Client-Side Web GIS

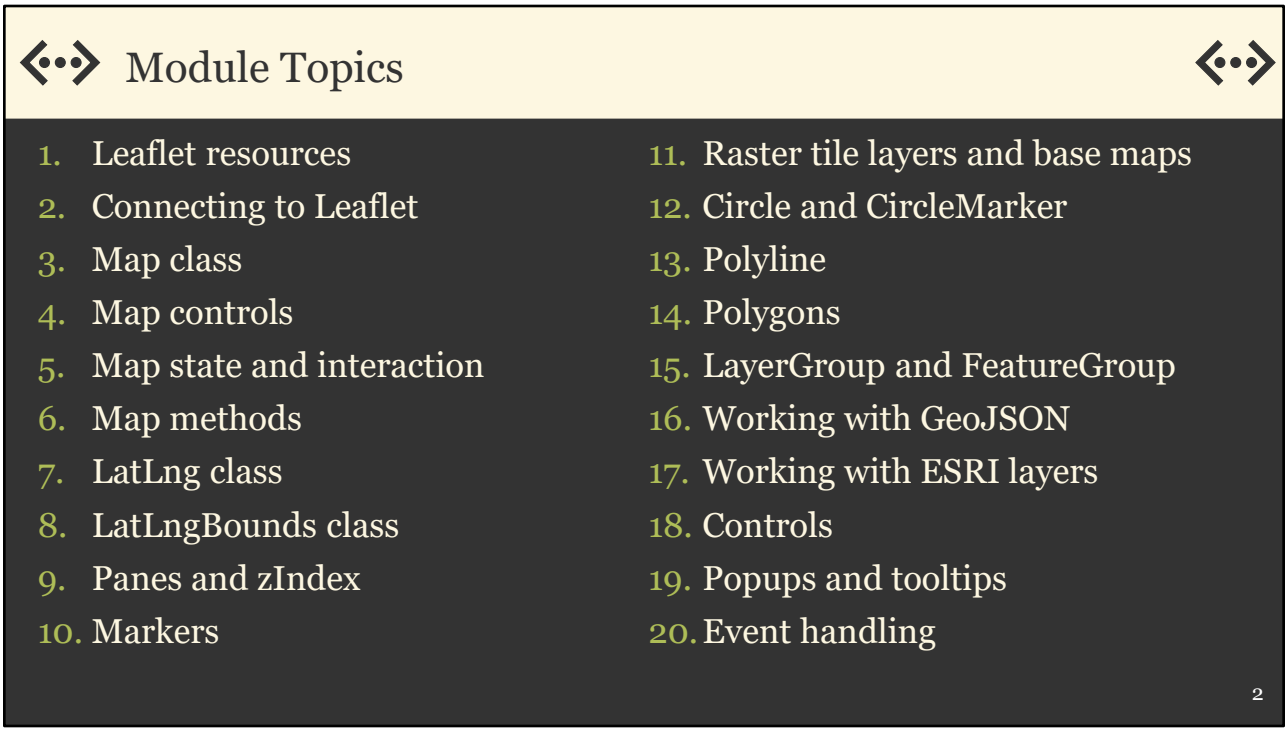
Leaflet JavaScript Library

Image from NASA: https://commons.wikimedia.org/wiki/File:Earth_Western_Hemisphere_transparent_background.png#filelinks

WV VIEW

AmericaView
Empowering Earth Observation Education
AmericaView.org - Est. 2003

Now that we have covered JavaScript and event handling, we are ready to start building interactive web maps using this language. We will focus on the free and open-source Leaflet JavaScript library.



These are the topics covered in this module.

Background

What is Leaflet?

- Web mapping JavaScript API
- Free and open-source
- Used by a variety of organizations:
  - NPR
  - Data.gov
  - Facebook
  - Pinterest, Etsy, Flickr
  - USA Today
  - The Washington Post
  - European Commission
  - Open Street Map
  - US Forest Service

https://leafletjs.com/

4

Leaflet is a free and open-source web mapping JavaScript library. As noted on this slide, Leaflet is used by a variety of organizations and companies.

Preparing to use Leaflet

- Download Leaflet JavaScript and CSS libraries and store in your website directory (in “src” folder and “css” folder)
- Link to Content Delivery Network (CDN)
- You do not need an API key to use Leaflet

```
<link rel="stylesheet" href="css/leaflet.css">


<script src="src/leaflet-src.js"></script>
```

https://leafletjs.com/download.html

5

Similar to other APIs, libraries, frameworks, or toolkits, to use Leaflet you can either download the JavaScript and CSS code from the Leaflet webpage and store it within your website directory or you can link to an associated content delivery network (CDN). For Leaflet, I generally prefer to host the source code along with my website, as the files are not very large. Since Leaflet is free and open-source, you do not need to acquire an API key to use it.

You can download the files from the Leaflet website on the Download page. This page also provides links to CDNs.

In your HTML file, you must use a <link> tag to link to the CSS and a <script> tag to link to the JavaScript component.

JavaScript Classes

- Blueprint for objects of the same type
- Use constructor method to create a new object of this class
- Pre-defined methods, properties, and options to work with for:
  - Creation
  - Interaction
  - Events
  - Get, set, and change settings and options

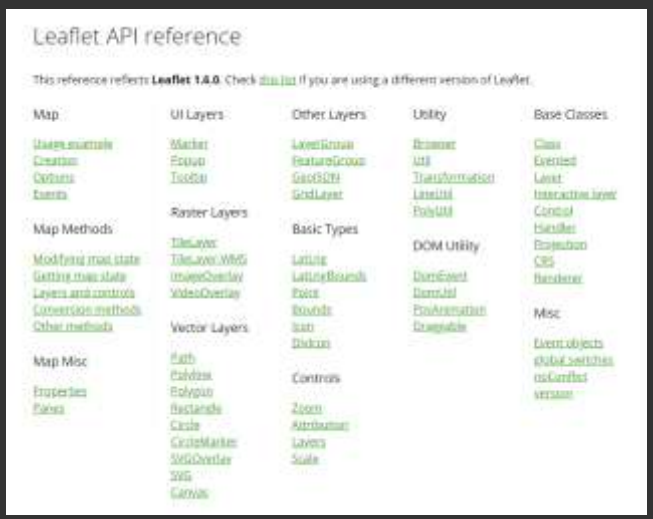
6

When using Leaflet, you will work with a variety of pre-defined classes that have associated methods to handle creation, interaction, events, and setting and changing options.

