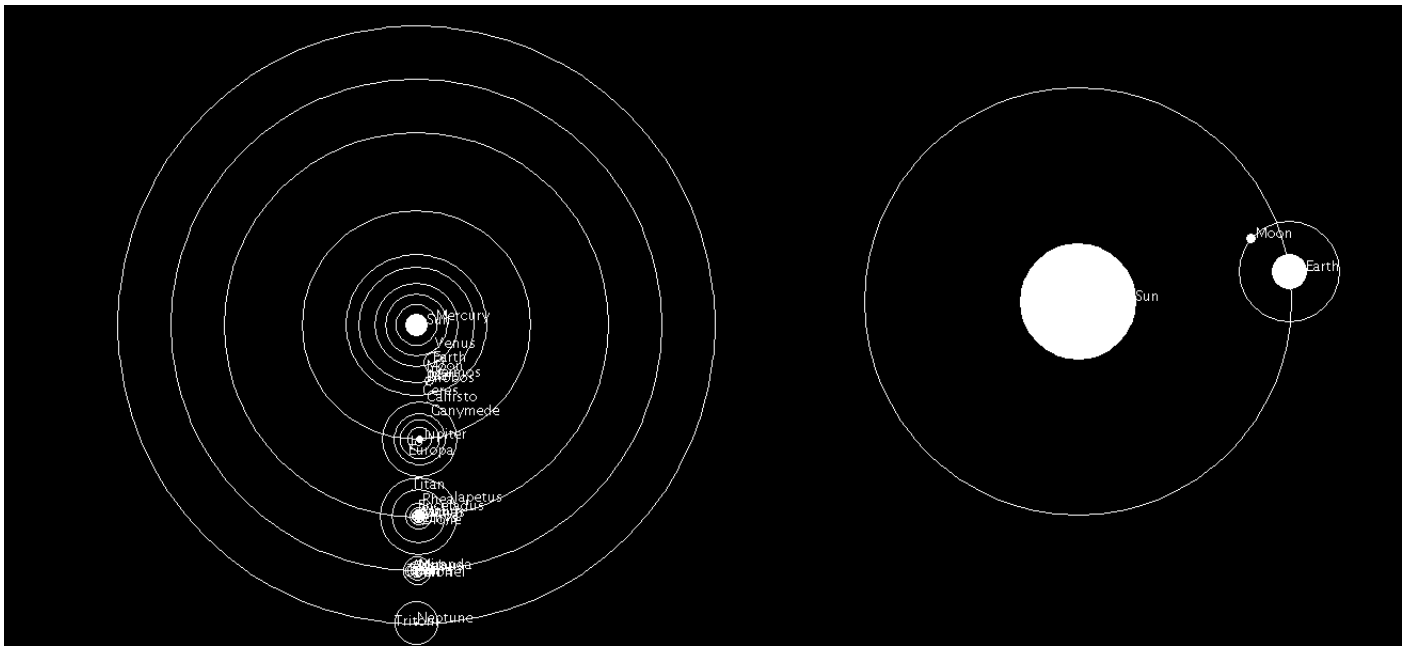
CPSC 231 – Fall 2012, L01/L03
Assignment 4: Drawing A Recursive Animation

Due: *Friday November 23 2012, 4:00PM*

In this assignment, you will exercise your Python skills on recursion. You will also need previous tools such as functions and dictionaries.

An Orbital System



An orbital system can be defined recursively. At its simplest, a single object can exist in space, with no satellites. Essentially, this is the simplest possible system aside from empty space. A more complex system can be built where a single object is orbited by a second, such as the Earth-Moon system. In this case, the Moon follows a path around the Earth, based on its gravity. An even more complex system is that of our solar system, in which (among other objects), Earth orbits the Sun, and in turn, the Moon orbits the Earth. These systems can be defined recursively, where the Earth-Moon system can be viewed as orbiting the Sun.

Data will be provided, defining the objects, the radius of their orbits, their orbital period, and their own radius. Satellites of the object will also be listed. Data will be similar to the following. (These object sizes are inaccurate. Drawing even something as small, astronomically, as our solar system to scale renders even the largest planets as less than a pixel in size. Orbital periods are accurate, however.)

```
Object: Sun
Satellites: Mercury,Venus,Earth,Mars,Jupiter,Saturn,Uranus,Neptune,Ceres,Pluto,Haumea,Makemake,Eris
Radius: 20890260
```

```
Orbital Radius: 0

Object: Earth
Orbital Radius: 77098290
Period: 365.256363004
Radius: 6371000.0
Satellites: Moon

Object: Moon
Orbital Radius: 18128500
Radius: 1737000.10
Period: 27.321582
```

The Orbital Radius represents the radius of the object's circular obit around its parent body. Radius is the size of the body. The period is the time (in days) that the object takes to complete a full orbit. There is no 'EOF' tag in the input file. You will have to handle terminating the file read yourself.

It is strongly recommended that you use a structure similar to a 2D dictionary to store this data. In this way, you may look up an object by its name (eg. data["Sun"]) or name and datum (eg. data["Mercury"]["Orbital Radius"]). This structure is very helpful because an object's satellites are listed by name, so being able to reference an item by name is invaluable. Note: An object may be listed as a satellite, but not defined in the input file. If this is the case, ignore the missing satellite.

For the purpose of this assignment, you may consider all orbits to be circular, and on the same plane.

Your program should also take in the name of the data file on which it runs as a command-line parameter. (Play around with sys.argv and try printing it out as an experiment) For example,

```
$ python a4.py solar.txt
```

should draw the orbital system defined in solar.txt.

```
from sys import argv
```

at the top of your program, you can access the file name provided at command line as argv[1].

A Recursive Drawing Algorithm

Drawing the system should be done recursively. At any particular level of recursion, your function should draw the object in question, as well as make a call to itself to handle the drawing of each of its satellites (if they exist). A circle indicating the object's orbit, as well as a filled circle to indicate the object itself should be drawn. Additionally, each object should be labeled, with its name indicated nearby, or the names placed into a legend. As your data structure will likely be a dictionary, one of the pieces of data your function will probably need is the name of the object that it is running on. In this way, it can reference the relevant data in your dictionary.

The position of a body in an orbit will be determined by a time value, which should increment as your program runs, and thus creates the animation. Combined with the orbital period given in the file, the positions of each object should change according to their own rates.

Sample input and output

Available at

`http://pages.cpsc.ucalgary.ca/~zongpeng/CPSC231-F12/assignments/A4/a4.html`

Submission

Submit your solution by sending an email to your TA, with your solution program in the attachment. Name your program in the form of `a4-sid.py`. For example, if your student ID is 123456, then name your program `a4-123456.py`.

Submissions received after the deadline will not be accepted.

Collaboration

Although it can be helpful to you to discuss your program with other people, and that is a reasonable thing to do and a good way to learn, the program you submit must ultimately be *your own work* that has been written by you independently.

All submissions from the class will be checked by `MOSS`, a computer program that identifies similar programs. Remember, if you are having trouble with an assignment, it is always possible to go to your TA and instructor to get help.