CPSC 217 Assignment 4

Due: Friday December 6, 2019 at 11:55pm
Weight: 7%
Sample Solution Length: Approximately 110 lines (not counting the comments)

Individual Work:

All assignments in this course are to be completed individually. Students are advised to read the guidelines for avoiding plagiarism located on the course website. Students are also advised that electronic tools may be used to detect plagiarism.

Late Penalty:

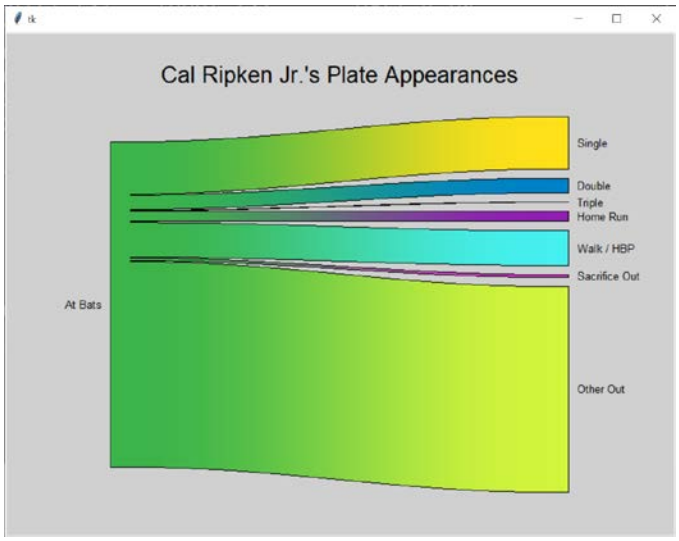
Late assignments will not be accepted.

Submission Instructions:

Submit your .py file electronically to the Assignment 4 drop box in D2L. You don't need to submit the data files or SimpleGraphics.py – we already have them.

Description

A Sankey diagram is a flow diagram where the width of each arrow is proportional to the amount of flow. A variation of such a diagram can be used to show how a source value is split into different destination components. For example, the baseball data set on the course website shows how Cal Ripken Jr.'s 12,883 at bats (the source) are divided among singles, doubles, triples, home runs, walks / hit by pitches, sacrifice flies / bunts, and other outs (the destinations). (Don't worry if you aren't familiar with baseball. What the different outcomes are isn't important to the assignment. It's just important that there are many different outcomes).



Your task is to build a program that reads data values from a file and draws a Sankey diagram (like the one pictured earlier in this section) that represents those values by following the steps outlined below.

Part 1: Getting Started

Several data sets can be found on the course website. Each data set is stored in a text file where the first line in the file is the title for the graph and the second line in the file is the label for the source value on its left side. These two lines are followed by a variable number of lines, each of which consists of the name of a destination category that is to appear on the right side of the diagram and the amount of flow to that category.

Begin by writing a program that opens a file for reading, displays the title, and displays the label for the source value on the left side of the graph. The name of the file will be provided as a command line argument to your program. If no command line argument is provided then your program should prompt the user and read the name of the file. Your program should quit with an appropriate error message if the user provides more than one file name on the command line, or if the file specified by the user (either on the command line or read using the input function) doesn't exist.

My solution to part 1 of the assignment is approximately 25 lines of code (including blank lines, without any comments). The output that should be displayed for this part of the assignment when the baseball data set is processed is shown below.

Part 2: Loading the Data

All of the data values must be loaded from the file and stored into a data structure before any of the data can be plotted. A dictionary is a good data structure to use in this case because the destination names can be the keys and the amounts of flow to the destinations can be the values. You will construct a dictionary in this part of the assignment and then draw the diagram using the values in the dictionary in the next parts of the assignment.

The lines in the file that contain data begin with the name of the destination, followed by the amount of flow from the source to that destination. The destinations are strings that can contain any character except for a comma. The flow is a floating-point number. These two values will be separated by a comma. Each destination will be unique.

Create a function that implements the functionality needed by this part of the assignment. The function will take a file object as its only parameter. This file object will have previously been opened for reading, and the first two lines from the file (the graph's title and the label for the source) will have already been read from it before it is passed to the function. The function will return a dictionary containing all of the data loaded from the file as its only result.