Here, I will discuss the most used classes and provide some examples.

Documentation

- Documentation available with API
- Some examples used here came from the API documentation



https://leafletjs.com/reference.html

7

Leaflet is very well documented. This slide provides a link to the library reference page. Many of the examples used here were taken from the API page.

If you use leaflet on a regular basis, you will get familiar with this documentation and learn to use it effectively.

Document Ready

```
$(document).ready(function () {});
```

8

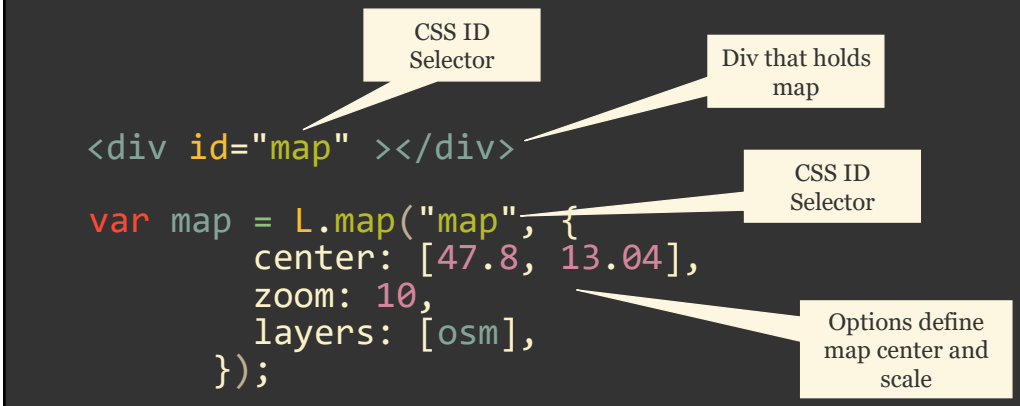
You may not want functions to be executed until the webpage is fully loaded. So, it is common to place JavaScript code that is used to generate map content and handle events within a document ready event, defined in the example with jQuery.

This will be demonstrated in the provided examples and is commonly recommended when using Leaflet.

Map Class

Generate a Map

- L.map()
- Must assign to a division on the page using a CSS ID.



10

The Map class allows for the generation of a web map that can be stored in a on a webpage.

Maps are generated using the L.map() constructor method. The first argument ("map" in this example) is the ID selector for the page in which you want to produce the map on the page.

You can also provide a variety of options. In this example, I am setting the map center, defined as an array that contains a latitude and longitude coordinate pair, and the map zoom level. I also add a layer.

Note that Leaflet expects the latitude to be provided first followed by the longitude within the array.

Controls

- AttributionControl = used to provide attribution/citation for basemaps and data layers (this is a legal obligation for using some data)
- zoomControl = add a default zoom controller to the map (will be added by default)

```
const tiles = L.tileLayer(
  "https://tile.openstreetmap.org/{z}/{x}/{y}.png",
  {
    maxZoom: 19,
    attribution:
      '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>',
  },
).addTo(map);
```

11

Maps have controls that are added by default but can be turned off. For example, the AttributionControl adds attribution for specific layers. For some layers, such as base maps and raster tile layers, attribution is a legal obligation. Many layers will include default attribution, but you can provide your own attribution. The slide provides an example attribution for the OpenStreetMaps base map.

The zoom control is added by default and provides buttons to zoom in and out.

Map State and Interaction

- center = set center of map view
- zoom = set map zoom level
- minZoom = set minimum allowed zoom level
- maxZoom = set maximum allowed zoom level
- Layers = add layers to the map
- maxBounds = set maximum bounds allowed
- dragging = define whether the map can be panned or dragged

12

This slide lists some common options used to define map interaction or change the map state in some way.

Center is used to define the map center coordinates while zoom sets the zoom level. The minZoom and maxZoom options can be used to limit the allowed zoom levels or how much the user can zoom in or zoom out.

The layers option can be used to add layers to a map and accepts an array of variables representing layers.

The maxBounds and dragging arguments are used to limit or hinder panning in the map space.

These options can be defined within the L.map() constructor method as a JavaScript object.

Map Methods

- addControl() = add control to map
- removeControl() = remove control from map
- addLayer() = add layer to map
- removeLayer() = remove layer from map
- setView() = set view using lat/long and zoom level
- setZoom() = set zoom
- fitBounds() = fit map to bounds (generally relative to a data layer)

13

This slide lists a variety of methods for altering the map.

The addControl() method allows the user to add controls to the map. In contrast, removeControl() removes the control.

The addLayer() and removeLayer() methods are used to add and remove layers from the map.

setView() is used to set or change the map center and zoom level while setZoom() just changes the zoom level. The fitBounds() method is used to fit the map extent to the bounds of a specific layer.

Note that most set methods have associated get methods to retrieve the current settings. This is useful for event handling.

Setting Center and Zoom

```
const map = L.map("map").setView([47.8, 13.04], 13);
```

14

This slide provides an example of setting the map center and zoom level using setView().

The map center is defined as an array that contains the latitude and longitude while the zoom is specified with a number that represents a zoom level.

Limiting Pan and Zoom

- Set a minimum zoom level with minZoom option
- Set a maximum zoom level with maxZoom option
- Limit panning using maxBounds option
- Not allow dragging or panning using dragging option

```
var map = L.map("map", {
  center: [47.8, 13.04],
  zoom: 10,
  layers: [osm],
  minZoom: 8,
  maxZoom: 11,
  maxBounds: bounds
});
```

15

Again, you can limit the user's ability to pan and/or zoom using the available options.

Working with LatLng

- Represent geographic points
- Provide latitude and longitude as an array or as an object
- Provide latitude before longitude when providing an array

```
var latlng1 = L.latLng(50.5, 30.5);
```

16

Another common class in Leaflet is the LatLng class that is used to represent geographic coordinates. Again, Leaflet expects the latitude to be provided before the longitude.

