Computer Science CS134 (Spring 2020)
Shikha Singh & Iris Howley
Laboratory 5
Presenting Data (due 11pm, ******Monday/Tuesday******)*

Objective. To learn how to set up a personalized environment and how to plot data.

This week we will use Python's `matplotlib` module to present data. Before we get started, however, we'll have to install this tool in our own directory. This is a simple process that you must do once for each computer you use. Once the tools have been installed, you may choose to *activate* them, or not.

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.

Virtual Environments & Installing Additional Packages.
Few Python environments have *all* the modules you might want to eventually have installed. This week we'll use the *virtual environment* system to allow us to experiment with adding new modules. First, we set up and *activate* the environment, then we install the new software. Here are the steps:

1. We assume that you're using Python 3.6 or greater. You can find python's version with

   ```
   -> python3 --version
   Python 3.7.0
   ```

2. Go to your home directory and create the environment in your `cs134` folder (the `->` is a prompt):

   ```
   -> python3 -m venv --system-site-packages cs134
   ```

   The phrase `--system-site-packages` indicates that you want to *extend* the environment you already have set up on your machine. The specification of the directory simply indicates where the environment is to be installed. Installing it in your `cs134` folder makes it easy to access in all of your future labs.

3. We now navigate to our `cs134` folder and *activate* this new environment:

   ```
   -> cd cs134
   -> source bin/activate
   (cs134) -> python --version
   Python 3.7.0
   ```

   If the activation worked, you'll see your usual prompt, prefixed with the name of the folder that contains your virtual environment. In addition, `python` now refers to `python3`. The `source` command is simply saying "execute the commands found in the activate script in the bin subdirectory." Those commands changed your prompt, and told bash where to install new modules.

Once an environment is activated, you can de-activate it (if you want to) by typing deactivate. Don't do that now, or everything will go back to python's normal behavior, with python refering to version 2 of Python and not 3 and not knowing how to install new modules..

4. New modules are installed with pip (a silly recursive acronym for "Pip Installs Python"). We would like to install the non-standard package, matplotlib:

```
(cs134) -> pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-3.2.0...
    100% |################################| 14.1MB 17.MB/s
... output omitted ...
Installing collected packages:...matplotlib
Successfully installed...matplotlib-3.2.0...
(cs134) ->
```

   The first time you import matplotlib it does some one-time initialization, like building fonts that work specifically with your machine. We'll force matplotlib to build those fonts now:

```
(cs134) -> python
>>> import matplotlib.pyplot as plt
...
UserWarning: Matplotlib is building.... This may take a moment.
...
>>> quit()
```

   The matplotlib environment is now installed and ready for use. You'll be able to use matplotlib for any new code you write, as long as you activate the environment at the beginning of each new session (*i.e.*. whenever you log in, open a Terminal window, *etc.*).

Cloning the lab repository. 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 lab05. Your line should look something like this:

```
git clone https://evolene.cs.williams.edu/cs134-s20/lab05/22xyz3.git lab05/
```

where your CS username replaces 22xyz3. The resulting directory contains a stars directory containing data samples, testStars.py, stars.py, and finally honorcode.txt. In the rest of the lab we read in data, organize and plot it.
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!

This week's tasks. This week's task requires you to read data from a file and write code using `matplotlib` to produce graphs of it. Data about stars is provided in the `.csv` files in the `stars` directory of the starter code. Each `.csv` file provides the same information about different groups of stars. You must flesh out the Python script `stars.py` by implementing the following functions.

1. `loadStars(filename)` This function takes the name of a csv file in the `stars` directory, loads the information from the file and returns it as a list of dictionaries. For example, the `neighbors.csv` file contains data for just three stars:

   ```
   α Centauri A,0.01,4.37
   α Centauri B,1.33,4.37
   Proxima Centauri,11.13,4.24
   ```

   Your code, when given the string `'stars/neighbors.csv'` as an argument, should produce the following list of dictionaries (keys are shown in order here for clarity, they are not ordered in Python).

   ```
   >>> loadStars('stars/neighbors.csv')
   [
   { "name": "α Centauri A", "brightness": 0.01, "distance": 4.37 },
   { "name": "α Centauri B", "brightness": 1.33, "distance": 4.37 },
   { "name": "Proxima Centauri", "brightness": 11.13, "distance": 4.244 },
   ]
   ```

   In `testStars.py`, you will find the list of dictionaries named `pleiades`, which stores the expected result of the function call `loadStars('stars/pleiades.csv')`. Note that your code must work for any of the files in the `stars` directory.

2. `compareStars(starList)` This function takes as an argument a list of dictionaries (in the same format as returned by `loadStars`). This function must create and display a bar graph of the brightness of each star in the dictionary, sorted by brightness in microlux (μlx) and labeled by name.

