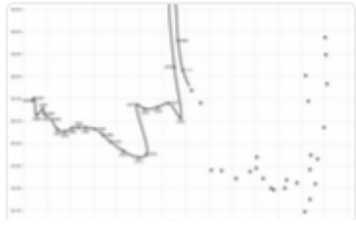
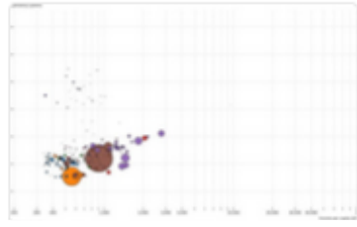

Animated treemap

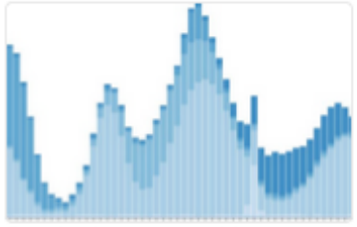

Te





Bar chart race

Data Visualization with D3.js


Stacked-to-grouped bars


Streamgraph transitions

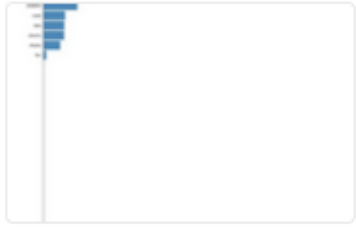
Smooth zooming

Zoom to bounding box

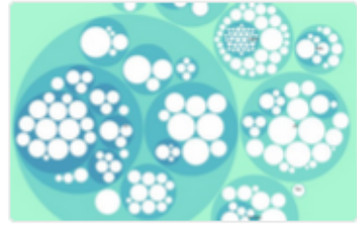
Orthographic to equirectang...

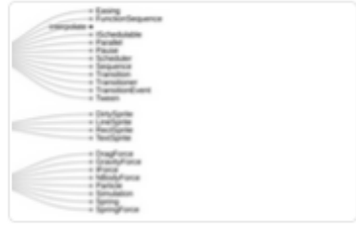

World tour


Walmart's growth


Hierarchical bar chart

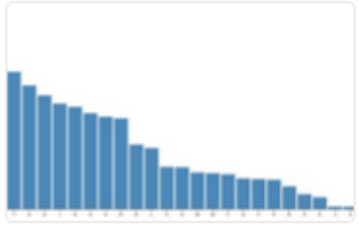

Zoomable treemap

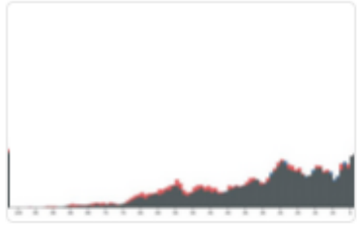

Zoomable circle packing


Collapsible tree


Zoomable icicle


Zoomable sunburst


Sortable bar chart


Icelandic population by age, ...

https://d3js.org/

Aline Menin

aline.menin@inria.fr

UNIVERSITÉ CÔTE D'AZUR

What is d3.js?

- It is a JavaScript library for manipulating documents based on data.
- D3 is built on top of common web standards like HTML, CSS, and SVG.
- Based on web standards to comply with capabilities of modern browsers without tying yourself to a proprietary framework;
- It combines powerful visualization components and a data-driven approach to DOM manipulation.

Overview of Web Standards

- HTML (HyperText Markup Language)
- CSS (Cascading Stylesheets)
- DOM (Document Object Model)
- SVG (Scalable Vector Graphics)

HTML (HyperText Markup Language)

- HTML use a set of tags to define the different structural components of a webpage:
  - **<h1>**, **<h2>, …, <h6>** tags define headers
  - **<p>** tags define paragraphs
  - **<ol>** and **<ul>** are ordered and unordered lists
- Browsers have common ways to display these tags:
  - lists show up like lists, and headers like headers.
- The ** ** and **** tags are special:
  - browsers do not apply default styles to them, so that they can be used to define custom groups.

index.html

```
<!DOCTYPE html>
<html>
        <head>
                <title>TITLE GOES
HERE</title>
         </head>
         <body>
                <p> MAIN CONTENT GOES
HERE</p>
         </body>
</html>
```

CSS (Cascading Stylesheets)

- CSS is a language for styling HTML pages.
- CSS styles (also known as selectors) are typically applied to HTML tags based on their **name**, **class**, or **identifier**.