Working with LatLngBounds

- Defines a rectangular extent
- Can be used to set map extent

```
var corner1 = L.latLng(40.712, -74.227),
corner2 = L.latLng(40.774, -74.125),
bounds = L.latLngBounds(corner1, corner2);

map.fitBounds(bounds);
```

17

You can also use the LatLngBounds class to specify a rectangular extent by providing latitude and longitude coordinates that represent opposite corners of a rectangle.

The fitBounds() method can then be used to fit the map to the defined bounds.

Working with Layers

Layer Types

- User Interface Layers
  - Marker
  - Popup
  - Tooltip
- Raster Layers
  - Tile Layer
  - TileLayer.WMS
  - ImageOverlay
  - VideoOverlay
- Vector Layers
  - Path
  - Polyline
  - Polygon
  - Rectangle
  - Circle
  - CircleMarker
  - SVGOverlay
  - SVG
  - Canvas
- Other Layers
  - LayerGroup
  - FeatureGroup
  - GeoJSON
  - GridLayer

19

Classes have been defined to generate a variety of layers. We will not cover all of these classes here, but the Leaflet documentation provides a discussion of each if you are interested.

Panes and zIndex

- The relative horizontal position of layers and layer groups is defined by assigning a zIndex
- You can also generate our own panes to store layers and content and define your own zIndex for each pane

Pane	Type	Z-index	Description
mapPane	HTMLElement	'auto'	Pane that contains all other map panes
tilePane	HTMLElement	200	Pane for GridLayers and TileLayers
overlayPane	HTMLElement	400	Pane for vectors (Paths, like Polylines and Polygons), ImageOverlays and VideoOverlays
shadowPane	HTMLElement	500	Pane for overlay shadows (e.g. Marker shadows)
markerPane	HTMLElement	600	Pane for Icons of Markers
tooltipPane	HTMLElement	650	Pane for Tooltips.
popupPane	HTMLElement	700	Pane for Popups.

20

The relative horizontal position, layering, or stacking of map features, layers, and groups is determined by the zIndex. The table provided on this slide is from the library documentation and lists the default zIndex for different types of layers.

Larger numbers indicate that layers will draw above layers with smaller numbers. For example, by default popups draw above Polygons and Polylines.

You can also define your own zIndex for layers by creating your own panes. I generally prefer this method if I have a lot of layers and want to have more control over the drawing order, but this is just personal preference. If you are using a lot of layers, I highly recommend defining you own panes.

Panes and zIndex

```
hpmap.createPane("pA");
hpmap.createPane("pB");
hpmap.createPane("pC");
hpmap.createPane("pD");
hpmap.createPane("pE");


                hpmap.getPane("pA").style.zIndex = 210;
                hpmap.getPane("pB").style.zIndex = 410;
                hpmap.getPane("pC").style.zIndex = 420;
                hpmap.getPane("pD").style.zIndex = 430;
                hpmap.getPane("pE").style.zIndex = 440;
```

21

This slide demonstrates how to create new panes and define a zIndex for each pane. Once panes are generated, layers can be assigned to them using the pane option.

Markers

Marker

- Show geographic point as an icon 
- L.marker()
- Options
  - Icon
  - Opacity
  - riseOnHover
  - Pane
  - Attribution

```
L.marker([50.5, 30.5]).addTo(map);
```

23

Markers are user interface (UI) layers that are used to display clickable and draggable icons on the map. By default, the blue tear drop symbol is used. However, you can specify custom icons to use.

Markers have a variety of options. For example, you can choose an icon, set an opacity, allow the marker to be raised when it is hovered over, assign it to a pane, and provide attribution.

This slide provides an example of adding a marker using the L.marker() constructor method and by providing a coordinate pair to locate the marker. It is then added to the map using the addTo() method.

Change Icon

```
var myIcon = L.icon({
  iconUrl: "images/myIcon-01.png",
  iconSize: [32, 32],
});
```

```
var marker = L.marker([47.798714, 13.045149], {
  icon: myIcon,
}).addTo(map);
```

24

You can use custom icons. In the provided example, I am linking to an orange dot icon that I created and have stored in the images folder within the website directory. The link is defined using iconUrl while its size, in pixel units, is defined using iconSize. The variable associated with the icon can be referenced when creating the marker using the icon argument.

Marker Methods

- getLatLng/setLatLng() = get or set the coordinate of the marker
- getIcon()/setIcon() = get or set the icon used to represent the location
- setOpacity() = change the opacity of the icon
- toGeoJSON() = convert the icon to GeoJSON

```
marker2.on("mouseover", function () {
  this.setIcon(myIcon3);
});
```

25

Here are some common methods used to work with Markers.

The coordinates for the marker can be returned or set using getLatLng() or setLatLng() while the icon used can be returned or set using getIcon() or setIcon(). The opacity can be set with setOpacity().

Markers can be converted to GeoJSON using the toGeoJSON() method.

The example code on this page shows how to use an event handler to change the icon. When the Marker is moused over, the icon will change.

Add and Style Marker

```
var myIcon = L.icon({
  iconUrl: "my-icon.png",
  iconSize: [38, 95],
  iconAnchor: [22, 94],
  popupAnchor: [-3, -76],
  shadowUrl: "my-icon-shadow.png",
  shadowSize: [68, 95],
  shadowAnchor: [22, 94],
});
L.marker([50.505, 30.57], { icon: myIcon }).addTo(map);
```

- Set location
- Define icon to use (default to blue tear drop icon)
- Can use custom icons or icons from libraries

26

This slide provides an example of how to define and style a Marker.

A Marker icon can be referenced from a URL or via a file stored within your website directory. You can also manipulate the size and position of the icon and the size and position of the icon shadow.

When the marker is generated, the icon can be referenced using a variable.

Font Awesome Icons

```
var m2 = L.AwesomeMarkers.icon({
  icon: "tree",
  prefix: "fa",
  markerColor: "green",
});
```

https://github.com/lennardv2/Leaflet.awesome-markers

```
<script src="src/leaflet.awesome-markers.js"></script>
```

27

The Awesome-Markers plugin for Leaflet supports the use of icons from Font Awesome as Marker icons. The example code on this page shows how to define an icon using this library. Once defined, the icon can be used via the icon options within L.marker().

