Interactive Web Programming

1st semester of 2021

Murilo Camargos
(**murilo.filho@fgv.br**)

Heavily based on **Victoria Kirst** slides

Schedule

Today:

- Making a bar chart with D3
  - Bind CSV files to SVG shapes
  - Introduction to axes and scales

HW5 is out! Due Jun 03.

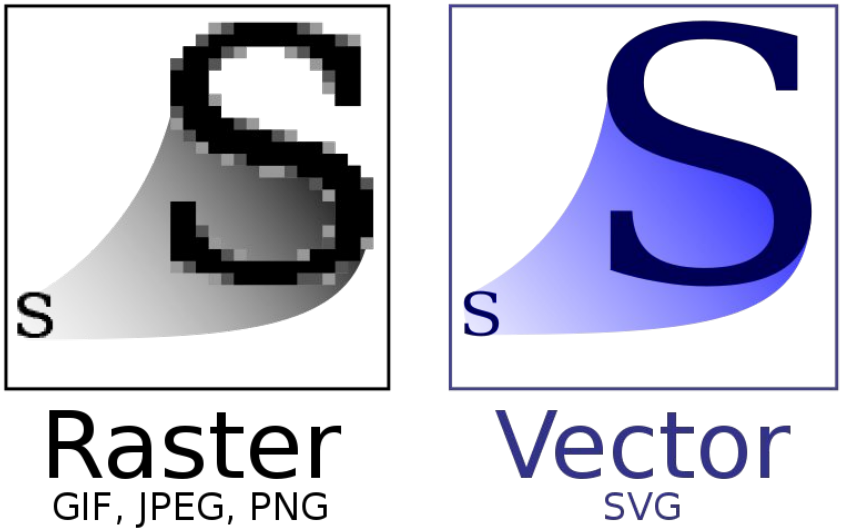
Credits:

- https://www.youtube.com/watch?v=_8V5o2UHG0E
- Intro to Data Viz at Ohio State University
  - http://web.cse.ohio-state.edu/~shen.94/5544/

SVG

SVG as an image format

SVG: Stands for **S**calable **V**ector **G**raphics



https://commons.wikimedia.org/wiki/File:Ghostscript_Tiger.svg

SVG with HTML

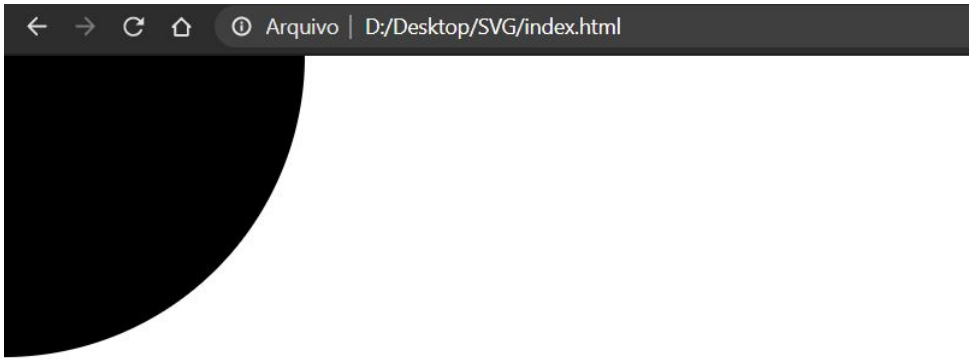
You can use the **svg** HTML tag to create an SVG canvas:

```
<body>
  <svg></svg>
</body>
```

You can use different shape SVG elements inside this tag:

- <circle>, <ellipse>, <line>, <polygon>, <polyline>, <rect>

```
<svg>
  <circle r="100"></circle>
</svg>
```



