# CPSC 231 Assignment 3

**Due Date**: Friday, November 8th, 2019, at 11:59 PM

**Weight**: 8%

**Collaboration**

Discussing the assignment requirements with others is a reasonable thing to do, and an excellent way to learn. However, the work you hand-in must ultimately be your work. This is essential for you to benefit from the learning experience, and for the instructors and TAs to grade you fairly. Handing in work that is not your original work, but is represented as such, is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. Cite all sources of code that you hand-in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:
   ```
   # the following code is from
   https://www.quackit.com/python/tutorial/python_hello_world.cfm.
   ```
   Use the complete URL so that the marker can check the source.
2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. However, you may still get a low grade if you submit code that is not primarily developed by yourself.
3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's console, then this code is not yours.
4. **Collaborative coding is strictly prohibited.** Your assignment submission must be strictly your code. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing code itself, or modelling code after another student's algorithm. **You can not use (even with citation) another student's code.**
5. We will be looking for plagiarism in all code submissions, possibly using automated software designed for the task. For example, see Measures of Software Similarity (MOSS - https://theory.stanford.edu/~aiken/moss/).

Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize.

**Late Penalty:**

Late assignments will not be accepted.

**Submission Instructions:**

Your program must be submitted electronically. **Use the Assignment 3 dropbox in D2L** for the electronic submission. You can submit multiple times over the top of a previous submission. Don't wait until the last minute to attempt to submit. You are responsible if you attempt this and time runs out.

## Description (Constellation drawing)

In this assignment, you will create a program that plots stars and constellations from input files. These files, for the most part, will be entered via command-line arguments. However, there will be an optional mode where a filename may be entered via a user prompt for input.

Expected usage:

**python CPSC231A3Zaamout.py**
(prompt for a *stars-location-file*, then loop prompting for valid *constellation files* until the user enters an empty string (“”))

**python CPSC231A3Zaamout -names**
(prompt for a *stars-location-file* and when drawing named stars write their names to drawing window, then loop prompting for valid *constellation files* until “” entered”)

**python CPSC231A3Zaamout.py <arg1>**
(use **arg1** as *stars-location-file*, and loop prompting for valid *constellation files* until “” entered)

**python CPSC231A3Zaamout.py <arg1> -names**
(use **arg1** as *stars-location-file* and when drawing named stars write their name to drawing window, loop prompting for valid *constellation files* until “” entered)

**python CPSC231A3Zaamout.py -names <arg2>**
(use **arg2** as *stars-location-file* and when drawing named stars write their name to screen, loop prompting for valid *constellation files* until “” entered)

**python CPSC231A3Zaamout.py <arg1> <arg2>**
(exit program with descriptive error indicating invalid argument as neither input was **-names**)

**python CPSC231A3Zaamout.py <arg1> <arg2> <arg3>**
(exit program with descriptive error indicating too many arguments)

If a *stars-location-file* does not exist, then the program should exit after printing a descriptive error message. On the other hand, if an invalid *constellation file* is entered (information about how to check for valid files is detailed in the appropriate sections below), the program should re-prompt until a valid filename is given or “” is entered to exit the loop. **sys.exit(1)** can be used for exiting the program at a specific point with error_code=1.

