Assignment 4

DUE DATE: FRIDAY DECEMBER 9, 2022 AT 11:59PM

Material Covered:

- 2D Arrays & Plotting
- Objects
- Recursion (Bonus marks)

Assignment Guidelines:

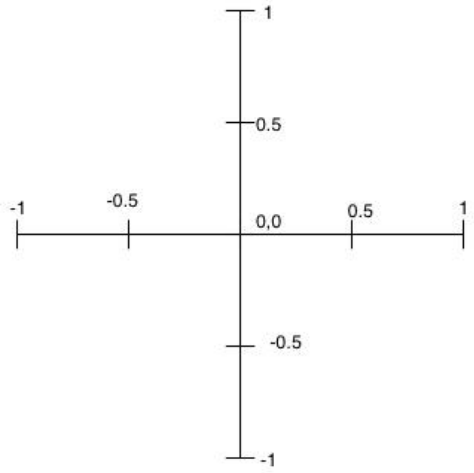
- **All students in this course must read and meet the expectations described in the Expectations for Individual Work in Computer Science (follow the link and read all information provided).**
- Assignments must be completed using course material. Do not use advanced material we have not covered in the course, even if you have prior programming experience. Assignment 4 should be completed using concepts from Weeks 1-10 (+ week 11 for the bonus).
- Assignments must follow the programming standards document published on the course website on UM Learn.
- Submit one .py file per question. Name the files using your name, the assignment number, and the question number, *exactly* as in this example: **`LastnameFirstnameA4Q2.py`**. Use your name *exactly* as shown in UM Learn (without hyphens, if applicable).
- Do NOT submit any input files. Your assignment will be tested with input files that may be different from the ones provided in UM Learn.
- Do NOT zip the files that you submit.
- You may submit the assignment multiple times, but only the most recent version will be marked.
- After the due date and time, a late penalty of 2% per hour, or portion of hour, will be applied. After 49 hours, the penalty is 100% and submissions will no longer be accepted. The date and time of the last file submitted controls the mark for the entire assignment.
- These assignments are your chance to learn the material for the exams. *Code your assignments independently*. We use software to compare all submitted assignments to each other, and pursue academic dishonestly vigorously. You must complete the Honesty Declaration before you will be able to submit your assignment.

Question 1: Star Chart & Constellations [16 marks]

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

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

- The first three fields are the x, y, and z coordinates for the star. We will ignore the z coordinate, and use only the x and y coordinates. Each axis in the coordinate system goes from -1 to +1, and the centre point is 0,0. (See the figure below.) These coordinates have been converted from sky coordinates to values that will give you a plot of the northern sky, with the North Pole at (0,0).



Star Catalog Coordinate System

- The fourth field is the Henry Draper (HD) number, which is simply a **unique** identifier for the star.
- The fifth field is the magnitude (or brightness) of the star. Larger positive numbers (e.g. 4 or 5) indicate fainter stars. Small positive numbers (e.g. 1 or 2) indicate bright stars, and zero and negative numbers (e.g. -1) indicate the brightest stars in the sky.
- The sixth field is the Harvard Revised number, another identifier.
- The seventh field exists only for a small number of stars and is a semicolon-separated list of names for a star. A star may have several names, or no common names and simply be known by its catalogue numbers.

Two unique identifiers appear in the data because the star data has been collected from different sources, and the catalogues 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 (or names) of each star.

Step 1: Build a dictionary of stars

Your first task is to load all 3526 stars into a dictionary. The HD number should be used as the key, and all of the other fields should be packed into a tuple and used as the value. The alternate names for a star should be stored into a list before packing into the tuple.

Prompt the user to enter the name of a text file containing star data. Pass the filename to a function that opens and reads the file, builds the dictionary, and returns the dictionary of stars.

Step 2: Filter by magnitude

Prompt the user to enter a limiting magnitude, representing the faintest stars that should be plotted. Pass the dictionary of stars and magnitude to a function that will identify which stars are brighter than the input magnitude (recall from above that larger magnitudes are fainter). The function should return a 2D array containing the stars that will be plotted in step 3. Column 0 should contain the x coordinates, column 1 should contain the y coordinates, and column 2 should contain the magnitudes. The number of rows should exactly match the number of stars that meet the magnitude criteria.

Step 3: Plot the stars

Create a plot displaying all of the stars that met the magnitude criteria in step 2. Import matplotlib.pyplot, for example:

```
import matplotlib.pyplot as plt
```

Use a scatterplot to display all of the stars. Usually when plotting star charts, brighter stars are represented by larger dots, and you will follow this convention. The line

```
plt.scatter(starX, starY, s=starSize, c='k')
```

includes two options for the scatterplot. s is a list of sizes for the dots that will be drawn. The list should be the same length as the list of x coordinates and the list of y coordinates. c is the colour to use when plotting ('k' is black, 'b' is blue, 'r' is red, etc.)