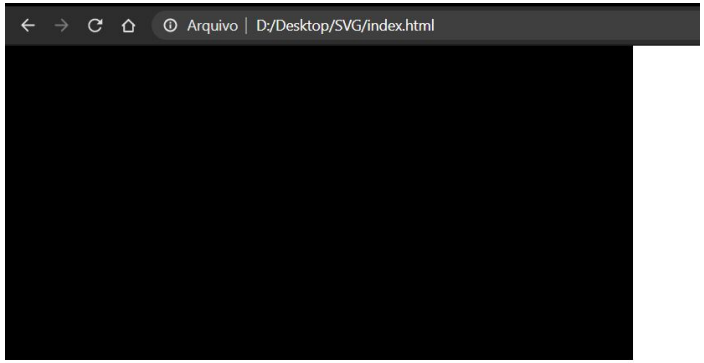
Codepen

SVG with HTML: canvas size

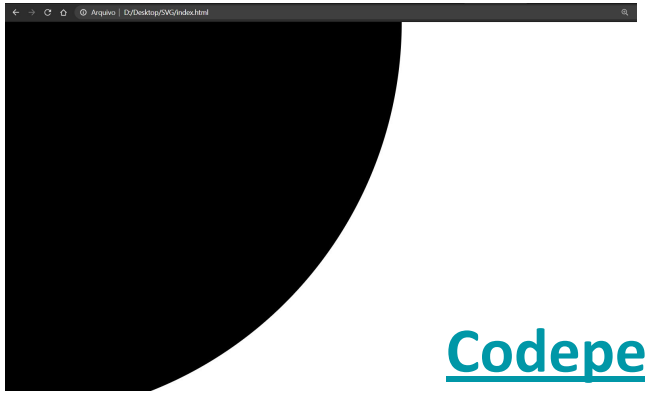
SVG's standard canvas size is **300×150**:

- Other browsers might implement 100vw and 100vh.

```
<svg>
  <circle r="400"></circle>
</svg>
```



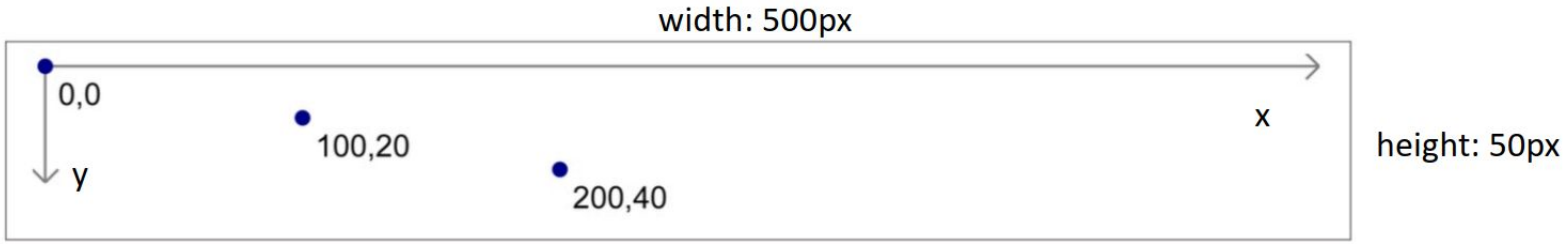

```
<svg width="400" height="400">
  <circle r="400"></circle>
</svg>
```



[Codepen]

SVG with HTML: coord. system

SVG's coordinate system:



HTML

```
<svg width="50" height="50">
  <circle cx="50" cy="50" r="50"></circle>
</svg>
```

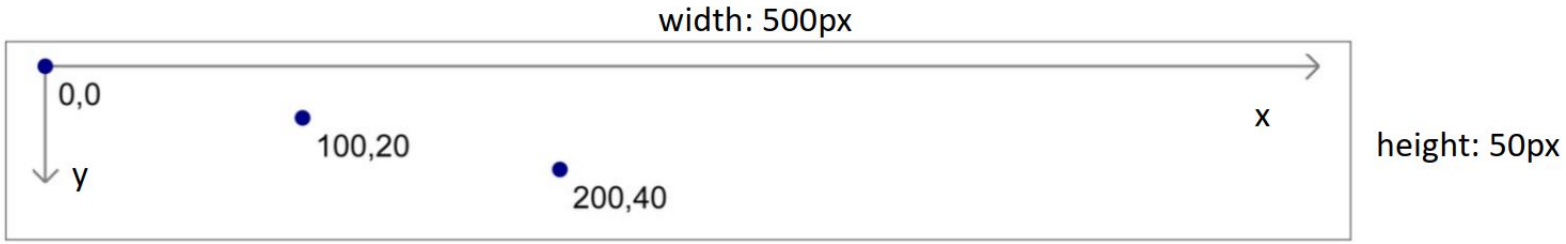
CSS

```
svg {
  background-color: red;
}
```

[Codepen]

SVG with HTML: coord. system

SVG's coordinate system:

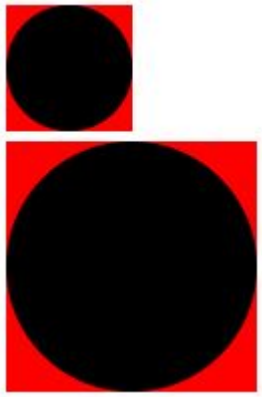


viewBox to define a virtual box that can be resized

- Needs fours numbers: min-x, min-y, width and height

HTML

```
5  <svg width="50" width="50" viewBox="0 0 100 100">
6    <circle cx="50" cy="50" r="50"></circle>
7  </svg>
8  <br>
9  <svg width="100" width="100" viewBox="0 0 100 100">
10   <circle cx="50" cy="50" r="50"></circle>
11 </svg>
```



[Codepen]