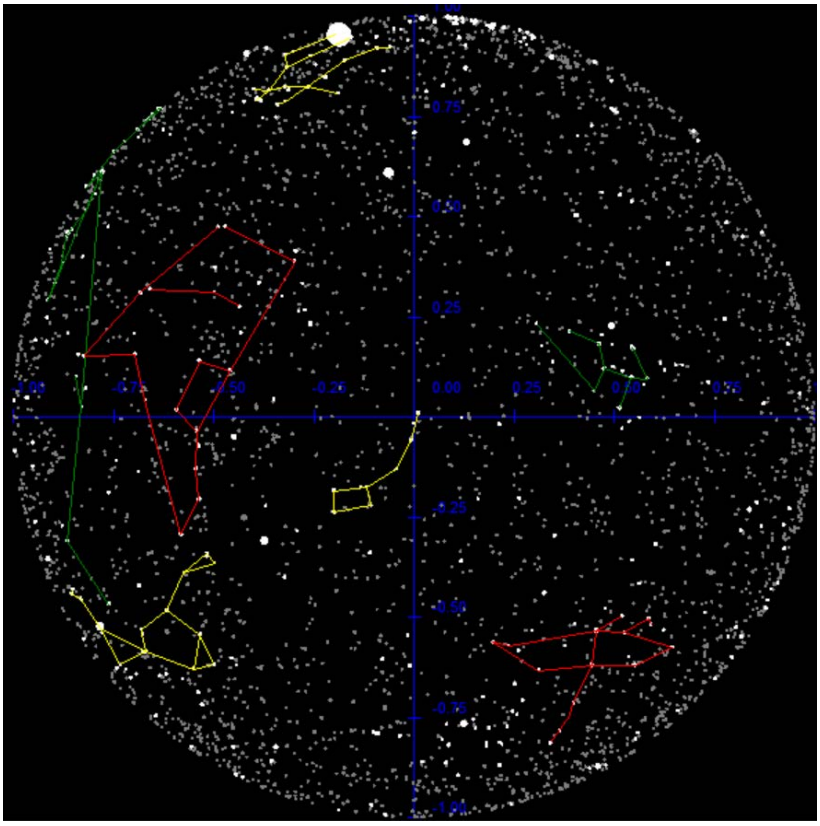
You can expect the filenames to be in the correct format and for every star named in a *constellation file* to exist in the *stars-location-file*. All (x,y) coordinates and the magnitude for stars will be floating-point values. The background colour should be **black**; the axes should be drawn in **blue**, the named stars in **white**, and the un-named stars in **grey**. The constellation edges will be drawn in **red**, **green**, or **yellow**. Each new constellation will be drawing using a different one of these colours. Once all 3 colours have been used, the next constellation will cycle back to use the first colour again.

Completing this assignment will require the use of **system arguments, file input, functions, data structures, error handling, loops**, and **if** statements. You will have to write the complete program

yourself for this assignment. **Global variables are explicitly not allowed**, so you must have a main function, and you must pass all the information you need in and out of the functions using parameters and returned values. The only variables allowed outside of functions are your constants. All of your functions must be properly documented with a description, parameter information, and return information.

**The following sections provide additional requirements** and suggestions for implementing your program. Example output for stars_all.dat and all *constellation files*:

```
python CPSC231A3Solution.py stars_all.dat
Enter constellation filename:BigDipper.dat
Enter constellation filename:Bootes.dat
Enter constellation filename:Cas.dat
Enter constellation filename:Cyg.dat
Enter constellation filename:Gemini.dat
Enter constellation filename:Hydra.dat
Enter constellation filename:UrsaMajor.dat
Enter constellation filename:UrsaMinor.dat
Enter constellation filename:
```



## Input Files

You will be provided with input files. **stars_all.dat** is a *stars-location-file* that contains an extensive list stars and their information, **stars_named.dat** is another version that only contains the named stars. **stars_10.dat, stars_5.dat, stars_1.dat** contain 10%, 5%, and 1% of the stars that are in **stars_all.dat** with all the necessary named stars to draw the provided constellations.

The *stars-location-files* are CSV (comma-separated value file). Each line in a *stars-location-files* contains the information about one star. The format is as follows.
**x,y,z,id1,mag,id2,names**
You will only need to use the **x**, **y**, **mag,** and **names** data for each star. **(x,y)** are chart coordinates between **-1 <= x <= 1 and -1 <= y <= 1**. The magnitude **mag** is a value **-2 > mag <= 10.5**. **names** contains a list of 0, 1, or more string identifiers (names) for a star, each name is separated by a semi-colon ";".