CSS

```
/* Applied to all <p> tags */
p {
	color:blue;
}

/* Applied to all tags with the class “red” */
.red {
	background: red;
}

/* Applied to the tag with the id “some-id” */
#some-id {
	font-style: italic;
}

/* Applied only to <p> tags inside <li> tags */
li p {
	color: #0C0;
}
```

HTML

```
<!DOCTYPE html>
<html>
	<head>
		<title>TITLE GOES HERE</title>
		</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>
		</body>
```

Result (HTML + CSS)

Normal paragraph

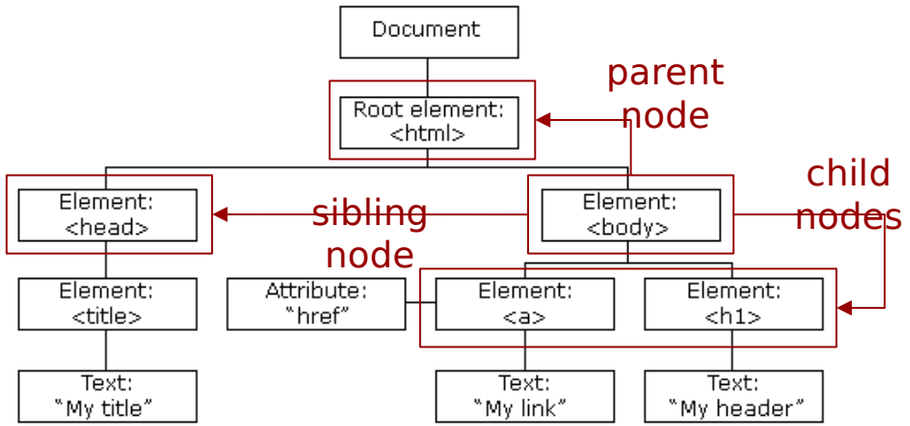
Red paragraph

1. *Unique element*
2. Another list element
3. Paragraph inside list element

Second paragraph

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
- In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**



JavaScript HTML DOM

- JavaScript is the programming language of HTML and the Web.
- DOM and JavaScript allow the creation of dynamic HTML:
  - changing/adding/removing HTML elements and its attributes;
  - changing the CSS style of elements and attributes;
  - react to existing HTML events in the page;
  - creating new events.

```
<!DOCTYPE html>
<html>
      <head>
            <title>TITLE GOES HERE</title>
                  </head>
            <body>

                  <h1>My First JavaScript</h1>

                  <button type="button" onclick="document.getElementById('demo').innerHTML
            = Date()">
                  Click me to display Date and Time.
                  </button>

                  <p id="demo"></p>

            </body>
            </html>
```

DOM + JavaScript D3 API

The standard DOM API is somewhat verbose, therefore libraries such as D3 provide syntactic sugar to ease the manipulation of HTML elements, styles and attributes.

HTML

```
<!DOCTYPE html>
<html>
	<head>
		<script type="text/javascript"
	src="https://d3js.org/d3.v5.min.js"></script>
		<title>TITLE GOES HERE</title>
		</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>
		</body>
		</html>
```

DOM API (JavaScript)

```
document.getElementById("some-id")
// <li id=”some-id”>Unique element</li>

document.getElementsByTagName("p").length;
// 4

var reds = document.getElementsByClassName("red")
// [<p class=”red”>Red Paragraph</p>]

reds[0].innerText
// “Red Paragraph”
```

D3.js

```
d3.select("#some-id")
// [Array(1)]

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

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

reds.text()
// “Red Paragraph”
```

DOM + JavaScript D3 API

- The DOM also handles tracking elements as they are rendered, e.g. mouse movement.
  - Listeners may be attached to these events to add various levels of interactivity to the web page, e.g. mouseover, mouseleave
- D3 has some nice helpers for working with events as well.
  - **Note:** We can chain d3 methods, usually starting with a selection (i.e. they return themselves, so we can group them visually).

