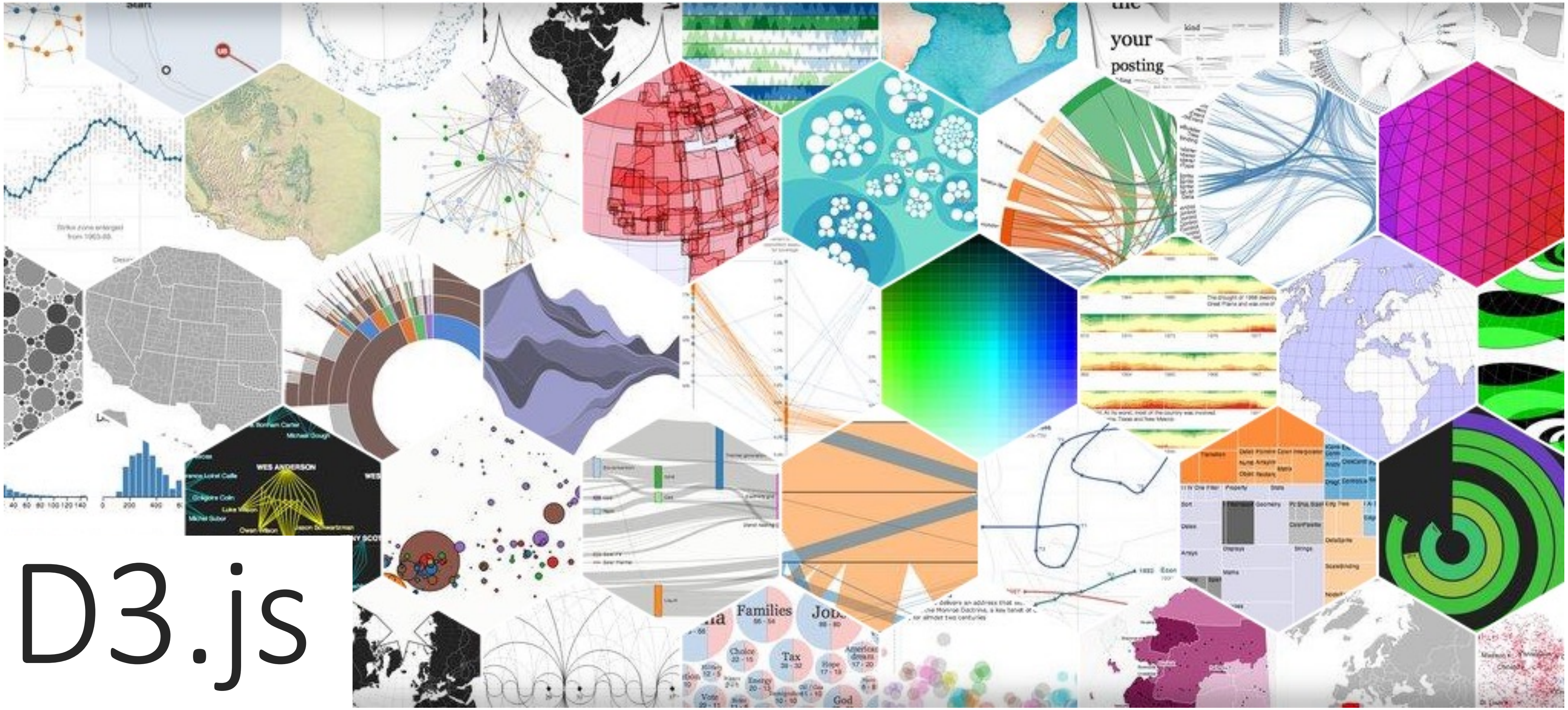


D3.js

DATA-DRIVEN DOCUMENTS (D3JS.ORG/)

What is D3?

D3 stands for Data-Driven Documents

Open-source JavaScript library developed by Mike Bostock to create custom interactive data visualizations

Official website: d3js.org

D3 Source code: https://github.com/d3/d3

D3 features

Uses Web Standards: SVG, HTML, and CSS

Driven by static or fetched data from a remote server in different formats (Arrays, Objects, CSV, JSON, XML, etc.)

Manipulate the Document Object Model (DOM) based on your data

No standard visualization format → complete control over your visualization

`transition()` → interpolate between values

Great support for animation (`duration()`, `delay()`, `ease()`)

Animations are fast and responsive to user interactions

DOM (Document Object Model)

When a web page is loaded, the browser creates a Document Object Model of the page.

