One type of question encountered in the Test of English as a Foreign Language (TOEFL) is the "Synonym Question", where students are asked to pick a synonym of a word out of a list of alternatives. For example:

```
 1. vexed                                          (Answer: (a) annoyed)
(a) annoyed
(b) amused
(c) frightened
(d) excited
```

For this assignment, you will build an intelligent system that can learn to answer questions like this one. In order to do that, the system will approximate the *semantic similarity* of any pair of words. The semantic similarity between two words is the measure of the closeness of their meanings. For example, the semantic similarity between "car" and "vehicle" is high, while that between "car" and "flower" is low.

In order to answer the TOEFL question, you will compute the semantic similarity between the word you are given and all the possible answers, and pick the answer with the highest semantic similarity to the given word. More precisely, given a word w and a list of potential synonyms s_1, s_2, s_3, s_4, we compute the similarities of (w, s_1), (w, s_2), (w, s_3), (w, s_4) and choose the word whose similarity to w is the highest.

We will measure the semantic similarity of pairs of words by first computing a *semantic descriptor vector* of each of the words, and then taking the similarity measure to be the *cosine similarity* between the two vectors.

Given a text with n words denoted by $(w_1, w_2, ..., w_n)$ and a word w, let $desc_w$ be the semantic descriptor vector of w computed using the text. $desc_w$ is an n-dimensional vector. The i-th coordinate of $desc_w$ is the number of sentences in which both w and w_i occur. For efficiency's sake, we will store the semantic descriptor vector as a dictionary, not storing the zeros that correspond to words which don't co-occur with w. For example, suppose we are given the following text (the opening of *Notes from the Underground* by Fyodor Dostoyevsky, translated by Constance Garnett):

> I am a sick man. I am a spiteful man. I am an unattractive man. I believe my liver is diseased. However, I know nothing at all about my disease, and do not know for certain what ails me.

The word "man" only appears in the first three sentences. Its semantic descriptor vector would be:

```
{"i": 3, "am": 3, "a": 2, "sick": 1, "spiteful": 1, "an": 1, "unattractive": 1}
```

The word "liver" only occurs in the second sentence, so its semantic descriptor vector is:

```
{"i": 1, "believe": 1, "my": 1, "is": 1, "diseased": 1}
```

We store all words in all-lowercase, since we don't consider, for example, "Man" and "man" to be different words. We do, however, consider, *e.g.*, "believe" and "believes", or "am" and "is" to be different words. We discard all punctuation.

The cosine similarity between two vectors $u = \{u_1, u_2, \ldots, u_N\}$ and $v = \{v_1, v_2, \ldots, v_N\}$ is defined as:

$$\text{sim}(u, v) = \frac{u \cdot v}{|u||v|} = \frac{\sum_{i=1}^{N} u_i v_i}{\sqrt{\left(\sum_{i=1}^{N} u_i^2\right)\left(\sum_{i=1}^{N} v_i^2\right)}}$$

We cannot apply the formula directly to our semantic descriptors since we do not store the entries which are equal to zero. However, we can still compute the cosine similarity between vectors by only considering the positive entries. See the starter code for how this can be accomplished.

For example, the cosine similarity of "man" and "liver", given the semantic descriptors above, is

$$\frac{3 \cdot 1 \text{ (for the word "i")}}{\sqrt{(3^2 + 3^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2)(1^2 + 1^2 + 1^2 + 1^2 + 1^2)}} = 3/\sqrt{130} = 0.2631\ldots$$

Question 1.

Implement the following functions in `synonyms.py`. Note that the names of the functions are case-sensitive and must not be changed. You are not allowed to change the number of input parameters, nor to add any global variables. Doing so will cause your code to fail when run with our testing programs, so that you will not get any marks for functionality. We provide you with a starter version of `synonyms.py`

Part (a) `get_sentence_lists(text)`

This function takes in a string `text`, and returns a list which contains lists of strings, one list for each sentence (as defined below) in the string `text`. A list representing a sentence is a list of the individual words in the sentence, each one in all-lowercase.

