CSSE290
Homework Assignment 6: Baby Names

This assignment is about using Ajax to fetch data from files and web services in text, HTML, XML, and JSON formats. You must match in appearance and behavior the following web page:

Popular Baby Names

First Name:

Claude Search

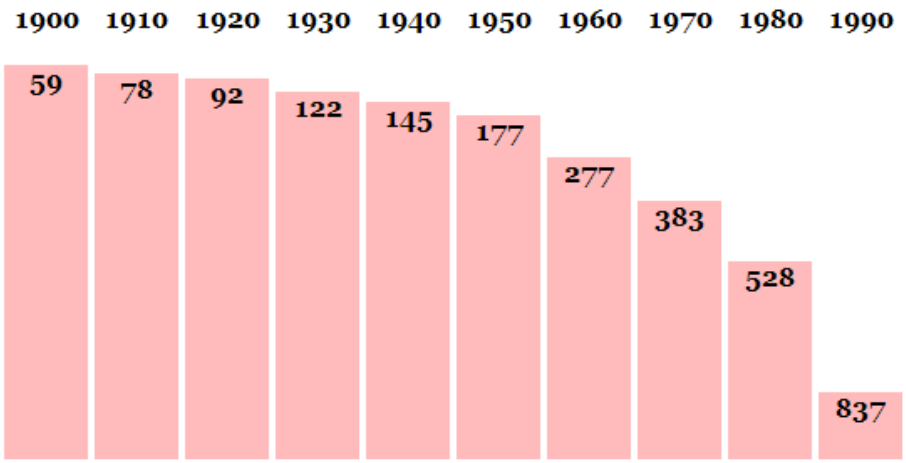
Male Female

Origin/Meaning:

The name **CLAUDE** means ...

"French, English French masculine and feminine form of CLAUDIUS."

Popularity:



Celebrities with This First Name:

- Claude King (129 films)
- Claude Dauphin (116 films)
- Claude Brasseur (115 films)
- Claude Rich (109 films)
- Claude Pieplu (100 films)
- Claude Akins (93 films)
- Claude Gillingwater (92 films)
- Claude Brosset (81 films)
- Claude Payton (63 films)
- Claude Chabrol (60 films)

Every 10 years, the Social Security Administration provides data about the 1000 most popular boy and girl names for children born in the US for each gender, at **http://www.ssa.gov/OACT/babynames/** . Your task for this assignment is to write JavaScript code for a web page to display the baby names, popularity rankings, and meanings.

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

- **names.js**, the JavaScript code for your Baby Names web page's behavior
- **names.css**, the CSS styles for your web page's appearance

This program uses Ajax to fetch data from a web server. Ajax can only connect to a web server from a page located on that same server. This means that **you must upload your page to the wwwusers.csse.rose-hulman.edu server to test it**. If you try to fetch data while viewing the page from your local hard drive, the request will fail with an exception.

Data:

Your program reads data from a web service at
http://wwwuser.csse.rose-hulman.edu/babynames/babynames.php.
You may assume that all data sent to your program from **babynames.php** is valid and follows the formats below.

This web service accepts four different types of queries, specified using a query string with a parameter named `type`. Each type of query produces output in text, HTML, XML, 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. list: The first query type is `list`, which outputs plain text containing all baby names on file in a plain text format, with each on its own line. The following query would return the results below (abbreviated):

babynames.php?type=list

```
Aaden
Aaliyah
Aarav
Aaron   ...
```

The provided web service generally returns the names in alphabetical order, but your code should not rely on this.

2. meaning: The second query type is `meaning`, which outputs information about that baby name's meaning in HTML format as a web page fragment. In addition to the always-required `type` parameter, the `meaning` query requires a second parameter named `name`. The following query returns the results below:

babynames.php?type=meaning&name=morgan

```
<div><p>The name <strong>MORGAN</strong> means ...</p> <hr />
     <p><q>Welsh, English From the Old Welsh masculine name Morcant, which was
        possibly derived from Welsh mor "sea" and cant "circle".</q></p></div>
```

When this data comes back to your page from the server, inject it into the `div` with `id` of `meaning`:

```
<div id="meaning">  <!-- insert the HTML fragment here -->  </div>
```

If the name you pass doesn't have any meaning data (such as "Mogran"), the **babynames.php** service will just output a general message about how that name has no known meaning; you can handle this case the same way as usual, by injecting the HTML that is returned into the page.

3. rank: The third query type is `rank`, which returns XML data about that baby name's popularity rank in each decade. The `rank` query requires two additional parameters named `name` and `gender`. The gender should be `m` or `f`. The server has separate data for each gender; the same name might be more/less popular for boys than for girls.

The overall XML document tag is called `<baby>` and has an attribute name for the baby's first name and gender. Inside the `<baby>` tag are several `<rank>` tags, one for each decade. Each `<rank>` tag contains an attribute named `year` representing the year of data. The text inside the `<rank>` tag is that name's popularity in that year.

