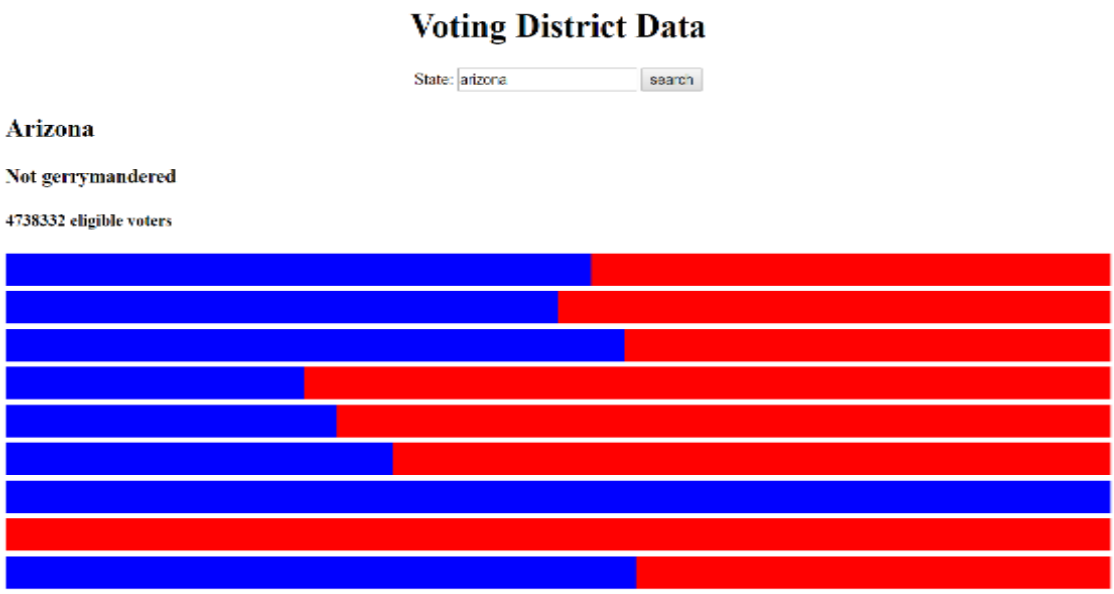
University of Arizona, CSc 337
Homework Assignment 7: Gerrymandering

Thanks to Marty Stepp and Jessica Miller for pieces of this assignment write up

This assignment is about using **Ajax** to fetch data in text, and JSON formats.



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. Even if you disagree with this definition of gerrymandering, it is interesting to understand it better as the courts are currently debating it.

We will provide you with the HTML (**gerrymandering.html**) and CSS (**gerrymandering.css**) code to use. Turn in the following files:

- **gerrymandering.js**, the JavaScript code for your weather web page's behavior
- **gerrymandering.css**, the CSS styles for your web page's appearance *(you don't need to modify it, but you can if you like)*
- **gerrymandering.html**, the weather web page *(you don't need to modify it, but you can if you like)*

You are allowed to modify gerrymandering.html and **gerrymandering.css** but are not required to do so. You can change the page's general appearance, so long as it meets the requirements in this spec. We encourage you to be creative and have fun.

This program uses Ajax to fetch data from a web service. We will also provide you with a web service called **gerrymandering_service.js** and two .txt files. You must put these in a directory on your computer and run **gerrymandering_service.js** with Node as shown in class on Friday 3/16.

Data:

Your program reads data from the web service **gerrymandering_service.js**. You may assume that all data sent to your program from **gerrymandering_service.js** is valid and follows the formats below.

This web service accepts two different types of queries, specified using a query string with a parameter named `state` and a parameter named `type`. Each type of query produces output in text or JSON format. *(You can test queries by typing in their URLs in your web browser's address bar and seeing the result.)* If you submit an invalid query, such as one missing a necessary parameter, your request will return an HTTP error code of 400 (Invalid Request) rather than the default 200.

1. voters: The first query type is `voters`, which outputs plain text containing the number of eligible voters in the passed in state. The following query would return the results below:

http://localhost:3000?state=arizona&type=voters

```
4738332
```

If the city you pass doesn't have any data (such as "arrrrizona"), your request will return an HTTP error code of 410 (Gone).

2. districts: The second query type is `districts`, which returns JSON data about that city's voting districts.

http://localhost:3000?state=arizona&type=districts

```
{"state":"Arizona",
 "districts": [[97391,87723],
               [109543,109704],
               [58192,46185],
               [45179,122560],
               [54596,124867],
               [70198,129578],
               [54235,0],
               [0,128710],
               [88609,67841]
              ]
}
```

If you submit a query for an unknown state, your request will return an HTTP error code of 410 (Gone).

