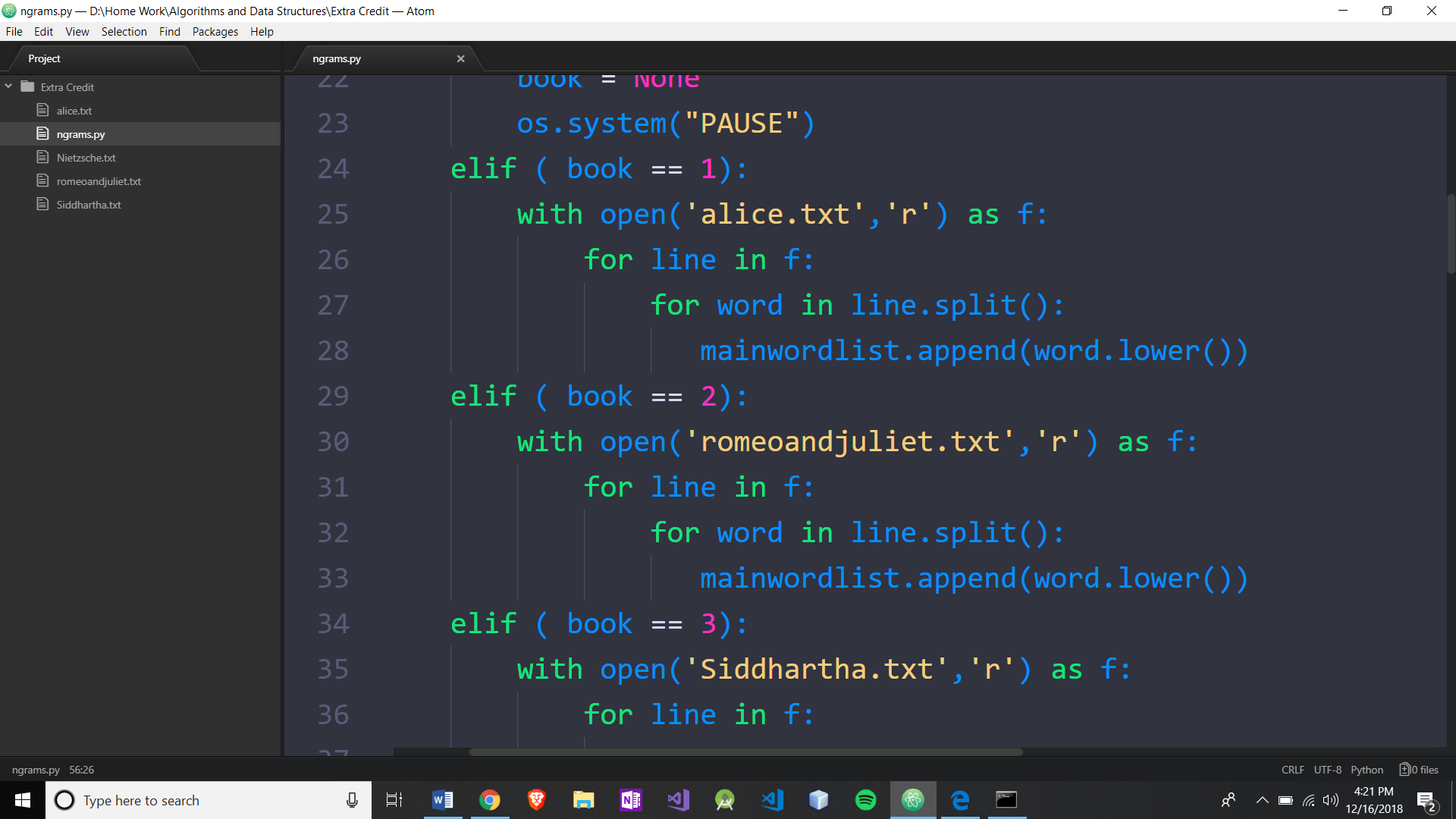
**Figure 1**



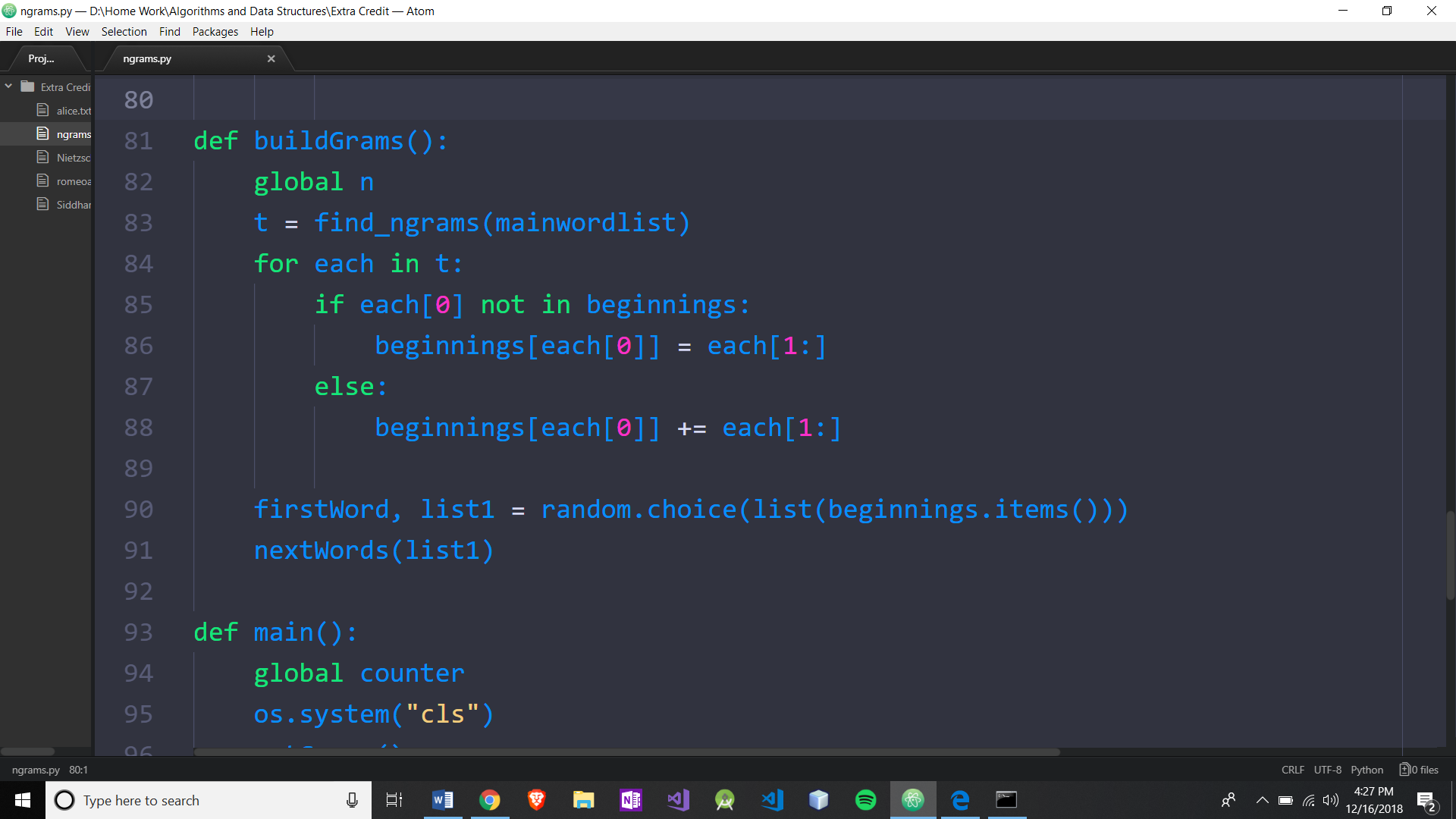
This is the output (Figure 1) when n = 2 with “Alice in Wonderland”. It uses a list it builds from all of the words in a selected text. This paragraph here lies within the most coherent paragraph space. Using this method when n = 2, the list of available next words is limited to words that occurred directly afterwards of the key value. The power of only considering the very next words is watered down you start including the next n terms, which is what occurs as n grows > 2.

