Python Server: `python -m SimpleHTTPServer,` **then open** http://localhost:8000/

JavaScript

- Interactive window: **confirm(**"<string>"**)**
- Input window: **prompt(**"<string>"**)**
- Comment: // and Commenting multiple lines: /* multiple lines */
- Length: "<string>"**.length**
- Print out: **console.log( )**
- If: **if(**<condition>**) {**Code to run when if condition is true**} else if ( ) {** Code to run when else if condition is true **} else {**Code to run when else condition is true**}**
- Substring: "<string>"**.substring(**start, end**)**
- Saving values by defining a variable: **var** varName = data
- Function: **var** functionName = **function (**<input**>) {**code to do something**; return** result**};**
- The variables defined outside a function are accessible once they have been declared – global variables (scope is global)
- Random number between 0 and 1: **Math.random( )**
- For loop: **for (var** i = 1; i < n; i = i + 1**) {**code to iterate**}**
- Increment by 1: i**++** and increment by n: i **+=** n
- Decrement by n: i **-=** n
- Arrays: **var** arrayName = **[**data, data, data, …, data**]**
- Method to append element to an array: arrayName**.push(**<value>**)**
- Rounding: **Math.floor( )**
- While loops: **while(**condition**) {**code to iterate**}**
- Is NAN: **isNaN(**<string or value>**)**
- JavaScript with try to match the expression between the **switch( )** parentheses to each **case**. It will run the code below each case if it finds a match, and execute **default** code if no match is found.

```
switch (/*Some expression*/) {
    case 'option1':
        // Do something
        break;
    case 'option2':
        // Do something else
        break;
    case 'option3':
        // Do a third thing
        break;
    default:
       // Do yet another thing
}
```

- Logical operators: AND - **&&** OR - || NOT - **!**
- Convert to upper case letters: **.toUpperCase( )**
- Convert to lower case letters: **.toLowerCase( )**
- **for (var x in y) {}**
- Object using *literal notation*: **var objectName= {propertyName: value, }**
- Accessing property of an Object using 'dot notation': **objectName.propertyName**
- Accessing property of an Object using 'bracket notation': **objectName["propertyName"]**
- Object using *constructor notation*: **var objectName = new Object( )** then assign properties like **objectName.propertyName = value**

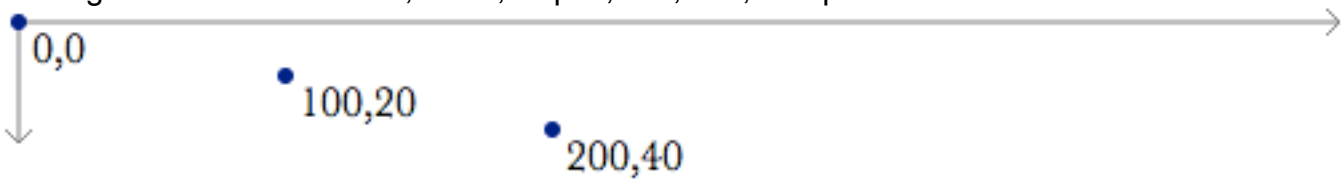
- Variables associated with an object – properties and function associated with an object – method: **objectName.methodName = function (**<input**>) {**code to do something**; return** result**};**
- Methods can be used to make calculations based on object properties.
- Calling method: **objectName.methodName(**<input**>)**
- Constructor: **function functionName (x) {this.x = x;}** and **var** temp **= new functionName(**<value>**)**
- **typeof** varName
- objName**.hasOwnProperty( )**
- **objectName.prototype.methodName(**<input**>)**