The HTML DOM is a standard **object** model and a **programming interface** for HTML. It defines:

- HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

The HTML DOM is a standard for how to get, change, add, or delete HTML elements

Selecting DOM elements

To manipulate DOM elements we first need to select a particular element

`d3.select(css-selector)` → returns the first matching element in the HTML document based on specified `css-selector`

- Typical `css-selector` : the element's unique identifier

`d3.selectAll(css-selector)` → returns all matching elements

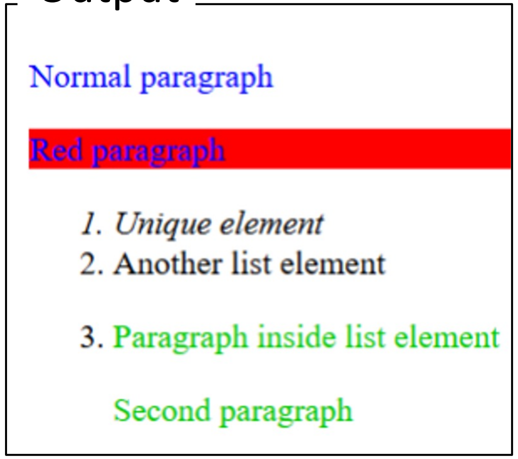
- Typical `css-selectors`: the element's class or name

CSS (Cascading Stylesheets)

CSS styles (a.k.a selectors) are applied to HTML tags by **name**, **class**, or **identifier**

```
<html>
    <head> <title>TITLE GOES HERE</title></head>
    <style>
        p { color: blue; } /* Applied to all <p> tags */
        .red {  background: red; } /* Applied to all tags with the class “red” */
        #some-id { font-style: italic; } /* Applied to the tag with the id “some-id” */
        li p { color: #0C0; } /* Applied only to <p> tags inside <li> tags */

    </style>
    <body>
        <div> <p> Normal Paragraph</p> <p class=“red”>Red Paragraph</p> </div>

        <ol>
            <li id=“some-id”>Unique element</li>
            <li>Another list element</li>
            <li> <p>Paragraph inside list element</p> <p>Second paragraph</p></li>
        </ol>
    </body>
</html>
```

Output



Normal paragraph

Red paragraph

1. *Unique element*
2. Another list element

3. Paragraph inside list element

   Second paragraph

D3 selection: example

Import the d3.js library

```
<html>
    <head> <title>TITLE GOES HERE</title>
        <script type="text/javascript" src="https://d3js.org/d3.v5.min.js"></script>
    </head>
    <body>
        <div> <p> Normal Paragraph</p> <p class=“red”>Red Paragraph</p> </div>

        <ol>
            <li id=“some-id”>Unique element</li>
            <li>Another list element</li>
            <li> <p>Paragraph inside list element</p> <p>Second paragraph</p></li>
        </ol>

        <script>
            d3.select("#some-id") // [Array(1)]

            d3.selectAll("p").size(); // 4

            let reds = d3.selectAll(".red") // [Array(1)]

            console.log(reds.text()) // “Red Paragraph”
        </script>
    </body>
</html>
```

The css selector must be prefixed by
- A hash sign (#) for identifiers
- A dot (.) for classes

Modifying selected DOM element(s)

`text(“content”)` → gets or set the text

`append(“element name”)` → adds an element inside it

`remove()` → removes it from the DOM

`html(“content”)` → gets or sets the inner HTML

`attr(“name”, “value”)` → gets or sets an attribute

`property(“name”, “value”)` → gets or sets a property

`style(“name”, “value”)` → gets or sets the style

`classed(“css class”, bool)` → gets, adds or remove a css class

Method chaining in D3

In D3 methods are chained together using a period

```
d3.select(“body”)                 // selects the HTML element “body”
    .append(“p”)                  // adds a paragraph element to it
    .attr(“id”, “myParagraph”)    // defines a unique identifier
    .classed(“par”, true)         // assign the css class “par” to the element
    .style(“color”, “red”)        // set the color of text to red
    .text(“Hello World”)          // define the text to display
```

Function of Data

Each DOM manipulation methods can take a constant value or function as parameter

- This function is a function of data

Each method is called for each of our data values bound to the DOM

- We can apply any logic to manipulate data

```
.text(function (d, i) {
    console.log(d); // the data element
    console.log(i); // the index element
    console.log(this); // the current DOM object

    return d;
});
```

Other than the data (or d) parameter, there are two other parameters available to us.

Event Handling

`D3.selection.on()` → bind an event listener to any DOM element ; it takes 2 parameters

- an event type → click, mouseover, etc.
- a callback function → executed when and event occurs

`D3.mouse(container)` → gets the x and y coordinates of the current mouse position in the specified DOM element

`D3.event` → contains event data such as `timestamp` and methods such as `preventDefault`

Animations

Animation : a transition from one form to another

D3.selection.`transition()` → makes a transition on any DOM element

Useful methods:

`selection.transition()` → schedules a transition for the selected element

`transition.duration()` → sets the animation duration in milliseconds for each element

`transition.ease()` → sets the easing function (linear, elastic, bounce, etc.)

