Web Mapping Exercise 4
GISC 27105/37105
ArcGIS Maps SDK for JavaScript, I

This assignment will use ArcGIS JavaScript API to consume a variety of web GIS data types. ArcGIS JavaScript API allows developers to create enormously complex web GIS applications, however the steps are somewhat different than those we encountered in earlier assignments.

In this exercise, you will learn to add tiled web map services as well as Esri feature services; you will practice consuming and customizing external JSON files for your map using renderers; and you will work with generating a dynamic legend based on your rendering options. **Pay close attention to the order of functions** and variables you call/use in the HTML document. If something is not loading in, it's most likely because it hasn't yet been defined.

Loading in services

Let's begin with the very BASIC start of an ArcGIS Maps SDK for JavaScript map:

```
<html>
 <head>
  <meta charset="utf-8" />
  <meta name="viewport" content="initial-scale=1,maximum-scale=1,user-scalable=no"/>
  <title>Assignment 4 | YOUR NAME</title>
  <style> html, body, #viewDiv {padding: 0; margin: 0; height: 100%; width: 100%;}
</style>

 <link rel="stylesheet" href="https://js.arcgis.com/4.29/esri/themes/light/main.css"/>
 <script src="https://js.arcgis.com/4.29/"></script>

    <script>
      require(["esri/Map", "esri/views/MapView"], function(Map, MapView) {
        var map = new Map({
          basemap: "streets"
        });

        var view = new MapView({
          container: "viewDiv",
          map: map,
          zoom: 16,
          center: [-96.68, 40.81] // longitude, latitude – change this as needed
        });
      });
    </script>
  </head>

  <body>
    <div id="viewDiv"></div>
  </body>
</html>
```

- We will want to add a custom basemap, rather than drawing from the Esri prepackaged sets (e.g. "streets"). We first need to create a variable to define the base map we want to use. We will borrow from Lincoln-Lancaster County, Nebraska's GIS base map for this, since I showed it in class this past week. The basemap is located on their GIS server "gisext.lincoln.ne.gov," and it's in their GIS folder, under LancoBasemap. The variable should be identified as a "TileLayer" because this is an Esri-specific call to a tileset in the ArcGIS for Server model, as they're using. How do we know they're using ArcGIS for Server? First, when you go onto the county's GIS pages and inspect the web page closely, you can see in the developer tools (under sources) that there are numerous calls to data at https://gisext.lincoln.ne.gov/arcgis/rest/services/. When we look on that page, we see that it's a directory for a hosted instance of ArcGIS for Server.

  See the following code box for how to create a variable for a tiled map service. Note that when you are calling in an *entire* tiled base map like this, you'll want to ensure that your URL ends with **/MapServer/** rather than ending with a number. MapServer files often contain a great number of subfolders, each represented by a number at the end, and each representing an individual layer of data that comprises a map service (think of this like your layer list in ArcGIS or QGIS, listing out each layer. In fact, that's exactly what it is, the order of layers as they were in ArcGIS Pro or ArcGIS Desktop when the service was published). We want the entire, composite image of all the features in our base map, not one specific sublayer. *You can grab a specific sub layer if you want to bring it in as a feature, and we'll do that later on*. Go ahead and add a base map variable. Generally, you should place this code somewhere inside of the brackets that contain the required Esri/Dojo functions, in the head, in the same script, but *before* you load in the map and map view sections. In the previous example code, for instance, you would place this just ahead of the "view" variable:

```
var thebasemap = new TileLayer({
url:"https://gisext.lincoln.ne.gov/arcgis/rest/services/GIS/LancoBasemap/MapServer/"
});
```

- Next, since we are adding variables anyway, we should also create a variable to bring in the GeoJSON file we intend to use, and put it in the same area of the HTML file as the previous variable. I grabbed a GeoJSON file of Lancaster County's parcels that was hosted on their Open Data site (originally: https://opendata.arcgis.com/datasets/5aa5f17d117f409d8608ec1bf6de06d6_0.geojson) and filtered it down to a smaller dataset to promote faster load times and ensure future access for this assignment. Thew new location is listed in the URL below.