Raster Tile Layers

Raster Layers

- TileLayer = represent raster tile layer data
- TileLayer.WMS = display raster tile layers from a Web Map Service (WMS)
- ImageOverlay = load and display a single image on the map over a given bounds
- VideoOverlay = extends ImageOverlay to allow for video data to be added to map over a given bounds (such as a time series animation)

29

Raster Tile Layers can be added to leaflet maps using the L.tileLayer() constructor method. If the tiles are provided by a web map service (WMS) specifically, then you can use L.TileLayer.WMS().

These are common methods for adding base maps or operational layers that have been cached to raster tiles. Note that you will not be able to change the symbology of Raster Tile Layers since you are loading in pre-rendered tiles.

L.ImageOverlay() and L.VideoOverlay() can be used to add a single image or video file to a map that is referenced to a specified geographic bounds. Video overlays can be used to include video animations, such as a time series video.

Note that image and video overlays can only reference a single file. They are not meant to reference Raster Tile Layers that include multiple files and change with the extent and zoom level.

Add Raster Tile Layer

- L.tileLayer()
- Options
  - minZoom
  - maxZoom
  - opacity
  - zIndex
  - bounds
  - pane
  - className

'http://{s}.somedomain.com/blabla/{z}/{x}/{y}{r}.png'

30

There are some options available for Raster Tile Layers. You can set the maximum and minimum zoom levels at which the tiles will be visible. You can also set an opacity, zIndex, and pane in which to draw the tiles.

The bounds option can be used to limit the geographic bounds in which the tiles are drawn.

The “className” option is used to associate the tiles to a CSS class.

This slide provides an example nomenclature for linking to Raster Tile Layers. {s} indicates subdomains to help the browser with parallel requests per domain limitations. {z} indicates the zoom level being requested while {x} and {y} indicate the tile coordinates. {r} is used for loading tiles appropriate for high resolution retina display.

Raster tiles can be provided in several different compressed raster formats. PNG and JPEG are most common.

Add Base Maps

- Base maps are generally added as raster tile layers

```
const tiles = L.tileLayer(
  "https://tile.openstreetmap.org/{z}/{x}/{y}.png",
  {
    maxZoom: 19,
    attribution:
      '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>',
  },
).addTo(map);
```

31

This slide provides an example of adding base maps by referencing Raster Tile Layers. Specifically, I am linking to OpenStreetMap (OSM) tiles.

Note that OSM data are open-source and free; however, you still need to provide attribution.

Add Base Maps (leaflet-providers)

- Leaflet-providers plugin offers an easy means to add a variety of different base maps

```
var Esri_WorldImagery = L.tileLayer(
  "https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}",
  {
    attribution:
      "Tiles &copy; Esri &mdash; Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community",
  },
);
var Esri_WorldStreetMap = L.tileLayer(
  "https://server.arcgisonline.com/ArcGIS/rest/services/World_Street_Map/MapServer/tile/{z}/{y}/{x}",
  {
    attribution:
      "Tiles &copy; Esri &mdash; Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, 2012",
  },
);
var Esri_NatGeoWorldMap = L.tileLayer(
  "https://server.arcgisonline.com/ArcGIS/rest/services/NatGeo_World_Map/MapServer/tile/{z}/{y}/{x}",
  {
    attribution:
      "Tiles &copy; Esri &mdash; National Geographic, Esri, DeLorme, NAVTEQ, UNEP-WCMC, USGS, NASA, ESA, METI, NRCAN, GEBCO, NOAA, iPC",
    maxZoom: 16,
  },
);
```

https://leaflet-extras.github.io/leaflet-providers/preview/

32

Instead of using the default method for adding Raster Tile Layers and base maps, I prefer to use the leaflet-providers plugin. I find that the syntax is simpler, and you have easy access to a variety of base maps.

When connecting to multiple base maps, you should only add one to the map initially. You can then use layer controls to switch between base maps, which will be discussed later in the module.

The link provided on this page offers previews for a variety of base maps.

Note that some base maps require an API key or token for use or may limit the number of downloads available within a certain time period. So, they all may not be free for unlimited use. However, some are, such as OSM base maps.

Points

Circle

- L.circle()
- Options
  - Radius (radius defined in meters so symbol does not change size with map scale)
  - Options: stroke, color, weight, opacity, lineCap, lineJoin, dashArray, dashOffset, fill, fillColor, fillOpacity, renderer, className, pane, attribution

```
L.circle(latlng, {radius: 5000,color: "#3388ff",fillColor: "#3388ff",
fillOpacity: 0.3})
```

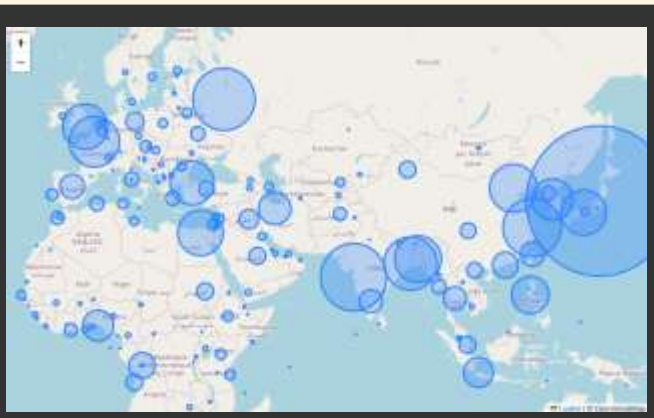
34

Point data can be added to maps as a Circle using the L.circle() constructor method. When using this method, the radius option defines the radius of the Circle in meters, so the Circle size maintains the same radius in map units as the user zooms in and out. In other words, the symbol size on the screen will change as the user zooms in and out.

As noted on this slide, a variety of styling options are available for Circles.

Note that Circles, CircleMarkers, Polylines, and Polygons all reference the Path abstract class that defines some common behaviors and styles. They all also reference the Layer class.

Circle



```
L.circle(latlng, {
              radius: feature.properties.POP_MAX * 0.05,
              color: "#3388ff",
              fillColor: "#3388ff",
              fillOpacity: 0.3,
            })
```

35

This slide provides an example for adding Circles that represent populated places. The size of the Circle is defined by the population (POP_MAX) attribute; however, the value has been scaled so that the Circles use a reasonable range of sizes.

I have also defined an outline color, fill color, and fill opacity.

