Astro Data Science Seminar
Lab 3 - 2016 April 26
Algorithm 2: Try to break your code

The goal today is to learn about testing your code. Good software is written with "unit tests" in mind, which are basic conditions the code must meet every time it is run. Think of these like boundary conditions (e.g. the mass of the Earth from Lab 1 had *better* be close to the known value), or limiting cases (e.g. what happens when I enter a zero into the algorithm?). Writing good tests will ensue that your code works for all possible use cases, and means it's less likely to break when somebody else tries to use it!

Tests can also measure the performance of your code/algorithm. This is very important when you need to run code on a massive dataset, or are putting a job on an expensive supercomputer and need to get answers as quickly (cheaply) as possible.

The best thing to do is write a series of these basic tests that your code must pass, so today that's what we're going to do!

Part 1
Update your Fork (again)

Remember 2 weeks ago you "Forked" the master version of the Seminar repository on GitHub. Now I've again 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, if you didn't last week, add upstream master to your local clone, pull latest version!

Other options include:
- do a *reverse* Pull Request on GitHub to pull the latest version of the master repo back to your Fork
- brute force way to do this is to delete your fork and your local clone, and re-fork it and clone it from scratch

Be sure to "Duplicate" the lab3-example.ipynb file, and make one called lab3-YOURNAME.ipynb

Part 2
Speed Racer!

We'e going to use solutions from the Constellation Algorithm from last week. There are 2 solutions in the "solutions.py" file I have provided.

2.1: If you have a solution from last week, add it to the "solutions.py" file

- Only Holly provided a solution, so she is the winner from last week!
- If you have it, awesome! If not, that's ok.
- Thus you will test either 2 or 3 algorithms. Either is fine

2.2: Write a function to test the speed of the 2 (or 3) constellation algorithms.

- This function should take N as the input argument (the number of coordinates to test),
- The function must generate N random coordinates (both RA and Dec), pass these to the algorithms, and measure the time it takes each
- The function should return 2 (or 3) runtimes.
- Note the Davenport code can accept arrays of coordinates, and the Christenson code cannot.

2.3: Plot the performance as a function of N for the 2 (or 3) algorithms

- You'll need to save the outputs for each N. Maybe do a FOR loop over a range of N values?
- How efficient are the algorithms? Do they do better in different regimes? Do they "flatten" or does runtime keep increasing?
    - Bonus: Look at the algorithms and think about WHY they behave the way they do?

Part 3
Write Some Unit Tests

3.1 Write a function to test if Vega is in the constellation of Lyra (abbreviation is "LYR").

- Be sure to test all 2 (or 3) algorithms.
- Method should return True or False for each algorithm.
- Reminder, coordinates for Vega are (`ra=18.62, dec=38.78`)

3.2 Make a copy of your Speed Racer code, and test if the outputs for each algorithm match.

- Write a function to compare the outputs of the algorithm using random coordinates.
- Pick some fixed number of coordinates to test each time
- Don't return the constellations, just return the fraction that agree!

3.3: Explore the Edge Cases

- What happens to each algorithm when you pass an invalid location (e.g. negative RA)
- What happens to each algorithm when you pass a nonsense input (e.g. a string instead of a float)?
- For one of the solution algorithms (or your own), add error traps for these cases (negative numbers and string inputs)
    - you could use if statements
    - you could use try/except statements
    - Decide if your code will try to convert the bad input to a useful one, or just quit and return an error.
    - Be sure your code prints something to the screen if an error trap is triggered.