Final Project Due Friday, May 3 by 11:59 PM or possibly Saturday, May 4th at final time

Worth **200** points in homework category (equivalent of two homeworks).
These projects are going to be challenging, but that's why you have 2+ weeks with multiple class periods in between to complete them. We recommend getting started early, asking plenty of questions, and saving often.
Keep track of how long this project takes you - we'd like to know for future planning purposes. You'll get 5 points for including amount of time worked in each git commit.

Solar System N-body simulation:

- **Create a simulation of our solar system with the Sun and 8 major planets.** Incorporate a keyword to choose the duration of the simulation, and have an if statement to create a default if the user doesn't specify. Make sure you get the direction of orbit right! `[20%]`

- **Simulation remains accurate for at least 100 years:** Credit is for making sure the simulation is reasonably accurate and representative of reality and the default system stays together for at least 100 years (a counter, either as a title or legend, would be helpful). This will require accurate initial positions (especially velocity) and a sufficiently small timestep. Get polar coordinate positions from `initial_solar_positions.pdf` (you will need to convert to cartesian coordinates). In a circular orbit, the direction of the velocity is always at a right angle from the (current) heliocentric longitude (angle from the sun). *Include notes about how you determined a reasonable timestep.* `[15%]`

- **Implement at least two of the following options** (or your own idea approved by Cameron and Adam), choosable by the user with read or keyword prompts: `[25%] 8%` extra credit per extra implementation up to 24%

- **Interactive way for the user to define mass, position and velocity of a given object**, preferably in terms of ratios. User does not open the file and modify variables. Can use series of read prompts, or keyword arguments.
- **Simulate what happens if a rogue stellar object encounters the system.** The rogue object could be a brown dwarf, white dwarf or black hole came through the solar system. The user should be able to specify mass, position and velocity, but have well-functioning defaults if they don't. You'll probably want to have a separate initial conditions function.
- **Model a real exoplanetary system with at least 5 planets and the star.**
  Go to: `http://en.wikipedia.org/wiki/List_of_multiplanetary_systems`, most star pages linked from there have a table with the exoplanet's mass and position (you may have to guess at the angles. Don't worry about eccentricity). You'll probably want to have a separate initial conditions function.
- **Incorporate the asteroid belt and/or trojan asteroids into the simulation** (this can run by default, doesn't need a keyword). You'll probably want to make use of the random functions instead of explicitly defining the properties of each asteroid.
- **Make a cool visualization**
  - Change the angle of perspective from 90 deg (looking down on solar system) to 0 deg (looking at it from flat to elliptical) and back as simulation runs (make it look good)
  - Zoom in and out of the system and pan around it
  - Track a planet in its orbit, then switch to another planet and track it

- **Understandable, visually pleasing output.** Use plot symbol keyword settings or `draw_circle` function to make Sun and planets easily recognized with size, shape and color. (Rough size scale, doesn't have to be realistic). Animation is not required, but probably useful. `[20%]`

- **Simulation has an informative, understandable and manageable user interface.** Code is well formatted and commented so it could be easily understood by your peers. `[20%]`

Image Analysis/Photometry:

- Make an image photometry package

Create several procedures and functions (called by one parent procedure) that, given an image containing stars, find the position, brightness, and radius of all stars above a brightness threshold.

- Create a function to identify all pixels associated with a given star [20%]
- Create a function to determine the *centroid position* of a star. The centroid is the brightness-weighted average of the positions. [10%]
- Create a function to determine the *brightness* of a star. The brightness is the sum of all the pixels associated with a star. [5%]
- Create a function to measure the radius of each star. The 'effective radius' is given by $r_{eff} = (N_{pixels} * Area)^{1/2}/\pi$, where *Area* is the area of 1 pixel (which would just be 1 pixel^2 in pixel units). [5%]
- The parent function should save the data in a useful data format, e.g. a hash or a struct [10%]
- Make plots of the star positions, brightnesses, and colors [10%]

You will find chapter 20 section 6 (pages 33-47) extremely helpful.

- Implement at least one of the following options (or your own idea approved by Cameron and Adam), selectable by the user with read or keyword prompts: [20%]
(8% extra credit per extra implementation up to 24%)

- Create an output function that will save the data to disk in human-readable format, e.g. a comma-separated-value table, *and* allow the user to sort on any variable!
- Measure the radial profiles of the star data you collected.
- Fit Gaussian profiles to the stars, and save the parameters in a table
- Make an N-body initializer function that uses the measured positions of the stars, replaces their brightnesses with (reasonable guesses) at the corresponding masses, and velocities that can be anything. Use it in your N-body code!

- Simulation has an informative, understandable and manageable user interface. Code is well formatted and commented so it could be easily-understood by your peers. [20%]