420-LCW-05 Programming Techniques and Applications - Assignment 1

January 29, 2019

This is our first regular assignment in the course. To start, we'll mention some general requirements that will apply to all of these assignments:

1. **Identification section** Do this for *every* Python file in every assignment in this course. This section must be either in a comment, with a '#' preceding each line, or enclosed within triple quotes ("'). The grader and I need this section for the *accurate processing of your assignment*. Assignments missing this may lose up to 5% of the total mark.

   Example:

   ```
   """
   Justin Trudeau, 1234567
   Sunday, February 31
   R. Vincent, instructor
   Assignment 1
   """
   ```

   Obviously substitute your name, Marianopolis ID, and the correct date for the appropriate fields!

2. Always include additional comments with your code. These need not explain every individual line of your program, but consider using comments for the following situations:

   - A brief explanation of a particular variable's purpose, included on the first line where the variable is defined, e.g.:

     ```
     hi = 100 # Define the upper limit of the range.
     ```

   - A note mentioning any website or person you may have consulted with to help with the assignment.
   - A comment describing any constant value that appears in your code.

   In addition, each `def` statement, whether for a global function or class method should include at least a brief docstring.

3. Your submission for assignment will typically include multiple Python files, which have the extension `.py`. Before submission, these files must be combined into a single ZIP archive file (extension `.zip`). If you do not know how to create a ZIP file, I will demonstrate this in lab.

4. Be sure to respect other instructions specified in the assignment. Part of each assignment is to correctly follow the instructions as closely as possible.

Exercise 1 - Star chart and constellations

This section is mostly a review of Python dictionaries and file handling. It is based on a assignment created by Karen Reid at the University of Toronto.

Star catalogs and coordinate systems

Astronomers collect lots of data about stars and there are many catalogs that identify the locations of stars. In this assignment, you will use data from a star catalog to create a picture that plots the locations of stars.

Since a real data set often has some incorrect data and the occasional field missing, a cleaned up catalog has been prepared for your use in this assignment. The file `stars.txt` contains one line for each star that is represented in the catalog of northern hemisphere stars. The meaning of each field (column) is described below.

1. The first field is an identifier which you should ignore.

2. The second field is the Henry Draper number, which is a unique identifier for the star.

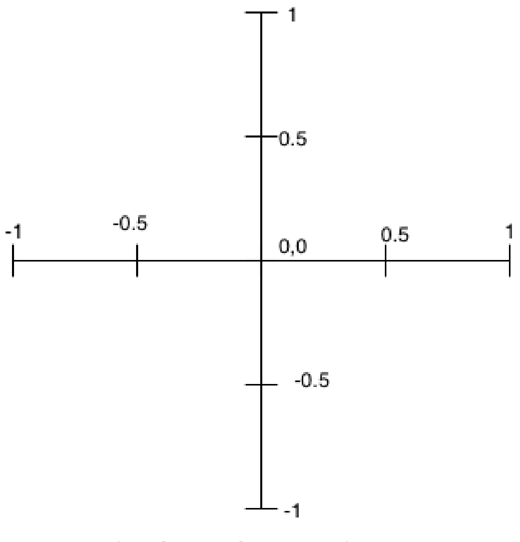
3. The third and fourth fields are the y and x coordinates for the star. Each axis in the coordinate system goes from -1 to +1, with the centre point at 0,0. (See the figure below).

4. The fifth field is the magnitude (or brightness) of the star. Somewhat counterintuitively, the *smaller* the magnitude, the brighter the star. A negative magnitude corresponds to a very bright star.

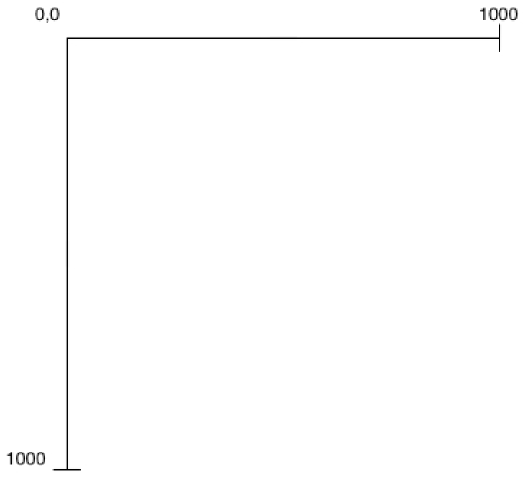
5. The sixth field exists only for a small number of stars and is a standard name for a star.

