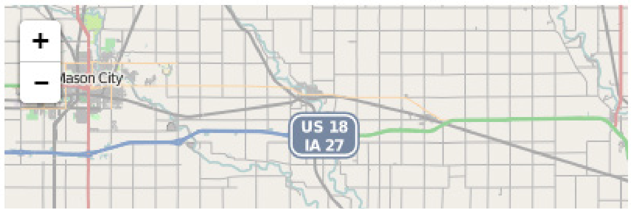
Mapping API's: Leaflet - Introduction to Legends

Welcome to the Essential ArcGIS Task Sheet Series. This series supplements the Iowa State University Geospatial Technology Training Program short course series. The task sheets are designed to provide quick, easy instructions for performing mapping tasks.

This task sheet builds off of *Mapping API's: Leaflet - Circles and Circle Markers* **PM2082-15b**. The code for that task sheet can be found in the ISU Geospatial Technology Program GitHub page at https://github.com/ISUEOGTP/GISTaskSheets within the **Leaflet-Tutorials** repository and is named **LegendLeaflet.html**. This task sheet will give you an introduction to generating legends with Leaflet.

1. Introduction and Setup

a. First, you will need to start with a basic leaflet map setup. Reference the task sheet: *Mapping API's: Leaflet - Getting Started* **PM2082-14r** to learn how to get this set up, or get the starter code from our GitHub page at https://github.com/ISUEOGTP/GISTaskSheets/blob/master/Leaflet-Tutorials/helloLeaflet.htm.



b. Instead of using an external JSON data file for this project, the data will be added directly after the map constructor and referenced using a variable called **myPoints**. Each point contains three values, a number (0-25), latitude, and longitude.

```
//data
var myPoints - [
        ["22",42.99497,-93.50808],
        ["20",42.10269,-93.23696],
        ["15",43.2,-93.1],
        ["19",42.98585,-94.50659],
        ["12",42.93163,-93.81726],
        ["5",42.5183,-93.89],
        ["14",42.42079,-93.5783],
        ["23",42.08414,-93.96632],
        ["6",42.51285,-93.0],
        ["14",42.013997,-93.635769],
];
```

c. Next, add two arrays. The first array called **breaks**, contains values defining the minimum value for each corresponding label. The second line of code is a variable named **labels**. This array contains the category names for sorting data. In this case there are three values, good, fair and poor. The order in which these appear is important as we will next associate 'good' with high values and poor with low values.

```
//used by color and legend functions to define
data breaks
var breaks = [17, 14, 0];
var labels = ['good', 'fair', 'poor'];
```

d. Add the function **getColor** to set the color for each marker based on a value in the data. In this function the data value is passed as variable **d** and tested to see if it is greater than or equal to (**>=**) the value stored in positions 0,1 or 2 of the **breaks** array. If the value meets the criteria the color identified is returned. You will see that there is no **breaks[2]** in the function because all values that are not greater than **breaks[1]** use the default color red. *Note that HTML colors or HEX colors can be used.*

```
//set color of marker function
function getColor(d) {
    return d >= breaks[0] ? 'green' :
    d >= breaks[1] ? "#ffff00" :
    "red";
}
```

e. Next, add a for loop to extract the data from **myPoints** and test to see what color to apply based on the value stored in position **[i][0]** of the **myPoints** data array. *Note: the radius style option is commented out. Uncommenting this will display the circles in various sizes. The number is also displayed in a pop-up.*

```
for (var i = 0; i < myPoints.length; i++) {
      marker = new L.circleMarker([myPoints[i]
[1],myPoints[i][2]], {
      //radius: myPoints[i][0]/2,
      fillColor: getColor(myPoints[i][0]),
      color: "#000",
      stroke: true,
      weight: 1,
      opacity: 1,
      fillOpacity: 0.9
})
.bindPopup("Value: "+myPoints[i][0])
.addTo(map);
}
```