Some Commands for JavaScript Console

- Clear console: **clear( )** or **command + k**
- Return DOM document contain within it is HTML of the page: **document**
- Browser window: **window**
- Selecting elements of webpage by their class, id, or tag, while working with D3, type: **document.get**FUNCTION**(" ")**
- Query Selector allows to CSS selector syntax to grab DOM elements: **document.querySelector (. Or #)**
- Selecting element with D3: **d3.select(**id or class or tag**)** and remember: # is ID and . is CLASS
- Manipulating CSS style of selection using D3 transformation function: **d3.select(**id or class or tag**).style(**'option', value**)**
- Selecting all elements with D3: **d3.selectAll()**
- Make variable and then type variableName. for auto completion
- Get method documentation – type method without (): d3.select
- If there are two classes **a** and **b**, then selection is like: d3.select('tag**.a.b**')
- Selection with ID **a**: d3.select('tag**#a**)
- Changing any attribute defining HTML tag using attribute method: **.attr()**
- Removing an element: **d3.select( ).remove( )**
- selection**.html()**
- Converting data values to pixel values: **d3.scale()**
- Data Values: .domain(min, max); Pixel Values that we want to map Data Values to: .range(min, max) i.e. Converting Data Values to Pixel Values using a function (linear, log, etc.). Function that maps domain into the range. Webpage Coordinate: Top left 0: Value increases from left to right and top to down. var x = d3.scale.log().domain([min, max]).range([pxMin, pxMax]) is a function
- Appending circle to an element: element.append('circle').attr()
- Circle: cx, cy, fill, r,
- Debugging: **debugger;**
- To inspect a table on Chrome console: **console.table(**data**)**
- To take a slice of an array: data**.slice(**start, end**)**
- Return the min and max value in the given arrary: **d3.extent(**array, accessor**)**
- Dealing with time: **d3.time**
- Formatting time: **d3.time.format** then **parse**
- Converting string to a number using **Unary Plus** or **+**: **+ "7"** = 7 (for negative integer use **-)**
- Longitude: x; Latitude: y
- Converting Lat/Long data to pixel values: **d3.mercator**
- Grouping data: **d3.nest()** -> **.key** and **.rollup**

Scott Murray

- **d3.select(**"element"**) –** Selects "element: from DOM using CSS selector syntax
- **d3.select(**"element"**).append(**"newElement"**) –** Creates "newElement" and appends to the end of our selection i.e. before </element> tag
- **.text(**"someString"**)** or **.text(**someFunction( )**)** for setting text content
- D3 employs *chain syntax*
- API Reference on functions/methods: https://github.com/mbostock/d3/wiki/API-Reference
- Data binding – attaching or associating input data to specific elements in the DOM. Two things are needed: (1) Data; (2) Selection of DOM elements
- **.data(**data**) –** Counts and parses data values
- **.enter( )** – This method looks at the DOM, and then at the data handed to it. If there are more data values than corresponding DOM element, **enter( )** creates new placeholder element. Then **.append(**"element"**)** takes placeholder element and inserts "element" into the DOM
- After you call **data( )** and use *chaining methods* you can create an anonymous function that accepts **d** as input. e.g. function(d){return d;};
- Writing your own function: **function(**input_value**) {**// Calculate something
  return output_value;**}**
- **.style(**"property", "value"**)** or **.style(**"property", someFunction( )**)** – For setting CSS properties on selection
- **.attr( )** – For setting HTML attributes
- **.classed(**"className", true**)** – Remove or apply classes from elements
- **for (var** i = 1; i < n; i = i + 1**) {**code to iterate**}** - For loop syntax
- arrayName**.push(**<value>**)** - Method to append new value to the end of an array
- svg visual elements: rect, circle, ellipse, line, text, and path



- Styling svg elements: **fill** – Color value for inside; **stroke** – Color value for border; **stroke-width** – Numeric value for border width; **opacity** – Numeric value (0.0 transparent to 1.0 opaque); text: **font-family**; **font-size**
- **svg.attr(**"width", num value**).attr(**"height", num value**)** – Setting up svg
- **function(d, i){**calculate something with position i**};**
- Object using *literal notation*: **var objectName= {propertyName: value, }**
- Accessing property of an Object using 'dot notation': **objectName.propertyName**
- Accessing property of an Object using 'bracket notation': **objectName["propertyName"]**
- Object + Arrays: **var** objArray = **[{**property: value**}, {**property: value**}, {**property: value**}]** and accessing value: objArray**[**index**].**property
- JSON: **var** jsonData = **{**"property-1": "value-1", "property-2": "value-2"**}** and GeoJSON – Long/Lat
- When **d** is an array of values (not just single value, like 3.14), use bracket notation to access its values: **d[0]**
- **Scales are functions** that map from an input domain to an output range. Call the scale function, pass it a data value, it returns a scaled output value. Remember in JavaScript **variables** can store **functions**