You will be provided with a selection of *constellation files*. Each *constellation file* describes a constellation with a name and a list of constellation edges. Each constellation edge consists of two named stars, which can be found in the *stars-location-file*. The *constellation file* is a CSV file (comma-separated value file). The first line of the file will be the constellation name, and the following line(s) will each be a pair of star names separated by a comma.

## Setup Stage

You must use this setup function to create your original window. Not using this setup function will result in lost marks. The WIDTH and HEIGHT constants should be 600,600 and the BACKGROUNDCOLOUR constant should be "black".

```
def setup():
    pointer = turtle.Turtle()
    screen = turtle.getscreen()
    screen.setup(WIDTH, HEIGHT, 0, 0)
    screen.setworldcoordinates(0, 0, WIDTH, HEIGHT)
    pointer.hideturtle()
    screen.delay(delay=0)
    turtle.bgcolor(BACKGROUNDCOLOUR)
    pointer.up()
    return pointer
```

**You are not allowed to have any code outside functions, except for import statements, and constants. The only exception is the call to your main function, so you must also have a main function. Moreover, you are not allowed to use global variables unless they are constants.** You should only need to import sys and turtle libraries. Your main function should deal with your system arguments, call your setup function to get your turtle pointer, and then call other functions that perform the reading of your input files and the drawing of the required information. You are free to use as many functions as you decide; however, there is a minimum required. You are expected to have at least:

1. A main function that handles the input
2. A function that reads an input file and returns a data structure with star information (a list of star tuples and dictionary with names as keys and star tuples as values is recommended)
3. A function that draws your axes.
4. A function that draws your stars and possibly draws the names of the named stars
5. A function that reads a *constellation file* and returns the constellation name, and a list of edges which are easiest to store as tuples of stars names.
6. A function that draws a constellation given its name, edges, and the dictionary of star tuples stored by name.

7. Any additional functions you may need for the bonus; if you decide to solve the bonus. You must implement the bonus using additional functions. You may call the additional functions from existing functions, but you may not change the existing functions to solve the bonus.

**Submissions that do not contain at least the first 6 functions will be penalized.**

## Command-line Arguments

You should check the command-line arguments (stored in **sys.argv**) as described earlier in this document in your main function, or a function called from your main function. If there is an error in the command-line arguments, your program should print a descriptive message and exit. Your program should only draw something if your arguments are correct, and the star information can be loaded from the given filename. You should use **os.path.isfile(filename)** to determine if a filename is a valid file. It is recommended that you don't solve the rest of the assignment (reading the input files and drawing) until you have correctly implemented the command-line argument handling as described earlier in the assignment.

## Read Star Information

This should be accomplished within a function. This function can make use of others if you like, but can also be self-contained.

If the information in the star filename is of the wrong type or doesn't have the required amount of entries separated by commas, you should print a descriptive error and exit. Remember to use try/except blocks to catch any errors during the process of opening/closing a file, and while reading the star information.

You should be able to use string.split(",") and string.split(";") to split the star information, and then the names into lists of strings. Remember, you only need the (x,y) information and magnitude for a star which can be stored as a tuple (x,y,mag).

You will want to return a list of all these tuples to your main function. You will also want to return a dictionary that stores a tuple of star information as a value using the name of the star as a key. This information will help the constellation drawing function access the location information of the stars that define the endpoints of constellation edges.

Your star reading function should print the following information for every named star in the input file. Information about un-named stars should not be printed:
**NAME is at ( x , y ) with magnitude mag**
ex.
**ALPHERATZ is at ( 0.873265 , 0.031968 ) with magnitude 2.07**
**CAPH is at ( 0.512379 , 0.020508 ) with magnitude 2.28**
**…**

## Drawing the Axes

This should be accomplished within a function. This function can make use of others if you like, but can also be self-contained.