Once the function has been returned you should print out the dictionary that it returned so that you can verify that all of the data was loaded and stored into the dictionary correctly. The dictionary for the baseball data set is shown below:

{'Single': 2106.0, 'Double': 603.0, 'Triple': 44.0, 'Home Run': 431.0, 'Walk / HBP': 1402.0, 'Sacrifice Out': 137.0, 'Other Out': 8160.0}

My solution to Part 2 of the assignment is approximately a dozen lines of code (including blank lines, but without any comments) beyond what was needed for Part 1.

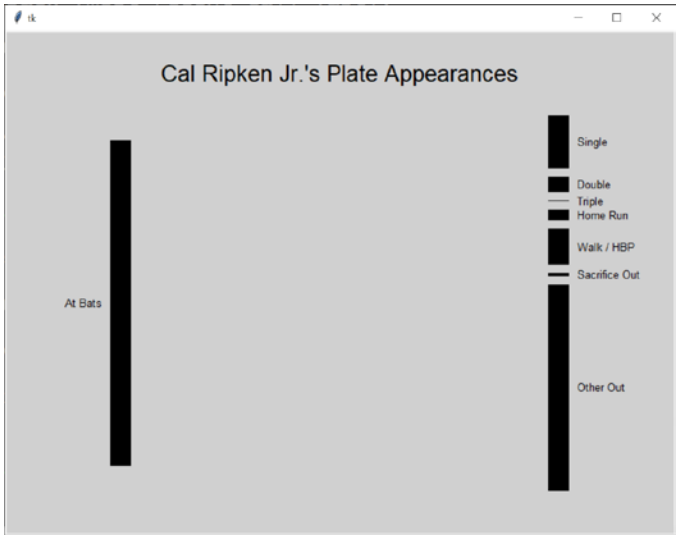
Part 3: Plotting the Source and Destinations

In this part of the assignment you will begin to write a function that draws a Sankey diagram. Your function will take a dictionary holding destinations (keys) and amounts of flow (values) as its only parameter. Your function will not return a result.

The source will be plotted on the left side of the graph and the destinations will be plotted on the right side of the graph, with the number of pixels allocated to the source and each destination being directly proportional to its magnitude. In addition, a small (10 pixel) gap should be left between each destination to make it easy to determine the relative sizes of the destinations, and the bar for the source should be centered within the vertical space needed for the destinations. The height of the bar drawn for each destination can be calculated in the following manner:

- Determine the total flow to all of the destinations. This should be 12883 for the baseball data set.
- Compute the number of available pixels as 450 – (number of destinations - 1) * 10. This should be 390 for the baseball data set.
- Compute the number of pixels per unit of flow as the number of available pixels divided by the total flow.
- Compute the height of each destination rectangle as the amount of flow to that destination multiplied by the number of pixels per unit of flow. The height of the source bar can be computed as the sum of the flows to all destinations, multiplied by the number of pixels per unit of flow.

The result of performing this task for the baseball data set is shown below. Notice that the labels have been drawn for each destination, centered vertically, next to the rectangles.

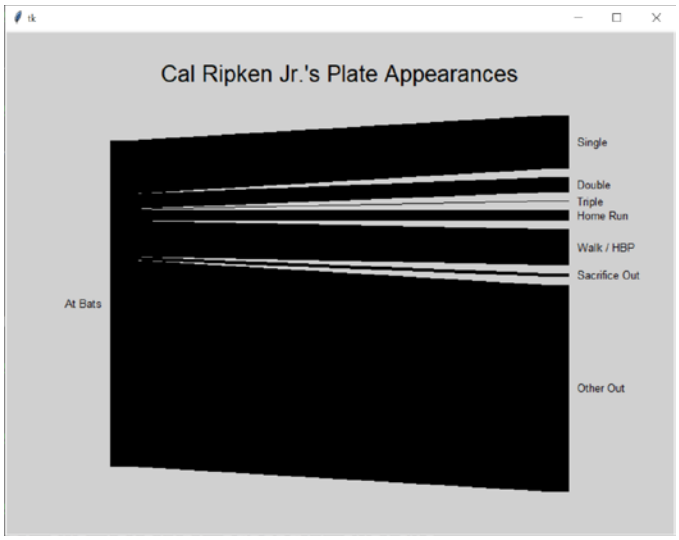


My solution to Part 3 of the assignment is approximately 20 lines of code (including blank lines, but without any comments).

