Objective. This week we're going to take a closer look at the faculty of Williams. We have a (mostly) up-to-date set of data about the faculty, presented here as a CSV file. Your goal is to write some small, but powerful, functions to answer trivia questions about the Williams faculty (in our solutions, each question can be answered in 10 lines or less!).

Getting Started. As usual, you should clone the starter repository for this week's lab into your cs134 directory:

```
git clone https://evolene.cs.williams.edu/cs134-labs/22xyz3/lab09.git ~/cs134/lab09
```

where your CS username replaces 22xyz3.

This week’s tasks. We have two Python scripts that need to be completed. The first file, faculty.py, contains a pair of classes—Degree and Instructor—and a few tools that will help you construct a database of faculty and query the results. The second file—trivia.py—contains functions to solve the trivia questions described in each docstring.

First, let’s address the functions in faculty.py:

1. Read through the implementations of the Instructor and Degree classes. Once you’ve done that, resolve the error in the current implementation of readDB() so that all its doctests pass. (Ignore, for the moment, failed doctests for population.)

2. We have provided two helper functions, maxes and mins. maxes returns a list containing all the largest values in a list according to a supplied key function. The implementation of maxes uses a filter function to arrive at a correct solution. We expect maxes to work as follows:

```plaintext
>>> maxes(['a', 'aa', 'aaa', 'a', 'aaa'])
['aaa', 'aaa']
>>> maxes([1, 1, 2, 4, 4, 4], key = lambda x: -x)
[1, 1]
```

mins is the opposite of maxes; it returns a list containing the smallest values according to a supplied key function. We expect it work as follows:

```plaintext
>>> mins(['a', 'aa', 'a'])
['a', 'a']
>>> mins([1, 1, 2, 3, 4, 4, 4], key = lambda x: x%2)
[2, 4, 4, 4]
```

Note that the current implementation uses a filter function, just like maxes. Replace this implementation with one that uses a list comprehension instead.

---

1We apologize in advance for any mistakes regarding your favorite faculty member; web scrapping is an error-prone task.
3. Implement the population method. This method takes a list, lst, and returns a list of tuples. Each tuple is of the form (val, number of occurrences of val in lst) for each val in the original list. Ideally the order of the tuples should be the same as the order each val appears in the original list. Some examples of how population should work are below:

```python
>>> population(['Physics', 'Physics', 'Math', 'English', 'Physics'])
[('Physics', 3), ('Math', 1), ('English', 1)]

>>> population([])
[]

>>> population([1, 2, 1, 2, 1, 2])
[(1, 3), (2, 3)]

>>> population([1.0, 4.1, 5.2, 10.3])
[(1.0, 1), (4.1, 1), (5.2, 1), (10.3, 1)]
```

After finalizing the faculty module, you should turn your attention to solving the trivia questions in trivia.py. To receive full credit on your trivia solutions, your implementations need to match the following requirements. Partial credit will still be given to solutions which do not meet these requirements, but successfully find the correct solution to the trivia question programmatically.

You need to complete 4 trivia questions total and each solution should use map and/or filter. In addition, you should:

1. Complete 1 of either Q1 or Q2
2. Using the sorted function, complete at least 1 of either Q3 or Q4
3. Using maxes and/or mins, complete at least 1 of either Q5, Q6, and Q7
4. Complete an addition question of your choice

Submitting your work. When you're finished, add, commit, and push your work to the server as you did in previous labs. Remember that you must certify that your work is your own, by signing the honorcode.txt file, committing and pushing it along with your work. Make sure you have: (a) finished all the necessary work in faculty.py and (b) answered trivia questions according to the requirements above in trivia.py.
Extra credit. Work on more than 4 of the trivia questions or invent additional questions and find their answers!

Perpetual Grading Expectations. In addition to the lab-specific grading guidelines contained above and in GradeSheet.txt, remember that you are assessed on whether your code meets the general course expectations. For full coverage of what style entails, please see the Style Guide.

As a guide to avoiding common errors, make sure to follow these guidelines:

⋆ Always sign the honor code. It protects your work.
⋆ Follow the 80-character rule. This makes your code more readable.
⋆ Use meaningful variable names. Help the reader understand your code.
⋆ Comment your code. Logic that was hard to develop is often hard to understand.
⋆ Write entirely new tests of your own design. Ours can be improved.
⋆ Every doctest should be meaningfully different. Address standard use and unusual “corner” cases.
⋆ When done, review the lab handout and make sure you’ve addressed the requirements.

⋆