HTML

```
<!DOCTYPE html>
<html>
	<head>
		<title>TITLE GOES HERE</title>
		</head>
		<body>
			<h1 id=”click-me”>
				Click on me!
			</h1>
			<p class=”hover-me”>
				Hover over me!
			</p>
			<p class=”hover-me”>
				OK now hover over here!
			</p>
			<p class=”hover-me”>
				Hover here too!
			</p>

		</body>
		</html>
```

DOM API (JavaScript)

```
var clickMe = document.getElementById(“click-me”);

clickMe.onclick = function() {
	if (this.style.backgroundColor) {
		this.style.backgroundColor = “ ”;
	} else {
		this.style.backgroundColor = “red”;
	}
}
```

D3.js

```
d3.selectAll(".hover-me")
	.on(“mouseover”, function() {
		this.style.backgroundColor = “yellow”;
	})
	.on(“mouseout”, function() {
		this.style.backgroundColor = “”;
	})
```

SVG (Scalable Vector Graphics)

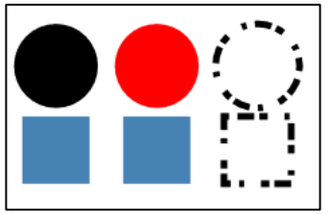
- SVG is a XML format used for drawing.
- Similar to DOM, SVG has elements with parents, children and attributes.
  - The elements also respond to the same mouse/touch events.
- SVG defines tags for lots of basic shapes:
  - <rect> for rectangles
  - <circle> for circles
  - <line> for straight lines
- Some CSS syntax used for DOM are different for SVG elements.
  - e.g., background-color: red; → fill: red;

HTML

```
<!DOCTYPE html>
<html>
<head>
<title>TITLE GOES HERE</title>
</head>
<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>
```

CSS

```
.red {
	fill: red; /* not background-color */
}

.fancy {
	fill: none;
	stroke: black;
	stroke-width: 3pt;
	stroke-dasharray: 3,5,10;
}
```



SVG - Groups

- Grouping elements:
  - DOM: ** ** and **** tags
  - SVG: **<g>** tag
- The tag **<g>** is widely used to create graphs in d3
  - to apply styles or reposition all the elements of a group at once.

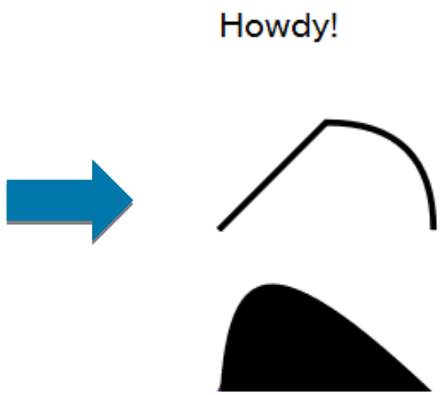
```
<svg viewBox="0 0 100 100" >
     <!-- Using g to inherit presentation attributes -->
     <g fill="white" stroke="green" stroke-width="5">
          <circle cx="40" cy="40" r="25" />
          <circle cx="60" cy="60" r="25" />
     </g>
</svg>
```

SVG - Paths

The **<path>** tag

- Powerful (and complex)
- It can be used for drawing lines or arbitrary shapes
- Largely used with d3 to create graphs (e.g., pie charts, line charts, etc)
  - d3 provides methods to automatically create the paths according to the data

```
<svg width="300" height="100" >
<g transform="translate(5, 15)">
      <text x="0" y="0">Howdy!</text>
      </g>
      <g transform="translate(5,55)">
       <!-- M: move to (jump)
            L: line to
            Q: quadratic curve to -->
       <path d="M0,50 L50,0 Q100,0 100,50" fill="none" stroke-
width:"3" stroke="black" />
      </g>
      <g transform="translate(5, 105)">
       <!-- C: curve to (cubic)
            Z: close shape -->
       <path d="M0,100 C0,0 25,0 125,100 z" fill="black" />
      </g>
      </svg>
```



Note: The **transform** attribute allows to move, rotate, scale, etc., elements.