- A scale's input **domain** is the range of possible input data values (min and max values). A scales output **range** is the range of possible values in pixel units.
- Creating scale: **var** scale = **d3.scale.**typeOfScale()**.domain(**[min, max].**).range(**[pxMin, pxMax]**)**
- Maximum value in data: **d3.max(**data, function(d){return d}**)**
- Remember need to scale "x" and "y" values in attributes of visual elements and texts
- Scales: **linear, sqrt, pow, log, quantize, quantile, ordinal**
- **d3.scale.linear()** has some methods: **nice(), rangeRound(), clamp()**
- Scale can be used on "fill" also for gradient colors
- **Axes are functions** that generate visual elements of the axis (*lines, labels, ticks*): **d3.svg.axis()**
- Axis need information about scale to operate
- AXIS Need TO do

- **d3.csv(**"data.csv", Callback Function **function(){})** – Loading a csv
- In case of an error: Use **console.log(d)** before **return d**

d3Vienno – YouTube

- **D3.js** – JavaScript library for manipulating documents based on data.
- API Reference for d3: **https://github.com/mbostock/d3/wiki/API-Reference**
- Inside Head element: <script src = '**http://d3js.org/d3.v3.min.js'**></script>
- **d3** – Refers to an object
- **console.log(**object**)** – To inspect the object in console
- **SVG** – Scalable vector graphics
- **.style("**property**", "**value**")** – Method to change stylistic properties of elements
- **.attr("**attribute**", "**value**")** – Method to change attributes of elements (SVG)
- To create SVG, start by creating a **container** or **canvas** for visualization. Done by appending **svg** element to page body. Specify **width** and **height** attributes. Then shapes can be appended to canvas.
- In SVG to color an element use "**fill**" attribute
- **Circle position attributes:** "cx", "cy", "r"
- **Rectangle position attributes:** "width", "height"
- **Line position attributes:** "x1", "y1", "x2", "y2". Other attributes: "stroke" and "stroke-width"
- **.data(**dataVariable**)** – This method binds data to DOM elements
- **.enter( )** – Contains placeholder for data elements for which there are no corresponding DOM elements
- **.scale( )** – Takes input data (domain) and transforms it into a range that will fit on canvas. Scales: **linear, sqrt, pow, log, quantize, quantile, ordinal**.
- **.domain([**min dataValue, max dataValue**])**
- **.range([**0, canvas dimension**])**. For color .range([color-1, color-2]) is used
- **Group –** Grouping SVG elements together to transform them as whole. **.append(**"g"**)** to canvas. Then use **transform** attribute to move group up or down or left or right with value **translate(x pixels, y pixels)**
- **Axes are functions** that generate visual elements of the axis (*lines, labels, ticks*): **d3.svg.axis()**. Axis need information about scale to operate
- **.call(**axis**)** – Calling a variable

- **Enter, Update, Exit** – (1) DOM Elements < Data Elements – Resulting selection: **enter** (2) DOM Elements > Data Elements – Resulting selection: **exit** (3) DOM Elements = Data Elements – Resulting selection: **update**
- Transitions – Animations. **.transition( )**
- **.duration(**milliseconds**)** – Transition time
- **.delay(**milliseconds**)** – Transition only after delay of specified milliseconds
- **.each**('end', function( ){ }**)** – Event listener
- Loading external data: JSON – **d3.json(**'dataFileName', **function(**data**){**code…**})**. CSV – **d3.csv(**'dataFileName', **function(**data**){**code…**})**.
- Paths – Component of SVG. Paths can be used to create any shape
- Arc – Part of circle.