SVG with HTML and CSS

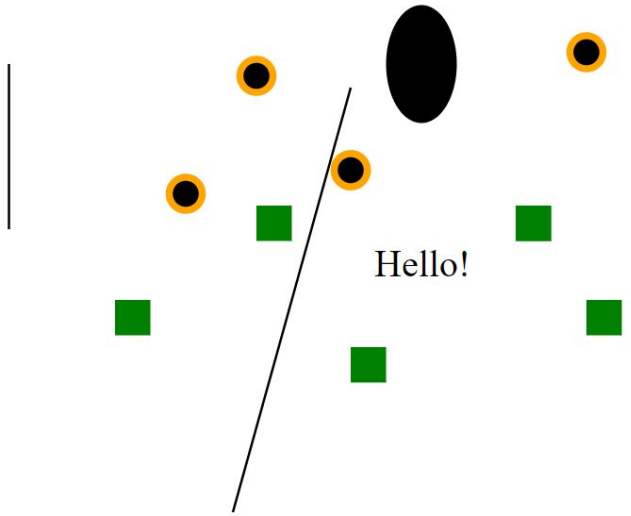
You can style the shapes with CSS code:

HTML

```
<svg width="400" height="220">
  <line x1="5" x2="5" y1="100" y2="30" stroke="black"></line>
  <line x1="100" x2="150" y1="220" y2="40" stroke="black"></line>

  <rect x="150" y="150" width="15" height="15"></rect>
  <rect x="220" y="90" width="15" height="15"></rect>
  <rect x="110" y="90" width="15" height="15"></rect>
  <rect x="220" y="90" width="15" height="15"></rect>
  <rect x="50" y="130" width="15" height="15"></rect>
  <rect x="250" y="130" width="15" height="15"></rect>

  <circle cx="250" cy="25" r="7"></circle>
  <circle cx="150" cy="75" r="7"></circle>
  <circle cx="80" cy="85" r="7"></circle>
  <circle cx="110" cy="35" r="7"></circle>
  <circle cx="150" cy="75" r="7"></circle>

  <ellipse cx="180" cy="30" rx="15" ry="25"></ellipse>

  <text x="160" y="120">Hello!</text>
</svg>
```

CSS

```
rect {
  fill: green;
}
circle {
  stroke: orange;
  stroke-width: 3px;
}
```



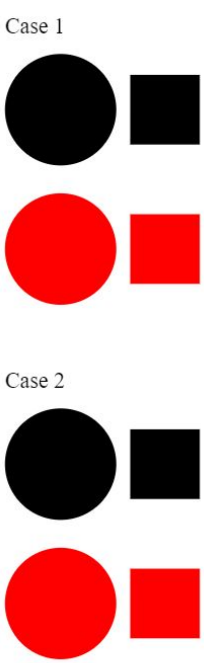
[Codepen]

SVG groups

You can group the shapes and apply rules to all of them at once:

HTML

```
<svg widht="300" height="250">
  <text x="10" y="15">Case 1</text>

  <circle cx="50" cy="70" r="40"></circle>
  <rect x="100" y="45" width="50" height="50"></rect>

  <circle cx="50" cy="170" r="40" fill="red"></circle>
  <rect x="100" y="145" width="50" height="50" fill="red"></rect>
</svg>
<br>
<svg widht="300" height="250">
  <text x="10" y="15">Case 2</text>

  <circle cx="50" cy="70" r="40"></circle>
  <rect x="100" y="45" width="50" height="50"></rect>

  <g fill="red" transform="translate(0,100)">
    <circle cx="50" cy="70" r="40"></circle>
    <rect x="100" y="45" width="50" height="50"></rect>
  </g>
</svg>
```



[Codepen]

SVG Paths

You can also create a **path**:

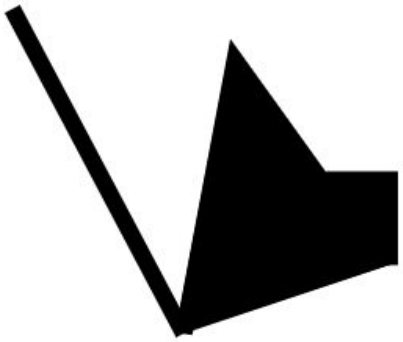
- M x y – Move to (x,y)
  - m dx dy – Move by (dx,dy)
- L x y – Line to (x,y)
  - l dx dy
- H x, V y – draw horizontal and vertical lines
  - h dx, v dy
- Z, z close path
- Curve commands (Bezier Curves and Arcs)