The data has 12 rankings per name, from 1900 to 2010, from 1 (popular) to 999 (unpopular). A rank of 0 means the name was not in the top 1000. The following query would output the results below (abbreviated):

babynames.php?type=rank&name=morgan&gender=m

```
<baby name="Morgan" gender="m">
    <rank year="1900">70</rank>
    <rank year="1910">86</rank>
    ...
    <rank year="2010">258</rank>
</baby>
```

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

Though the provided web service always returns data starting at 1900 and running until 2010, you should not rely on the starting year being 1900. Your code should examine the XML to determine the starting year. The provided web service generally returns exactly 12 rankings for each name, but your code should not rely on this. If te first or last years have 0 for their rank, do not display bars for them.

4. celebs: The fourth query type is `celebs`, which outputs JSON data about actors who have the same first name as the given baby name. The `celebs` query requires two additional parameters named `name` and `gender`.

The overall JSON data object contains a single field named `actors`, which is an array of matching celebrities (they come from the `actors` table of the `imdb` database). If no actors match, the array will be empty (`[]`).

Each matching actor is a nested object with three fields: strings called `firstName` and `lastName` representing the actor's name, and `filmCount` representing the number of films in which the actor has appeared.

The following query would return the results below (abbreviated):

babynames.php?type=celebs&name=morgan&gender=m

```
{
  "actors": [
    {"firstName": "Morgan", "lastName": "Alling", "filmCount": 6},
    {"firstName": "Morgan", "lastName": "Andersson", "filmCount": 11},  ...,
    {"firstName": "Morgan", "lastName": "Lund", "filmCount": 7}
  ]
}
```

Appearance and Behavior:

All style or appearance aspects not mentioned below are subject to the preference of the web browser. The screenshot in this document was taken in Chrome on Windows, which may differ from your system.

The font for the page body (including form controls such as text boxes and buttons) is 12pt Georgia, with a fallback of the default serif font on the system. Heading fonts are proportional to this.

The HTML page given to you shows a heading of "First Name:" followed by a `select` element with `id` of `allnames`. When the page loads, the box is empty and disabled. Using Ajax, the page should fetch the list of all names from the web service as described previously, fill itself with an `option` for each name, and then enable itself.

The rest of the page is inside of a large `div` with the `id` of `resultsarea`. Initially this `div` is hidden, but when the user clicks Search after selecting a name from the `select` box, you should make this area appear. *(You can make an area of the page appear by calling the `.show()` method on its HTML DOM object, or `.hide()` to make it disappear.)* Nothing should happen and no data should be fetched if the user selects the initial blank "(choose a name)" option in the list.

Meanings: The next section of the page has the heading **"Origin/Meaning:"**. It is initially blank. When the user chooses a name from the `select` box, that name's meaning data should be fetched using Ajax and injected into the page into the paragraph with `id` of `meaning`. The meaning has a `q` (quote) element that should appear in italic.

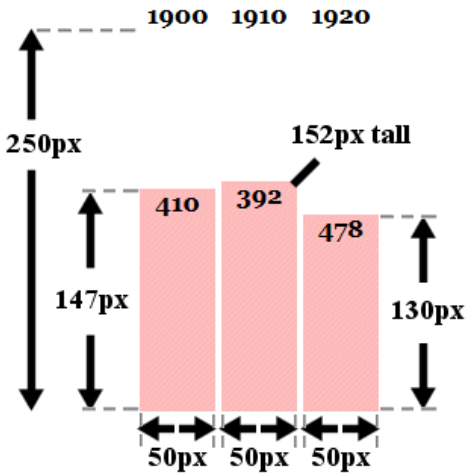
Rankings: The next section of the page has the heading **"Popularity:"**. It is initially blank. When the user chooses a name from the `select` box, that name's ranking information should be fetched using Ajax and injected into the page into this area. Specifically, there is an HTML table with the id of graph in this area, initially empty:

```
<table id="graph"></table>   <!-- baby name ranking data should be inserted here -->
```

If a name is selected from the selection box, you should use the DOM to fill the table with two rows of data: the first lists each of the decades as table header cells (`th`), and the second shows a vertical bar representing that name's ranking for that year in the data. For example, the table after searching for male name "Morgan" might look like this:

```
<table id="graph">
  <tr><th>1900</th><th>1910</th><th>1920</th>...<th>2010</th></tr>
  <tr>
    <td><div>70</div></td>   <!-- ranking for 1900 -->
    <td><div>86</div></td>   <!-- ranking for 1910 -->
    ...
    <td><div>258</div></td>  <!-- ranking for 2010 -->
  </tr>
</table>
```

The ranking bars are drawn as `div`s inside each table cell `td` of the second row of the table. Each ranking bar uses a pinkish background color of #FFBBBB. The bar's width is exactly 50px. Its height should be one fourth as many pixels as the "inverse ranking" for that decade. The inverse ranking is defined to be 1000 minus the ranking. For example, a ranking of 880 would lead to a bar with a height of 30 because one fourth of (1000 - 880), or 120, is 30. *(Note that division is exact in JavaScript; `13 / 4` is `3.25`. Round down height values using `parseInt`.)*



The overall height of each `td` cell in the second row of the `graph` table is 250px. The bottoms of the bars are aligned vertically; achieve this by setting the `td` table cells to have a `vertical-align` of `bottom`.