You will need to calculate a list or 1D array of sizes (one size per star to be plotted). Because a larger magnitude indicates a fainter star, we need to use something such as `size=(5-magnitude)` to give us a larger number for brighter stars. (The faintest stars visible in the night sky are magnitude 5.) You will then need to multiply by a constant factor, or square the result, to get a noticeable difference between dot sizes on the plot. Play with your formula, and find something that is pleasing to you.

Step 4: Plot a constellation

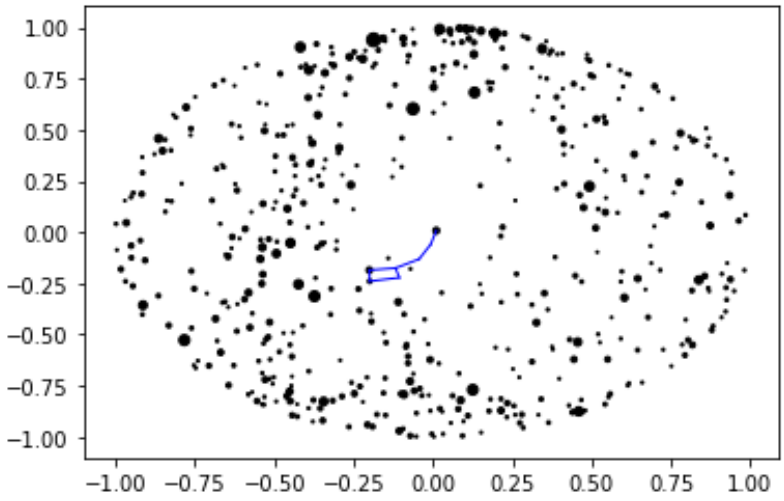
Prompt the user to enter the name of a file containing a constellation. (Several have been provided in the Assignment 4 folder.)

For each line in the file, add a line to your star chart. Each line in a constellation file contains common names for two stars, separated by a comma. You will need to look in your star dictionary for the stars that have those names in their list of names, and extract their (x,y) coordinates. To draw a line on your plot, use something similar to

```
plt.plot([x1,x2], [y1,y2], lw=1, c='b')
```

In the above line, `[x1,x2]` is a list of x coordinates, `[y1,y2]` is a list of y coordinates, `lw` specifies the line width, and `c` specifies the colour for the line. Note that if you list more that two coordinates, lines will be drawn from one point to the next. However, the constellations are not necessarily made up of lines where the end of one line is the start of the next line. You will need to draw one line on the plot for each line in the constellation file.

Your final plot should look similar to the plot below.



Sample console output (**`black text`** printed by program, **`green text`** entered by user):

```
Please enter the name of the file containing the stars > stars.txt
Please enter the limiting magnitude > 4
Please enter the name of a constellation file > UrsaMinor_lines.txt
Program Terminated Normally.
```

Question 2: Canoe Routes [24 marks + 5 bonus]

A canoe trip consists of paddling across lakes, and hauling gear across portage trails between lakes. In this question you will create a representation of lakes and portages in a park. You will then provide a means for park staff to obtain the total distance of all portages (e.g. to plan staffing to maintain trails) and to determine whether there is a trail between two given lakes (e.g. to assist patrons inquiring about possible trips). Optionally (for bonus marks), you will provide a means for park staff to determine whether a route exists between two lakes.

Warning: This question is longer than most you have written (~150 lines plus ~50 lines for the bonus). Use incremental development. Test each method as you write it. Thorough testing means that when you run into a bug, you should know exactly which method is causing the problem. When writing longer methods, test those as they are developed, so that, for example, you know that the bug is in one loop rather than anywhere in the method.

Read through this question in entirety and plan your code on paper first before coding.

The Lake Class [5 marks]:

- A Lake object has a name (a string), a water quality rating (a string, such as 'excellent' or 'poor'), and a list of all Portages that lead out of that lake (see Portage class below).
- The Lake class should have the following methods:
    - a constructor that, given a lake name and rating, will create a Lake object.
    - a method to print the object, in a readable format that includes all of the information that you have on the lake, including its name, water quality rating, and list of neighbouring lakes. Example: "Crabclaw has excellent water quality, and is connected to: Manomin, Upper Stewart"
    - a method that will return the lake name.
    - a method that will return the lake water quality.
    - a method, named addPortage, that given a Portage object will add it to the list of Portages.
    - a method that will return the list of Portages.