SVG Paths

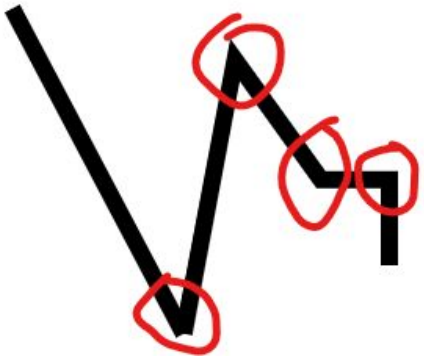
HTML

```
1  <svg widht="200" height="200">
2    <text x="10" y="15">Case 1</text>
3    <g stroke="black" stroke-width="5">
4      <line x1="10" y1="25" x2="60" y2="120"></line>
5      <path d="M60 120 L75 40 L100 75 H120 V100"></path>
6    </g>
7  </svg>
8  <br>
9  <svg widht="200" height="200">
10   <text x="10" y="15">Case 2</text>
11   <g stroke="black" stroke-width="5">
12     <line x1="10" y1="25" x2="60" y2="120"></line>
13     <path d="M60 120 L75 40 L100 75 H120 V100" fill="none"></path>
14   </g>
15 </svg>
```

Case 1



Case 2



[Codepen]

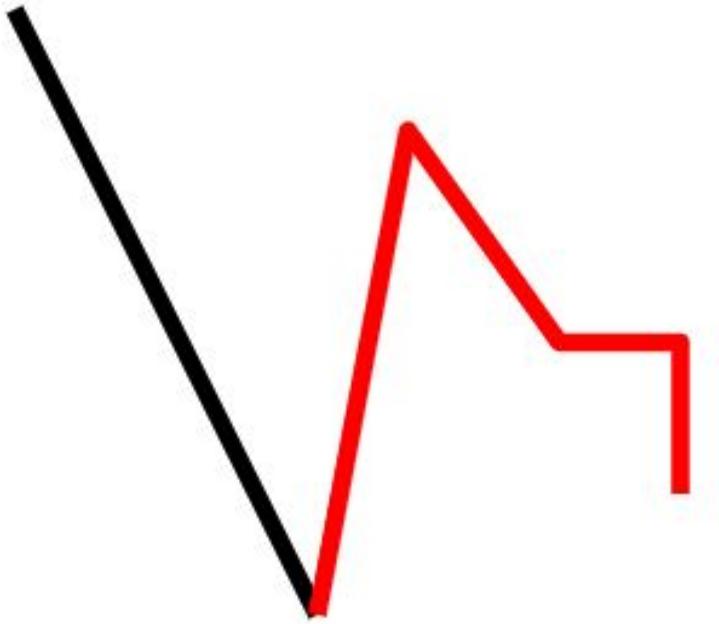
SVG Paths and CSS

HTML

```html
<svg widht="200" height="200">
  <text x="10" y="15">Case 3</text>
  <g class="lines">
    <line x1="10" y1="25" x2="60" y2="120"></line>
    <path d="M60 120 L75 40 L100 75 H120 V100"></path>
  </g>
</svg>
```

CSS

```css
.lines {
  stroke: black;
  stroke-width: 5;
}
.lines path {
  fill: none;
  stroke: red;
  stroke-linejoin: round;
}
```



[Codepen]

Introduction to D3

What is D3?

D3: Data-Driven Documents

- A **JS** library by Mike Bostock
- It allows you to **bind** arbitrary **data** to the **DOM**, and then apply data-driven **transformations** to the document.
- You can use it to generate an HTML table from an array of numbers. Or, use the same data to create an interactive SVG bar chart with smooth transitions and interaction.

How to load D3?

- The whole lib can be loaded from a single JS file:
  - https://d3js.org/d3.v6.min.js
  - https://cdnjs.cloudflare.com/ajax/libs/d3/6.7.0/d3.min.js
  - https://unpkg.com/d3@6.7.0/dist/d3.min.js
  - …
- When loaded, it'll introduce a global object **d3**

HTML

```
1 <html>
2   <head>
3     <script src="https://d3js.org/d3.v6.min.js" defer></script>
4   </head>
5   <body></body>
6 </html>
```

D3 Selections

- You can select DOM elements in an easy way with D3
- Say you want to change the text color of paragraph elements; we would do something like this:

```
const paragraphs = document.querySelectorAll("p");
for (const p of paragraphs) {
  p.style.color = 'blue';
}
```

- With D3, we could achieve the same result with:

```
d3.selectAll('p').style('color', 'blue');
```

