Penn Engineering
ONLINE LEARNING

**Part 4: Implementing the Solution in Python**

***Summary***

In this final part of the course project, you will implement the solution in Python.

***Description***

Throughout this course project, we have been applying computational thinking to solve the problem of counting the number of occurrences of a word and its synonyms in a corpus of text documents.

In Part 1, we applied the pillars of computational thinking to decompose this problem into two smaller problems, then used pattern recognition to find commonalities between the problems, used data representation and abstraction to identify the data we needed, and developed an algorithm.

In Part 2, we represented that algorithm using a flowchart, and in Part 3 we developed the pseudocode for that algorithm as follows:

```
1   All_words ← [Keyword]
2   For each Entry in Thesaurus
3           If Keyword = Entry.Word
4           Then
5                   For each Word in Entry.Synonyms
6                           Add Word to All_words
7                   Stop
8
9   For each Search_word in All_words
10          Count ← 0
11          For each Document in Corpus
12                  For each Word in Document
13                          If Search_word = Word
14                          Then Count ← Count + 1
15          Output: Search_word, Count
```

Now, to finish the computational thinking problem solving process, implement this algorithm in Python in the space below by completing the “search” function.

As you can see, the parameter to the function is the “keyword” for which to search. Your program can also access two other variables that you will need to complete the program:

- **Thesaurus**, which is a list of Entry objects; each Entry object has a **word** attribute, which is a string of characters; and an attribute called **synonyms**, which is a list of strings
- **Corpus**, which is a list of lists of strings

For example, the Entry class and Thesaurus variable may be defined like this:

```
1  class Entry :
2      def __init__(self, input_word, input_synonyms) :
3          self.word = input_word
4          self.synonyms = input_synonyms
5
6  e1 = Entry("dog", ["doggie", "puppy"])
7  e2 = Entry("cat", ["kitty"])
8
9  Thesaurus = [e1, e2]
```

Note that the first argument to the Entry constructor is the word in the thesaurus, and the second is the list of words that are its synonyms. All words consist only of lowercase letters.

And the Corpus variable may be defined like this:

```
1  doc1 = [“this”, “is”, “a”, “single”, “document”]
2  doc2 = [“here”, “is”, “another”, “document”]
3
4  Corpus = [doc1, doc2]
```

Each document is represented as a list of words, which are all lowercase letters, and the Corpus is a list containing those lists.

You can access the **Thesaurus** and **Corpus** variables in your program without having to define them; they have already been defined and initialized with words and phrases describing a person’s emotions -- happy, sad, angry, etc. -- in the setup of this activity. For your own testing purposes, you can create your own Thesaurus and Corpus and populate them with your own data, but please do so **outside** the function definition, and keep in mind that the correctness of your function will be determined using the Thesaurus and Corpus that we have provided.

Your “search” function should implement the algorithm described by the pseudocode above by using the Thesaurus to find the keyword’s synonyms, and then reporting the number of occurrences of the keyword and its synonyms in all documents in the Corpus.

The output of your function should be a list of tuples, in which the first element of the tuple is the word that was searched for (either the keyword or one of its synonyms) and the second is its total number of occurrences.

For instance, if the keyword was “cat” and it occurred 120 times, and its synonym was “kitty” and it occurred 84 times, the output should be:

[ (“cat”, 120), (“kitty”, 84) ]

*Hint:* You can create a tuple variable like this:

result = (“cat”, 120)

And then add it to a list using the list’s “append” function.

As in Parts 2 and 3, you may assume that:

- The Thesaurus is not empty, i.e., there is at least one Entry in the Thesaurus.
- The Corpus is not empty, i.e. there is at least one document (list of strings) in the Corpus.
- Each document contains at least one string.

You can also assume that all words consist only of lowercase letters and that you don’t have to account for punctuation.

```
1  def search(keyword) :
2
3    # implement the function here
4
5    return # modify to return a list of tuples
6
7  input = “happy”
8  output = search(input) # invoke the method using a test input
9  print(output) # prints the output of the function
10 # do not remove this line!
```

Your program will be evaluated using the inputs “happy” and “sad” for the Thesaurus and Corpus we have provided, along with other test inputs as well. Keep in mind that you can print out the Thesaurus and Corpus objects to see what is in them and to get an idea of whether your program is producing the correct output for those inputs.

*Hints*

- *Recall that using the computational thinking pillar of decomposition, we broke this problem into two small problems: finding the synonyms and counting the number of occurrences for each word. Rather than writing the entire function all at once, start on the first part (finding the synonyms) and make sure that works before moving on to the second.*
- *Review previous activities and lessons for examples of iterating over lists, looking for individual values, and updating variables to keep count of something.*
- *Part of the challenge of this activity is knowing in advance whether your program is generating the correct output. As mentioned above, you can create your own Thesaurus and Corpus based on the examples described above, and use these to determine whether your program is correctly finding the synonyms and correctly counting the number of occurrences; be sure to do this* ***outside****the function definition. Keep in mind, though, that your function will be graded using the Thesaurus and Corpus that we have provided.*
- *If you'd like to see the Thesaurus and Corpus that we have provided, don't forget that these are variables just like any others, and you can print out their values.*
- *As in previous activities, don't forget that you can use the "print" function to print out intermediate values as your code is performing operations so that you can see what is happening, and you can use the "Run" button to run the program and see those outputs.*

*Common Errors*

You may run into Python syntax or runtime errors while developing your solution. Here are some of the common ones:

- If you encounter **TypeError** or **AttributeError** related to the Entry class, be sure you are correctly using its attributes, based on the example above. Note that you should not need to create any new Entry objects unless you are creating a test Thesaurus.
- Likewise, if you encounter a **NameError** or **AttributeError** related to the Thesaurus or Corpus variables, check the examples above and make sure you understand their structure. Note that the Corpus is a list of lists, and not a list of "documents," i.e. there is no "document" class in this program.
- Last, if you encounter an **IndentationError,** this means that your code is not properly indented. Keep in mind that all the code you write as part of the solution should be within the body of the "search" function and indented by at least one tab; you may have additional code outside the search function, e.g. creating a test Thesaurus or Corpus, but the code inside the function must be indented.

After you have run your program using the "Run" button and believe that it is producing the correct output, accept the terms of the Coursera Honor Code and then click the "Submit Quiz" button below to submit this assignment and have it graded.

A few seconds after you submit the quiz, you will see the result at the top of the screen. If it reads “Congratulations! You passed!” then you are done!

However, if it reads “Try again once you are ready.” then this means that the automatic grading program indicated that your program is not correct. In that case, click the “Retake” button to go back to the quiz and try again.

If you believe that your code was correct, be sure to click the “Run” button before submitting and inspect the result of the “print” statement that is right before the “return” statement. This allows you to see the output that your code is producing before it is evaluated. If it looks right, but the automatic grading utility is still indicating that it is incorrect, then post a message on the discussion board to ask for assistance.