`transition.delay()` → sets the animation delay in milliseconds for each element

Data Binding

→ Bind data to DOM elements

→ Create new elements based on data

`data()` → joins data to the selected elements

`enter()` → creates a selection with placeholder references for missing elements

`exit()` → removes nodes and adds them to the exit selection, which can be later removed from the DOM

`datum()` → injects data to the selected element without computing a join

The data() function

Input : an array of values (number or object) or a function of data

```
<p>D3 Tutorials</p>     → <p>Hello World</p>
<p> </p>                → <p>Hello D3</p>
<p> </p>                → <p>Hello JavaScript</p>

<script>
        // the data is an array of strings
        let myData = ["Hello World!", "Hello D3", "Hello JavaScript"];

        let p = d3.select("body")
            .selectAll("p")      // select all <p> elements from the page
            .data(myData)        // join the new data
            .text(function (d) {    // create the text to be displayed based on the data
                return d;           // d is a value of myData
            });
</script>
```

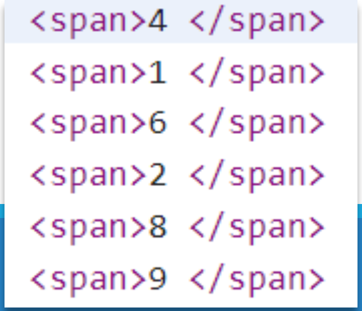
Try it at https://www.tutorialsteacher.com/codeeditor?cid=d3-23

The enter() function

What if number of elements and data values do not match?

- Lesser elements than the dataset or no selection at all (no HTML code in place)

```
<body>          // the HTML code contains only the body, with no other elements
<script>
    let data = [4, 1, 6, 2, 8, 9]; // there are 6 data values in our data array

    let body = d3.select("body")
                 .selectAll("span")
                 .data(data)
                 .enter()             // creates 6 reference placeholders
                 .append("span")      // append 6 span elements
                 .text(function(d) { return d + " "; });
</script>

</body>
```



```
<span>4 </span>
<span>1 </span>
<span>6 </span>
<span>2 </span>
<span>8 </span>
<span>9 </span>
```

Try it at https://www.tutorialsteacher.com/codeeditor?cid=d3-25

The `exit()` function

More elements than data values

- `.exit().remove()` → remove additional elements

```
<body>
    <p>D3 Tutorials</p>   → <p>Hello World</p>
    ~~<p></p>~~
    ~~<p></p>~~
    <script>

    let myData = ["Hello World!"];      // 1 data value
    let p = d3.select("body")
                .selectAll("p")         // 3 <p> elements selected
                .data(myData)
                .text(function (d, i) { return d; })
                .exit()      // place the 2 additional <p> elements in an exit selection
                .remove();              // remove the 2 <p> elements
    </script>
</body>
```

Try it at https://www.tutorialsteacher.com/codeeditor?cid=d3-28

The `datum()` function

Typically used for static visualizations that do not need updates

- It binds data directly to an element

```
<body>
    <p>D3 Tutorials</p>    → <p>100</p>
    <script>

    d3.select("body")
        .select("p")
        .datum(100)
        .text(function (d, i) {
            return d;
        });
    </script>
</body>
```

Try it at https://www.tutorialsteacher.com/codeeditor?cid=d3-29

Data Loading

Different types of data defined either locally in variables or from external files

Methods to load data from external files

- `d3.csv()` → for loading CSV files
- `d3.json()` → for loading JSON files
- `d3.tsv()` → for loading TSV files
- `d3.xml()` → for loading XML files

Parameters

- URL (local or distant) to the file
- callback function to treat the parsed data objects (all data becomes a JSON object after loading)

Using Promises to load data files

Promise → object that represents the eventual completion (or failure) of an asynchronous method and its resulting value

- Use it to wait for an asynchronous method to finish executing

```
d3.json(“filepath.json”).then(data => {
    // do something with data
})
```

Loading multiple files

```
Promise.all([d3.json(“filepath1.json”), d3.csv(“filepath2.csv”)]).then(datafiles => {
    let data1 = datafiles[0],
    let data2 = datafiles[1]
    // do something with data
})
```

More about promises at https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Creating SVG Elements

SVG is a XML format used for drawing

Similar to DOM, it has elements with parents, children, and attributes

- They also respond to the same mouse/touch events

SVG defines tags for basic shapes

- `<rect>` for rectangles
- `<circle>` for circles
- `<line>` for straight lines

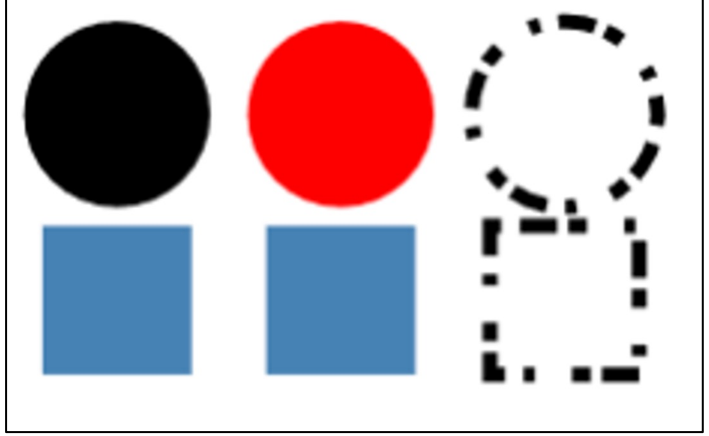
Some CSS syntax used for DOM change for SVG elements

