Lab: Handling Web-based File Formats and Displaying Markers and Graduated Symbols

GEOG 319/658 Exercise #7 FALL 2014

Handling Web-based File Formats and Displaying Markers and Graduated Symbols

Due: Monday, October 27

Download the code for Chapter 12 of the Peterson text from http://maps.unomaha.edu/cloud/code.html

In this exercise you will be learning how to use two web-based file formats: XML and GeoJSON. In addition you will learn how to display markers and circles representing metropolitan statistical area (MSA) data in these formats.

A. Storing Data in Arrays and Displaying Markers Representing City Names and Population Values

Before working with the various file formats, we'll simply store the data in arrays.

Edit the code for Figure 12.5 (use file 12_05_Wayne_sites in the folder 12_05_Points_in_Arrays) so that it accomplishes the following:

1. Center the map on the 48 contiguous United States.
2. Set the zoom level to a value that will allow you to see just the 48 contiguous United States.
3. Using the table shown below, replace the arrays for lats, longs, hovertitle, and htmlAll with arrays depicting the four columns shown in this table. The values for latitude and longitude were determined using Google Geocoder.

City Name	Population	Latitude	Longitude
Chicago-Naperville-Elgin	9,461,105	41.87811	-87.62980
Dallas-Fort Worth-Arlington	6,426,214	32.78014	-96.80045
Houston-The Woodlands-Surgar Land	5,920,416	29.76019	-95.36939
Los Angeles-Long Beach-Anaheim	12,828,837	34.05223	-118.24368
New York-Newark-Jersey City	19,567,410	40.71278	-74.00594
Philadelphia-Camden-Wilmington	5,965,343	39.95233	-75.16379
Washington-Arlington-Alexandria	5,636,232	38.90719	-77.03687

4. Modify the marker section of the code as follows:
    a. Set up the loop so that it only handles the number of cities specified in the arrays.
    b. Specify the title of the marker as a city name.
    c. Specify a contentString that concatenates the word Population with the actual population.
    d. Replace "html: htmlAll[i]" with "myContent:contentString".
    e. In the line "infowindow.setContent(this.html)", replace "html" with "myContent".

5. Experiment with the program to make sure that it works.

a. When you hover over a marker, you should see the name of the associated MSA.
b. When you click on a marker, you should see the population of the associated MSA.

6. Document and format all of your code.

B. Storing the Data in an XML File

Now we begin to experiment with storing the data in an XML file. Download the SevenCities.xml file for this exercise from Blackboard. Open this file up in Notepad++ and examine its structure.

Within the Scripts directory for Peterson's code for chapter 12, you will see the file downloadxml.js. Copy this file to the same directory where you will be editing code. This file enables you to actually read the XML files. It will need to be copied to the web hosting service when you have finished your program.

Edit the code for Figure 12.6 (use 12_06_Mapping_pts_from_XML_file) so that it accomplishes the following:

1. Modify the function createMarker as follows:
   a. Change the third parameter so that it is a population value (you might call it pop).
   b. Change the contentString so that it concatenates the word Population with the actual population.
2. Set the zoom level so that it is appropriate for the U.S.
3. Center the map on the U.S.
4. Change the line for var fileName so that SevenCities is the xml file used.
5. Modify the following line of code so that it deals with the city info in the XML file.

```
var markers = xmlDoc.documentElement.getElementsByTagName("marker");
```

6. Modify the subsequent for loop so that it handles the proper number of cities.
7. Within the body of this same for loop, modify the code so that it fits the attributes shown in the SevenCities XML file.

8. Experiment with the program to make sure that it works. If you click on a marker, you should see the population displayed. If you click on a city name in the upper right, you should see the population displayed for that city.

9. Document and format all of your code.

C. Storing the Data in a GeoJSON file

Download the SevenCities.json file for this exercise from Blackboard. Open this file up in Notepad++ and examine its structure.

Download the cityIcon.png file for this exercise from Blackboard. We will use this icon to represent each metropolitan statistical area (MSA).