Codepen

Making a smiley face with D3

Codepen

Adding a simple transition

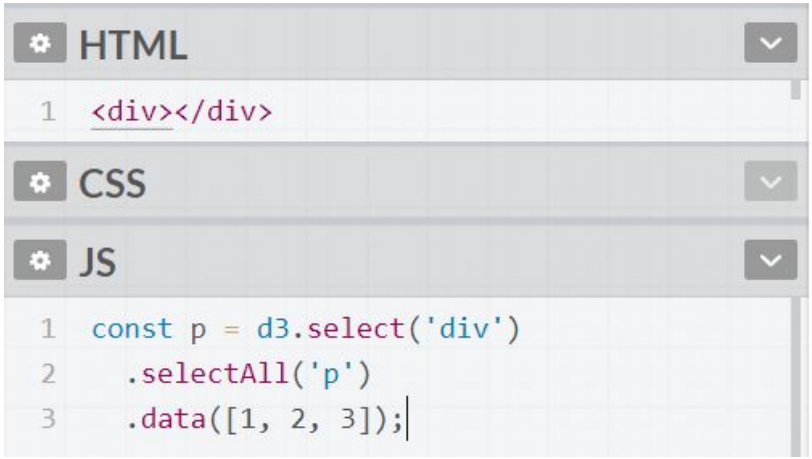
```
40  const eyebrowGroup = eyeGroup
41    .append('g')
42      .attr('transform', `translate(0,${eyebrowYOffset})`);
43
44  eyebrowGroup
45    .transition().duration(2000)
46      .attr('transform', `translate(0,${eyebrowYOffset-30})`)
47    .transition().duration(2000)
48      .attr('transform', `translate(0,${eyebrowYOffset})`)
49
50  const leftEyebrow = eyebrowGroup
51    .append('rect')
52      .attr('width', eyebrowWidth)
53      .attr('height', eyebrowHeight)
54      .attr('x', -eyeSpacing - eyebrowWidth/2);
```

[Codepen]

Data binding with D3

D3 Data Bind

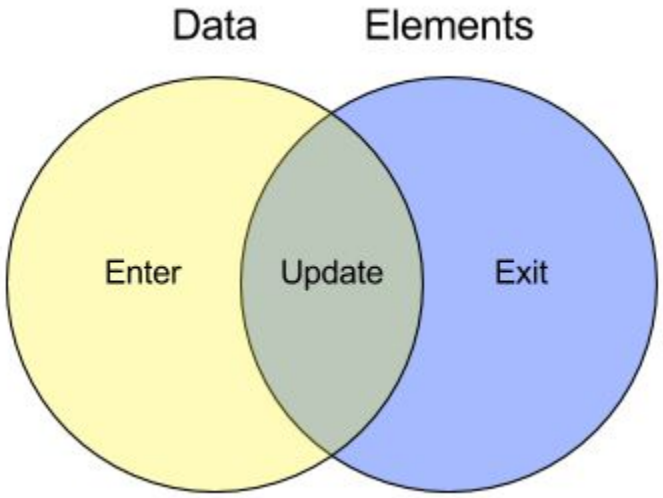
- We can Bind data to a D3 selection by calling the method “data” (even if the DOM elements does not exist yet):



- When data is bound to a selection, each element in the data array is paired with the corresponding node in the selection.

D3 Data Join

- D3 uses a declarative style of programming to bind data to DOM elements and returns three virtual selections:
  - **Enter:** New data, no selection elements
  - **Update:** Data item mapped to existing selection element
  - **Exit:** Selection elements with no data item mapped to them



https://www.amphinicy.com/blog/post/manipulating-svg-using-d3js-library

D3 Data Join: enter

- Since we don’t have any “p” inside our “div”, the join must happen at the “enter” stage:

HTML

```
1  <div></div>
```

CSS

JS

```
1  const p = d3.select('div')
2    .selectAll('p')
3    .data([1, 2, 3])
4    .enter().append('p')
5      .text(d => `Paragraph ${d}`);
```

Paragraph 1

Paragraph 2

Paragraph 3

[Codepen]

D3 Data Join: update

- What if we had one “p” inside our “div”?

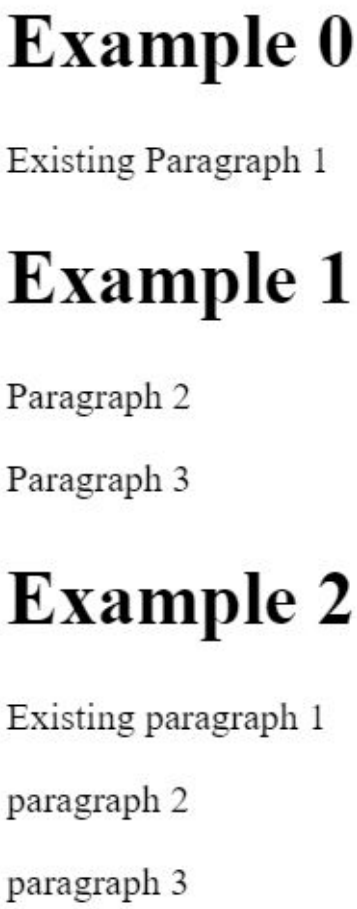
HTML

```html
<h1>Example 0</h1>
<div id="ex0">
  <p></p>
</div>

<h1>Example 1</h1>
<div id="ex1">
  <p></p>
</div>

<h1>Example 2</h1>
<div id="ex2">
  <p></p>
</div>
```