For our purposes, sentences are separated by one of the strings `"."`, `"?"`, or `"!"`. We ignore the possibility of other punctuation separating sentences. We also assume that every period separates sentences. For example, the string

```
"The St. Bernard is a friendly dog!"
```

is considered to be two sentences: `"The St"` and `"Bernard is a friendly dog"`.

The words in the list that represents the sentence must be in the order in which they appear in the sentence, and must not begin or end with punctuation.

You should assume that only the following punctuation signs are present in the text file: `","`, `"-"`, `"--"`, `":"`, `";"`, `"!"`, `"?"`, `"."`, the single quote, or the double quote. That the single quote is considered punctuation means that, for example, "don't" is considered to be two words, "don" and "t". The possesive form "School's" is also considered to be two words, "School" and "s".
For example, if the text file contains the following (and nothing else):

```
Hello, Jack.  How is it going?  Not bad; pretty good, actually...  Very very
good, in fact.
```

then the function should return the following list:

```
[['hello', 'jack'],
 ['how', 'is', 'it', 'going'],
 ['not', 'bad', 'pretty', 'good', 'actually'],
 ['very', 'very', 'good', 'in', 'fact']]
```

Part (b) `get_sentence_lists_from_files(filenames)`

This function takes in a list of strings `filenames`, each one the name of a text file, and returns the list of every sentence contained in all the text files in `filenames`, in order. The list is in the same format as in Part a, but with the files rather than a string serving as the source of the text..

Part (c) `build_semantic_descriptors(sentences)`

This function takes in a list `sentences` which contains lists of strings representing sentences, and returns a dictionary `d` such that for every word `w` that appears in at least one of the sentences, `d[w]` is itself a dictionary which represents the semantic descriptor of `w` (note: the variable names here are arbitrary). For example, if `sentences` represents the opening of *Notes from the Underground* as above, part of the dictionary returned would be:

```
{ 'man': {'i': 3, 'am': 3, 'a': 2, 'sick': 1, 'spiteful': 1, 'an': 1,
          'unattractive': 1},
  'liver': {'i': 1, 'believe': 1, 'my': 1, 'is': 1, 'diseased': 1},
  ... }
```

with as many keys as there are distinct words in the passage.

Part (d) `most_similar_word(word, choices, semantic_descriptors)`

This function takes in a string `word`, a list of strings `choices`, and a dictionary `semantic_descriptors` which is built according to the requirements for `build_semantic_descriptors`, and returns the element of `choices` which has the largest semantic similarity to `word`, with the semantic similarity computed using the data in `semantic_descriptors`. If the semantic similarity between two words cannot be computed, it is considered to be -1. In case of a tie between several elements in `choices`, the one with the smallest index in `choices` should be returned (*e.g.*, if there is a tie between `choices[5]` and `choices[7]`, `choices[5]` is returned).

Part (e) `run_similarity_test(filename, semantic_descriptors)`

This function takes in a string `filename` which is a file in the same format as `test.txt`, and returns the percentage of questions on which `most_similar_word()` guesses the answer correctly using the semantic descriptors stored in `semantic_descriptors`.

The format of `test.txt` is as follows. On each line, we are given a word (all-lowercase), the correct answer, and the choices. For example, the line:

```
feline cat dog cat horse
```

represents the question:

```
feline:
(a) cat
(b) dog
(c) horse
```

and indicates that the correct answer is "`cat`".

Question 2.

Add code to test each of the functions in 1a-1d You are encouraged to test your helper functions, but that will not be graded. You should submit all the files needed for the testing of your functions, so that we can run the tests if we so choose.

Question 3.

Download the novels *Swann's Way* by Marcel Proust, and *War and Peace* by Leo Tolstoy from Project Gutenberg, and use them to build a semantic descriptors dictionary. Report how well the program performs on the questions in `test.txt`, using those two novels at the same time. Note: the program may take several minutes to run (or more, if the implementation is inefficient). Include the code used to generate the results you report. The novels are available at the following URLs:

`http://www.gutenberg.org/cache/epub/7178/pg7178.txt`
`http://www.gutenberg.org/cache/epub/2600/pg2600.txt`