Astro Data Science Seminar
Lab 2 - 2016 April 19
Algorithm 1: Do Something!

Today we are thinking about *algorithms,* which is just a fancy word for "code that does a useful thing". This encompasses both the technical or mathematical challenge of solving problems, and the functional need for writing well documented code that works as expected.

Part 1
Update your Fork

Remember last week you "Forked" the master version of the Seminar repository on GitHub. Now I've updated the project, and it's time for you to pull the latest revisions into your Fork.

https://help.github.com/articles/syncing-a-fork/

So add upstream master to your local clone, pull latest version!
(By the way: a brute force way to do this is to delete your fork and your local clone, and re-fork it and clone it from scratch)

NOTE:
Today be sure to make a copy of the example code instead of replacing it. This makes turning in much easier!

Part 2
Make a basic function

One goal of writing good computer code is to never repeat yourself if possible. You achieve this by taking individual, repeatable tasks, and wrapping them up as "functions" (or classes, or methods, or what have you).

Some rules of the road:

- Python calls a function a "Method"
  - https://docs.python.org/2/tutorial/modules.html
- A function should perform a single task
- Every function should be completely documented, both docstrings and inline comments.
- Similar or related functions should be packaged together into modules
- In Python, use classes to combine attributes and methods for objects.
  - https://docs.python.org/2/tutorial/classes.html
  - Won't discuss today, but very important.
- Python methods take "args" and "kwargs".

2.1: Write a function called K2F that converts Kelvin to Fahrenheit.

a. Be sure to include all the required documentation
b. Explore what happens when you give it unphysical values...
c. Stretch goal: make the conversion go either way (F2K or K2F), using a flag/ boolean

2.2: Put your function in a new file and import it into your IPython notebook.

a. Create a file called LASTNAME_func.py.
b. Cut/Paste K2F in to it
c. Then try importing and using your function like:
   - `from LASTNAME_func import K2F`
d. Does it still work? (be sure K2F isn't defined elsewhere in the notebook!)
e. Stretch goal: put another function in that same file. Switch your import statement to be more general:
   - `import LASTNAME_func as func`
   - `func.K2F`

Part 3
Constellation Algorithm

Today's real challenge is to implement the algorithm described by N. G. Roman 1987, PASP, 99, 695 (see handout). In the `data/` directory I have included a few files you might find useful:

data/data.txt	The table from Roman (1987)
data/const-names.txt	A list of abbreviations and full constellation names
data/VI-49_bound_20.dat.txt	The full constellation boundary data file from the IAU

3.1: Implement the algorithm from Roman (1987)

a. Skip "step 1", since we'll ignore precession.
   - Note, precession of stars from 1875 to today is significant! Like, ~30 arcmin...
b. Use the provided `data.txt` file
c. Make this it's own method/function!
d. What constellation does Vega lie in? (`ra=18.62, dec=38.78`)
e. What constellation does HD 129078 lie in? (`ra=14.78, dec=-79.03`)
f. Stretch goal: Plot the constellation boundaries and the location of the input star as part of your method.

Thought problem: How would you make this algorithm work with the full IAU boundary file?