Be certain the file jquery-1.7.1.min.js is at the same directory level as the code you will edit below. Note that this file also will need to be copied to your web hosting site when you have wrapped up your program and are ready to distribute it.

Edit the code for Figure 12.7 (use 12_07_Mapping_pts_from_JSON_file) so that it accomplishes the following:

1. Change the title to reflect the type of data that we are dealing with.
2. Change the zoom level and center as we did above.
3. Change the file name for the json file so that it matches our desired data file.
4. Within the function addLocation make the following changes:
    a. Modify the for loop so that it handles the proper number of cities.
    b. Modify the body of the for loop so that it fits the attributes shown in the GeoJSON file.
   For example, the following line would handle the name attribute.

```
var name = json.Cities[i].name;
```

5. Replace the addLocation function with the following code:

```
function addLocation(json) {
 var n=json.Cities.length;
 for (var i = 0; i <n; i++) {
        var name = json.Cities[i].name;
        var pop = json.Cities[i].population;
        var location = new google.maps.LatLng(json.Cities[i].lat,json.Cities[i].lon);
        addMarker(gmap, name, pop, location)
                }
                        }
```

6. Replace the addMarker function with the following code. Note that we are using the cityIcon.png icon file to represent cities. You may want to use a different icon.

```
function addMarker(gmap, name, pop, location){
        var cityIcon = 'cityIcon.png';
        var marker = new google.maps.Marker({
                position:location,
                map:gmap,
                icon:cityIcon});
 attachSecretMessage(marker, name, pop);
}
```

7. Replace the attachSecretMessage function with the following code:

```
function attachSecretMessage(marker, name, pop){
        var infowindow = new google.maps.InfoWindow({
                content: name + ': '+ pop,
                size: new google.maps.Size(50,50)});

        google.maps.event.addListener(marker,'click', function(){
        infowindow.open(gmap, marker);
        });
 }
```

8. Experiment with the program to make sure that it works. When you click on a city icon, you should see the name of the associated MSA and population displayed.

9. Document and format all of your code.

D. Displaying Graduated Circles Using GeoJSON

In this section we will start with the code from exercise #5 and add code that will enable us to display circles representing each MSA. Here we will assume that you wish to use a GeoJSON input file.

Load the code that you created for exercise #5 into Notepad++. Save this code with a new name. Now edit the code as follows:

1. Delete the code originally found in the body of your program.

2. Insert the following line of code in the head section of your code; this indicates that you will use the JSON library.

<script type="text/javascript" src="jquery-1.7.1.min.js"></script>

3. Create a function that will draw a circle on the map. This function will have three arguments: the name of the object representing the map, the name of the object representing the location, and the radius of the circle. You can use Peterson's program 12_21_Circles_US_cities as a guide to what to include in the body of the function. Your function will return a drawn circle. Thus, your return statement might look like the following:

return new google.maps.Circle(populationOptions);

4. The rest of the body will be structured as follows:

a. Create a map

Here you can use the approach from Peterson's 12_21 program where you specify the mapOptions and create a map object. Remember that you also will need an associated div element. You will not need to put this in the initialize function.

b. Get the city data from the JSON file

Here you could use the code from the earlier GeoJSON example, but you will modify the name of the function call to indicate that you will AddCityCirclesToTheMap.

c. Create a function AddCityCirclesToTheMap which does the following:

i. Determine the number of cities.
ii. Put the population values from the JSON file in an array.
iii. Call the function you wrote for exercise #5 to determine the max population of values in the array from ii.
iv. Specify a largest radius; we found that a value around 100,000 seemed about right.
v. Add the circles to the map.
Here you create a loop in which you go through each city to:

1) determine an object specifying the location for the city
2) compute the radius of the circle for that city (you call the function you wrote in exercise 5 to compute a radius)
3) draw the circle (you call the function described in 3 above).

5. When done editing your program, test it to make sure that it displays circles correctly.

6. Document and format all of your code.

Create a folder for exercise 7 on the webhosting service. Load all files related to parts A, B, C, and D into this folder. Post the resulting URLs to the following Google worksheet . Hand in a hard copy version for all four parts of the exercise.