Your program should draw the axes in blue. When you draw the axes, they must be oriented so that the centre of the Cartesian coordinate system (0, 0) is at the center of the screen (300, 300). The star coordinates will be given such that the x and y values are between -1 and 1. The screen height is 600 pixels. **Therefore, each unit in the Cartesian coordinate system must be represented by 300 pixels**. Both axes will go from -1 to +1. The step size for the axis ticks will be 0.25. For example, the x-axis would be labeled with $-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1$.

Your set of axes should include a tick mark and a label for each unit (you can optionally omit the 0s at the origin for a cleaner look). While it is possible to create the axes using a (long) sequence of function calls alone, **you must use loops appropriately** to keep the total number of lines of code to a more reasonable amount. Draw your axes in blue on a black background created by the setup() function so that the constellations will be easy to see when drawn on top of them.

## Drawing the Stars

This should be accomplished within a function. This function can make use of others if you like, but can also be self-contained.

Your program should draw every star in the input file. Each star without a name should be drawn in "grey" and every named star in "white". You should be able to do this by looping through your list of stars returned from your star information reading function. Draw every star in "grey", then loop through all the keys in your named star information dictionary and draw them in "white". If the user has given the argument to write the name of a star you can write it using
**pointer.write(name,font=("Arial", 5, "normal"))**
this should keep it small enough.
Each star should be drawn as a circle. To get the radius of the circle, you can use
**radius = (10 / (magnitude + 2)) / 2**
you will also want to use
**pointer.begin_fill()**
**…#Draw circle for star**
**pointer.end_fill()**
to draw your stars as filled circles.

## Reading a Constellation File

Reading the *constellation file* should be accomplished within a function. This function can make use of others if you like, but can also be self-contained.

When the drawing of stars is complete, your program will enter a loop prompting the user for a valid *constellation file* from which it can read constellation information. This loop should exit only when the user enters "", or if the contents of the input file are incorrect. If the user gives a filename such that
**os.path.isfile(filename) == False**
the program should continue re-prompting for a valid filename.

If the information in the file doesn't have the required amount of entries separated by commas, you should print a descriptive error and exit. Remember to use try/except blocks to catch any errors during the process of opening/closing a file, and while reading the constellation information.

You should be able to use string.split(“,”) to split the constellation information. Remember, the first line of the file is the constellation name, and the remaining lines are pairs of named stars. Each of these lines represents a single constellation edge. The information for an edge can be easily stored as a tuple (star_name1,star_name2).

You will want to return a list of all these tuples to your main function. You will also want to return the constellation name.

Your constellation reading function should print the following to the console for every constellation read: the name of the constellation and every star named in its edges. No star name should be printed more than once.
Ex.
**Enter constellation filename:BigDipper.dat**
**BIG DIPPER constellation contains {'BENETNASCH', 'PHECDA', 'DUBHE', 'MEGREZ', 'MIZAR', 'MERAK', 'ALIOTH'}**

## Drawing the Constellation

Drawing the edges of a constellation should be accomplished within a function. This function can make use of others if you like, but can also be self-contained.

For each constellation, your function must loop through every pair of stars that constitute its edges. To draw an edge, you will need 4 values: an (x,y) coordinate for the start of the edge and an (x,y) coordinate for the end of the edge. To get these, you should look up the named star in the star information dictionary to get the tuple of (x,y,mag) star information.
The colour of a constellation line will be either red, green, or yellow. Each constellation will have its edges drawn in one colour, and the next constellation will cycle to the next colour. You can look at Assignment 2 for an idea of how to use a counter to accomplish the cycling of constellation colours.