**Figure 2**

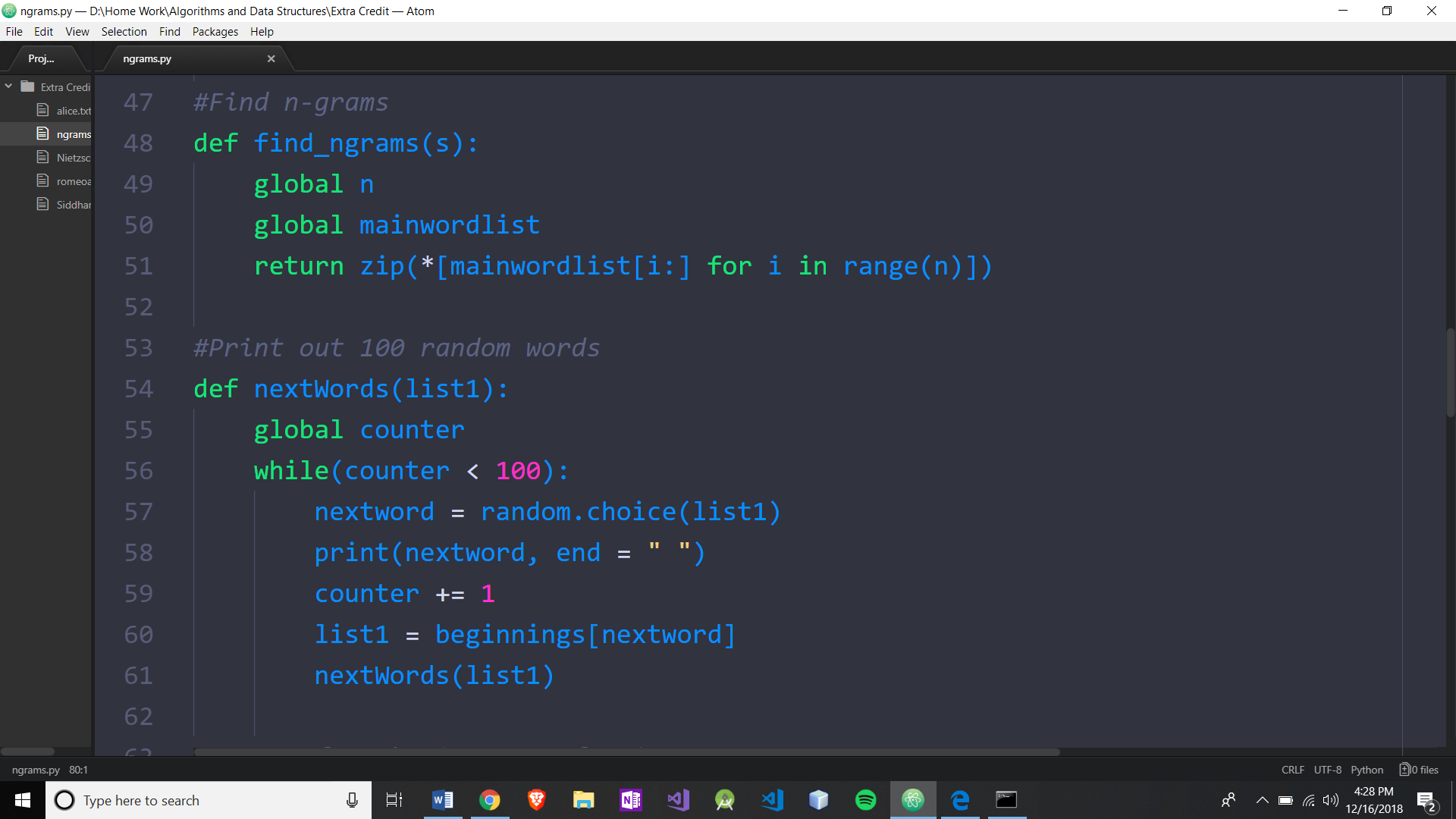


I argue that these two for loops (see Figure 2) are a list of all of the words that are come after it n-1 terms later. If the word is already in beginnings{} then just append words to the list.

**Figure 3**



It then uses the Python built