JS

```js
const ex0 = d3.select('#ex0')
  .selectAll('p')
  .data([1, 2, 3])
    .text(d => `Existing Paragraph ${d}`);

const ex1 = d3.select('#ex1')
  .selectAll('p')
  .data([1, 2, 3])
  .enter().append('p')
    .text(d => `Paragraph ${d}`);

const ex2 = d3.select('#ex2')
  .selectAll('p')
  .data([1, 2, 3])
    .text(d => `Existing paragraph ${d}`);

ex2.enter().append('p')
  .text(d => `paragraph ${d}`);
```

[Codepen]



D3 Data Join: exit

- What if we had more than three “p”s inside our “div”?

HTML

```
<div>
  <p></p>
  <p></p>
  <p></p>
</div>
```

JS

```
const div = d3.select('div')
  .selectAll('p')
  .data([1, 2, 3])
    .text(d => `Paragraph ${d}`);
```

Paragraph 1

Paragraph 2

Paragraph 3

```
<div>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
  <p>Paragraph 3</p>
  <p></p>
</div>
```

JS

```
const div = d3.select('div')
  .selectAll('p')
  .data([1, 2, 3])
    .text(d => `Paragraph ${d}`);

div.exit().remove();
```

Paragraph 1

Paragraph 2

Paragraph 3

```
<div>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
  <p>Paragraph 3</p>
</div>
```

Codepen

Making a bar chart with D3

CSV

CSV: **C**omma **S**eparated **V**alue

- Allows data to be save in tabular format
- Each row is a record
- Each record has one or more fields separated by commas

E.g.:
https://raw.githubusercontent.com/murilocamargos/iwp/main/pages/countries/countries.csv

```
country,population
China,1407692960
India,1376238018
United States,331449281
Indonesia,271350000
Pakistan,225200000
Brazil,213057783
Nigeria,211401000
Bangladesh,170566060
Russia,146171015
Mexico,126014024
```

CSV

We want to load this CSV file in our JS code using D3.

- Use the **d3.csv** method.
- It returns a **Promise** (like fetch)!
- It takes care of data structure processing and returns a list of records with **named fields**.

JS

```
1  d3.csv('https://raw.githubusercontent.com/murilocamargos/iwp/main/pages/countries/countries.csv').then(data => console.log(data))
```

Codepen

Console

```
// [object Array] (10)
▾ [// [object Object]
▾ {
    "country": "China",
    "population": "1407692960"
  },// [object Object]
▾ {
```

CSV

- We can process each record in our CSV:

JS

```
d3.csv('https://raw.githubusercontent.com/murilocamargos/iwp/main/pages/countries/countries.csv').then(data => {
  data.forEach(item => {
    item.population = +item.population;
  });
  console.log(data)
});
```

Console

```
    "country": "Russia",
    "population": 146171015
  },// [object Object]
  {
    "country": "Mexico",
    "population": 126014024
  }]
```

[Codepen]

CSV Join and rectangle creation

- We want to create a rectangle for each country.
- We can use selects and data joint to do that!

JS

```
const svgWidth = 960;
const svgHeight = 500;

const svg = d3
  .select('svg')
    .attr('width', svgWidth)
    .attr('height', svgHeight);

const render = data => {
  svg.selectAll('rect').data(data)
    .enter().append('rect')
      .attr('width', 300)
      .attr('height', 100);
};

d3.csv('https://raw.githubusercontent.com/murilocamargos/iwp/main/pages/countries/countries.csv').then(data => {
  data.forEach(item => {
    item.population = +item.population;
  });
  render(data);
});
```

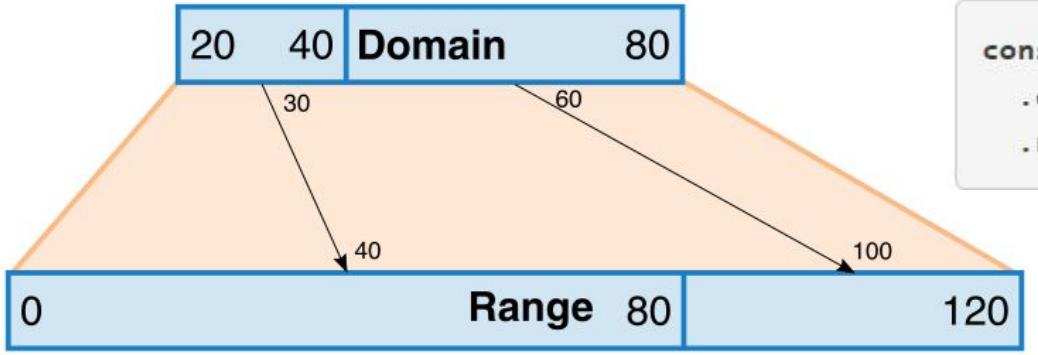
```
<svg width="960" height="500">
  <rect width="300" height="100"></rect>
  <rect width="300" height="100"></rect>
  <rect width="300" height="100"></rect>
  <rect width="300" height="100"></rect>
  <rect width="300" height="100"></rect>
  <rect width="300" height="100"></rect>
  <rect width="300" height="100"></rect>
  <rect width="300" height="100"></rect>
  <rect width="300" height="100"></rect>
  <rect width="300" height="100"></rect>
</svg>
```