If the given name was not in the top 999 rankings for that decade, its rank in the XML will be 0. You should handle this special case by drawing the ranking bar `div` with a height of 0 pixels tall.

Within each ranking bar appears the ranking number for that decade. The numbers appear in bold text at the top of the bar, horizontally centered within the bar. If the ranking is very popular (1 through 10 inclusive), the ranking number should appear in red. Some less popular rankings (around 900 and up, or 0) have numbers that drop below the bottom of the graph; this is expected and you don't need to treat this as a special case.

Some name/gender combinations (such as Aaliyah/male) in the overall list do not have any ranking data. In such a case, the **babynames.php** query will return an error code of 410. You should handle this case by displaying a message indicating that there was no ranking data for the chosen name. There is already a `div` in the page with the `id` of `norankdata` that contains such a message, but it is hidden by default, so you must show it in such cases.

```
<div id="norankdata">   <!-- show me when the name has no rank data -->
  There is no ranking data for that name/gender combination.
</div>
```

You can check what kind of Ajax error occurred by examining the `ajax.status` field in your failure handler.

Celebs: The next section of the page has the heading **"Celebrities with This First Name:"**. It is initially blank. When the user chooses a name from the `select` box, that name's celebrity information should be fetched using Ajax and injected into the page into this area. There is an HTML bulleted list (`ul`) with the `id` of `celebs`, initially empty:

```
<ul id="celebs"></ul>   <!-- baby name celebrity data should be inserted here -->
```

If a name is selected from the selection box, you should use the DOM to fill the bulleted list with bullet items (`li`) of data, one for each celebrity that is returned in the JSON data on the server. Each bullet item should contain the actor's first and last name, followed by the actor's number of films in parentheses. For example, for male name Morgan, the list contains the following bullets (abbreviated):

- Morgan Wallace (119 films)
- Morgan Freeman (99 films) ...
- Morgan Jones (18 films)

If there are no celebrities with the given first name, the query will return JSON data with an empty array of actors. In this case, your page should just leave the bulleted list empty with no bullets in it. You don't need to do anything else or display an error message in this case.

Subsequent Searches: The user can use the page to make multiple searches. When making a search, any data from a previous name 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`:

```
$("example").innerHTML = "";     // clear out any child elements from this area
```

Be careful to test your page with several searches in a row. For example, if one search has no ranking data (such as Aaliyah/male), you will have shown the `grapherror` message. But on the next search, you should hide this message.

Loading Feedback: In each section of the page where data is shown, there is a small `div` with an animated "loading" GIF image that should be displayed while the data is being fetched from the server. For example:

```
<div class="loading" id="loadinggraph"> <img src="loading.gif" alt="icon" /> Searching...</div>
```

This HTML is already in the provided page, but it is your job to make it appear or disappear at appropriate times.

There is a loading area with the `id` of `loadingnames` to the right of the Search button. It should be visible until the page is done loading the list of names from the server, then it should disappear for the remainder of the page view.

There are three other loading areas with `id`s of `loadingmeaning`, `loadinggraph`, and `loadingcelebs`. All three should be shown when the user clicks Search to search for data about a name. When the data for one of the Ajax requests arrives from the server and you are finished processing that data, hide the corresponding loading `div`. The page should work for multiple requests; show/hide these loading `div`s properly on each subsequent search as well.

*(The provided **babynames.php** web service delays itself by 1-2 seconds to help you test your loading behavior. If you want to test other delays, pass an optional parameter `delay` to the service for the number of seconds you want it to pause before returning its data.)*

Error Handling: If an error (`onFailure`) 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 -->
```

You should also hide all "loading" animation images on the page if any error occurs. 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.php**. This will trigger an HTTP 404 File Not Found error on every request.

Implementation and Grading:

For reference, our .js file has roughly 160 lines (120 "substantive") and our CSS file has 30 lines (13 substantive).

Fetch your data using **Ajax**. We suggest Prototype's `Ajax.Request` object rather than the raw `XMLHttpRequest`, but either is acceptable if the code is not redundant. Process XML data using the JavaScript XML DOM, and process JSON data using `JSON.parse` rather than trying to examine/parse the returned string directly.

For full credit, your CSS code must pass the **W3C CSS validator**. 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.

You should separate content (HTML), presentation (CSS), and behavior (JS). As much as possible, your JS code should **use styles and 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 heights of the ranking bar `div`s drawn in the graph. You must 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. Capture common operations as functions to keep code size and complexity from growing.

No global variables are allowed on this assignment; values should be declared at the most local scope possible. If a constant value is used frequently, you may declare it as a global "constant" variable named `IN_UPPER_CASE`.

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 reasonably. Properly use whitespace and indentation. Use good variable and method names. Avoid lines of code more than 100 characters wide.