CSc 110, Spring 2018
Programming Assignment #7: Gerrymandering (20 points)
Due: Tuesday, March 20, 2018, 7:00 PM

Thanks to and to Marty Stepp and Stuart Reges for parts of this document.

This assignment focuses on reading input files. Turn in a file named `gerrymandering.py`.
You will need `DrawingPanel.py`, which you used on previous assignments, to write this program.

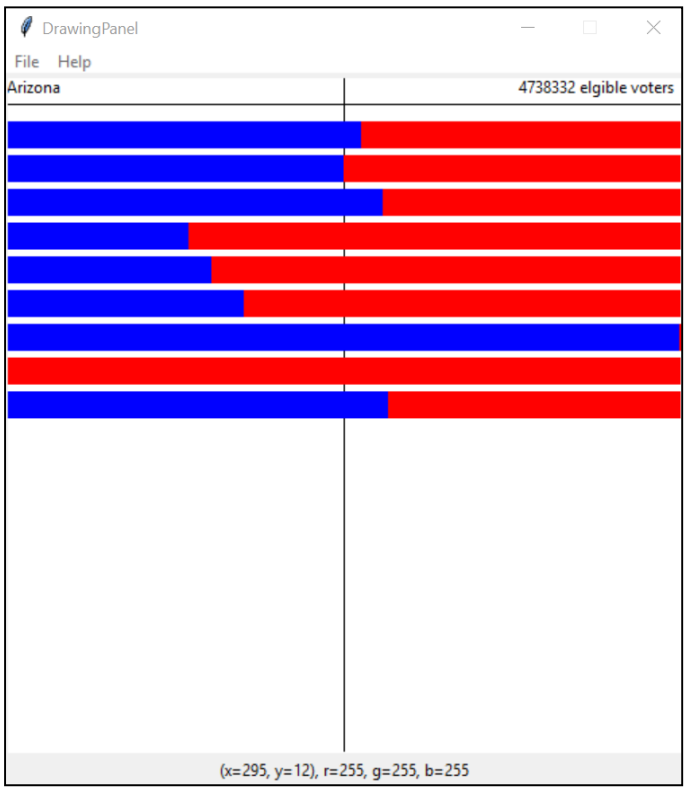
Program Description:

Recently gerrymandering has been a popular topic in the news and the subject of an ongoing Supreme Court case. Gerrymandering is creating voting districts in such a way that gives one party an unfair advantage over the other. Political parties do this to try to help themselves stay in power. Recently, a research group came up with a mathematical definition for what constitutes gerrymandering. For this project, you will prompt the user for a state name, and then determine whether the state's districts are gerrymandered, according the researchers' definition, and output a graphical representation of the districts on a `DrawingPanel`. The input data about states' districts and total voters comes from two input files provided on the course web site. Even if you disagree with this definition of gerrymandering, it is interesting to understand it better as the courts are currently debating it.

Your program should give an introduction and then prompt the user for a state name. It should then read the district file searching for that state, case-insensitively (that is, you should find the state regardless of the capitalization the user uses when typing it in). If the state is found in the file, your program should print the total wasted votes of democrats, the total wasted votes of republicans, the total voters in the state, who has gained an advantage from gerrymandering the state, if it is gerrymandered, and display information about the state graphically.

```
This program allows you to search through
data about congresssional voting districts
and determine whether a particular state is
gerrymandered.

Which state do you want to look up? Arizona
Total Wasted Democratic votes: 327852
Total Wasted Republican votes: 369697
4738332 elgible voters
```



Input Data and Files:

Your program reads data from two files. Download them from our web site to the same folder as your program.

1. districts.txt: *congressional district date*

Each line of districts.txt contains a state name followed by district information in groups of three. The first of the three is the district name, the second the democratic votes in that district and the third the republican votes in that district. Depending on the size of the state, there will be a different number of districts. For example:

```
Arkansas,1,63555,124139,2,103477,123073,3,0,151630,4,87742,110789
Alaska,AL,114596,142566
Rhode Island,1,87060,58877,2,105716,63844
```

Notice that the data is separated with commas (",") instead of the spaces we have used so far. We are using commas because some state names contain spaces and we want to preserve those spaces.

You can split a string called line on commas with the following call to split: line.split(",").

Note that the district names will sometimes be numbers and sometimes strings. The vote counts will always be integers.

Once the user types a state name, search each line of districts.txt to see if it contains data for that state. If the state name is found, output its data line to the console, then construct a DrawingPanel to graph the data (see next page). **Your code should not assume that the file is sorted alphabetically**.