Codepen

D3 Linear scales

- We can map our values in pixels using D3 scales.
  - “Scales are functions that map from an input domain to an output range.” - **Mike Bostock**

- The linear scale is useful with quantitative attributes.




```
const myScale = d3.scaleLinear()
  .domain([20, 80])
  .range([0, 120]);
```

https://github.com/d3/d3-scale/blob/master/README.md
https://www.d3indepth.com/scales/

Back to our example

```
const render = data => {
  const xScale = d3.scaleLinear()
    .domain([0, d3.max(data, d => d.population)])
    .range([0, svgWidth]);

  svg.selectAll('rect').data(data)
    .enter().append('rect')
      .attr('width', d => xScale(d.population))
      .attr('height', 100);
};
```

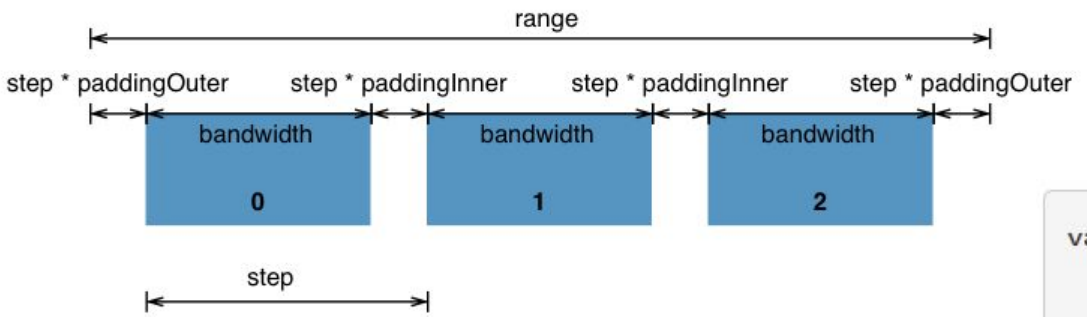
```
▼<svg width="960" height="500">
   <rect width="960" height="100"></rect>
   <rect width="938.5487708058155" height="100"></rect>
   <rect width="226.03743770942776" height="100"></rect>
   <rect width="185.0517175279473" height="100"></rect>
   <rect width="153.5789452268057" height="100"></rect>
   <rect width="145.298355175407" height="100"></rect>
   <rect width="144.16848401372982" height="100"></rect>
   <rect width="116.32040668868586" height="100"></rect>
   <rect width="99.68379354543337" height="100"></rect>
   <rect width="85.93739293830097" height="100"></rect>
 </svg>
```

- D3 has a nice “max” function to help.
- Our max width allowed will be “svgWidth”.
- The width will vary according to the population.

Codepen

D3 Band scales

- What about the vertical positioning?
- Band scales are great when the domain is ordinal but the range is continuous and numeric.




```
var bandScale = d3.scaleBand()
  .domain(['Mon', 'Tue', 'Wed', 'Thu', 'Fri'])
  .range([0, 200]);

bandScale('Mon'); // returns 0
bandScale('Tue'); // returns 40
bandScale('Fri'); // returns 160
```

https://github.com/d3/d3-scale/blob/master/README.md
https://www.d3indepth.com/scales/

Back to our example

```
const render = data => {
  const xScale = d3.scaleLinear()
    .domain([0, d3.max(data, d => d.population)])
    .range([0, svgWidth]);

  const yScale = d3.scaleBand()
    .domain(data.map(d => d.country))
    .range([0, svgHeight]);

  svg.selectAll('rect').data(data)
    .enter().append('rect')
      .attr('y', d => yScale(d.country))
      .attr('width', d => xScale(d.population))
      .attr('height', d => yScale.bandwidth());
};
```

[Codepen]

Let’s clean up a little bit:

Removing duplicated logic.

