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.

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

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

Part 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`

Subpart (a) `cosine_similarity(vec1, vec2)`

This function returns the cosine similarity between the sparse vectors `vec1` and `vec2`, stored as dictionaries. For example,

```
cosine_similarity({"a": 1, "b": 2, "c": 3}, {"b": 4, "c": 5, "d": 6})
```

should return approximately 0.70 (as a `float`).

Subpart (b) `build_semantic_descriptors(sentences)`

This function takes in a list `sentences` which contains lists of strings (words) 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* above:

```
[["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"]],
```

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.

Computational complexity Beware of writing the function so that its complexity if $\mathcal{O}(n^2)$ where n is the number of words in the text. This will result in your not being able to run the code once you complete all the functions.

A hint for how to achieve this is at the end of this handout.

Subpart (c) `build_semantic_descriptors_from_files(filenames)`

This function takes a list of `filenames` of strings, which contains the names of files (the first one can be opened using `open(filenames[0], "r", encoding="latin1")`), and returns the a dictionary of the semantic descriptors of all the words in the files `filenames`, with the files treated as a single text.

You should assume that the following punctuation always separates sentences: `"."`, `"!"`, `"?"`, and that is the only punctuation that separates sentences. You should also assume that that is the only punctuation that separates sentences. Assume that only the following punctuation is present in the texts:

```
[",", "-", "--", ":", ";"]
```

Subpart (d) `most_similar_word(word, choices, semantic_descriptors, similarity_fn)`

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` and the similarity function `similarity_fn`. The similarity function is a function which takes in two sparse vectors stored as dictionaries and returns a `float`. An example of such a function is `cosine_similarity`. 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).

Subpart (e) `run_similarity_test(filename, semantic_descriptors, similarity_fn)`

This function takes in a string `filename` which is the name of a file in the same format as `test.txt`, and returns the percentage (i.e., `float` between `0.0` and `100.0`) of questions on which `most_similar_word()` guesses the answer correctly using the semantic descriptors stored in `semantic_descriptors`, using the similarity function `similariy_fn`.

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) dog
(b) cat
(c) horse
```

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

For example, `rum_similarity_test` could be used as follows:

```
sem_descriptors = build_semantic_descriptors_from_files(["wp.txt", "sw.txt"])
res = run_similarity_test("test.txt", sem_descriptors, cosine_similarity)
print(res, "of the guesses were correct")
```

1 Correctness and complexity

You should test your individual function using small examples where you know the correct restult.

You should also attempt to run your functions on the recommended novels (together): *War and Peace* and *Swann's way*, available for Project Gutenberg.

`http://www.gutenberg.org/files/2600/2600-0.txt`

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

Because different people will make different choices regarding punctuation other than what is outlined in 1(c) , results will vary. You should expect to get between 67.5% and 72.5% correct guesses when all functions are implemented correctly.

Computational complexity hint: think of a way to update the semantic descriptor dictionary sentence-by-sentence.