How to create a chart?

- Chart Elements
  - The Data
  - The Scales
  - The Axes
- Using SVG
- Using D3
  - Helpers
  - Defining the Scales
  - Defining the Axes
  - Binding the Data

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

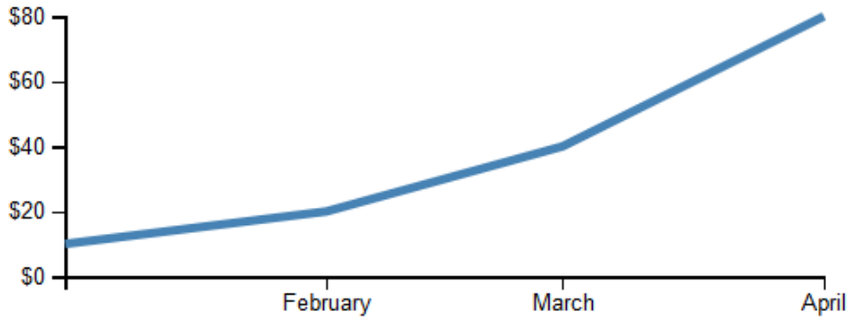
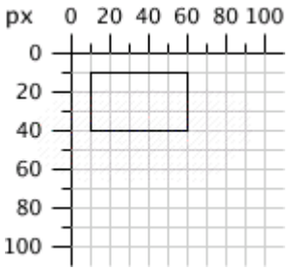


Chart Elements

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

- The Scale: refers to the coordinate system
  - x-axis: from January 2014 to April 2014
  - y-axis: from $10 to $80
  - SVG dimensions: 350 by 160 pixels
    - Specify the mapping between data (i.e. values of variables) and pixels of the screen
  - **Note:** y-axis flips as the SVG origin (i.e. the coordinates 0, 0) is in the top left
- The Axes: refer to the labels
  - Labels such as “$20” and “February” have to get to our screen somehow.
  - They also need to be formatted correctly according to the data type.
- The Data
  - Each row of our dataset will become a point over the line.
  - The points in the line must fit into the defined coordinate system.

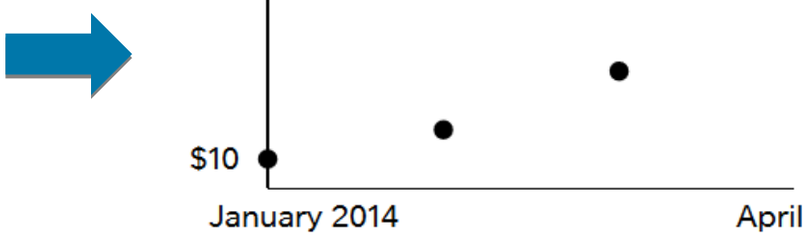


Default coordinate system

Using SVG

- Points and lines are drawn manually using SVG tags
  - The **<path>** tag is complex to be used by hand

```
<svg width="350" height="160">
  <!-- 60px x 10px margin -->
  <g class="layer" transform="translate(60,10)">
    <!-- cx = 270px * ($X / 3)
                    ^      ^   ^
    width of graph  x-value max(x)

          cy = 120px - (($Y / 80) * 120px)
                  ^       ^     ^       ^
      top of graph   y-value  max(y)  scale -->
    <circle r="5" cx="0"   cy="105" />
    <circle r="5" cx="90"  cy="90"  />
    <circle r="5" cx="180" cy="60"  />
    <circle r="5" cx="270" cy="0"   />

    <g class="y axis">
      <line x1="0" y1="0" x2="0" y2="120" />
      <text x="-40" y="105" dy="5">$10</text>
      <text x="-40" y="0"   dy="5">$80</text>
    </g>
    <g class="x axis" transform="translate(0, 120)">
      <line x1="0" y1="0" x2="270" y2="0" />
      <text x="-30"   y="20">January 2014</text>
      <text x="240" y="20">April</text>
    </g>
  </g>
</svg>
```