in zip() function to create n-tuples from the text stream. This zip() goes through every word in the selected text and creates n-tuples. So, O(n). Which turns out to be the dominating

**Figure 4**

**Figure 5**

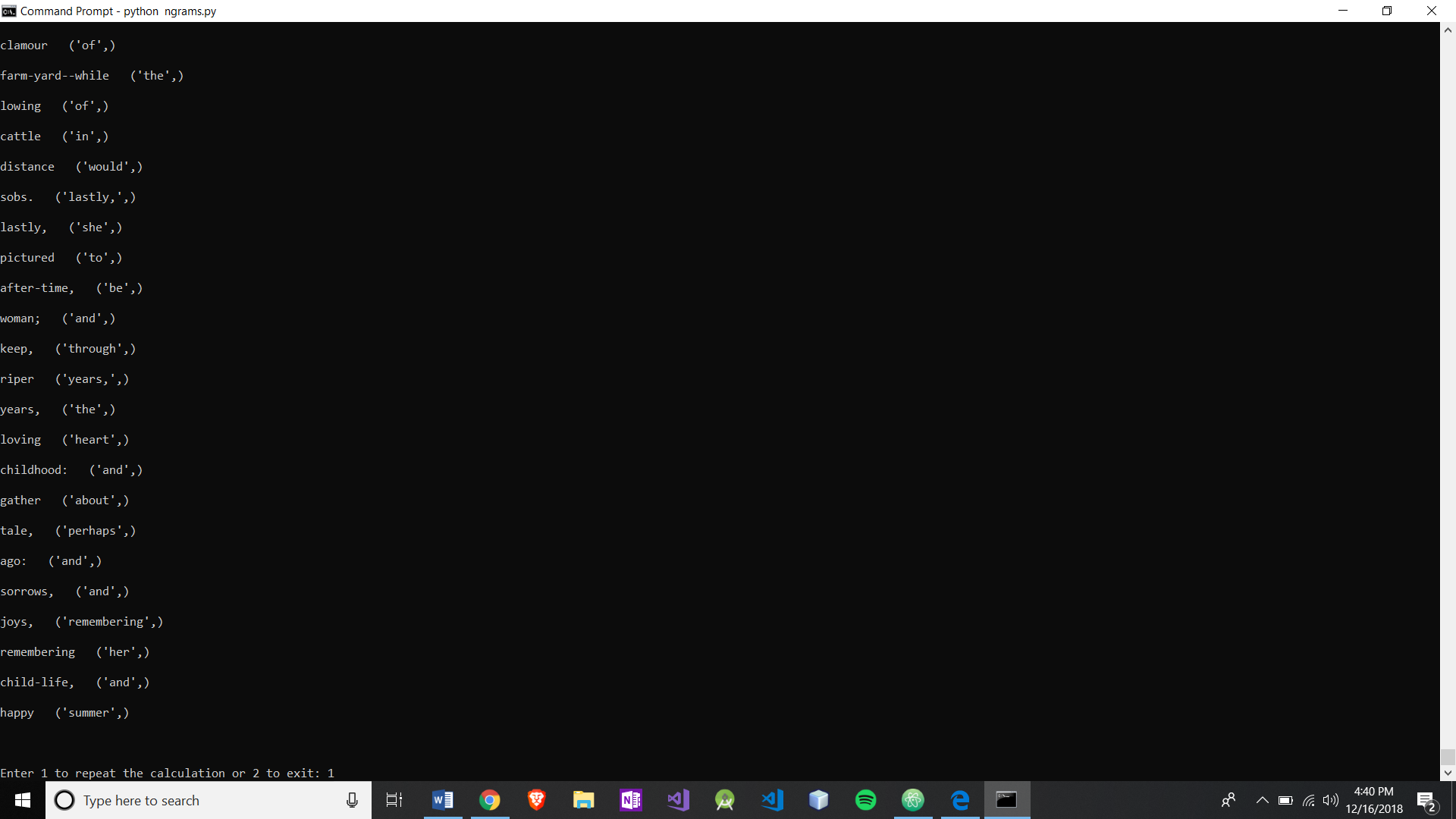
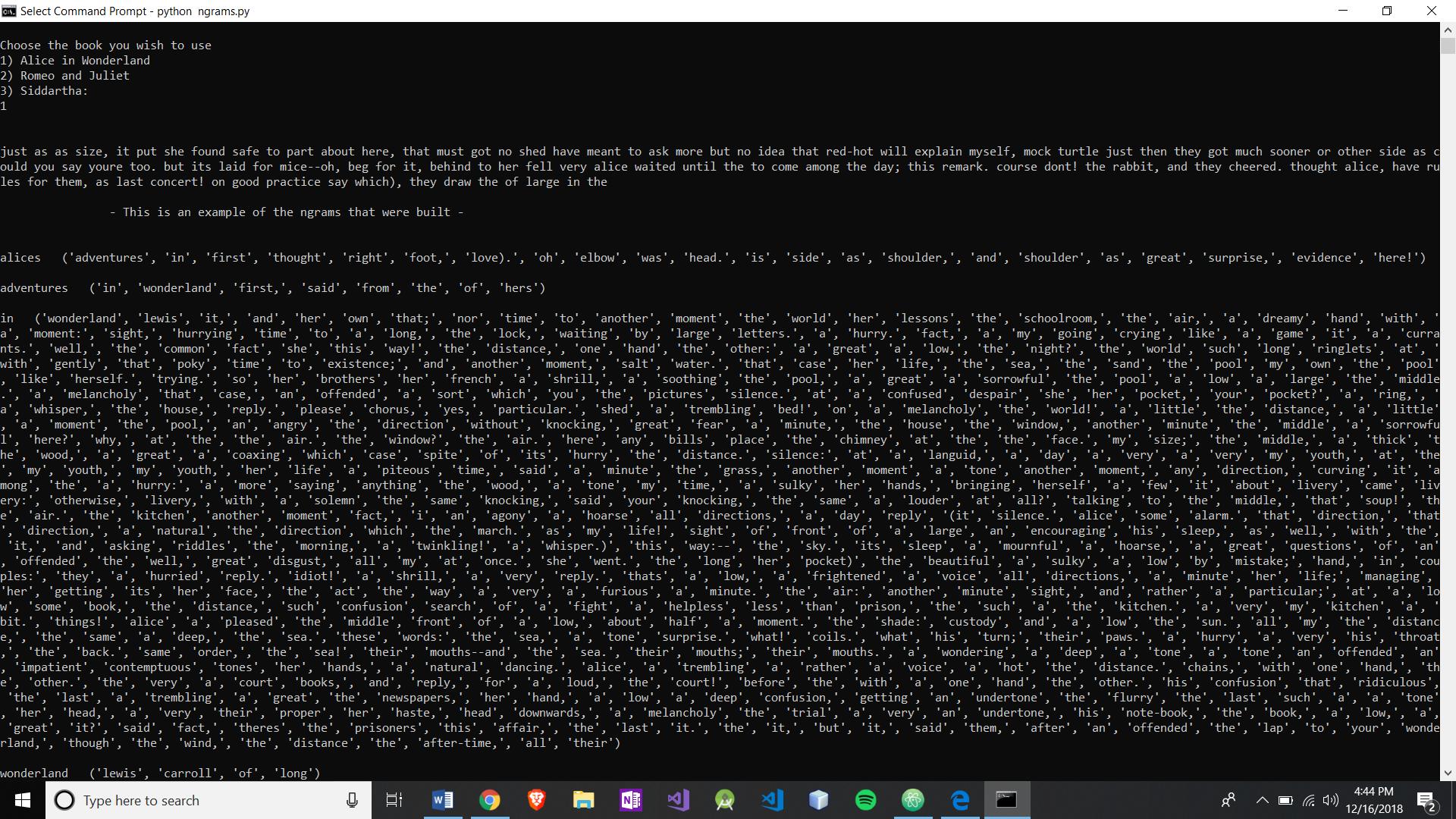