- First, create a variable for the URL that is hosting the GeoJSON file and place the code for it just above your base map variable you had created in the above steps:

```
const url = " https://rcshepard.github.io/downtownlincoln.geojson";
```

- Then, you create below that section another section to declare that you want to bring in a GeoJSON file hosted at that location.

```
const geojsonLayer = new GeoJSONLayer({
url: url,
copyright: "GISC27105"
//renderer info would go here, and you add a comma to the field above
});
```

- Next, you need to tell Esri's API that you want to do something with Tile Layers and GeoJSON files. This happens in the top part, in both the require and function lists where you can declare these modules.
- Add "esri/layers/TileLayer" and "esri/layers/GeoJSONLayer" to your require list, and then also add them to the function list to execute on the code. See bolded examples of me adding these features:

```
require([
    "esri/Map",
    "esri/layers/TileLayer",
    "esri/layers/GeoJSONLayer",
    "esri/views/MapView"
], function(Map, TileLayer, GeoJSONLayer, MapView) {
```

Finally, you have the basics to load in the new tiled layer and GeoJSON, and now you just need to write it into the code that these layers should be added to the map as well. Making these changes is fairly easy now that you've defined the parameters of the objects.

- Navigate to the **map** variable. We no longer want to use the default "streets" base map provided by Esri, and so we can remove the "basemap:" definition altogether from that part of the code. Notably, the standard "basemap" meaning here refers only to Esri-owned base maps, where you can use shorthand code (like "topo" or "streets") to call in one of their own maps. Your base map is technically a tile ***layer*** that sits underneath, so we'll call it in that way.
- Add your two layers to the **map** variable. Add the variable names for both the base map and GeoJSON, in brackets, separated by a comma, as shown below:

```
var map = new Map({
          layers: [thebasemap, geojsonLayer]
        });
```

At this point, your map will load in the layers if you run the code, however there will be no real styling to your GeoJSON layer. You can do this on the fly, using renderers.

Adding a FeatureLayer

"Feature Layers" are ArcGIS for Server or ArcGIS Online feature services that have been published, so these are Esri-specific data much like the Tiled map service we added. Technically, they CAN be restyled on the fly using renderers, but generally we just call them in because the style has already been supplied by the desktop software.

- To consume and use a Feature Layer service, we first need to create a variable to define it. As you might expect, incorporating feature layers is exceptionally easy with ArcGIS JavaScript API.

```
var featureLayer = new FeatureLayer({
  url:
"https://gisext.lincoln.ne.gov/arcgis/rest/services/Parks/Park_Boundaries/MapSer
ver/0"
});
```

- Next, near the top of your code where you add all the modules, you need to add the Feature Layer module just as you added the other modules, and then also execute it as a function. Note the bolded additions in the code snippet here:

```
      require([
          "esri/Map",
          "esri/layers/TileLayer",
          "esri/layers/GeoJSONLayer",
          "esri/views/MapView",
          "esri/layers/FeatureLayer"
      ], function(Map, TileLayer, GeoJSONLayer, MapView, FeatureLayer) {
```

- Finally, you just need to add the feature layer itself. Make sure you add it somewhere before the ending bracket of functions near the end of the page, but *after* you define the map and map view.

```
  map.add(featureLayer);
```

- Now you have the layers you want, although they haven't been customized yet with styles.
- We will use a simple renderer to redesign the incoming symbols from the GeoJSON layer. Although these are polygon symbols, we will restyle them as *graduated symbols*. This is fairly easy to do, although it may involve a lot of code.

Restyling a Layer

- You do not have to add “renderer” to the require list, and you do not have to add it as a function, either. You just have to create a variable for it, and you also have to point to that variable

- First, assign to your GeoJSON layer a renderer (bold, below). Notice that render is set to a variable, “renderer,” which we haven’t yet set. We’ll do that in the next step.

```
const geojsonLayer = new GeoJSONLayer({
url: url,
copyright: "Rob Shepard",
renderer: renderer
});
```

- First, you set its properties, “simple” type. Point it to the County Assessed Value field from the JSON “CNTASSDVAL.” You also give it a title here, and then define the symbol styles. You also set legend options here. We haven’t yet created a legend, but let’s set that here too:

- In the visualVariables section, we enclose in brackets all the properties that should vary across symbols. To set specific breaks, we set the visualVariables properties to vary based on type: “size”; set to vary based on data in field: “CNTASSDVAL”, and then finally set the “stops,” numeric breaks between the data classes and what symbols belong in each category

```
const renderer = {
   type: "simple",
   field: "CNTASSDVAL",
      legendOptions: {title: "Assessed values in Lincoln"},
   symbol: {
     type: "simple-marker",
     color: "yellow",
     outline: {
       color: "black"
     }
   },
    visualVariables: [
     {
       type: "size",
       field: "CNTASSDVAL",
       stops: [
         {
           value: 25000,
            size: "5px",
          label: "Under 25,000"
         },
         {
            value: 100000,
             size: "10px",
```

```
                label: "20,000 to 99,999"
            },
          {
              value: 500000,
                 size: "15px",
                label: "100,000 to 500,000"
            },
            {
                value: 2500000,
                 size: "25px",
                label: "Astronomical"
            }
          ]
        }
      ]
    };
```

Adding the Legend

- Finally, you will also need to define a legend if you'd like one to show. A legend is technically a module that we'll need to load, so start there:

```
require([
    "esri/Map",
    "esri/layers/TileLayer",
    "esri/layers/GeoJSONLayer",
    "esri/views/MapView",
     "esri/layers/FeatureLayer",
     "esri/widgets/Legend"

], function(Map, TileLayer, GeoJSONLayer, MapView, FeatureLayer, Legend) {
```

- Next, you will define how this legend should appear, and you will finally add it to the map.
- Add the legend variable like this, near the very end of your script section, but right before the end of the initializing function ending bracket }. Assign the layer to your geojsonLayer.

```
            const legend = new Legend({
            view:view,
            layerInfos: [{
            layer: geojsonLayer,
            title: "Legend"
```

```
            }]
            });


            view.ui.add(legend, "bottom-right");
```

- Notice that you will also want to add in the location for the legend by appending it to the view properties.

- Review your page. Try to visualize your HTML document as a series of operations in which each attribute or style or function is assigned to a variable, and each variable should appear sequentially. That is, if you're using a renderer in the style of a layer, your renderer should appear first in order. Or, if your layers are referenced by the "map" properties variable, then your layers should appear first before you reference them, et cetera. While the specific order can be changed a little bit, a rough organization might look like this:

    - Base map variable
    - Renderer variable
    - GeoJSON URL
    - GeoJSON layer variable
    - FeatureLayer URL variable
    - Map variable referencing layers
    - View variable referencing map
    - Add Feature *map.add(featureLayer)*
    - Legend

- In addition to the order of operations here, your last check should be to confirm that all of your styles and variables are being fired within the scope of the Esri functions (between the parentheses and squiggly brackets). If you look up to the list of widgets/functions near your dojo require section at the top of the script, you'll notice it opens with require **(**[*list of functions*] and also function(*list of functions*) **{** which means everything, including the legend should be closed out with **});** to end the process of firing these functions. If you do not enclose something, such as your legend, it won't fire at all, even if other elements work. On page 1 with the beginning script, note that this item was present right above the </script> tag near the bottom of the code.

- Finally, make some final changes, outlined in the following rubric and upload the HTML page to your GitHub. Submit a word file with links to your finished work.

Checklist/Rubric

☐ Change the graduated symbol color to red (5pts)
☐ Change the graduated symbol outline to white or gray (5pts)
☐ Change symbol classes to 3, with breaks at 100,000, 500,000 and 1,000,000 (5pts)
☐ Change the graduated symbol labels to something like “low” “medium” “high” (5pts)
☐ Resize the value of the circles to 7px, 17px, and 26px (5pts)
☐ Change the text of the legend title to say “Assignment 4 Symbols” (5pts)
☐ Give your map a copyright statement with your name (5pts)
☐ Move the legend to the bottom-right of the screen (5pts)

☐ Add any other GIS layer to this map. You can create it, find it on the GIS server linked here, or use Lincoln’s open data portal (https://opendata.lincoln.ne.gov/) (10 pts)