Using D3

```
<!DOCTYPE html>
<html>
<head>
<title>TITLE GOES
HERE</title>
</head>
<body>
        <svg></svg>
        </body>
        </html>
```

1. Define the dataset. The data is always represented as plain Javascript array objects.

```
var 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 }
            ]
```

2. Define the dimensions of the chart

```
var 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
```

3. Set the SVG dimensions

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

4. Create the chart group using the **<g>** tag

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

Using D3 – Helpers (data)

- D3 can handle different types of data defined either locally in variables or from external files
- D3 provides the following methods to load different types of data from external files

Method	Description
d3.csv()	Sends http request to the specified url to load .csv file or data and executes callback function with parsed csv data objects.
d3.json()	Sends http request to the specified url to load .json file or data and executes callback function with parsed json data objects.
d3.tsv()	Sends http request to the specified url to load a .tsv file or data and executes callback function with parsed tsv data objects.
d3.xml()	Sends http request to the specified url to load an .xml file or data and executes callback function with parsed xml data objects.

Using D3 – Helpers (data)

- D3 handle loading a single or multiple files using promises

Loading a single data file

```
d3.json("filepath.json").then(data => {

     // do something with data

}
```

Loading multiple data 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

Using D3 – Helpers (**selections**)

- D3 selections are a group of elements that match a query or could match a query later (the elements may not have been constructed yet).
- How to:
  - d3.select(<key>) will find one element
    - typically using the element's **unique** identifier
  - d3.selectAll(<key>) will match all available elements
    - typically used to select elements with a same class or name

Using D3 – Helpers (min, max, extent)

- Automate operations such as finding the **min**imum and **max**imum values of a dataset (or both at the same time, the "**extent**").
- The data is always represented as Javascript array objects

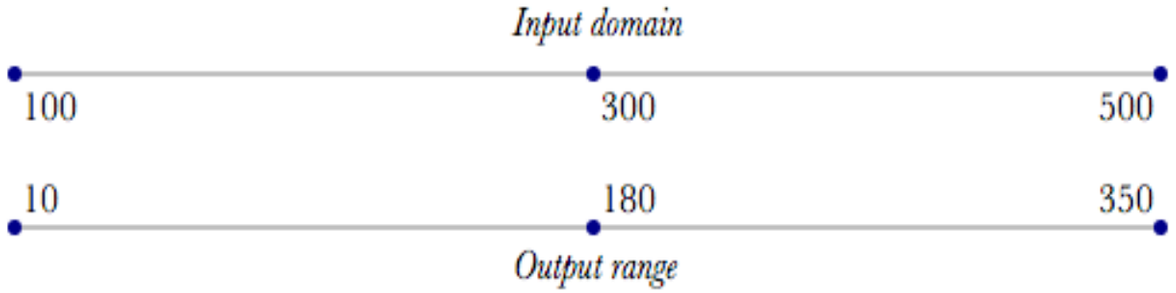
```
var values = [10, 20, 40, 80],
      data = [
            { date: "2014-01-01", amount:10 },
            { date: "2014-02-01", amount:20 },
            { date: "2014-03-01", amount:40 },
            { date: "2014-04-01", amount:80 }
            ]

            d3.min(values)
            // 10
```

- When working with JSON objects, we use callback functions to recover the values we want to use in the helper method.
  - The callback function has normally two arguments: the element and its index.
  - These arguments are commonly named ***d*** and ***i***, respectively.

```
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]
```

Using D3 – Defining the Scales

- d3-scale module (https://observablehq.com/@d3/introduction-to-d3s-scales) : map values across coordinate systems
- Types of scales: *linear, logarithmic, time*
- Scales are configured with a domain and a range:
  - they map values from the data (the domain) to the ap




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

- Defining the domain with helpers:

```
yScale.domain(d3.extent(data, d => d.amount)
```

- The object **yScale** is a function
  - We use it to translate values from one coordinate to another (i.e. between domain and range values):

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

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

Using D3 – Defining Scales (for time)

- The same things with dates!

