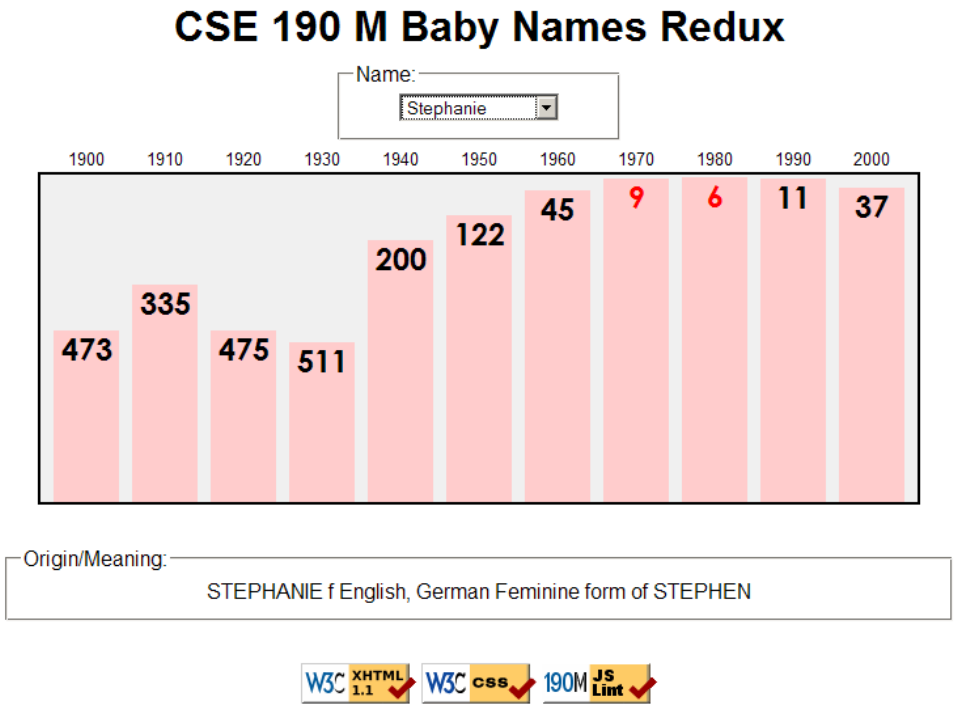
University of Washington, CSE 190 M
Homework Assignment 7: Baby Names

This assignment tests your understanding of fetching data from files and web services using Ajax (Asynchronous JavaScript and XML). You must match in appearance and behavior the following web page:



Background Information:

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

You do not need to submit any HTML or CSS file. We will provide you with the HTML (**names.html**) and CSS (**names.css**) code to use. Turn in the following file:

- **names.js**, the JavaScript code for your Baby Names web page

This program uses Ajax to fetch data from the Webster 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 Webster to test it**. If you try to fetch data from Webster while viewing the page from your local hard drive, the request will fail with an exception.

Data:

Your program will read its data from a web service located at the following URL:

https://webster.cs.washington.edu/cse190m/babynames.php

This web service accepts three different types of queries, specified using a query string with a parameter named `type`. Each type of query returns text or XML as its output. (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 rather than the default 200. If you submit a query for an unknown name, your request will return an error code of 404.

1. list: The first type of query is `list`, which returns plain text containing all baby names on file, with each on its own line. The following query would return the results below (shortened by ...):

https://webster.cs.washington.edu/cse190m/babynames.php?type=list

```
Aaliyah
Aaron
Abagail
...
```

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

2. rank: The second type of query is `rank`, which returns an XML document about that baby name's popularity rank in each decade. The `rank` query requires a second parameter named `name`.

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

The data has 11 rankings per name, from 1900 to 2000, from 1 (popular) to 999 (unpopular). A rank of 0 means the name was not in the top 1000. The following query would return the results below:

https://webster.cs.washington.edu/cse190m/babynames.php?type=rank&name=morgan

```
<baby name="Morgan">
    <rank year="1900">430</rank>
    <rank year="1910">477</rank>
    ...
    <rank year="2000">25</rank>
</baby>
```

Though the provided web service always returns data starting at 1900 and running until 2000, 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 always returns exactly 11 rankings for each name, but your code should not rely on this.

3. meaning: The third type of query is `meaning`, which returns text about that baby name's meaning. The `meaning` query requires a second parameter named `name`. The following query would return the results below:

https://webster.cs.washington.edu/cse190m/babynames.php?type=meaning&name=martin

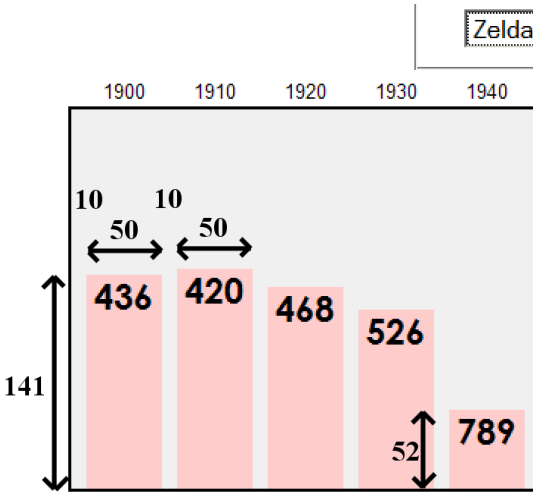
```
MARTIN m English, French, German From the Roman name Martinus, which was derived
from Martis, the genitive case of the name of Roman god MARS.
```

All names returned by the list query have ranking data, but not all will have meaning data (such as "Mogran"). In such a case, do not display any text in the Origin/meaning area (and clear any text that used to be there).