## Additional Specifications:

Ensure that your program meets all the following requirements:

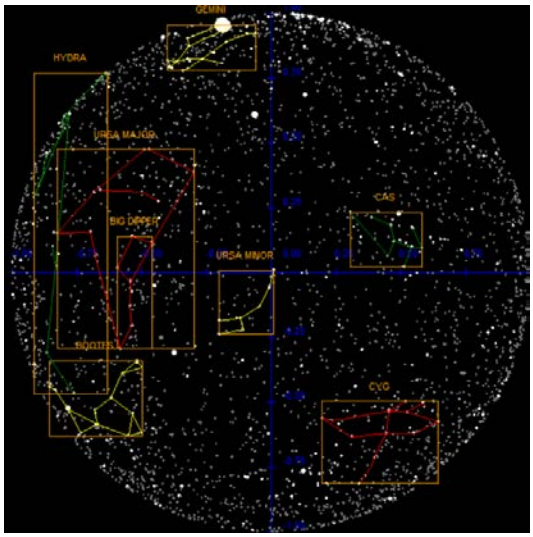
- You should have the class, your name, your tutorial, your student id, the date, and description at the top of your code file in comments. Marks are given for these.
- The code file should be **CPSC231A3<LastName>.py** (ex. Mine would be CPSC231A3Zaamout.py)
- Import the necessary libraries
- Use constants appropriately. Your TA may note one or two magic numbers as a comment, but more will result in lost marks.
- Use descriptive naming of variables. The use of variables that follow the math descriptions given is fine for those purposes.
- Use the correct colouring as requested.
- Use in-line comments to indicate blocks of code and describe decisions or complex expressions.
- You should comment your functions descriptively. (see Assignment 2)
- Break and continue are generally considered bad form. As a result, you are **NOT** allowed to use them when creating your solution to this assignment. In general, their use can be avoided by

using a combination of if statements and writing better conditions on your while loops. You are allowed to use return within a loop inside a function, or sys.exit(1) in a loop of a function when finding an error to exit.

- You will have to perform error checking when reading an input file. Your program should never crash with a Python error. All errors should be avoided by checking if the file exists before opening it or caught by try/except structures if there is an error in a file. Descriptive messages should be produced instead of errors.

## Bonus:

Looking for an A+? Improve the program so that each constellation has a box drawn around it. This box should contain within it every edge and star drawn. You can use padding, such as 15 pixels, to make the box slightly bigger than the constellation inside of it. Draw the constellation name (obtained from the *constellation file*) next to the box. Use a different colour that can be seen easily (such as orange) to differentiate this box from constellations/stars/axes. (Hint: you will need to track the largest and smallest x and y coordinates for each constellation, and then use these 4 values to draw a box around each of the constellations.). Write the min x, max x, min y, max y that define this box out to file <Name>_box.dat where <Name> is the name of the constellation as shown in its corresponding input file.



## Grading:

This assignment will be graded on a combination of functionality and style. A base grade will be determined from the general level of functionality of the program (Does it handle arguments correctly? Does it read input files correctly? Does it draw correctly? Does it print the requested output correctly?). The base grade will be recorded as a mark out of 12.

The assignment will be graded out of 12, with the grade based on the program's level of functionality and conformance to the specifications. The program must not have syntax errors to get more than an F. A program with runtime errors that occur **during proper usage** of the program, every time it runs, will not get better than a C grade. Runtime errors are your program crashing.

Your TA will begin by grading your code with a general functionality grade starting point and subtract marks when smaller specifications are unfilled.

The total mark achieved for the assignment will be translated into a letter grade using the following table:

| Mark | Letter Grade | Starting Point Guidelines (not a final grading scheme!) |
|---|---|---|
| 13 | A+ | Appears to fulfill assignment and bonus spec |
| 12 | A | Appears to fulfill assignment spec |
| 11 | A- | |
| 10 | B+ | |
| 9 | B | Can read files and draw stars but not constellations (one component missing) |
| 8 | B- | |
| 7 | C+ | |
| 6 | C | Only handles arguments and drawing axes (couple large components missing) |
| 5 | C- | |
| 4 | D+ | |
| 3 | D | Only some of arguments/file reading/axes (some basic components work) |
| 0-2 | F | Syntax errors or barely started code |