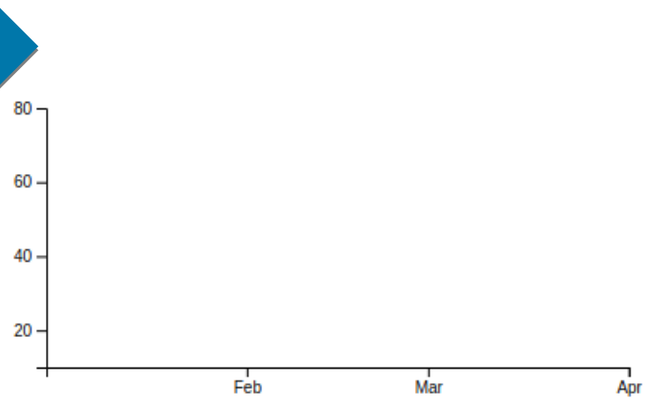
```
var xScale = d3.scaleTime()
     .domain([
          new Date("2014-01-01"),
          new Date("2014-04-01")
     ])
     .range([0, width])

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

- Scales are not just for linear transforms (continuous or quantitative scales), but also for arbitrary transforms (discrete or ordinal scales)
  - e.g., mapping between data and colors.

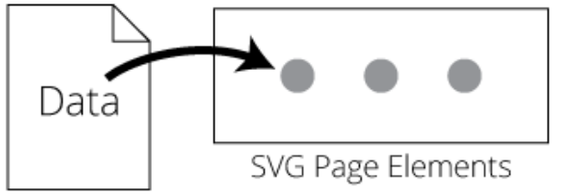
Using D3 – Defining the Axes

- We want labels and thick marks:
  - d3 can handle this automatically.
- When building an axis object, we give to it the scale we want to use as argument to the function.

```
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)
```



Using D3 – Data Bin



Data binding might require the following operations:

- Selections:
  - var selection = d3.selectAll(<key>)
- Joining the data:
  - selection.data(<data-object>): for binding data to multiple elements
  - selection.datum(<data-object>): for binding data to a single element
- Creating elements
  - selection.enter()
  - selection.append(<element-name>): when data won’t change
  - selection.join() (v5+): when data update is required (i.e. filters)
- Transitions
  - selection.transition()
  - selection.duration(<time-milliseconds>)
- Tooltips

Using D3 – Data Binding (data)

- We use selections to map pieces of our data to elements in the DOM.
- We want to display the data as a line chart. We want each object to be a circle and to connect them with lines. Thus, (1) each object would become a **<circle>** tag, inside of our **<g>** tag:

```
<!-- before -->
<svg>
<g>
      <g class="x-axis"></g>
      <g class="y-axis"></g>
</g>
</svg>
```

```
<!-- after -->
<svg>
<g>
      <g class="x-axis"></g>
      <g class="y-axis"></g>
      <circle /> <!-- {date: "2014-01-01", amount: 10} -->
      <circle /> <!-- {date: "2014-02-01", amount: 20} -->
      <circle /> <!-- {date: "2014-03-01", amount: 40} -->
      <circle /> <!-- {date: "2014-04-01", amount: 80} -->
      </g>
      </svg>
```

- We select the <g> element and bind our data using the *data(<data-obj>)* function:

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

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

```
var circles = chartGroup.selectAll("circle")
      .data(data);

circles.size() // 0 -- not <circle> tag exists
yet!
```

Now we have a selection but still no elements! We have more work to do…

Using D3 – Data Binding (enter)

- Goal: one circle per data record.
- We add a new **<circle>** tag for each data point.
- Since this is the first data binding (there are no circles in the page yet), the process is straightforward.
  - **Note:** For the next selection, we must handle the fact that there will already be circles drawn on the page.
- We use selection.enter(), to indicate that we want to add new elements to the page.