Two unique identifiers appear in the data because the star data has been collected from different sources, and the catalogs have several different ways to uniquely identify stars. The fields that you will need for this assignment include the x and y coordinates, the magnitude, the Henry Draper number, and the name of each star where available.

The coordinate system used for pixels in a picture has position (0,0) in the upper left corner of the picture, and the maximum x and y values are the height and width of the picture in pixels. In this assignment, all pictures will be square. See below for a comparison of the two coordinate systems.



Star Catalog Coordinate System



Picture Coordinate System
(with an example width and height of 1000 pixels)

Functions to create

I've provided a skeleton file, `star_chart.py`, that implements some key `tkinter` functionality you will need to implement this assignment. Be sure to do all of your work in this file, and submit your modified file.

Translate coordinates

The first task is to write a function `coords_to_pixel(star_x, star_y, size)`. Given the x and y coordinates of a star (`star_x` and `star_y`), and the size in pixels of the picture (which should be 1000), return the x, y location of the star in terms of pixels in the picture. Return this as a tuple with two elements.

For example, if the size of the picture is 1000 by 1000, then the (0, 0) point in the star catalog is at pixel (500, 500), and (0.5, 0.5) from the catalog is at pixel (750, 250).

You should use this as a helper function in the remainder of the assignment.

Read coordinates

Write a function `read_coords(file)`. The single argument is an open text file that contains a star catalog as specified above, return three dictionaries. The first is keyed on the Henry Draper number and the values are tuples containing the x and y coordinates of each star.

The second dictionary is also keyed on the Henry Draper numbers and contains the magnitudes (float) of the stars.

The third dictionary is keyed on the names of the stars and the values are the Henry Draper numbers. If a star does not have a name it will not be represented in the third dictionary.

Plot stars

Write a function `plot_by_magnitude(size, coords, magnitudes)`. This function should plot all of the stars in the dictionaries. Use the given function `draw_star(x, y, radius, color)` to draw the stars. Note that the coordinates used by `draw_star` are pixel coordinates! Experiment with different choices for color and radius. In my example I computed the radius from the magnitude using the equation: $r = \log(8 - m)$. You can improve the look of the star chart by varying the color with magnitude as well. The color argument to `draw_star` is a string, which can be a hexadecimal RGB value such as `'#FF0C1B'`, where the first two digits give the red value, the second two give the green value, and the last two give the blue value. Therefore `'#000000'` is black and `'#FFFFFF'` is white.