Part 4: Connecting the Sources to the Destinations

A polygon can be used to connect the source to each destination. In order to draw the polygon the y position within the source and the y position of the destination must both be known. As a result, you will probably find that you want one variable to keep track of the y position at the source and a second variable to keep track of the y position at the destination. As each polygon is drawn the y position at the source should be increased by the height of the destination while the y position at the destination should be increased by the height of the destination plus 10 (to account for the gap between the destinations). You likely already introduced such a variable for the destinations when drawing the bars in the previous part of the assignment, and you can continue to use that variable in this part of the assignment.

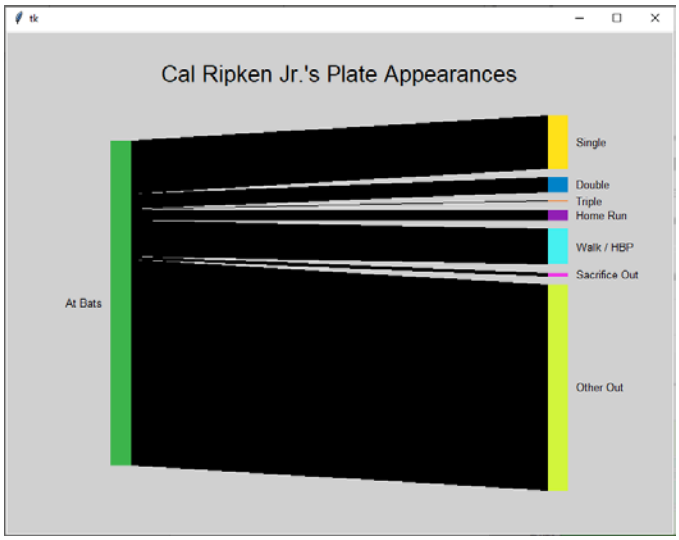
The output generated when the polygons are added is shown below. I added 3 lines of code (one of which spanned multiple lines) to complete this part of the assignment.



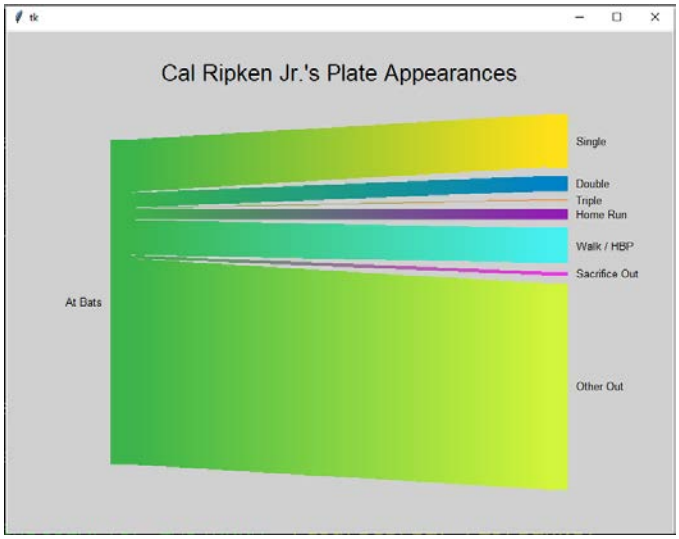
Part 5: Adding Color

A Sankey diagram can be more attractive and easier to understand when each destination is a different color and the color of each connector between the source to the destination gradually transitions from the source color to the destination color. Black outlines can also improve the look and clarity of the diagram. You will improve your program so that it includes all of these features in this part of the assignment.