We will talk about how to link to data layers later in this module. For now, we are just focused on symbology.

CircleMarker

- L.circleMarker()
- Options
  - Radius (with radius defined in pixels so symbol does change with map scale)
  - Options: stroke, color, weight, opacity, lineCap, lineJoin, dashArray, dashOffset, fill, fillColor, fillOpacity, renderer, className, pane, attribution

```
var cMrk1 = L.circleMarker(latlng, {
  radius: 10,
  weight: 1,
  color: "#111111",
  fillColor: "#eeeeee",
  fillOpacity: 0.9,
  pane: "pD",
});
```

36

A CircleMarker is similar to a Circle except that the radius is defined in pixel units as opposed to map units,. This results in the symbol maintaining the same size on the screen as the user zooms in and out.

A variety of options are available, similar to those for Circles. CircleMarkers also reference and extend the Path and Layer classes.

CircleMarker (Single Symbol)

```
L.circleMarker(latlng, {
                radius: 6,
                fillColor: "#ff0000",
                color: "#000000",
                weight: 1,
                opacity: 1,
                fillOpacity: 0.8,
              })
```

37

This slide demonstrates adding populated places as CircleMarkers. All features use the same symbol.

CircleMarker (Unique Colors)

```
var fill;
switch (feature.properties.MEGACITY) {
  case 1:
    fill = "#fc8d62";
    break;
  case 0:
    fill = "#66c2a5";
    break;
  default:
    fill = "#cccccc";
}
return L.circleMarker(latlng, {
  radius: 6,
  weight: 1,
  color: "#111111",
  fillColor: fill,
  fillOpacity: 0.9,
})
```

38

I am now using unordered colors to show a nominal variable: whether the city is classified as a megacity. The symbology is defined using a switch statement. Note that I specify a default color.

CircleMarker (Size)

```
var r = feature.properties.POP_MAX / 500000;
return L.circleMarker(latlng, {
  radius: r,
  weight: 1,
  color: "#111111",
  fillColor: "#8da0cb",
  fillOpacity: 0.9,
})
```

39

Now, I am using a continuous size ramp to show the population of each city.

Lines and Polygons

Polyline

- L.polyline()
- Accepts array of arrays of latitude and longitude coordinates
- Can use a multidimensional array to represent MutliPolyline shape
- Options: stroke, color, weight, opacity, lineCap, lineJoin, dashArray, dashOffset, fill, fillColor, fillOpacity, fillRule, renderer, className

```
var latlngs =
    [[45.51, -122.68],
    [37.77, -122.43],
    [34.04, -118.2]];
```

```
var polyline = L.polyline(latlngs, { color: "blue" }).addTo(map);
```

41

Polylines are used to represent linear features. The linear feature must be represented by a series of latitude and longitude coordinates using an array of arrays. MultiPolyline features, or a single feature made up of multiple lines, can be generated using a multidimensional array.

A variety of styling options are available.

Polyline extends the Path and Layer classes.

Polyline

```
lyrLine = L.geoJSON.ajax("data/worldRivers.geojson", {
          style: function (feature) {
            return {
              color: "#0000EE",
              weight: 4,
            };
          },
          onEachFeature: function (feature, layer) {
            layer.bindPopup(getPopup(feature));
          },
        });
```

42

This is an example of displaying line features, in this case major rivers. The same symbology is used for each feature.

Polygons

- L.polygon()
- Accepts array of arrays of latitude and longitude coordinates
- First and last should not be the same coordinate
- Can use multidimensional arrays to represent MultiPolygon shape
- Can include holes

```
var latlngs = [
  [
    [37, -109.05],
    [41, -109.03],
    [41, -102.05],
    [37, -102.04],
  ], // outer ring
  [
    [37.29, -108.58],
    [40.71, -108.58],
    [40.71, -102.5],
    [37.29, -102.5],
  ], // hole
];
```

43

Polygons extend the Polyline class and also the Path and Layers classes.

Similar to Polylines, Polygons are defined using an array of arrays, and MultiPolygons, where a single feature is made up of multiple separated polygons, are defined using a multidimensional array. Note that it is also possible to define holes along with an outer ring.

In contrast to GeoJSON, the first coordinate should not be repeated as the last coordinate in the array. The last coordinate will automatically connect back to the first.

Polygon Styling

- Options: stroke, color, weight, opacity, lineCap, lineJoin, dashArray, dashOffset, fill, fillColor, fillOpacity, fillRule, renderer, className

```
var latlngs = [
  [37, -109.05],
  [41, -109.03],
  [41, -102.05],
  [37, -102.04],
];
var polygon = L.polygon(latlngs, { color: "red" }).addTo(map);
```

44

Similar to Circle, CircleMarker, and Polyline, a variety of styling options are available for the Polygon class.

Styling options are provided using JavaScript object notation (note the curly brackets).

The Polygon can then be added to the map using the addTo() method.

Polygon (Single Symbol)

```
lyrPolySingle = L.geoJSON.ajax("data/worldCountriesWGS84.geojson", {
  style: function (feature) {
    return {
      color: "#111111",
      weight: 1,
      fillColor: "#e5d8bd",
      fillOpacity: 1,
    };
  },
  onEachFeature: function (feature, layer) {
    layer.bindPopup(getPopup(feature));
  },
});
```

45

This slide demonstrates a single symbol symbology for Polygon features representing country boundaries.

Polygon (Categorical)

```
lyrPolyCategory = L.geoJSON.ajax("data/worldCountriesWGS84.geojson", {
          style: function (feature) {
            var fill;


            switch (feature.properties.INCOME_GRP) {
              case "1. High income: OECD":
                fill = "#7fc97f";
                break;
              case "2. High income: nonOECD":
                fill = "#beaed4";
                break;
              case "3. Upper middle income":
                fill = "#fdc086";
                break;
              case "4. Lower middle income":
                fill = "#ffff99";
                break;
              case "5. Low income":
                fill = "#386cb0";
                break;
              default:
                fill = "#000000";
            }
            return {
              color: "#333333",
              weight: 1,
              fillColor: fill,
              fillOpacity: 1,
            };
          },
          onEachFeature: function (feature, layer) {
            layer.bindPopup(getPopup(feature));
          },
        });
```

46

I am now using a switch statement to apply different symbology based on the income type for each country.

