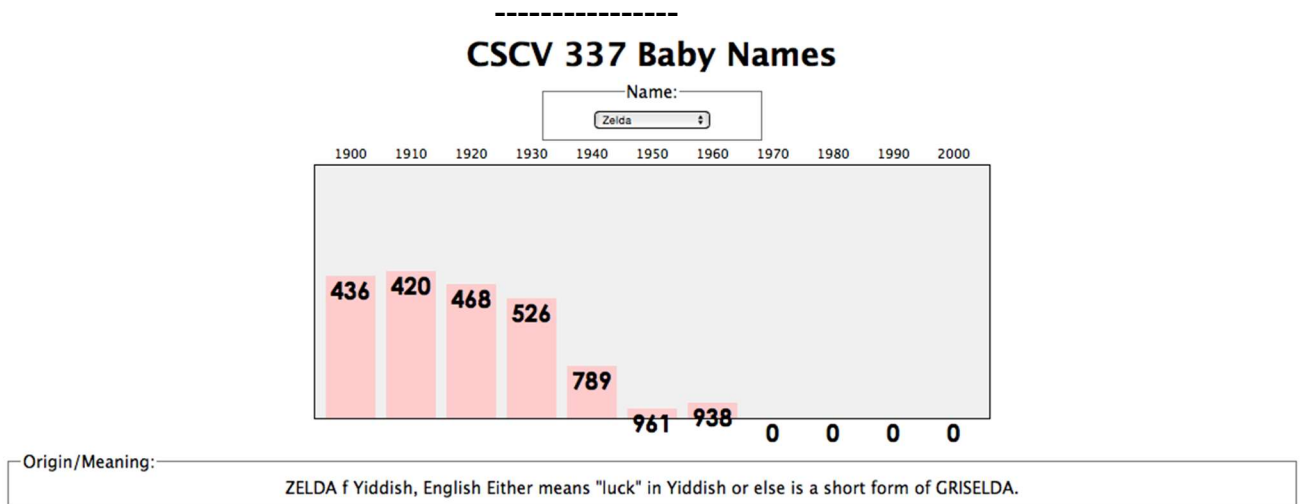# Assignment 5 Baby Names

What you need to do

Fetch data from files and web services using Ajax (Asynchronous JavaScript and XML).

Requirements

You must match in appearance and behavior the following web page:

Every 10 years, the Social Security Administration provides data about the 1000 most popular boy and girl names for children born in the US. 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. I 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 web server. Please note that 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 web and test it by viewing it**

on the web. If you try to fetch data from the web 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:

http://u.arizona.edu/~milazzom/cscv337/sp19/hw5/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. 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 ...):

   http://u.arizona.edu/~milazzom/cscv337/sp19/hw5/babynames.php?type=list

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

2. 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:

   http://u.arizona.edu/~milazzom/cscv337/sp19/hw5/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 this.

3. 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:

http://u.arizona.edu/~milazzom/cscv337/sp19/hw5/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. 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. You may also assume that the web app is reachable at the time your code runs.

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
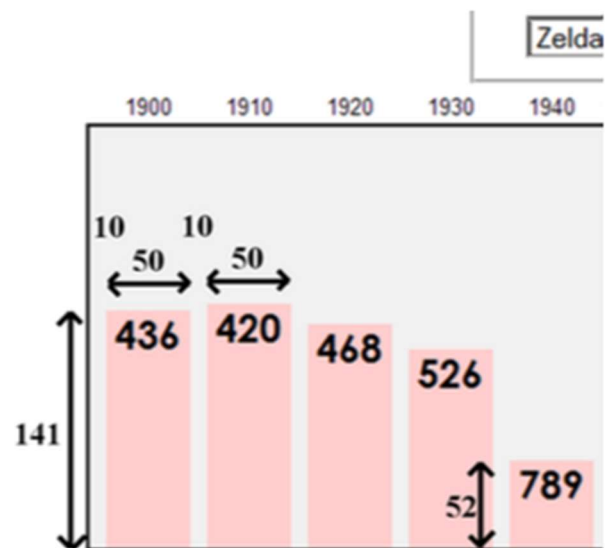
the top of the shaded region and centered horizontally within the ranking bar's width. The provided CSS file contains a class named year that contains all necessary properties for these elements other than their x positions, which differ for each year label.

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. 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 may differ from your system.

Tips and Hints to Create the Graph Bar Section

To create the graph bar section, you need to create DOM elements using JavaScript methods dynamically based on the data you can collect from the available services. For example, in case that a user selects "Abel" from the baby name list, you can query its ranking data and the response data can be used to create p (of class year) and div (of class ranking) elements in the graph section of the web page.

The CSS rules for classes *year* and *ranking* are provided in the file *names.css*. Your code should focus on calculating values for the heights of the ranking bars as well as their positions. The below screenshot includes the look of the web page when a web user chooses "Abel" as well as the HTML source code focusing on the graph section. Note that the graph section code should be dynamically generated by your JavaScript code. (Don't hard code this part in your names.html, which is provided and should NOT be modified at all!)