You may assume that all XML and text data sent to your program is valid and does not contain any errors.

Appearance and Behavior:

The HTML page given to you shows a heading followed by a select element with `id` of `babyselect`. When the page first loads, the box is empty and disabled. In the background the page should fetch the name data from the web service as described in this document, fill itself with an option for each name, and then enable itself.



There is also a large 670x250px shaded bar graph section in the center of the page that is initially blank. If a name is selected from the selection box, this shaded section is filled with bars representing that name's popularity for each year in the data. Any data from a previous name should be cleared from the graph. Nothing should happen if the user selects the "Select a name..." option in the list.

The bars are positioned absolutely with respect to the overall shaded graph section. Each ranking bar's width is 50px; its height is one fourth as many px as the "inverse ranking" for that decade. The inverse ranking is defined to be 1000 minus the ranking. For example, the ranking of 880 leads to a bar with a height of 30 (one fourth of 1000 - 880, or 120). (Note that division is exact in JavaScript; `13 / 4` is `3.25`. You will want to round down any height values you compute using `Math.floor` or `parseInt`. Their x-coordinates are such that the first bar's left edge is 10px from the left edge of the graph section, and there are 10px horizontal space between bars. The first bar's left corner is at x=10, the second is at x=70, the third at x=130, and so on. All bars' y positions are such that their bottoms touch the bottom of the shaded graph area.

Within each ranking bar appears the ranking number for that name in that decade. The numbers appear in an 18pt sans-serif font, top-aligned and horizontally centered within the bars. If the ranking is very popular (1 through 10 inclusive), it should appear in red. Some less popular rankings (around 900 and up) have numbers that drop below the graph region's black border; this is expected behavior.

The provided CSS file contains a class named `ranking` that contains all necessary properties for these bars, except the x/y positions and height, which differ for each bar. Create a DOM element for each bar and assign this class to it. The element itself is the bar, and its inner text content is the ranking.

Above each ranking bar, at the top of the shaded graph region, are labels representing the corresponding years. These are positioned vertically 1.5em from the top of the shaded region and centered horizontally within the ranking bar's width. The provided CSS file contains a class named `year` that contains all necessary properties for these elements other than their x positions, which differ for each year label.

Underneath the ranking bars appears a paragraph of text explaining the meaning of the name shown. When a new name is chosen, that name's meaning information is loaded asynchronously and appears underneath the graph. Inject the relevant text into the paragraph with `id` of `meaning`.

Error messages: If an error or exception occurs during any Ajax request, 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 the HTTP error code and some descriptive error text. It is **not** an error if a name has no meaning data; this is expected for some names (such as "Mogran"). Do not show an error message in such a case.

All other style elements on the page are subject to the preference of the web browser. The screenshots in this document were taken on Windows XP in Firefox 3.5, which may differ from your system.

Extra Features:

A separate specification document on the course web site describes several additional features you can complete to add functionality to your program. But you must complete at least one of these extra features as part of your assignment; in other words, **one extra feature is required**. The one required feature can be any extra feature of your choice. If you complete more than one extra feature, you can receive additional points/rewards as described in the other spec document.

Implementation and Grading:

Submit your work from the course web site. For reference, our .js file has roughly 95 lines (54 "substantive").

Fetch the necessary data for the program using **Ajax**. We suggest using Prototype's `Ajax.Request` and `Ajax.Updater` objects rather than the raw `XMLHttpRequest` object, but either approach is acceptable if the code is not redundant. Process XML data by examining the request's XML tree using the XML DOM.

Make extra effort to minimize **redundant code**. Capture common operations as functions to keep code size and complexity from growing. You can reduce your code size by using the `this` keyword in your event handlers.

For full credit, your JavaScript code should pass the provided **JSLint** tool with no errors reported. You should follow reasonable style guidelines similar to those of a CSE 14x programming assignment. In particular, minimize global variables, avoid redundant code, and use parameters and return values properly.

A few **global variables** are allowed, but it is not appropriate to declare lots of them; values should be local as much as possible. If a particular constant value is used frequently throughout your code, declare it as a global "constant" variable named `IN_UPPER_CASE` and use the constant throughout your code.

You should separate content (XHTML), 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.

For full credit, you must write your code using **unobtrusive JavaScript**, so that no JavaScript code, `onclick` handlers, etc. are embedded into the XHTML code.

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.

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

Do not place a solution to this assignment on a public web site. Upload your files to the **Webster** server at:

https://webster.cs.washington.edu/*your_uwnetid*/hw7/names.html