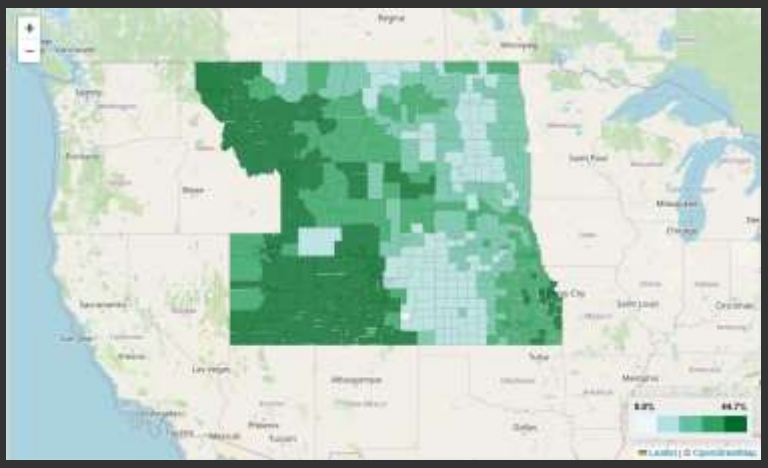
Polygon (Graduated Color)

```
lyrPolyGraduated = L.geoJSON.ajax("data/worldCountriesWGS84.geojson", {
        style: function (feature) {
          var val = feature.properties.POP_EST;
          var fill;


          // Example class breaks (replace with your own logic)
          if (val > 500000000) fill = "#bd0026";
          else if (val > 200000000) fill = "#f03b20";
          else if (val > 100000000) fill = "#fd8d3c";
          else if (val > 50000000) fill = "#fecc5c";
          else if (val > 25000000) fill = "#ffffb2";
          else fill = "#edf8e9";


          return {
            color: "#222222",
            weight: 1,
            fillColor: fill,
            fillOpacity: 1.0,
          };
        },
        onEachFeature: function (feature, layer) {
          layer.bindPopup(getPopup(feature));
        },
      });
```

47

I am now using ordered colors to show a continuous variable: estimated population. The symbols are defined using control flow.

Choropleth



- leaflet-choropleth plugin

https://github.com/timwis/leaflet-choropleth

```
<script src="src/choropleth.js"></script>
```

48

The leaflet-choropleth plugin makes it easier to create choropleth symbologies and associated legends.

Grouping

LayerGroup

- Group layers to handle as a single layer
- L.layerGroup()

```
L.layerGroup([marker1, marker2]).addLayer(polyline).addTo(map);
```

50

It is possible to merge multiple layers into a single group using the LayerGroup class. This can be useful if you want to work with layers collectively.

FeatureGroup

- L.featureGroup()
- Extends layer group
- Makes it easier to do the same thing to all members of the group

```
L.featureGroup([marker1, marker2, polyline])
  .bindPopup("Hello world!")
  .on("click", function () {
    alert("Clicked on a     member of the group!");
  })
  .addTo(map);
```

51

FeatureGroup extends FeatureLayer and makes it easier to do the same thing to multiple layers at once, such as add a popup to each layer in the group.

Both FeatureGroup and LayerGroup extend the Layer class.

Working with GeoJSON Data

GeoJSON

- Link to GeoJSON via a URL or stored within your website directory (for example, in “data” folder)
- Rendered using similar methods as those for makers, circleMarkers, circles, polylines, and polygons
- Extends featureGroup
- Use a function to define how GeoJSON spawns leaflet layers

53

Since JavaScript can directly read data stored in GeoJSON format, which is open-source and non-proprietary, this is a common format used in Leaflet. You can access GeoJSON files using a URL or by storing them in your website directory (for example, in the “data” folder). I generally use QGIS to convert files to GeoJSON for use in web mapping.

The GeoJSON Leaflet class extends the FeatureGroup class. So, GeoJSON features are read in and stored in a group. Remember that GeoJSON can store points, lines, and polygons in the same file, so the FeatureGroup can also contain these different geometries.

In order to symbolize the GeoJSON data in Leaflet, it is common to use a function to define how layers are spawned and what symbology is used.

Reading in GeoJSON

- L.geoJSON() or $.getJSON()
- L.geoJson.ajax() (from leaflet-ajax plugin)
  - Call JSON using AJAX

```
lyrpfor = L.geoJSON.ajax("data/per_for_wgs84.geojson", {
  style: stylePfor,
  onEachFeature: popPfor,
});

lyrcities = L.geoJSON
  .ajax("data/cities_wgs84.geojson", { pointToLayer: returnCtyStyle })
  .addTo(hpmap);
```

54

Leaflet provides the L.geoJSON() constructor method. You can also use $.getJSON() if you are storing the data locally.

You can use the “onEachFeature” option to specify actions to be performed when reading in each feature, such as binding a popup. The style argument can be referenced to a function that generates different symbology as defined with control flow statements or a switch.

For point data, the “pointToLayer” option is used to specify how Markers, Circles, or CircleMarkers are spawned from the GeoJSON data.

Another option for reading in GeoJSON is using the L.geoJSON.ajax() method form the leaflet-ajax plugin. This allows for AJAX to be applied to transfer the data. I prefer to use this method.

Styling GeoJSON

- Similar to Leaflet layers
- Can use a function

```
lyrstates = L.geoJSON
  .ajax("data/states_wgs84.geojson", {
    style: {
      fillColor: "#ffffff",
      color: "red",
      fillOpacity: 0,
      opacity: 1,
      weight: 1.5,
      pane: "pB",
    },
  })
  .addTo(hpmap);
```

55

If you want to apply the same style to all the features read in from the GeoJSON file, you can specify the styling options using JavaScript object notation as demonstrated here.

Styling GeoJSON

- Similar to Leaflet layers
- Can use a function

```
function stylePfor(json) {
  var att = json.properties;
  if (att.wester_158 <= 2) {
    var clrPfor = "#edf8fb";
  } else if (att.wester_158 > 2 && att.wester_158 <= 5) {
    var clrPfor = "#b2e2e2";
  } else if (att.wester_158 > 5 && att.wester_158 <= 10) {
    var clrPfor = "#66c2a4";
  } else if (att.wester_158 > 10 && att.wester_158 <= 15) {
    var clrPfor = "#2ca25f";
  } else {
    var clrPfor = "#006d2c";
  }
  return {
    fillColor: clrPfor,
    opacity: 1,
    color: "#222222",
    fillOpacity: 0.9,
    weight: 1,
    pane: "pC",
  };
}
```

56

If you want to apply different styling based on an attribute to represent different categories or ranges of values, this can be accomplished using a function and control flow.