- `background-color: red;` → `fill: red;`

SVG: example

```
<html>  <head> <title>TITLE GOES HERE</title> </head>
    <style>
        .red { fill: red; /* not background-color */ }
        .fancy { fill: none; stroke: black;
            stroke-width: 3pt;
            stroke-dasharray: 3,5,10; }
    </style>
    <body>
        <svg width="300" height="180">
            <circle cx="30" cy="50" r="25" />
            <circle cx="90" cy="50" r="25" class="red" />
            <circle cx="150" cy="50" r="25" class="fancy" />

            <rect x="10" y="80" width="40" height="40" fill="steelBlue" />
            <rect x="70" y="80" width="40" height="40" style="fill: steelBlue;" />
            <rect x="130" y="80" width="40" height="40" class="fancy" />
        </svg>
    </body>
</html>
```

Output



Groups in SVG

The <g> tag allows to group elements

- The grouped elements inherit attributes from the <g> tag (styles, position, etc.)

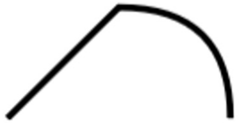
Widely used to create charts with D3

```
<svg viewBox="0 0 100 100" >
    <!-- Using g to inherit presentation attributes -->
    <g fill="white" stroke="green" stroke-width="5">
        <circle cx="40" cy="40" r="25" />
        <circle cx="60" cy="60" r="25" />
    </g>
</svg>
```

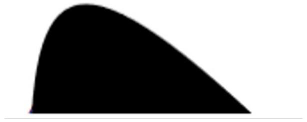
Paths in SVG

The **`<path>`** tag helps to draw lines and arbitrary shapes

- Powerful and complex



```
<path d="M0,50 L50,0 Q100,0 100,50"
    fill="none" stroke-width="3" stroke="black" />
```



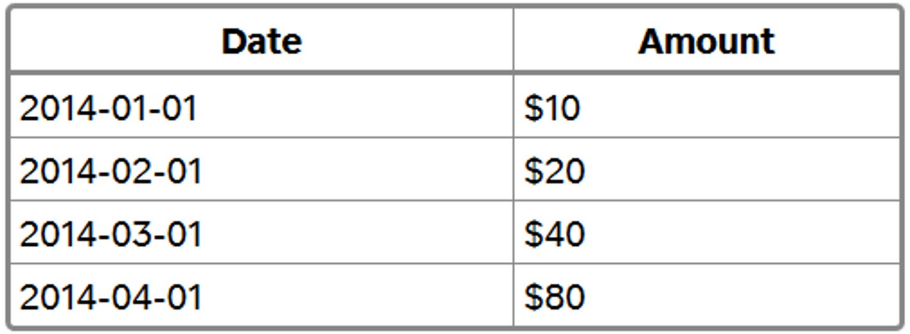
```
<path d="M0,100 C0,0 25,0 125,100 z" fill="black" />
```

Used in most D3 charts (pie chart, line chart, etc)

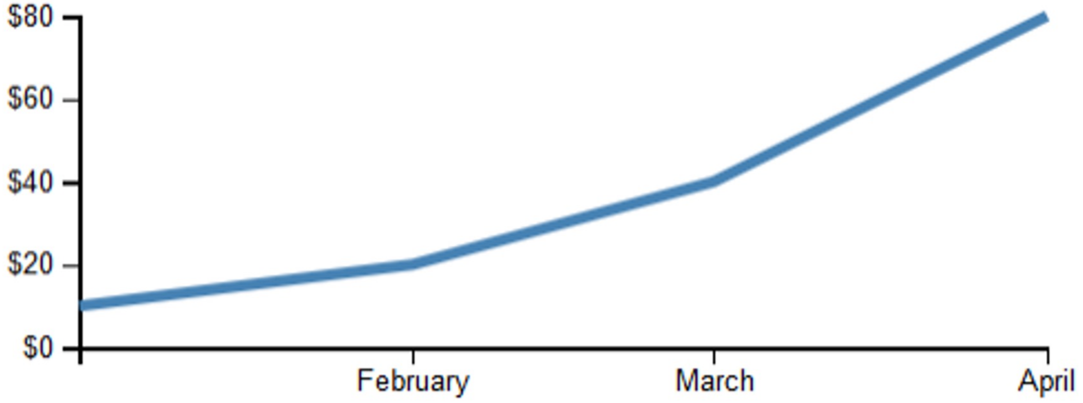
D3 has methods to automatically create paths according to the data

- **`d3.line`**, **`d3.arc`**, **`d3.pie`**, etc.