Figure 5 illustrates once again n-grams n = 2 and choosing “Alice in Wonderland”. These are end of the scanning process and therefore are the words with only one occurrence and therefore their list is limited to the very next word.

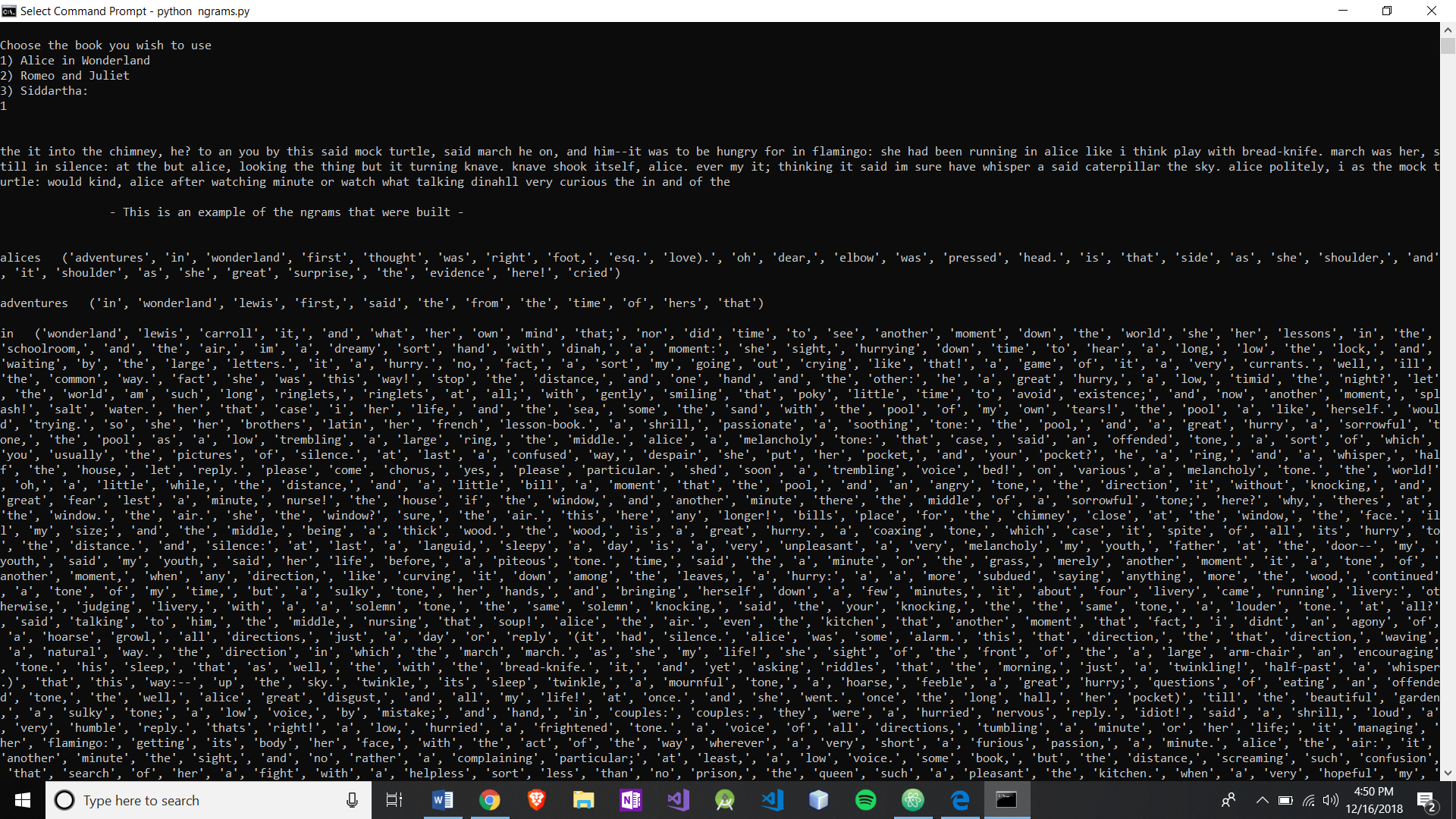
Now lets choose n = 3 and stick with “Alice in Wonderland”.