   Note that the original data stores brightness values as "magnitudes" where smaller numbers are brighter, but the graph you create must use units of microlux (μlx), where a large number represents a brighter star. To make this conversion easy we have provided a `magnitudeToLux` conversion function that converts the magnitude values of brightness to its microlux value. You must use the converted values, both when sorting, and when displaying the data.

   For your plot, you will need to create lists of x- and y-values to pass into the `bar` function. You can create these with a list comprehension, or by iterating over the sorted list of dictionaries. You will also need to use the following functionalities from the `matplotlib` `pyplot` submodule:

   - `figure` to create a new figure

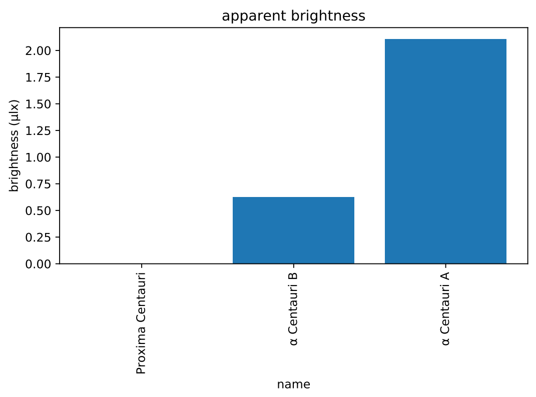

Figure 1: Bar plot generated for neighbors.

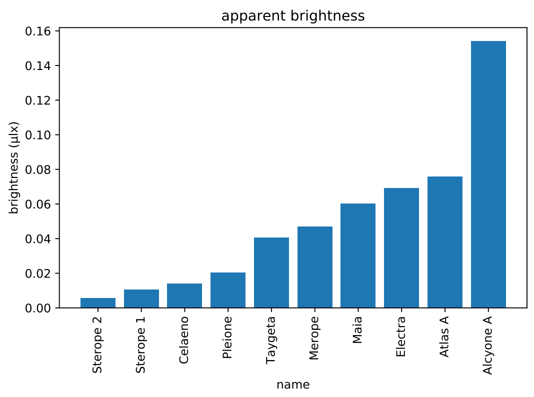

Figure 2: Bar plot generated for pleiades.

- `bar` to create your plot
- `xticks` to set the x-axis labels
- `title`, `xlabel`, and `ylabel` to set up the title and labels of the graph
- `rotation=` keyword argument to rotate the x labels by 90 degrees so they are vertical instead of horizontal and do not overlap
- `tight_layout` right before showing the plot to avoid having text cut off at the margins
- `show` to display the plot once it's set up and `savefig` to save the plot to a file.

We encourage you to learn about these functions and more by investigating the `matplotlib` tutorial at `http://matplotlib.org/users/pyplot_tutorial.html`.

Verifying your plots. We have provided testing functions in the `testStar.py` module to help you test your implementation of the above two functions. You can uncomment the function calls `compareStars(neighbors)` and `compareStars(pleiades)` in the `__main__` block to generate and verify your bar plots. You must save your plots for both the function calls as `lab5-neighbors.pdf` and `lab5-pleiades.pdf`. To do this, you will have to run your program on one set of data, rename the .pdf image generated, and then do the same for the second dataset. The graphs for `neighbors` and `pleiades` should should look as shown below in Figures 1 and 2.

Note that as long as your bars are the right heights, the exact values that `matplotlib` uses for the y-axis ticks need not be the same as in the figures.

Submitting your work. When you're finished, stage, commit, and push your work to the server. Make sure to include the `lab5-neighbors.pdf` and `lab5-pleiades.pdf` files along with your code in `stars.py` that generates it. 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`.

Working across different machines. If you clone the lab repository on two different machines and are planning to switch between them, you must always start by "pulling" your most recent work from the server to the local machine. 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 `Pull`.

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:

- Your code for `loadStars` must work for any of the files in the `stars` directory.
- Remember your `doctests`! We'd like to see at least 2 doctests for `loadStars`. Consider some of the parameters you're passing `loadStars` and what you'd expect to be returned. Write code to ensure that your doctests run when your `stars.py` is run as a script.
- The plots generated by your implementation of `compareStars` on arguments `neighbors` and `pleiades` should resemble the plots shown above.
- The functions defined in `stars.py` must follow our specifications. Do not modify the function names, their parameters, nor what is returned.
- 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: `stars.py`, `lab5-neighbors.pdf` and `lab5-pleiades.pdf`.

Stylistically, we expect to see programs that exhibit the following: meaningful names used in declarations, informative comments (both in-line and docstrings), 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 satisfies all the requirements for the assignment – a job well done.
A-	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.
B	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.

No extra credit. There is no extra credit for this week. You should spend time preparing for the midterm instead! The grading scale for this week has been adjusted accordingly.

Acknowledgement. This assignment is based on Wellesley CS111 Spring19 course materials. We thank Peter Mawhorter (`pmawhort@wellesley.edu`) for providing the data and resources for this assignment.

⋆