Creating a chart with SVG and D3



Date	Amount
2014-01-01	$10
2014-02-01	$20
2014-03-01	$40
2014-04-01	$80



Draw chart elements: data, scales, axes

Chart elements

The scale → the coordinate system

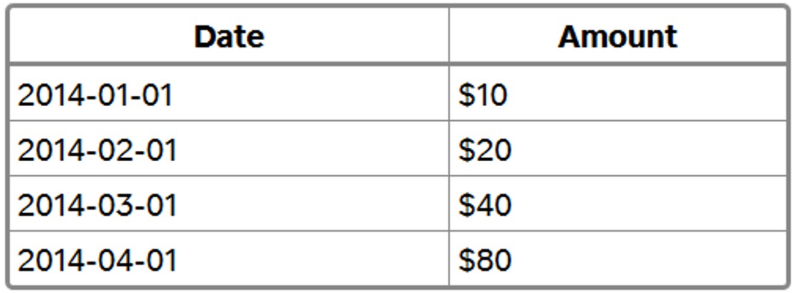
- X axis → from January 2014 to April 2014
- Y axis → from $10 to $80
- Specify the mapping between data values and pixels of the screen

The axes → the labels

- Text elements representing values such as “$10” and “February”
- Define the right display format according to the data type

The data

- Each row becomes a point over the line
- The points must fit the defined coordinate system



Date	Amount
2014-01-01	$10
2014-02-01	$20
2014-03-01	$40
2014-04-01	$80

Create a chart with SVG

```
<html> <head> <title>Line chart with pure SVG</title> </head>
     <style>
         .axis line{ stroke:  #000;}
         path { stroke: #000; fill: none; } </style>
     <body> <svg width="350" height="160">
        <g class="layer" transform="translate(60,10)">
         <circle r="5" cx="0" cy="105" />     // data value #1
         <circle r="5" cx="90" cy="90" />     // data value #2
         <circle r="5" cx="180" cy="60" />    // data value #3
         <circle r="5" cx="270" cy="0" />     // data value #4

         <path d="M 0 105 L 90 90 L 180 60 L 270 0" />  // line connecting dots

         <g class="y axis">
              <line x1="0" y1="0" x2="0" y2="120" />        // line that represent the y axis
              <text x="-40" y="105" dy="5">$10</text>       // label for lowest value
              <text x="-40" y="0" dy="5">$80</text> </g>    // label for highest value

         <g class="x axis" transform="translate(0, 120)">        // translate to the bottom of chart
              <line x1="0" y1="0" x2="270" y2="0" />             // line representing x axis
              <text x="-30" y="20" dx="5">January 2014</text>    // label for lowest value
              <text x="240" y="20" dx="5">April 2014</text> </g> // label for highest value
         </g> </svg> </body>
</html>
```

Create a chart with D3

Define the dataset

```
let data = [{ date: "2014-01-01", amount:10 },
            { date: "2014-02-01", amount:20 },
            { date: "2014-03-01", amount:40 },
            { date: "2014-04-01", amount:80 } ]
```

Define the dimensions of the chart

```
let margin = { left: 20, top: 10, bottom: 20, right: 10 }, // the margins of the chart
        width = 350, // the width of the svg
        height = 160; // the height of the svg
```

Set the SVG dimensions

```
let svg = d3.select("svg")
          .attr('width', width + margin.left + margin.right)
          .attr('height', height + margin.top + margin.bottom)
```

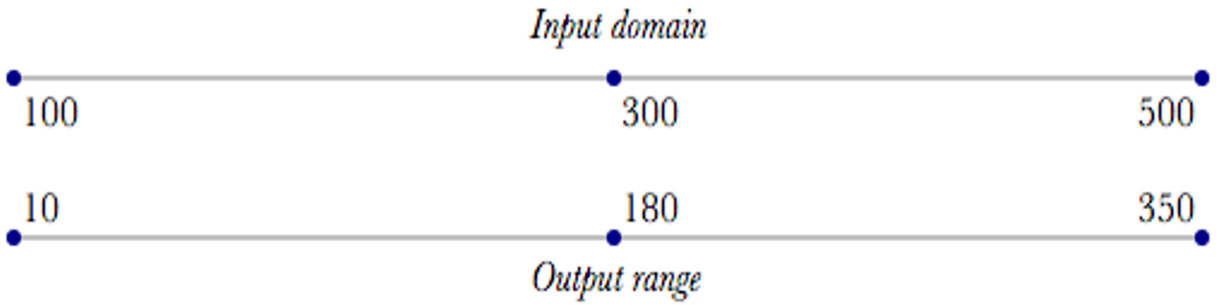
Create a chart group

```
let chartGroup = svg.append("g")
          .attr('transform', "translate(" + margin.left + "," + margin.top + ")")
```

Scales in D3

Scaling methods to different types of charts and data

- `d3.scaleLinear`, `d3.scaleTime`, `d3.scaleSequential`, `d3.scaleOrdinal`, etc.