**Figure 6**



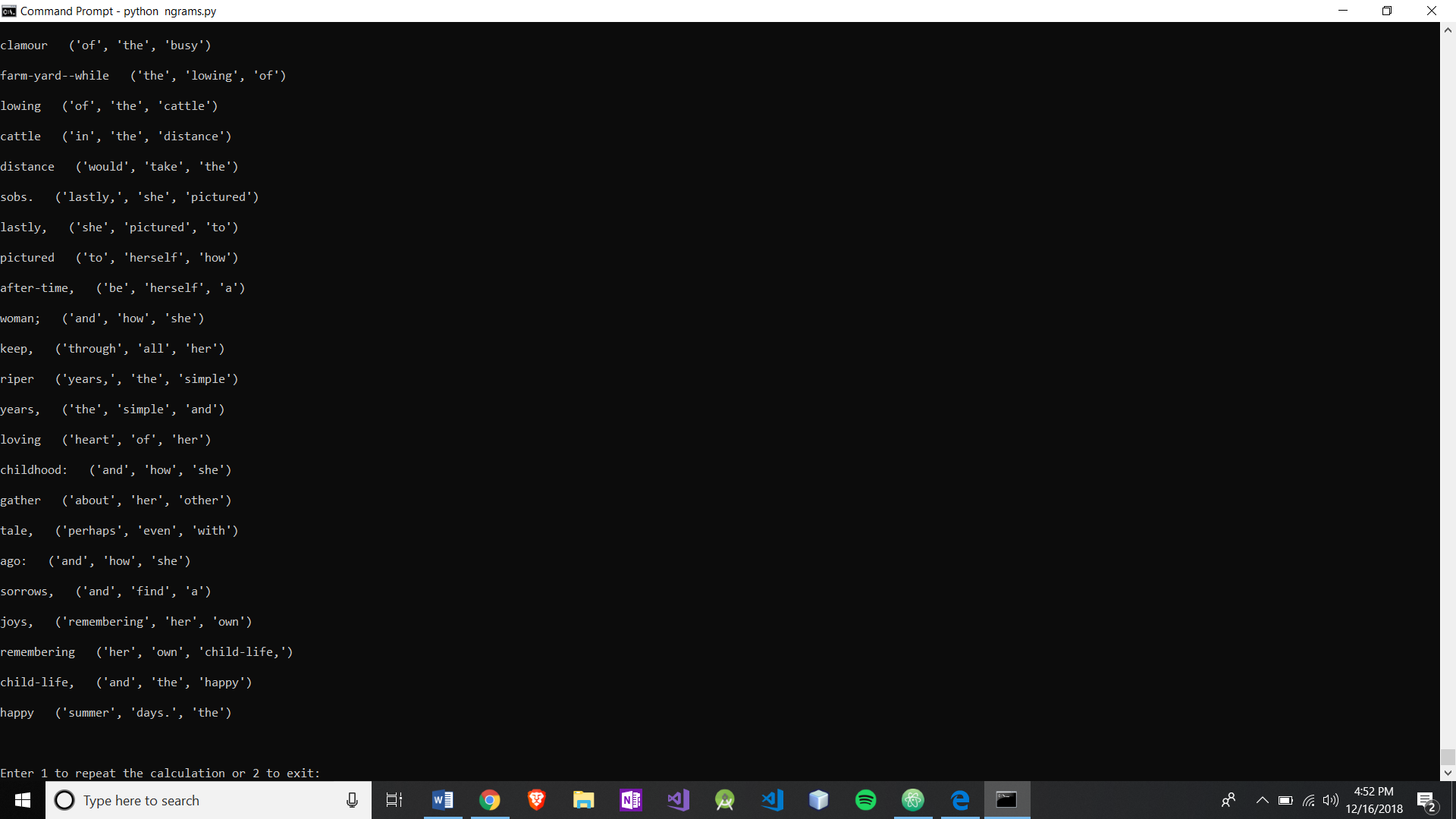
Now compare this paragraph from Figure 6 with the paragraph from Figure 1 and you will notice that it is making less sense. This is because the list of possible next words has now been constructed to include the next (n-1) terms and as n gets larger. This dilutes the effectiveness of the just including the very next term and that is all. It will be more apparent as I demonstrate this program with n = 4.