If the state name is not found, output a "not found" message and don't show any data. No DrawingPanel should appear.

```
This program allows you to search through
data about congresssional voting districts
and determine whether a particular state is
gerrymandered.

Which state do you want to look up? mErLin
"mErLin" not found.
```

The data displayed above has a different number of districts for each state. Your program should work properly with any number of districts of data greater than 0. Since there is a limit to the size of the DrawingPanel, you may not be able to see all decades worth of data at the default height, but your code should process as many districts of data as it finds in the line.

---

2. eligible_voters.txt: *the number of eligible voters in each state*

If the state name is found in districts.txt, you should also read eligible_voters.txt to find its total number of eligible voters. The number of eligible voters should be printed to the console and also drawn on the DrawingPanel. Every state name in districts.txt is also in eligible_voters.txt, so you do not need to worry about a state having district data but no eligible voter data.

Each line of eligible_voters.txt contains a state name, followed by the eligible voter count for that state. For example:

```
Alabama,3606103
Alaska,519501
Arizona,4738332
Arkansas,2148441
California,25278803
```

Though the two input files contain different data, the task of searching for a state name in districts.txt is very similar to the task of searching for a state name in eligible_voters.txt. Your code should take advantage of this fact and should avoid redundancy. You should write your code in such a way that you stop searching a file once you find a line that has the name you're searching for.

You may not assume the input files will have 50 lines. Your program must work on any length input file.

Graphical Output:

The panel's overall size is 500x500 pixels. Its background is white. It has a black line drawn horizontally 20 pixels from the top and a black line drawn vertically in the middle of the panel from top to bottom. The state name is drawn at (0, 0). The eligible voters data is drawn at a y of 0 and 120 pixels from the right hand edge of the panel.

Each district is represented by a horizontal bar that stretches across the entire width of the panel. These bars are 20 pixels tall each. The first one starts 25 pixels from the top of the panel. There are 5 empty pixels between each bar.

The blue part of the bar represents the democratic votes in the district. The red represents the republican votes in a district. The total votes for each party in each district are in the `districts.txt` file. The width of the democratic portion should be:

```
democratic-votes / (democratic-votes + republican-votes) * width-of-the-panel.
```

Determining Gerrymandering:

You can determine gerrymandering by counting up and comparing the wasted votes cast for each party. We will define a wasted vote as any vote not needed to win the election. That means all votes for the party that loses the district seat are wasted as well as all votes for the winning party other than the half + 1 they need to win the majority.

For example, imagine that there was a state with the following districts:

	Dem	GOP	Wasted Dem	Wasted GOP
District 1:	2	7	2	2
District 2:	4	5	4	0
District 3:	10	7	1	7
Total:	16	19	7	9

Having calculated this data, we can sum up the wasted votes for each district. We find that the democrats wasted 7 votes and the republicans wasted 9. It is impossible to make voting districts exactly fair and so we shouldn't expect the wasted vote counts to be equal. However, researchers have discovered that it is almost impossible for the disadvantaged party to recover if the difference in wasted votes is greater or equal to 7 percent. Therefore, the researchers, as well as us for the purposes of this assignment, will consider a state gerrymandered when there is a 7% or greater difference in the wasted votes.

Implementation Guidelines:

We suggest you begin with the text output and file processing, then any "fixed" graphical output, and then the bars.

Your program should work correctly regardless of the capitalization the user uses to type the state name. If the user types `"AlAbAmA"` or `"alabama"`, you should find it even though the input files have it as `"Alabama"`.

Stylistic Guidelines:

You should have at least these **two constants**. If the constant values are changed, your output should adapt.

- The **width** of the `DrawingPanel`, as an integer (default of `500`)
  e.g. If you change the width to 600, each bar should still be the width of the screen.
- The **height** of the `DrawingPanel`, as an integer (default of `500`)

We will be especially picky about redundancy. For full credit, your functions should obey these constraints:

- The `main` function should not draw on a `DrawingPanel`, nor read lines of input from a file.
- You may not calculate whether a state is gerrymandered in the same function that draws on the `DrawingPanel`, or reads from the file.

Your functions should be well-structured and avoid redundancy, and your `main` function should be a concise summary of the overall program. Avoid "chaining," which is when many functions call each other without ever returning to `main`.

For this assignment you are limited to the language features in lectures 1 - 23. Follow past stylistic guidelines about indentation, line lengths, identifier names, and localizing variables, and commenting at the beginning of your program, at the start of each function, and on complex sections of code. You may not have global variables or nest functions in one another. You may not use the list `count` or `sum` functions.