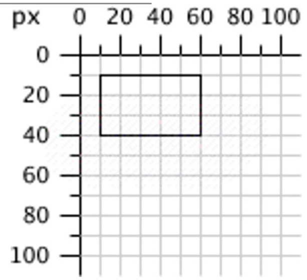


Domain → minimum and maximum values of the input data

Range → the output range that we would like our input values to map to

Defining the Y scale

```
let yScale = d3.scaleLinear()
    .domain([0, 80]) // $0 to $80
    .range([height, 0]) // seems backwards because SVG is y-down
```



Default coordinate system

`yScale` is accessed using the function syntax

- Used to translate values from one coordinate to another
- Domain ↔ range

```
yScale(10); // in: $10
// 200      // out: 200px (bottom of graph)

yScale(80);  // in: $80
// 0          // out: 0px (top of graph)
```

Defining the X scale

Transform the string object into a `Date` object recognizable by `d3.scaleTime`

```
let xScale = d3.scaleTime()
    .domain([
        new Date("2014-01-01"),
        new Date("2014-04-01")
    ])
    .range([0, width])

xScale(new Date("2014-02-01"))
// 124
```

Scales can also be used for arbitrary transformations

- e.g., mapping between data and colors

The `min`, `max`, and `extent` functions

Automate operations such as finding the minimum and maximum values of a dataset (or both → the extent)

Parameters : an array of values or a function of data

```
let values = [10, 20, 40, 80]
let data = [{ date: “2014-01-01”, amount:10 }, { date: “2014-02-01”, amount:20 },
    { date: “2014-03-01”, amount:40 }, { date: “2014-04-01”, amount:80 } ]

d3.max(values)
// 80
d3.max(data, (d,i) => d.amount)
// 80

d3.extent(values)
// [10, 80]
d3.extent(data, (d,i) => d.amount)
// [10, 80]
```

Define the yScale using the extent function

```
let yScale = d3.scaleLinear()
    .domain(d3.extent(data, d => d.amount))
    .range([height, 0])
```

Axes in D3

Render human-readable reference marks for scales

Made of lines, ticks and labels

Uses scales → each axis need to be given a scale to work with

`d3.axisTop()` → creates a top horizontal axis

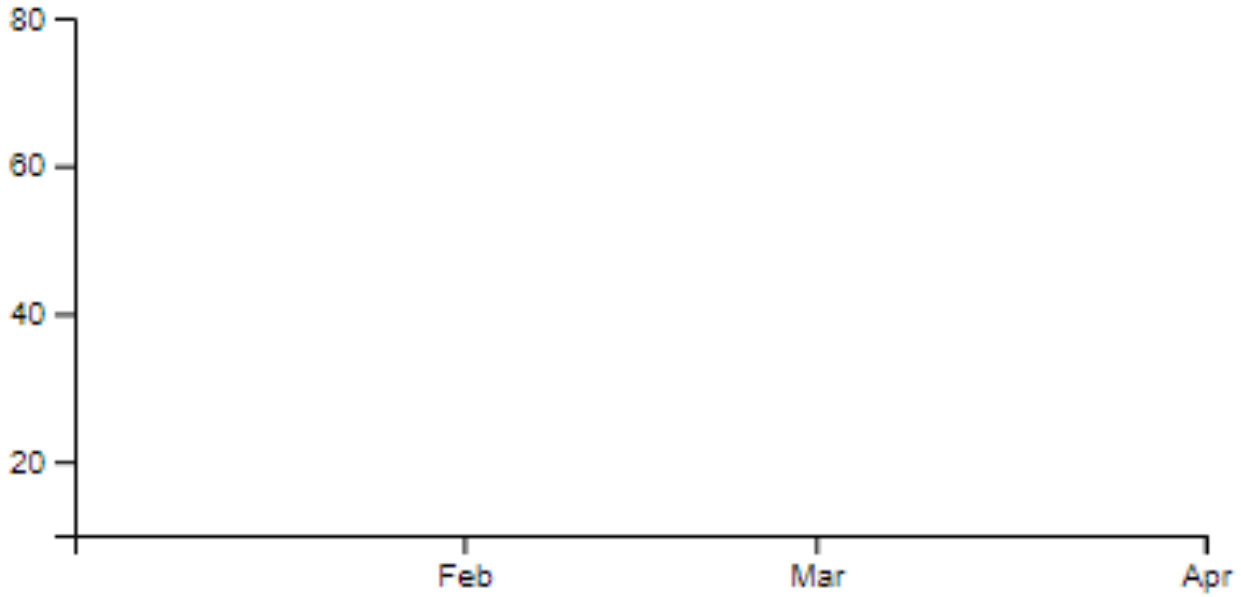
`d3.axisRight()` → creates a vertical right-oriented axis

`d3.axisBottom()` → creates a bottom horizontal axis

`d3.axisLeft()` → creates a left vertical axis

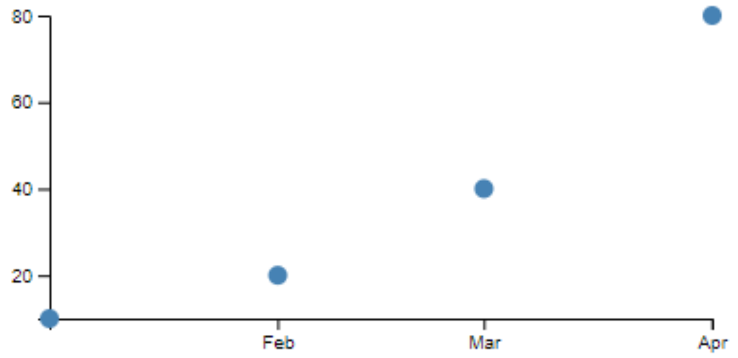
Defining the axes of our chart

```
var xAxis = d3.axisBottom(xScale)
    .ticks(4)
    .tickFormat(d3.timeFormat("%b"))

var yAxis = d3.axisLeft(yScale)
  .ticks(4);

chartGroup.append("g")
  .attr('transform', "translate(0, 0)")
  .classed('y-axis', true)
  .call(yAxis)

chartGroup.append("g")
  .attr('transform', "translate(0," + height + ")")
  .classed('x-axis', true)
  .call(xAxis)
```