**Figure 7**



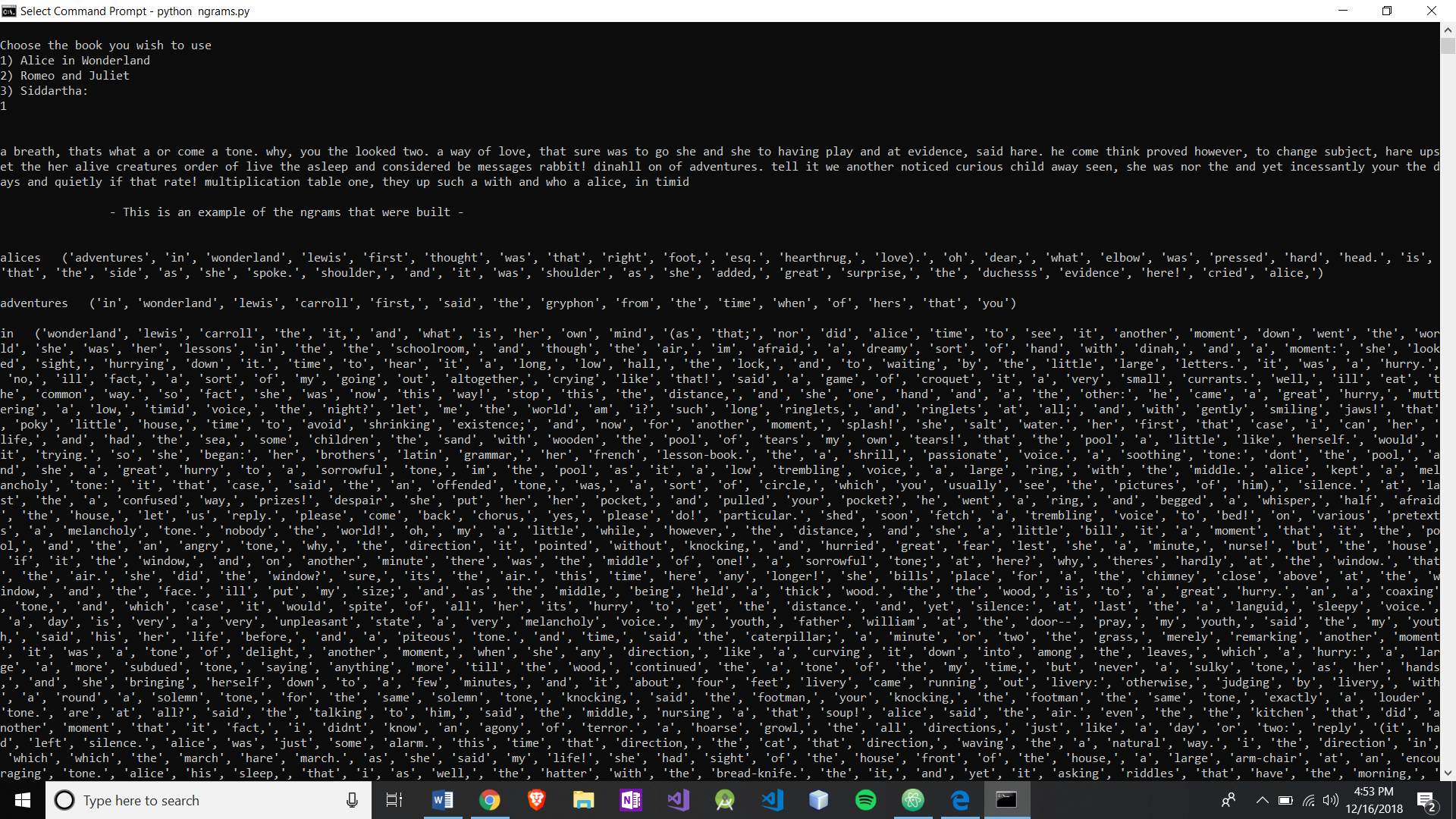
Less coherent than n = 2 or n = 3. Now notice the last entries once again.