Read constellations

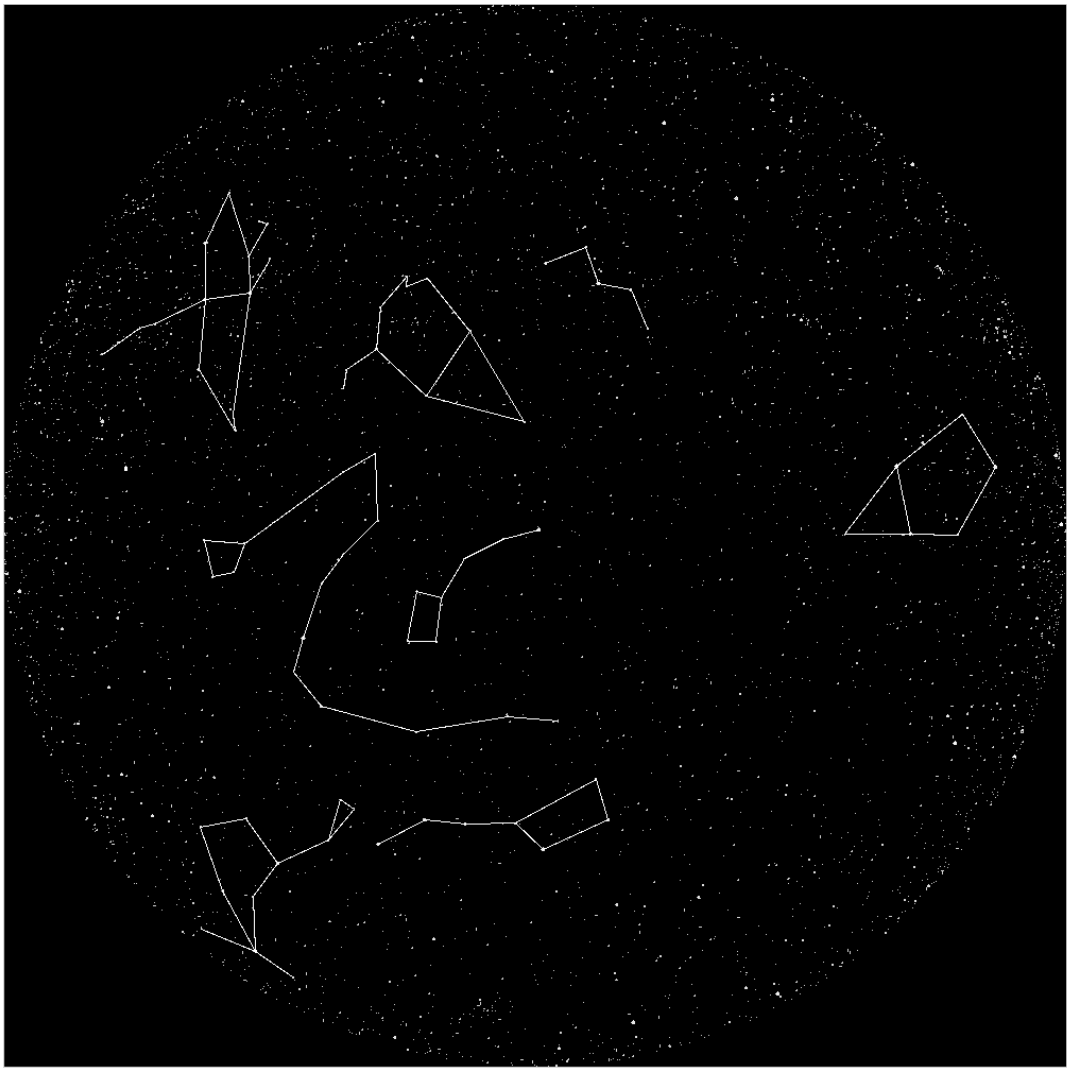
I have provided several text files that describe constellations. Each file contains several lines consisting of two comma-separated values. The values may either be a star name or a Draper number.

Write a function `read_constellation_lines(file)` that, given its argument, an open text file, reads the constellation data. The return value will be a dictionary keyed by the source (first) star name or number, with a value consisting of a list of star names or numbers. For each key, the associated list contains all of the stars connected by a line from the key star.

Draw constellations

Write a function `plot_constellation(coords, lines, names, color, size)` that draws the lines for a given constellation. The first and third arguments are the dictionaries returned by `read_coords`. Thed second argument is the dictionary returned by `read_constellation_lines`. The fourth argument is the color to use in drawing the constellation (you can choose). The fifth argument is the size of the picture (typically 1000). Use the provided function `draw_line(x0, y0, x1, y1, color)` to draw the lines for the constellation.

In your main program, use these functions to read the `stars.txt` file and each of the provided constellation files, then draw the associated stars and constellations. Your final image should look something like the image in the figure below:



Exercise 2 - Stacks

In this exercise, you will write a very simple syntax checker for Python. All your program will do is read a Python file and report whether or not the parentheses, square brackets, and curly braces are all properly matched. Since I don't want to make this too complicated, you do not have to not worry about recognizing strings or comments - you can assume that any curly brace, parenthesis, or square bracket we encounter is part of the program.

You'll want to use a stack to implement this. I have included the simple stack class shown in lecture that is derived from a Python list.

Your program should prompt for the name of a file, open and read the file, then run the syntax checking.

To perform the syntax checking, you will need to push each "left" bracket character onto a stack. Whenever you encounter a "right" bracket, you will check the stack and see if the top is the appropriate corresponding type. If the pair matches, pop the stack.

Your program should detect each of the following situations:

1. Unclosed (missing) brackets, e.g. `(1 + (2 * 3)`
2. Mismatched brackets, e.g. `(0, 1, 2]`
3. Extra brackets, e.g. `1 + (2 * 3))`

Verify that you detect each of these situations, and print a *different*, informative error message in each of the three cases. I have provided several test files `testN.py` that exhibit each of these errors. Be sure to print the line number where the error is detected! If no error is detected, your program should just print a brief message indicating that the checks all passed.

Submitting your work

When you have finished both sections, combine both of your Python files into a single ZIP file and upload that to Omnivox.