The Portage Class [3 marks]:

- A Portage object has a distance (a float, in km) and two references to Lake objects.
- The Portage class should have the following methods:
    - a constructor that, given references to two Lake objects and a distance, will create a Portage object.
    - a method to print the object, in a readable format that includes all of the information that you have on the portage, including the names of the lakes it connects and its distance. Example: "Manomin and Crabclaw are connected by a portage of 0.6 km."
    - a method that will return the portage distance.
    - a method that will accept one lake name and return the name of the lake at the other end of the portage (or a string containing an error message if the provided lake name is not actually one of the two lakes on that portage).

Load Data From Files [7 marks]:

- Your mainline should begin by loading the lake data into a list of Lake objects.
    - Ask the user to enter the name of a file containing lake data.
    - Read the file one line at a time, creating one Lake object for each line in the file. Add each Lake to the list of Lakes.
        - Each line in the file consists of a lake name and a water quality rating, separated by a comma. You can assume that every line will contain a name, a comma, and a rating.
- Next, load the portage data.
    - Ask the user to enter the name of a file containing portage data.
    - Read the file one line at a time. For each line:
        - Identify the lake names and distance. Each line consists of two lake names, and a distance, with commas between the three. You can assume that each line in the file contains 3 pieces of information. However, you should verify that you actually have Lake objects with the provided names before creating a Portage object. If you do not have a Lake object for one or both of the lake names, do not create a Portage object, and print an error message instead.
        - Create a Portage object from the data on the line. Remember that the Portage object has *references* to two Lakes, not simply the lake names.
        - You need to update the Lakes so that they have information on the portage. For each of the lakes, use the addPortage method in your Lake class to add the Portage to the list of Portages for each of the two Lakes.
        - There should NOT be a list of all Portages in the park – all of the portage information will be accessible via the list of Lakes.
- At this point, print your list of Lakes, and verify that the data has been correctly loaded.

Interactive Behaviour [9 marks + up to 5 bonus marks]:

- After loading data from files, the program will ask the user what they want to do, over and over, until the user decides to quit.
- Ask the user to enter **q** to quit, **d** to get the total portage distance, **e** to test for the existence of a portage, and (optionally, if you implement the bonus) **r** to test whether a route exists.
- Use functions to break up your program into smaller pieces, rather than having a long mainline.
- When the user enters **q**, the program should end. Print a message to indicate that the program ended normally.
- When the user enters **d**, calculate the total distance of all portage trails in the park, and print a message with the result.
- When the user enters **e**:
    - Ask the user to enter two lake names.
    - Determine whether there is a portage between those two lakes, and print a message with the result. (As stated above, there should not be a list of all portages in the park. You need to get the portage information through the list of Lakes.)

- (Bonus) When the user enters **r**:
  - Ask the user to enter two lake names.
  - **Use recursion** to determine whether there is a route between the two lakes. You only need to determine whether a route exists – you do not need to print the route, and you do not need to find the shortest possible route.
  - Consider the base case(s) – when do you know that a route does or does not exist?
  - Consider the recursive case – how can you create a smaller version of the same problem?
  - *Note: This problem is more difficult than anything you will be expected to do on a quiz, test, or exam in this course. As a bonus problem, you need to design the recursive function (and any helper functions) yourself.*

Sample program output:

Note that this is generated with a smaller dataset than is provided for your testing.

(**black text** printed by program, **green text** entered by user)

```
Please enter the name of the lake file > lakes.txt
Please enter the name of the portage file > portages.txt


Please enter an option (q to quit, d for portage distance, e to test for
existence of portage, r to find route: d
The total distance of all portage trails is 3.20 km.


Please enter an option (q to quit, d for portage distance, e to test for
existence of portage, r to find route: e
Please enter the name of the first lake: Eagle
Please enter the name of the second lake: Crabclaw
There IS a portage between Eagle and Crabclaw.


Please enter an option (q to quit, d for portage distance, e to test for
existence of portage, r to find route: e
Please enter the name of the first lake: Falcon
Please enter the name of the second lake: Crabclaw
There is NOT a portage between Falcon and Crabclaw.


Please enter an option (q to quit, d for portage distance, e to test for
existence of portage, r to find route: e
```

```
Please enter the name of the first lake: hello
Please enter the name of the second lake: Falcon
ERROR: 'hello' is not the name of a known lake.


Please enter an option (q to quit, d for portage distance, e to test for
existence of portage, r to find route: r
Please enter the name of the first lake: Upper Stewart
Please enter the name of the second lake: Manomin
There IS a route from Upper Stewart to Manomin.


Please enter an option (q to quit, d for portage distance, e to test for
existence of portage, r to find route: r
Please enter the name of the first lake: Falcon
Please enter the name of the second lake: Manomin
There is NOT a route from Falcon to Manomin.


Please enter an option (q to quit, d for portage distance, e to test for
existence of portage, r to find route: q
Program terminated normally.
```

[Programming Standards are worth 8 marks]