**Figure 8**

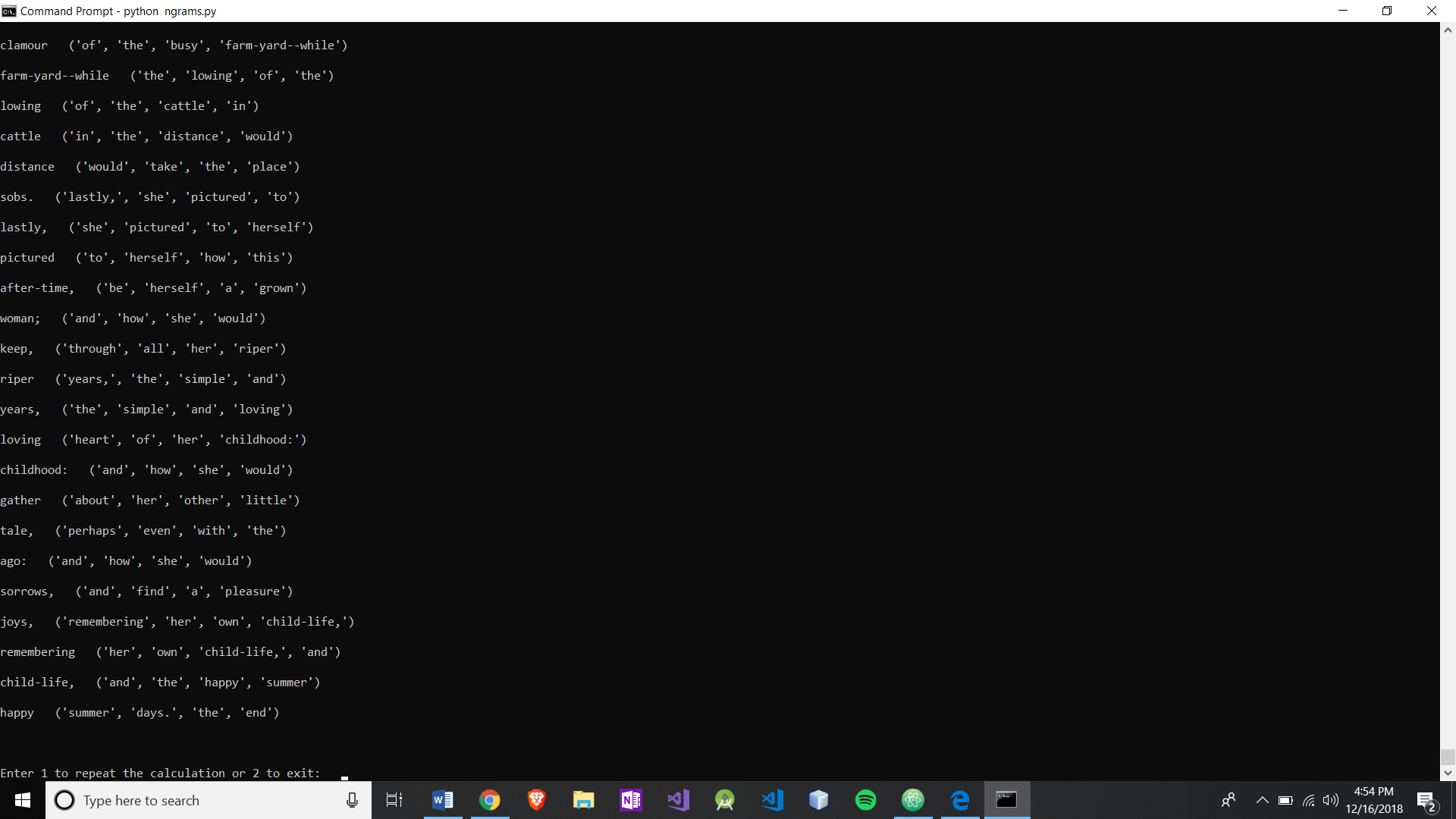


They include n-1 entries in the next list.