Appearance and Behavior:

All style or appearance aspects not mentioned in the provided CSS file are subject to the preference of the web browser. The screenshots in this document were taken in Chrome on Windows, which may differ from your system.

The HTML page given to you shows a heading of "Voting District Data" followed by an `input` element with `id` of `box`.

The rest of the page contains three `div`s in which to put the results. Initially these `div`s are empty, but when the user clicks Search after typing a name in the `input` box, you should add content to these `div`s.

Nothing should happen when the user types a name into the `input` box; wait until they click the Search button to search. You may assume that the user doesn't click Search again until the current search is done downloading.

Voters data: When the user chooses a state and clicks Search, that state's eligible voters information should be fetched using Ajax and injected into the page into the `div` with the id `voters`. It should be added as an `h4`.

Districts data: When the user chooses a state and clicks Search, that state's eligible voters information should be fetched using Ajax and injected into the page. The state name should be added to the `div` with the id `statename` as an `h2`. It should be added with the capitalization it is returned with from the Ajax request, not whatever capitalization the user typed it in.

For each inner list in the districts list you should create a `div` and give it the class `dem` and set its width to `dem-votes / (dem-votes + gop-votes)` percent. Create another `div` and give it the class `gop`. Append the `div` with the class `dem` to the `div` with the class `gop` and append the `div` with class `gop` to the `statedata div` on the page.

Determining Gerrymandering: You will need to use the districts data to determine whether the state is gerrymandered. 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.

Add this data as an `h3` to the top of the `statedata div`. If you determine it is gerrymandered use the text "Gerrymandered to favor the <replace-with-party-with less-wasted-votes> Party". If you determine it isn't gerrymandered use the text "Not gerrymandered".

Missing data: Some states do not have any data. In such a case, the web service query will return an error code of 410. You should handle this case by displaying a message indicating that there was no data for the chosen state. There is already a `div` in the page with the `id` of `errors`. Add your message to that `div`.

You can check what kind of Ajax error occurred by examining the `response.status` in your `checkStatus`.

Subsequent Searches: The user can use the page to make multiple searches. When making a search, any data from a previous search should be cleared from the screen. You can remove all HTML content inside an area of the page by setting its `innerHTML` to an empty string. For example, to clear out the `div` with the `id` of `example`:

```
document.getElementById("example").innerHTML = "";    // clear out any child elements
```

Be careful to test your page with several searches in a row. For example, if one search has an error you will have data in the `error div`. But on the next search, you should not show this message.

Error Handling: If an error occurs during any Ajax request, other than the expected HTTP 410 error when a name's ranking is not found, your program should show a descriptive error message about what went wrong. For full credit, your error message should not be an `alert`; it must be injected into the HTML page. The exact format of the error message is up to you, but it should at least include some descriptive error text about what went wrong. You can inject any error messages into the provided HTML page into the `div` with `id` of `errors`:

```
<div id="errors"></div>   <!-- an empty div for inserting any error text -->
```

In order to test your error-handling, try temporarily changing the URL of the web service in your code to a bogus file name such as **notfound.js**. This will trigger an HTTP 404 File Not Found error on every request.

Implementation and Grading:

For full credit, your HTML/CSS code must pass the **W3C validators**. Your JavaScript code should pass our **JSLint** tool with no errors. Your .js file must run in **JavaScript strict mode** by putting `"use strict";` at the top.

Follow the course **style guide**. Separate content (HTML), presentation (CSS), and behavior (JS). As much as possible, your JS code should **use styles/classes from the CSS** rather than manually setting each `style` property in the JS. In particular, no CSS styles should be set in JS other than the widths of the blue bars, or showing/hiding various elements on the page by setting their `display` property. Use **unobtrusive JavaScript**, so that no JavaScript code, `onclick` handlers, etc. are embedded into the HTML.

Follow reasonable style guidelines. In particular, avoid redundant code, and use parameters and return values properly. Make extra effort to minimize **redundant code**. Capture common operations as functions to keep code size and complexity from growing.

Use the **"module pattern"** shown in class to wrap your code in an anonymous function. **No global variables or symbols**, nor "module-global" vars, are allowed on this assignment; values should be declared at the most local scope possible. If a constant value is used, you may declare it as a module-global "constant" variable `IN_UPPER_CASE`.

Fetch data using **Ajax**. Be careful to avoid redundancy in your Ajax code; if you do similar Ajax operations in many places, make a helper function(s). Process JSON data using `JSON.parse`.

Your JavaScript code should have adequate **commenting**. The top of your file should have a descriptive comment header describing the assignment, and each function and complex section of code should be documented. If you make requests, comment about what you are requesting and what your code will do with the data that is returned.

Format your code similarly to the examples from class. Properly use whitespace and indentation. Use good variable and function names. Avoid lines of code more than 100 characters wide.