Here, I am using an attribute provided with the data layer to assign different colors to different ranges of values. The symbology is then returned by the function.

This function will then need to be called using the style option or pointToLayer option.

Styling GeoJSON

```
lyrPolyCategory = L.geoJSON.ajax("data/worldCountriesWGS84.geojson",
{
          style: function (feature) {
            var fill;
            switch (feature.properties.INCOME_GRP) {
              case "1. High income: OECD":
                fill = "#7fc97f";
                break;
              case "2. High income: nonOECD":
                fill = "#beaed4";
                break;
              case "3. Upper middle income":
                fill = "#fdc086";
                break;
              case "4. Lower middle income":
                fill = "#ffff99";
                break;
              case "5. Low income":
                fill = "#386cb0";
                break;
              default:
                fill = "#000000";
            }
            return {
              color: "#333333",
              weight: 1,
              fillColor: fill,
              fillOpacity: 1,
            };
          },
          onEachFeature: function (feature, layer) {
            layer.bindPopup(getPopup(feature));
          },
        });
```

57

As an alternative to control flow, you can also use a switch to define styling to be applied to different cases.

Working with ESRI Layers

Working with ESRI Data Layers

- ESRI Leaflet plugin allows you to use ESRI data layers in Leaflet
- Can connect to layers made available through a data service, stored locally with your webpage, or hosted on ArcGIS Online
- Can connect to: ESRI Basemaps, ESRI Vector Basemaps, feature layers, hosted feature layers, raster tile layers, dynamic raster tile layers, map layers, and image map layers
- Allow for generation of symbology and pop-ups

http://esri.github.io/esri-leaflet/

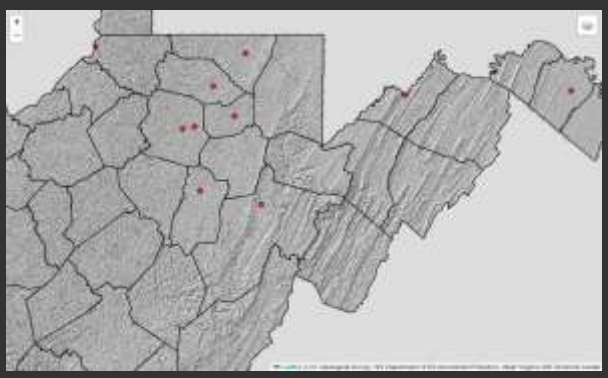
59

You can also use ESRI data layers in Leaflet with the esri-leaflet extension, including ESRI basemaps, ESRI vector tile basemaps, feature layers, hosted feature layers, raster tile layers, map layers, and map image layers. This plugin also allows for defining custom symbology and pop-ups.

You can link to data stored on ArcGIS-based data services, data stored locally in your website directory, or ArcGIS Online content. Data access may require licensing and fees.

Working with ESRI Data Layers

```
const imagery = L.esri.tiledMapLayer({
  url: "https://services.wvgis.wvu.edu/arcgis/rest/services/Imagery_BaseMaps_EarthCover/wv_imagery_NAIP_2011_1m_CIR/MapServer",
});

const counties = L.esri.featureLayer({
  url: "https://services.wvgis.wvu.edu/arcgis/rest/services/Boundaries/wv_political_boundary/MapServer/0",
  style: function (feature) {
    return {
      color: "#111111",
      weight: 2,
      fillColor: "#e5d8bd",
      fillOpacity: 0,
    };
  },
  onEachFeature: function (feature, layer) {
    layer.bindPopup(getPopupCounty(feature));
  }
});
```



http://esri.github.io/esri-leaflet/

60

Here are some examples of reading in raster tile layers and feature layers using the esri-leaflet package.

Controls

Zoom and Pan Control

- Leaflet has default zoom controls
- Can add different zoom controls with plugins
  - Example: Leaflet.zoomslider (http://kartena.github.io/Leaflet.zoomslider/)
  - Example: L.Control.ZoomBar (https://github.com/elrobis/L.Control.ZoomBar)
  - Example: Leaflet.twofingerzoom (https://github.com/aratcliffe/Leaflet.twofingerzoom)
  - Example: Leaflet.BoxZoom (https://github.com/gregallensworth/L.Control.BoxZoom)
- No pan control provide by base Leaflet so must use plug-ins
  - Example: Leaflet.Pancontrol (http://kartena.github.io/Leaflet.Pancontrol/)
  - Example: Leaflet.BorderPan (https://github.com/slara/Leaflet.BorderPan)

62

Leaflet provides a zoom control that is added by default. However, there are a variety of packages for adding other controllers. I have provided some examples here.

Layer Controls

```
ctlLayers = L.control.layers(objBasemaps, objOper).addTo(mymap);

objBasemaps = {
  OSM: lyrOSM,
  "OSM Terrain": lyrTer,
  "Night Lights": lyrNL,
  Dark: lyrDrk,
};

objOper = {
  Cities: lyrcities,
  "Percent Forest": lyrpfor,
  "Mean Annual Temp.": lyrtemp,
  "Total Annual Precip.": lyrprecip,
  States: lyrstates,
};
```

- L.control.layers()
- Can use as a base map switcher
- Can use to turn layers on and off

63

Layer controls are used to switch between base maps and/or turn operational layers on and off.

This is not added by default but can be turned on.

Only one base map can be active at once while multiple operational layers can be active.

In order to provide a list of layers to include in the control, you need to generate variables that store the base map layers and operational layers in object notation. The provided name will be used in the table.

The addTo() method is used to add the controls to the map.

Scale Control

- L.control.scale()
- Options:
  - Maxwidth
  - Metric
  - Imperial
  - undateWhenIdle
  - position

```
L.control.scale().addTo(map);
```

64

A scale bar is not added by default, but you can use the L.control.scale() method to add a scale bar. There are also several options available.