```
var newCircles =
circles.enter();
```

Using D3 – Data Binding (append / join)

- Then, we use selection.append(<key>) or selection.join(<key>) to add these new elements.

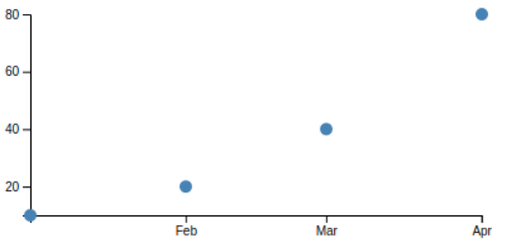
```
newCircles.join("circle")
  .attr("cx", d => xScale(new Date(d.date)))
  .attr("cy", d => yScale(d.amount))
  .attr("r", 5)
  .style("fill", "steelblue")
```

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

```
▼<svg width="390" height="190"> == $0
  ▼<g transform="translate(20,10)">
    ▶<g transform="translate(0, 0)" class="y-axis" fill="none" font-size="10" font-family="sans-serif" text-anchor="end">…</g>
    ▶<g transform="translate(0,160)" class="x-axis" fill="none" font-size="10" font-family="sans-serif" text-anchor="middle">…</g>
      <circle cx="0" cy="160" r="5" style="fill: steelblue;"></circle>
      <circle cx="124" cy="137.14285714285714" r="5" style="fill: steelblue;"></circle>
      <circle cx="236" cy="91.42857142857143" r="5" style="fill: steelblue;"></circle>
      <circle cx="360" cy="0" r="5" style="fill: steelblue;"></circle>
  </g>
</svg>
```

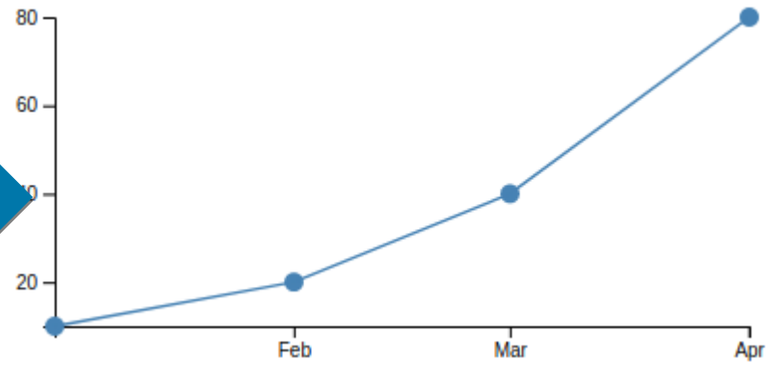


Using D3 – Data Binding (datum)

- We create a line through a path created with all the points in the data.

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

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

```
▼<svg width="390" height="190">
  ▼<g transform="translate(20,10)">
    ▶<g transform="translate(0, 0)" class="y-axis" fill="none" font-
    size="10" font-family="sans-serif" text-anchor="end">…</g>
    ▶<g transform="translate(0,160)" class="x-axis" fill="none" font-
    size="10" font-family="sans-serif" text-anchor="middle">…</g>
      <circle cx="0" cy="160" r="5" style="fill: steelblue;"></circle>
      <circle cx="124" cy="137.14285714285714" r="5" style="fill:
      steelblue;"></circle>
      <circle cx="236" cy="91.42857142857143" r="5" style="fill:
      steelblue;"></circle>
      <circle cx="360" cy="0" r="5" style="fill: steelblue;"></circle>
      <path fill="none" stroke="steelblue" stroke-width="1.5" d=
      "M0,160L124,137.14285714285714L236,91.42857142857143L360,0">
      </path> == $0
    </g>
  </svg>
```



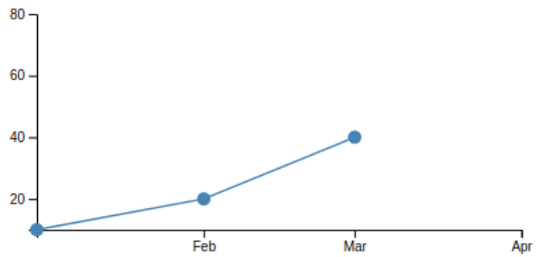
Using D3 – Data Binding (data update)

1. If we have more data, we add new elements
2. If we have less data, we remove the elements that do not correspond to the new data we want to join in our chart
3. We update the elements attributes and style according to the new data