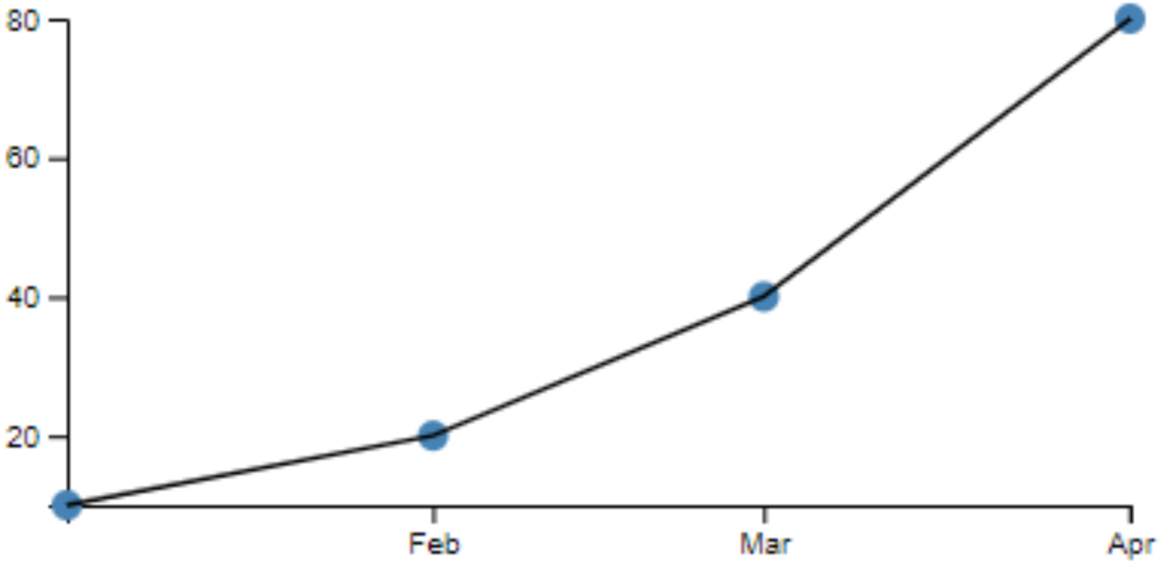
Binding the data and drawing the circles

```
chartGroup.selectAll("circle")   // select all circles in the page

          .data(data)            // bind the 4 data values

          .enter()               // create 4 placeholders

          .append("circle")      // create 4 circles

           // use the xScale to place the circle over the x axis
          .attr("cx", d => xScale(new Date(d.date)))

           // use the yScale to place the circle over the y axis
          .attr("cy", d => yScale(d.amount))

          .attr("r", 5) // set the radius of the circle

          .style("fill", "steelblue") // set a fill color
```



Adding the line across the dots

```
let lineGenerator = d3.line()
  .x(d => xScale(new Date(d.date)))
  .y(d => yScale(d.amount))

// draw the line
chartGroup.append("path")
    .datum(data)
    .attr("class", "line")
    .attr("fill", "none")
    .attr("stroke", "steelblue")
    .attr("stroke-width", 1.5)
    .attr("d", lineGenerator)
```



```
<path class="line" fill="none" stroke="steelblue" stroke-width="1.5"
d="M0,160L120.55555555555556,137.14285714285717L229.44444444444446,91.42857142857142L350,0"/>
```

Data update → the `join` function

1. More data values than elements → draw new elements
2. Less data values than elements → remove the extra elements
3. Update the attributes and style of elements to match new data values

```
chartGroup.selectAll("circle")
     .data(data)
     .enter()
     .append("circle")
     .attr("cx", d => xScale(new Date(d.date)))
     .attr("cy", d => yScale(d.amount))
     .attr("r", 5)
     .style("fill", "steelblue")
```

```
chartGroup.selectAll("circle")
  .data(data)
  .join(
         enter => enter.append(“circle”)
              .attr("r", 5)
              .style("fill", "steelblue"),
         update => update
                   .attr("cx", d => xScale(new Date(d.date)))
                   .attr("cy", d => yScale(d.amount)),
         exit => exit.remove()
  )
```

Animating our chart

```
chartGroup.selectAll("circle")
    .data(data)
    .join(
       enter => enter.append("circle")
           .attr("r", 5).style("fill", "steelblue"),
       update => update,
       exit => exit.remove()
 )
.transition()
.duration(500)
.attr("cx", d => xScale(new Date(d.date)))
.attr("cy", d => yScale(d.amount))

chartGroup.selectAll("path.line")
    .transition()
    .duration(500)
    .attr("d", lineGenerator(data))
```