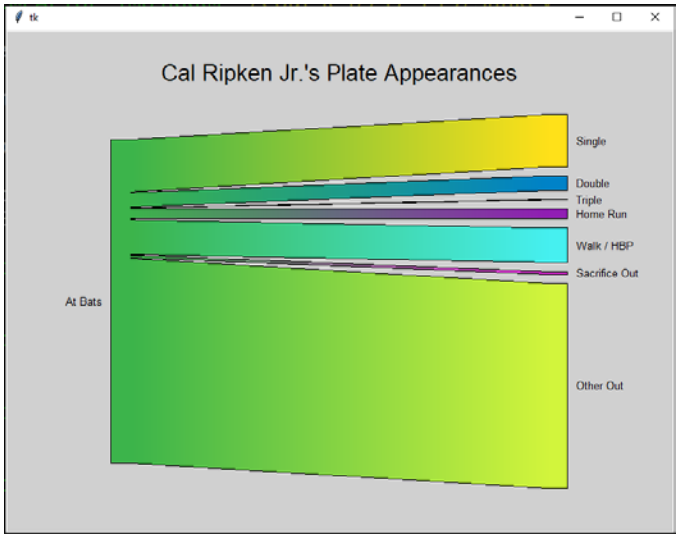
Pick a color for the source (which cannot be black) and create a list of no less than 12 unique colors for the destinations. (If you want to use colors that someone else has worked out, such as those at https://sashat.me/2017/01/11/list-of-20-simple-distinct-colors/, you are welcome to do so). Update your program so that the rectangles for the source and each destination are drawn in different colors, giving you a diagram similar to the one shown below.



Transitioning from the source color to the destination color will require you to replace the polygon with a series of lines (or narrow rectangles), each of which is drawn in a slightly different color. While you will need to work out the exact details of how to do this for yourself, I'd recommend constructing a loop that iterates over the x positions on the screen between the source and the destination. In the body of that loop you'll want a variable that describes how close you are to the destination, with 0 representing being at the source, increasing to 1 when the destination is reached. The value for this variable can be computed using the x value from the loop. Then the computed value can be used to compute both the y-position at which the line (or rectangle) should be drawn, and the red, green and blue components for its color.



The diagram will look nicer if there is a black outline around the perimeter of the colored region. Add additional black lines / rectangles to achieve the effect shown below.

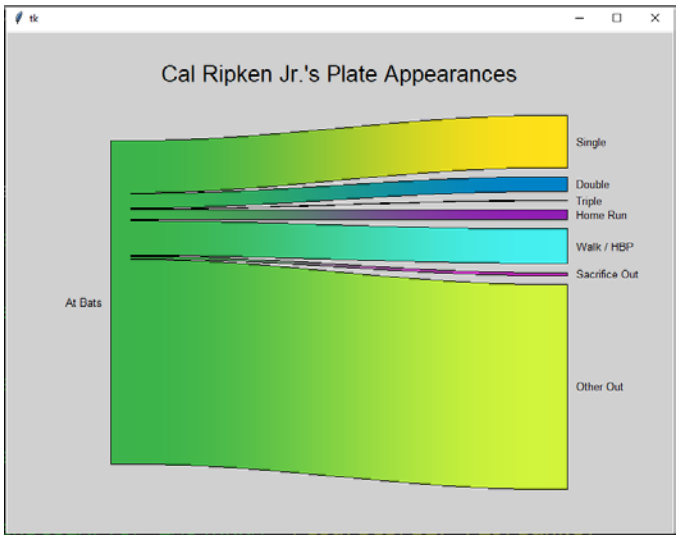


Finally, the diagram will look nicer if the paths from the source to the destination follow a curve instead of a straight line. This can be accomplished by taking your variable which increases from 0 to 1 and

modifying it so that the amount of increase follows a sin curve from -π/2 to π/2 instead of being a constant value. In particular, if your variable is named p, its value can be modified in the following manner:

```
p = p * pi - pi / 2
p = (sin(p) + 1) / 2
```

Adding these assignment statements (and updating the name of the variable to match what you selected previously) should result in the diagram shown below for the baseball data set.