You may want to download the image file and zoom it so that you can see the graph section HTML code with more details (e.g. what are the attribute values associated with p and div

elements in the graph section?) The details may give you some hints to code your graph section in JavaScript.
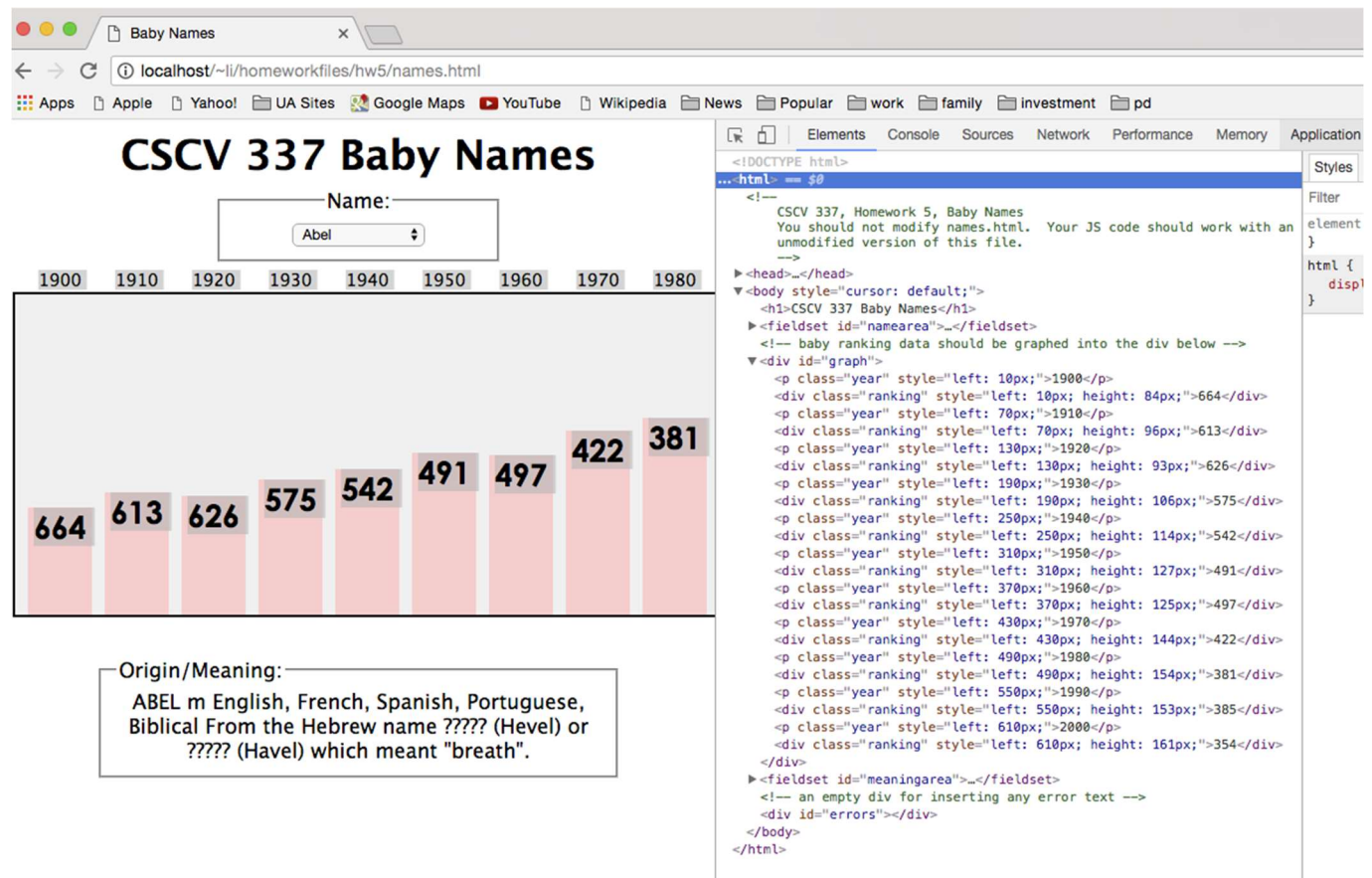
Turn-in

Please turn in a Zip archive where the filename is uanetid_h5.zip (not 7zip or some other format). Please submit your zip file to the D2L dropbox for Assignment 5 by the due date and post your homework files to your password-protected web space on the U-System.

When submitting your zip file, please specify the **URL** of your posted pages as well as the **password** for user **lxu** to access the pages in the drop box submission notes.

Rubric

This assignment is worth 100 points.

80 points will be awarded based on the appearance and behavior of your web page.

The remaining 20 points will be assigned based on your coding style.

- Fetch the necessary data for the program using Ajax. I 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. You can process XML data by examining the request's XML tree using the XML DOM.

- Your JavaScript code should minimize the number of global variables, utilize parameters and return values properly, correctly utilize the HTML DOM objects, correctly use indentation and spacing, and place a comment header at the top of your JavaScript file and at the top of every function explaining that function's behavior, and on complex code sections.

- You should minimize redundant code. You should also exercise good procedural decomposition, breaking down lengthy operations into functions, including parameters and returns over global variables when possible.

- You should separate content (HTML), presentation (CSS), and behavior (JS). As much as possible, your JS code should use styles from the CSS rather than manually setting each style property in the JS.

Part of your grade will come from successfully uploading your files to the web. **Please do not place a solution to this assignment online on a publicly accessible web site.**