Zoom Slider

- Leaflet.zoomslider

http://kartena.github.io/Leaflet.zoomslider/

```
<script src="src/L.Control.Zoomslider.js"></script>
```

65

The Leaflet.zoomslider plugin provides an alternative Zoom slider to replace the default leaflet zoom buttons.

Popups and Tooltips

Popups vs. Tooltips

- Generate a popup tied to a map feature or coordinate
- Tooltip generally used for text only

67

Popups and tooltips are both user interface (UI) layers similar to Markers.

A popup can hold a wide variety of content while a tooltip is meant to only hold text.

Build Popups

- Use .bindPopup() or L.popup()
- Options: maxWidth, minWidth, maxHeight, minHeight, keepInView, closeButton, className
- Content specified using HTML and attribute data
- Style with CSS using className option

```
L.circleMarker(latlng, optCty).bindPopup(
  "<p><b>" +
    att.NAME +
    ", " +
    att.ST +
    "</b><br>Elevation: " +
    att.elev +
    " <br>Annual Precip: " +
    att.precip +
    "</p>",
  popupOptions,
);
```

68

A popup can be generated using the L.popup() constructor method or within the bindPopup() method. A variety of option are available.

Note that popups can be styled with custom CSS using the “className” option. You can think of popups as small webpages within your map and webpage.

Popup content is defined using HTML. You can also call attribute names. It is possible to include text, links, images, and other HTML content.

A popup is being added to a layer in the provided example using bindPopup(). Note the use of HTML to define the popup content and the use of attribute variables. I also provide some popup options as a variable.

Build Tooltip

- .bindTooltip() or L.tooltip()
- Similar to methods for pop-up

```
return L.marker(latlng, optCty).bindTooltip("Font-Awesome Icon");
```

69

This is an example of binding a tooltip to a Marker.

Handling Events

Common Events

- Change view extent
- Change map center
- Change zoom level
- Add/remove a layer
- Change layer symbology
- Pan to a location

71

In order to add additional functionality to a map, you can design event handlers that accomplish a certain task. This slide lists some common web map events. In the following examples, I demonstrate some event handlers that make use of jQuery.

Set/Change Center and Zoom

- Example using jQuery
- Other useful map methods: setZoom(), zoomIn(), zoomOut(), fitBounds(), fitWorld(), panTo(), flyTo()

```
$("#btnSalz").click(function () {
  pmap.setView([47.798, 13.059], (zoom = 12));
});
```

72

In this example, I am creating an event to set or change the map view.

A button is assigned to the “btnSalz” class. When the button is clicked, the center and zoom level of the map will be changed for the “pmap” map object using the setView() method.

The zoom level can be set using setZoom(), zoomIn(), or zoomOut(). You can fit the map to a defined rectangular bounds or to the bounds of a data layer using fitBounds(). You can also pan or fly to a location using panTo() or flyTo(). You can zoom to a global extent using fitWorld().

Add and Remove Layers

- Useful methods: addLayer(), removeLayer(), hasLayer(), eachLayer()

```
$("#btnAddRem").click(function () {
  if (pmap.hasLayer(lyrPhotos) == true) {
    pmap.removeLayer(lyrPhotos);
  } else {
    pmap.addLayer(lyrPhotos);
  }
});
```

73

This slide demonstrates how to add or remove layers with the click of a button.

A button is assigned to the “btnAddRem” class. If the button is clicked, then a function is executed. If the “lyrPhotos” layer is loaded, then the layer will be removed. If it is not loaded, then it will be added.

To test to see if the layer is load, the hasLayer() method is used. To add or remove a layer, addLayer() or removeLayer() is used.

The control flow allows for different responses based on the map state.

Change Symbology or Style

```
$("#btnMrk2").click(function () {
  pmap.removeLayer(lyrPhotos);
  lyrPhotos = L.geoJSON
    .ajax("data/photos_points.geojson", { pointToLayer: returnPntStyle1 })
    .addTo(pmap);
});
```

74

In this example, the layer symbology is changed with the click of a button.

When a button assigned to the “btnMrk2” class is clicked, the “lyrPhotos” layer is removed form the “pmap” map object. It can be added to the map again with a different symbology, which is defined by the “returnPntStyle1” function.

Easy Button

- Plugin that allows you to easily add a button to a Leaflet map and set up an action or an event for it to handle
- Can use custom icons or icons from a library
- L.easyButton() = https://github.com/CliffCloud/Leaflet.EasyButton

```
ctlEasybutton = L.easyButton("fa-compress-alt", function () {
  ctlSidebar.toggle();
}).addTo(pmap);
```

75

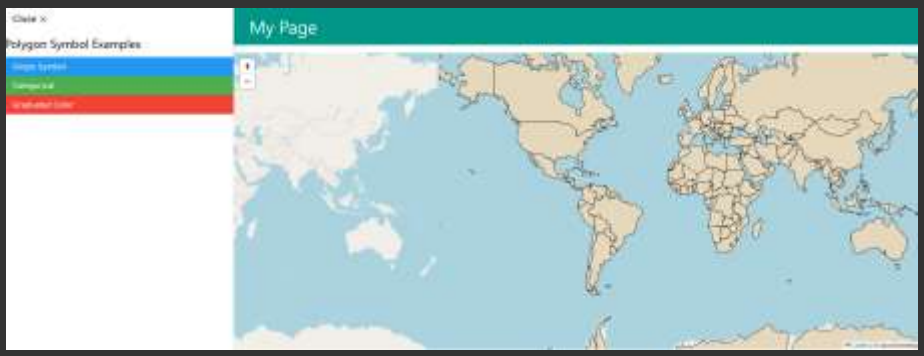
The Leaflet.EasyButton plugin allows you to add a button to the map that performs a certain action or event. You can even apply custom icons to the button.

In this example, the easy button is assigned a logo from the Font Awesome library. When the button is clicked, it causes a sidebar to toggle between open and closed.

Page Layout

Map is just one

- Can insert map into a page with a variety of content
- Elements outside of map can be used to interact with map



77

As a final note, maps can be a single component within a page with additional content. The map can even interact with other page content. In the example, the buttons in the side bar can be clicked to cause a response on the map.

This is the end of this lecture module.

Please return to the West Virginia View Webpage for additional content.

Thanks! Hope you found this useful.