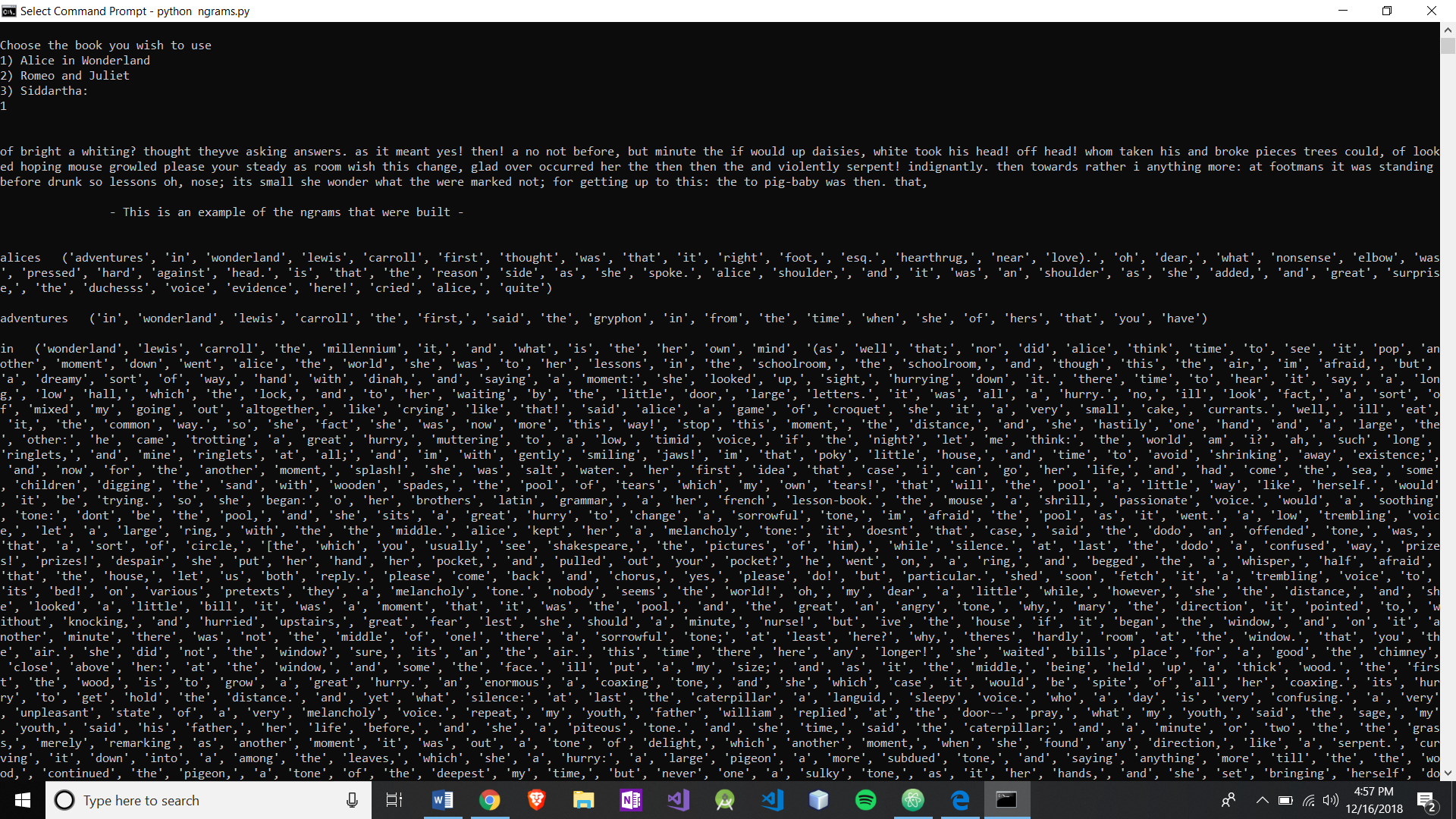
**Figure 9 ,** n = 5



**Figure 10**, The last entries once again.



**Figure 11**, n = 6



The paragraph has lost all of the magic of when n = 2. It has been diluted by all other words included. As our last picture expresses.

**Figure 12**

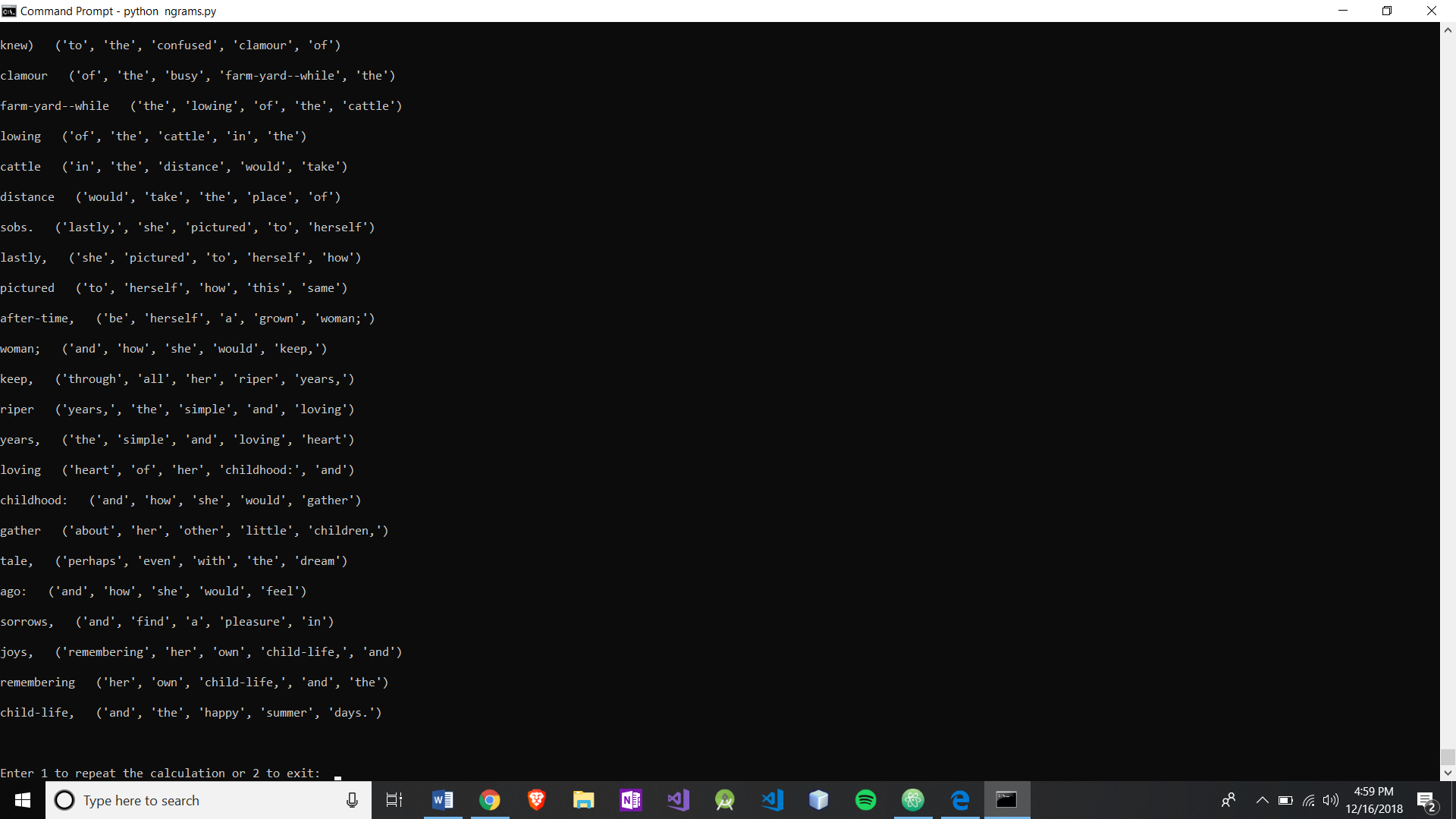


Figure 12 also demonstrates that the last original word was “child-life,” which is more of a token than word and as I did not account for parsing out all punctuation there may or may not exist just “child-life”. I searched and there is not an occurrence.

The rest of my functions, or methods, what have you; utilize direct access and simple assignment. There is a while loop with a global counter that runs 100 times. So technically perhaps this algorithm runs O(100n) which ~ is O(n).