Requirements

- Your program must be able to read the data file name as a command line argument. The data file will be the first and only command line argument, appearing in sys.argv[1]. If no command line argument is provided then your program should read the name of the file from the user. If more than 1 command line argument is provided then your program should display an appropriate error message and quit.
- Your program must perform basic error checking, such as ensuring that the file specified by the user exists. If an error is encountered while opening a file then your program should display an appropriate error message and quit.
- You must store the data in a dictionary where the keys are the destinations and the values are the amount of flow to the destination.
- Close every file that you open.
- Your program must make appropriate use of functions. In particular, it must include a function that reads the data values from the file and returns a dictionary representing those values (as described in Part 2) and a function that takes a dictionary as its parameter and draws the bars for that dictionary (as described in Part 3). Your solution will **not** receive full credit unless your code includes the functions described previously.

    - You may write additional functions if doing so helps you separate the problem into logical steps, reduces the amount of repeated code or otherwise results in a better solution to the problem.
- The only lines of code in your program that should be outside of a function definition are constants, import statements, and the call to the main function (which is normally the last line in the file).
- Do **not** define one function inside another function.
- Your program must **not** use global variables (except for constant values that are never changed). The data that you load inside a function must be returned as the function's result so that it can be used in subsequent functions.
- You may assume that the data in the files you are working with is correct. Once you open the file successfully you don't need to worry about problems such as reading a number where a word is expected, a missing comma, a blank line, etc.
- Your program must use good programming style. This includes things like appropriate variable names, good comments, minimizing the use of magic numbers, etc. Your program should begin with a comment that includes your name, student number, and a brief description of the program. **Each function should begin with a comment that describes its purpose, parameters (if any) and return values (if any).**
- 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.

Dealing with Command Line Arguments inside IDLE:

If you are working in IDLE instead of directly from the command prompt, you might find yourself wondering how to provide command line arguments when running inside IDLE. Unfortunately IDLE doesn't provide a good option for specifying command line arguments. Instead, the best strategy is probably to 'fake it' by adding the following line right after import sys:

```
sys.argv = ["Assignment4.py", "baseball.txt"]
```

This will overwrite whatever is in sys.argv with a list that has the name of your .py file as the first element and the name of the file that you want to process as the second element. You'll need to take this line out before you submit the assignment because your TA will run your submission from the command prompt, but you should be able to test all of the cases required for the assignment by changing the values in this list:

- You can test different files (including files that don't exist) by changing the second element in the list.
- You can test the case where the user provides too many command line arguments by adding another element to the list.
- You can test the case where the user doesn't provide a command line argument by removing the second element from the list.

Looking for an A+? Let Me (Optionally) Specify the Colors that Will be Used

Some diagrams benefit from having specific colors assigned to specific destinations. For example, a diagram showing what energy sources are used to supply the power needs for a region might want to color the most environmentally friendly sources with greens, hydroelectric with blue, and less environmentally friendly sources with yellow, orange or red.

Expand the capabilities of your program so that the data files can include optional color data after the source label, and after the amount of flow to each destination. This color data will consist of red, green and blue values that vary from 0 to 255. These three values will be comma separated, and a comma will also be present between the source label or the amount of flow to the destination and the red value. Some destinations may include color data while other destinations in the same file do not. When no color data is specified for the source or a destination your program should use a predetermined color, as described in Part 5. Multiple files that include optional color data are available on the course website.

Your solution to this part of the assignment must **not** use any global variables. In particular, your function for loading data from the file must return the color data as an additional result rather than modifying a global list of color values and then pass this additional color data to the function that draws the Sankey diagram.

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 open the file successfully (including error checking)? Is the data loaded and printed out successfully? Are the boxes drawn for the source and each destination with the correct sizes and locations? Is the source connected to the destinations? Is the diagram colored correctly?). The base grade will be recorded as a mark out of 12.

Style will be graded on a subtractive scale from 0 to -3. For example, an assignment which receives a base grade of 12 (A), but has several stylistic problems resulting in a -2 adjustment will receive an overall grade of 10 (B+). Fractional grades will be rounded to the closest integer.

Total Score (Out of 12)	Letter Grade
12	A
11	A-
10	B+
9	B
8	B-
7	C+
6	C
5	C-
4	D+
3	D
0-2	F