Problem 2: Markov text generation

Your tasks

Important

If you add test code to your ps8pr2.py file, please put it in one or more separate test functions, which you can then call to do the testing. Having test functions is not required. However, **you should *not* have any test code in the global scope (i.e., outside of a function)**. See the note in Problem 2 for more detail.

1. Write a function **create_dictionary(filename)** that takes a string representing the name of a text file, and that returns a dictionary of key-value pairs in which:
    - each key is a word encountered in the text file
    - the corresponding value is a list of words that follow the key word in the text file.

    For example, the dictionary produced for the text *"I love roses and carnations. I hope I get roses for my birthday."* would include the following key-value pairs, among others:

```
'I': ['love', 'hope', 'get']
'love': ['roses']
'roses': ['and', 'for']
'my': ['birthday.']
# as well as others!
```

    Guidelines:

    - You should *not* try to remove the punctuation from the words in the text file.
    - The keys of the dictionary must include every word in the file *except* the *sentence-ending words*. A sentence-ending word is defined to be any word whose last character is a period ('.'), a question mark ('?'), or an exclamation point ('!'). A sentence-ending word *should* be included in the lists associated with the words that it follows (i.e., in the value parts of the appropriate key-value pairs), but it must *not* appear as its own key.
    - If a word w1 is followed by another word w2 multiple times in the text file, then w2 must appear multiple times in the list of words associated with w1. This will allow you to capture the frequency with which word combinations appear.

- In addition to the words in the file, the dictionary must include the string `$` as a special key referred to as the *sentence-start symbol*. This symbol will be used when choosing the first word in a sentence. In the dictionary, the list of words associated with the key `'$'` must include:
    - the first word in the file
    - every word in the file that follows a sentence-ending word.

  Doing this will ensure that the list of words associated with `'$'` includes all of the words that start a sentence. For example, the dictionary for the text *"I scream. You scream. We all scream for ice cream."* would include the following entry for the sentence-start symbol:

```
'$': ['I', 'You', 'We']
```

- **You may find it helpful to consult the `word_frequencies` function from lecture. We will also discuss some additional strategies for `create_dictionary` in lecture.**

Testing your code

To test your code, download the following file:

sample.txt

and put it in the same folder that contains `ps8pr2.py`. This sample text file contains the following contents:

```
A B A. A B C. B A C. C C C.
```

Once this file is in place, run your `ps8pr2.py` in Spyder and test your function from the console:

```
>>> word_dict = create_dictionary('sample.txt')

>>> word_dict
result: {'A': ['B', 'B', 'C.'], 'C': ['C', 'C.'],
'B': ['A.', 'C.', 'A'], '$': ['A', 'A', 'B', 'C']}
```

The order of the keys–or of the elements within a given key's list of values–may not be the same as what you see above, but the elements of the lists must appear in the quantities shown above for each of the four keys `'A'`, `'B'`, `'C'`, and `'$'`.

Here are some additional files you can use for testing:

- edited_mission.txt - an edited version of BU's mission statement, and the dictionary that we derived from it.

- brave.txt - lyrics from the song *Brave* by Sara Bareilles, and its dictionary.

Here again, the ordering that you obtain for the keys and list elements in the dictionaries may be different. In addition, we have edited the formatting of the dictionaries to make them easier to read.

2. Write a function **generate_text(word_dict, num_words)** that takes as parameters a dictionary of word transitions (generated by the create_dictionary function) named word_dict and a positive integer named num_words. The function must use word_dict to generate and **print** num_words words, with a space after each word. ***The function must print the words that it generates. It must not return a value.***

   Guidelines:

   - The first word must be chosen randomly from the words associated with the sentence-start symbol, '$'. The second word must be chosen randomly from the list of words associated with the first word, *etc*. When the current word ends in a period ('.'), question mark ('?'), or exclamation point ('!'), the function must detect this and start a new sentence by again choosing a random word from among those associated with '$'.
   - Do not include '$' in the output text. It should only be used as an internal marker for your function.
   - You must use the random.choice function to choose from the list of possible words. Don't forget to include import random at the top of your file. Then, if wordlist is the list of possible words at a given point in the generated text, you can do something like the following:
   - next_word = random.choice(wordlist)
   - Here again, you shouldn't try to remove or change the punctuation associated with the words, and you don't need to worry if the generated text doesn't end with appropriate punctuation. The generated text won't be perfect, but most of the time it will at least be meaningful!
   - If your function encounters a word that doesn't have any words associated with it in the dictionary, the function must start a new sentence. This situation can occur if the last word in the file used

to create the dictionary was unique and did not end with punctuation.

- **Here again, we will discuss some strategies for `generate_text` in lecture.**

Examples

Here are two examples using the same text file as above. Your output may differ because of the randomness involved in the generation.

```
>>> word_dict = create_dictionary('sample.txt')

>>> generate_text(word_dict, 20)
B C. C C C. C C C C C C C C C C C C. C C C. A

>>> generate_text(word_dict, 20)
A B A. C C C. B A B C. A C. B A. C C C C C C.
```

Try some other examples using longer documents containing English words, such as the works of William Shakespeare. In particular, we are providing a text file containing the first act of *Romeo and Juliet*, along with the files that we provided above in the examples for `create_dictionary`.