Computer Science CS134 (Spring 2020)
Shikha Singh & Iris Howley
Laboratory 4
Parsing CSV Files (due Wednesday/Thursday at 11pm)

Objective. To analyze a CSV database.

This week we're going to take a closer look at the faculty of Williams. The Dean of Faculty has been kind enough to send us information[1] about the faculty here as a *comma separated values* (CSV) file.

We'll first build a small collection of methods to help recode the data. Then, we'll see if we can write a Python program to answer a number of trivia questions.

Working with a Partner. This week you are permitted and encouraged to find a partner (in the same lab section as yourself) to work on this lab together! To do this, you will decide to use one partner's repository, and then the chosen partner will add the other partner as a collaborator to their repository in GitLab. To invite your collaborator, select the repo, find the `Members` option under `Settings` and then on the left-hand side find your partner under `Select members to invite`. Select the `Maintainer` role for your partner (leave the expiration date black) and click on the green `Add to project`. Now when your partner logs into `evolene`, they will see your repository in their list of projects and will have access to stage, commit, and push to the repository.

Getting Started. As usual, you should clone the starter repository for this week's lab into your `cs134` directory, by going to `https://evolene.cs.williams.edu` and clicking `Clone` and the the `Copy URL to clipboard` button under the `Clone with HTTPS` text. Return to the `Terminal`, navigate to your `cs134` directory (with `cd ~/cs134`, if you are on a lab machine), and type `git clone` followed by the URL you just copied, followed by the name of the lab `lab04`. Your line should look something like this:

```
git clone https://evolene.cs.williams.edu/cs134-s20/lab04/22xyz3.git lab04/
```

where your CS username replaces `22xyz3`. The resulting directory contains a `README`, a CSV file that describes the faculty, two python starter files, `faculty.py` and `trivia.py`, `trivia.txt` for submitting trivia answers, and finally `honorcode.txt` .

Working across multiple machines. When working with a partner, both you and your partner may `git clone` a copy of the same repo to both your machines. If your partner makes edits to the repo, you'll need to *pull* them from the server to your local machine, so that you'll have access to their edits. You can do this by adding the correct project folder to Atom, toggling the `Git` tab, and clicking on the `Fetch` button at the bottom right of the `Git` tab, followed by the `Pull` button. When working with a partner, **always** start by `Fetch`ing and `Pull`ing!

Reading CSV files.

Python provides a module, `csv`, that will help you read (or write) CSV files. Here's how:

```
import csv
with open('faculty.csv','r') as f:
    csvf = csv.reader(f)
    for row in csvf:
        # process 'row', a list of fields separated by commas
        ...
```

[1] We asked the Dean of Faculty for this data in Spring of 2016, and so some more recent hires have been sadly left off the list. The Computer Science Department, however, keeps its records up to date, so you'll find all of us in there!

The `with` statement says: "open this file as f only for the duration of the following suite of statements." You can type `pydoc3 csv` for more details.

This week's tasks. We have two python scripts that need to be fleshed out. The first is a module, `faculty`, that contains a small set of tools that will help you build a database of faculty; those steps are discussed below. The second script—`trivia.py`—will contain routines that you use to compute the answers to the questions in `trivia.txt`.

First, let's write functions that will help us *recode* our database:

1. `parseName(name)`. Write a method `parseName` that takes a name (as a string) from our database and returns a list of names, the last of which is the surname (and possible suffix):

   ```
   >>> parseName('Nolan Jr.,James L.')
   ['James', 'L.', 'Nolan Jr.']
   >>> parseName('Rulikova Edwards,Marketa')
   ['Marketa', 'Rulikova Edwards']
   ```

   This method is *private*; we don't expect it to be needed outside the module, so its name should not appear in `__all__`.

2. `parseDegree(degreeInfo)`. Write a method `parseDegree` that takes a string and returns a list that describes the degree: a year (an integer), a degree (a string), and a granting institution (a string). If the degree string is empty, return the empty list.

   ```
   >>> parseDegree('2018, Ph.D., Stony Brook University')
   [2018, 'Ph.D.', 'Stony Brook University']
   ```

   This definition is private, as well.

3. `parseMember(rowList)`. Given a row (a list) read from the CSV file, return a list that describes the faculty member: item 0 is the parsed faculty name, item 1 is the faculty title, item 2 is the faculty member's department, item 3 is a parsed bachelor's degree, item 4 is the parsed master's degree, and item 5 is the parsed doctorate degree. (This routine is also private.)

   ```
   >>> parseMember(["Howley,Iris K.","Assistant Professor of Computer Science",\
       "Computer Science Department","2008, B.S., Drexel University","2012, M.S.,\
       Carnegie Mellon University","2015, Ph.D., Carnegie Mellon University"])
   [['Iris', 'K.', 'Howley'],
    'Assistant Professor of Computer Science',
    'Computer Science Department',
    [2008, 'B.S.', 'Drexel University'],
    [2012, 'M.S.', 'Carnegie Mellon University'],
    [2015, 'Ph.D.', 'Carnegie Mellon University']]
   ```

4. `readDB()`. Reads the database file `'faculty.csv'` and returns a list of parsed faculty members. This definition is used by others—it's *public*—so we include its name in `__all__`.