```
var 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 }
            ]
```

```
data.pop()

// [{ date: “2014-01-01”, amount:10 },
//  { date: “2014-02-01”, amount:20 },
//  { date: “2014-03-01”, amount:40 }]
```

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

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



Using D3 – Data Binding (Transitions)

- The operation selection.transition() allows temporal transitions to make transitions nicer.
- We can use attributes such as .duration(), .delay() and .ease(). We typically start by defining the duration to our transition:

```
chartGroup.selectAll("circle")
     .data(data)
     .enter()
     .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))
```

Using D3 – Data Binding (Tooltips)

- We must often provide the user with more information than what is being visually represented, such as the actual values of the represented variables
- We use then tooltips activated with mouseover and mouseout listeners that we add to our elements:

HTML

```
<div class="tooltip"> </div>
```

CSS

```
.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;
}
```

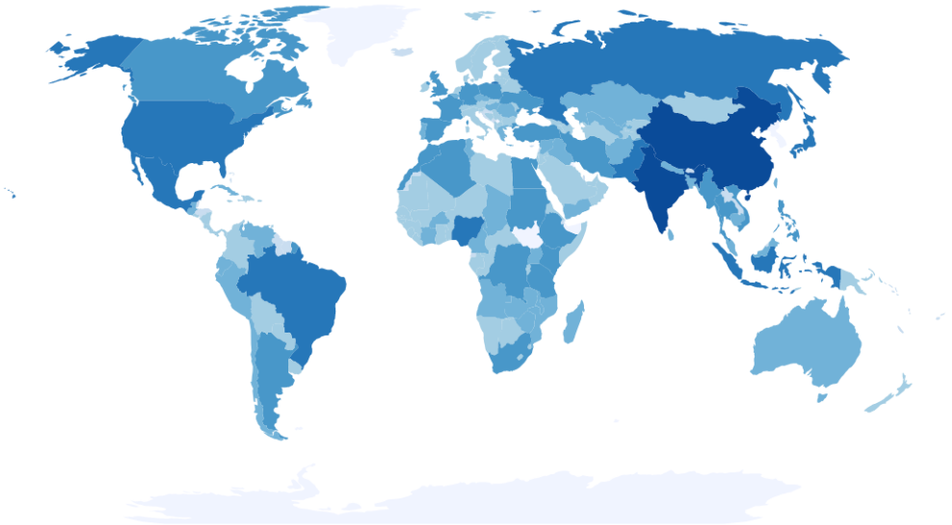
D3.js

```
chartGroup.selectAll("circle")
      .on("mouseover", function(d){
              var x = d3.event.pageX,
                y = d3.event.pageY;

              d3.select("div.tooltip")
                  .style("display", "block")
                  .style("left", x + "px")
                  .style("top", y + "px")
                  .html("Date: " + d.date + "<br>" + "Amount: " + d.amount)
      })
      .on("mouseout", function(d){
            d3.select("div.tooltip").style("display", "none")
            })
```

Working example: https://jsfiddle.net/amenin/yruefj3c/188/

Using Maps with d3

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

           // Load external data and boot
           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]

               var 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
                   .attr("fill", d => colorScale(getValue(d.id)))


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



References

- HTML + CSS + JavaScript:
  - https://www.w3schools.com/html/default.asp
  - https://www.w3schools.com/css/default.asp
  - https://www.w3schools.com/js/default.asp
- D3:
  - http://d3js.org/
  - https://www.d3-graph-gallery.com/

Note: There are big breaks in the API between versions v3 and v4+