Tooltips

```
<html>   <head> <title>TITLE GOES HERE</title> </head>
<style>
     .tooltip{
          position: absolute;
          z-index: 1000; /* to be in front of all other elements */
          display: none; /* initially invisible */
          background-color: #fff;
          box-shadow: 10px 5px 5px #ccc;
          border-radius: 5px; }
</style>
<body>
     <div class="tooltip"> </div> // <div> tag to represent the tooltip
     <script>
          chartGroup.selectAll("circle")
               .on("mouseover", function(d){
                       let x = d3.event.pageX, y = d3.event.pageY; // get the mouse coordinates

                       d3.select("div.tooltip“).style("display", "block")
                             .style("left", x + "px").style("top", y + "px") // place the tooltip at to the mouse position
                             .html("Date: " + d.date + "<br> Amount: " + d.amount)
               })
               .on("mouseout", function(d) { d3.select("div.tooltip").style("display", "none") })
     </script> </body> </html>
```

Provide the user with more information than what is being visually represented, e.g. the data values

Mouseover and mouseout events to activate tooltips

Maps in D3: projections

The concepts of latitude and logitude are unknown by the browser

Projections transform latitude and logitude coordinates to x and y coordinates on a flat surface (the screen)

- D3 projection methods : https://github.com/d3/d3-geo-projection

```
let projection = d3.geoMercator()
       .scale(200)
       .translate([width / 2, height / 2])
```

GeoJSON

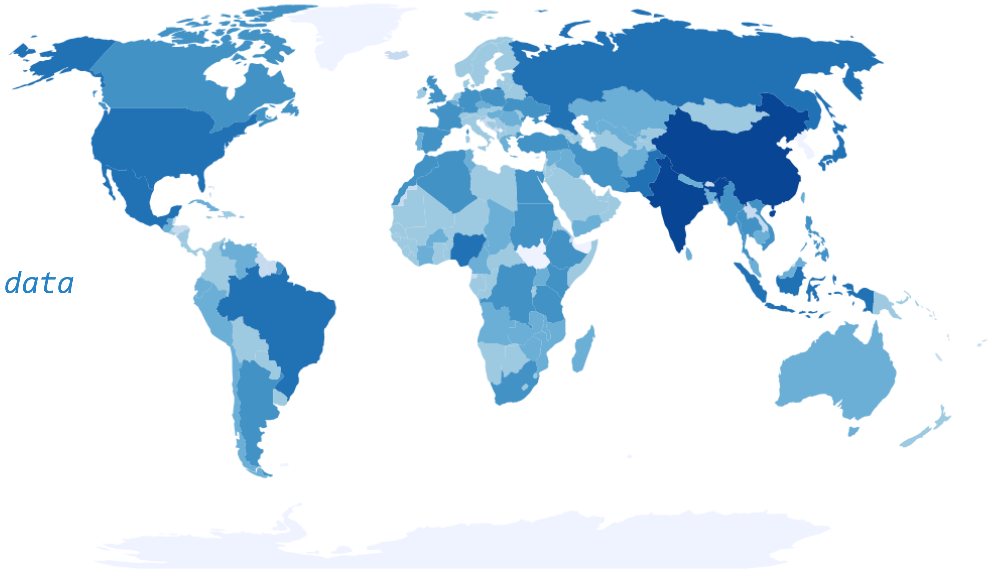
GeoJSON is a format for encoding geographic data structures

We can load `.geojson` data files using the `d3.json()` function

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

Example of map in D3

```
// Load external data
Promise.all([d3.json("https://raw.githubusercontent.com/holtzy/D3-graph-gallery/master/DATA/world.geojson"),
d3.csv("https://raw.githubusercontent.com/holtzy/D3-graph-gallery/master/DATA/world_population.csv")]).then(files => {
      let topo = files[0], data = files[1]

      let colorScale = d3.scaleThreshold()
            .domain([100000, 1000000, 10000000, 30000000, 100000000, 500000000])
            .range(d3.schemeBlues[7]);

      // Draw the map
      svg.append("g")
            .selectAll("path")
            .data(topo.features)
            .enter()
            .append("path") // draw each country
            .attr("d", d3.geoPath().projection(projection))
            // set the color of each country according to the value in the csv data
            .attr("fill", d => colorScale(getValue(d.id)))

      function getValue(countryId) {
            let item = data.find(d => d.code === countryId)
            return item ? item.pop : 0
      }
})
```



Ressources

D3:

- Official website: http://d3js.org/
- D3 gallery (examples with code source): https://www.d3-graph-gallery.com/
- A nice tutorial: https://www.tutorialsteacher.com/d3js

Web standards:

- HTML : https://www.w3schools.com/html/default.asp
- CSS: https://www.w3schools.com/css/default.asp
- JavaScript: https://www.w3schools.com/js/default.asp