5. `uniq(itemList)`. This routine takes an ordered list of values and returns a possibly smaller list with duplicate adjacent values removed. For example:

   ```
   >>> uniq([1988,1988,1988,1989,1990,1990,1991,1993,1993,1988])
   [1988, 1989, 1990, 1991, 1993, 1988]
   ```

   If the list had first been sorted, the result would have been a list of unique values.

   We attempted a solution, but it is broken. Please find the problem and fix it. (Apologies!)

6. `uniqCount(itemList)`. Given a list of values, possibly containing adjacent duplicates. The `uniqCount` function returns a list of pairs, `[value,count]`. Each pair corresponds to a value that occurs in the list and the number of times it occurs, adjacently. For example:

   ```
   >>> uniqCount([1988,1988,1988,1989,1990,1990,1991,1993,1993,1988])
   [[1988,3],[1989,1],[1990,2],[1991,1],[1993,2],[1988,1]]
   ```

When you are finished with these routines, make sure that your answers to all the trivia questions in `trivia.txt`, formatted using the `format` function, are appended to the `trivia.txt` file (by your code in `trivia.py`). Consider how to use the functions in `faculty.py` in your `trivia` solutions. Push your scripts and trivia responses by Wednesday/Thursday 11pm.

Submitting your work. When you're finished, stage, 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 typing out the Honor Code statement in the `honorcode.txt` file, committing and pushing it along with your work.

Late days. You are allowed a total of 3 late days over the semester, with at most 2 late days towards any one lab. You must request a late day in advance on the form, here: `http://bit.ly/s20late`.

Grading Guidelines. Both functionality and programming style are important when writing code, just as both the content and the writing style are important when writing an essay. In this program, some of the specific functional requirements we will test for include:

- The most efficient solutions are those with the fewest number of (nested) loops. Once you have a working solution, consider if you can make it more efficient!
- The best code will use the functions we built in our `faculty.py` toolbox. This is particularly relevant for Q5, and Q6 which should use `uniqCount()`.
- Your `trivia.py` should be able to print formatted answers to the questions (without any incorrect options), and append the answers to the `trivia.txt` file.
- Appropriate functions should be placed in your `__all__` variable, and the rest of your `faculty.py` module should contain the typical hallmarks of a well-built module: docstrings for every function & the module, a few doctests for each function with testable output.
- Documentation and testing are very important. Document your functions and test them to see what they do on the following inputs:

– `parseDegree("")`, `parseName('Olson Blair,Brooke Suzanne')`, `parseName('Bailey,Duane A.')`

– `parseMember(["Aalberts,Daniel P.",'Professor of Physics','Physics Department',"1989, B.S.,\`
`Massachusetts Institute of Technology",'',\`
`"1994, Ph.D., Massachusetts Institute of Technology"])`

– `len(readDB())`, `len(readDB()[0][0])`
– `uniq([1,1,2,-3,-3,-3,1,2])`, `uniqCount([1,1,2,-3,-3,-3,1,2])`

- Just like previous labs, we require that the functions defined in `faculty.py` follow our specifications, e.g., `parseDegree` takes in one parameter—a string `degreeInfo` — and returns a list `[int, str, str]`. Do not modify the function names, their parameters, nor what is returned. The examples above in **This week's tasks** provide details about these constraints.
- It is very important that your files be named properly so that we can run our tests for grading. The only acceptable names and capitalization are: `faculty.py`, `trivia.py`, `faculty.csv`, `trivia.txt`, and `honorcode.txt`.

Stylistically, we expect to see programs that exhibit the following: meaningful names used in declarations, informative comments (both in-line and docstrings), appropriate doctests for functions, good and consistent formatting, and good choice of Python commands. There is some subjectivity to what makes good style, but the basic goal is to make your ideas as clear and easy to follow as possible.

Grading scale. Programming labs will be graded on the following scale:

A+	An absolutely fantastic submission of the sort that will only come along a few times during the semester.
A	A submission that exceeds our standard expectations for the assignment. The program must reflect additional work beyond the requirements or get the job done in a particularly elegant way.
A-	A submission that satisfies all the requirements for the assignment – a job well done.
B+	Submission meets the requirements for the assignment, possibly with a few small problems.
B	A submission that has problems serious enough to fall short of the requirements for the assignment.
C	A submission that has extremely serious problems, but nonetheless shows some effort & understanding.
D	A submission that shows little effort and does not represent passing work.

Extra credit—expand your knowledge. Invent additional faculty trivia questions and find the answer!

⋆