2. Add CSS Styling

a. Data points should now appear on the map. To add a legend you first must provide some CSS code to define the appearance of the Legend elements. Add the CSS code to the right. The **.Legend** class sets the text color of the label and line weight. **.legend i** defines the size of the color boxes and **.info** defines the white legend box.

```
.legend {
    line-height: 18px;
    color: #555;
}

.legend i {
    width: 18px;
    height: 18px;
    float: left;
    margin-right: 8px;
    opacity: 0.9;
}
.info {
    padding: 6px 8px;
    font: 10px/18px Arial, Helvetica, sans-serif;
    background: white;
    background: rgba(255,255,255,0.8);
    box-shadow: 0 0 15px rgba(0,0,0,0.2);
    border-radius: 5px;
}
```

3. Add Legend Control

a. The final step is to add the legend control. This starts by creating a leaflet control and specifying where it will be displayed on the map. Positions can be **bottomright**, **topright**, **bottomleft** or **topleft**.

b. Create the legend function shown in the code to the right. This function will create a new DIV on the map that uses the legend and info class items defined in the CSS. The **label** and **breaks** arrays are used next along with a call to the **getColor** function to create legend boxes of the correct color.

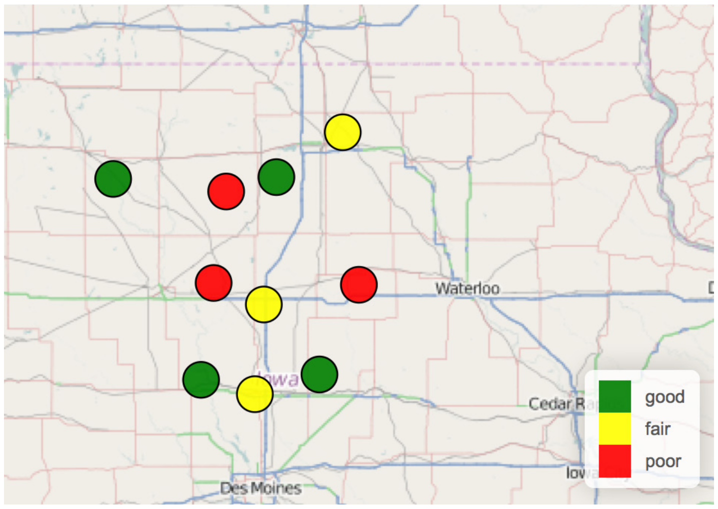
c. Add the legend to the map

```
var legend = L.control({position: 'bottomright'});

legend.onAdd = function (map) {
    var div = L.DomUtil.create('div', 'info
legend');

    //loop through items and generate legend
    for (var i = 0; i < breaks.length; i++) {
        div.innterHTML +=
          '<i style="background:' +
getColor(breaks[i]) + ' "></i> ' +
          labels[i] + (breaks ? ' ' + '<br>' : '');
    }
    return div;
};

legend.addTo(map);
```

4. Categories and Customization

a. The use of the **label** and **breaks** arrays are not necessary for creating a legend if the breaks are hard-coded into the **getColor** function or if the JSON data already includes category labels that accompany the values.

a. The code example in this task sheet however was designed to accommodate dynamic changes to the break values. For example, replace **var breaks = [17, 14, 0]**; with v**ar breaks = [12, 6, 0]**; and the map will appear with different colors. This particular example uses only three breaks, but with a little bit more customization you could make the entire **getColor** function dynamic and allow for additional break values as well as color options.



Contact:

Bailey Hanson bahanson@iastate.edu, 515-520-1436 or Professor Christopher J. Seeger, ASLA, GISP cjseeger@iastate.edu, 515-509-0651 for more information about the Geospatial Technology Program. This task sheet and more are available at www.extension.iastate.edu/communities/gis

Iowa State University Extension and Outreach does not discriminate on the basis of age, disability, ethnicity, gender identity, genetic information, marital status, national origin, pregnancy, race, religion, sex, sexual orientation, socioeconomic status, or status as a U.S. veteran. (Not all prohibited bases apply to all programs.) Inquiries regarding non-discrimination policies may be directed to Ross Wilburn, Diversity Officer, 2150 Beardshear Hall, 515 Morrill Road, Ames, Iowa 50011, 515-294-1482, wilburn@iastate.edu.

August 2015 PM2082-15o