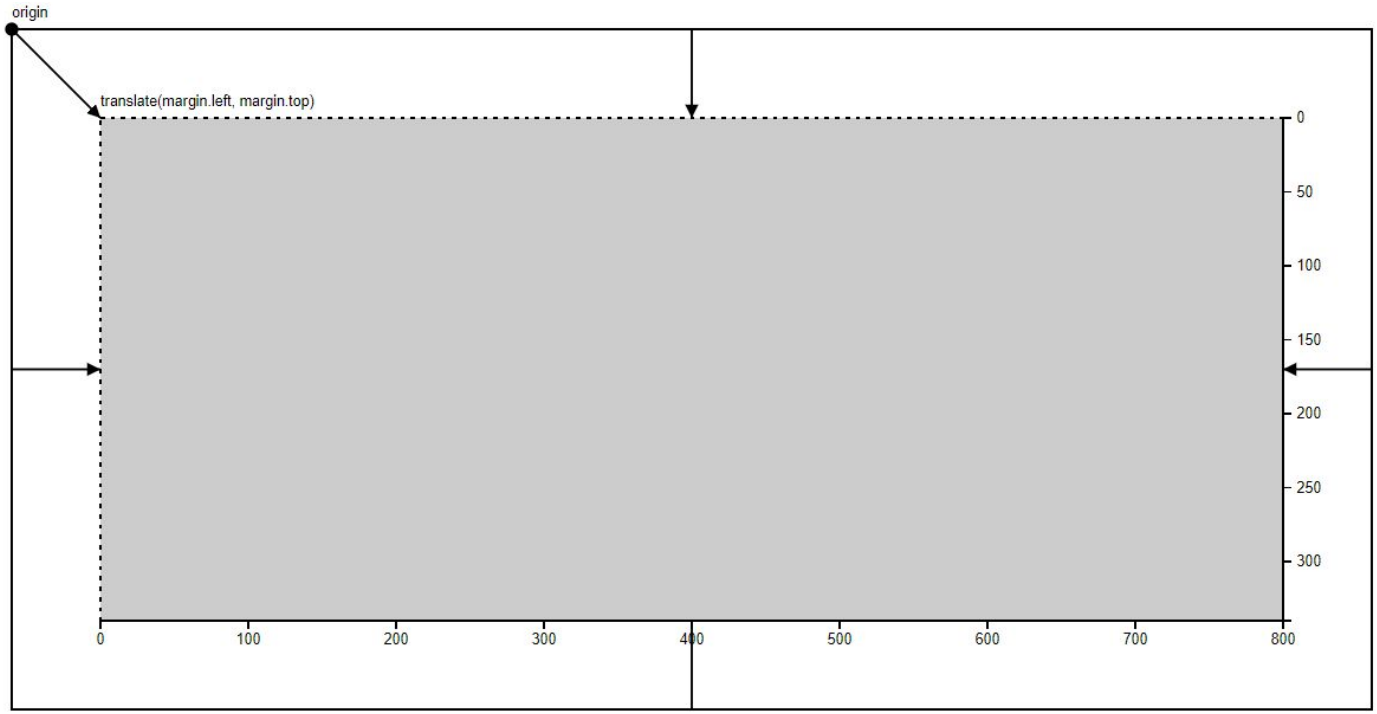
```
const render = data => {
  const xScale = d3.scaleLinear()
    .domain([0, d3.max(data, d => d.population)])
    .range([0, svgWidth]);

  const yScale = d3.scaleBand()
    .domain(data.map(d => d.country))
    .range([0, svgHeight]);

  svg.selectAll('rect').data(data)
    .enter().append('rect')
      .attr('y', d => yScale(d.country))
      .attr('width', d => xScale(d.population))
      .attr('height', d => yScale.bandwidth());
};
```

```
const render = data => {
  const xValue = d => d.population;
  const yValue = d => d.country;

  const xScale = d3.scaleLinear()
    .domain([0, d3.max(data, xValue)])
    .range([0, svgWidth]);

  const yScale = d3.scaleBand()
    .domain(data.map(yValue))
    .range([0, svgHeight])
    .padding(0.1);

  svg.selectAll('rect').data(data)
    .enter().append('rect')
      .attr('y', d => yScale(yValue(d)))
      .attr('width', d => xScale(xValue(d)))
      .attr('height', d => yScale.bandwidth());
};
```

Codepen

D3 Axis

- We need a place to put the axis, i.e., we need space around our bars.
- The margin convention



https://bl.ocks.org/mbostock/3019563

In our exemple

- We start by creating a “margin” object:

```
const margin = {top: 20, right: 20, bottom: 20, left: 20};
```

- Then, instead of appending our “rects” to “svg”, we’ll create a translated group:

```
const g = svg.append('g')
  .attr('transform', `translate(${margin.left},${margin.top})`);

g.selectAll('rect').data(data)
```

- Now, we need to fix our scales to consider the inner width and height:

```
const xScale = d3.scaleLinear()
  .domain([0, d3.max(data, xValue)])
  .range([0, svgWidth - margin.left - margin.right]);
```

```
  .range([0, svgHeight - margin.top - margin.bottom])
```

Codepen

D3 left axis

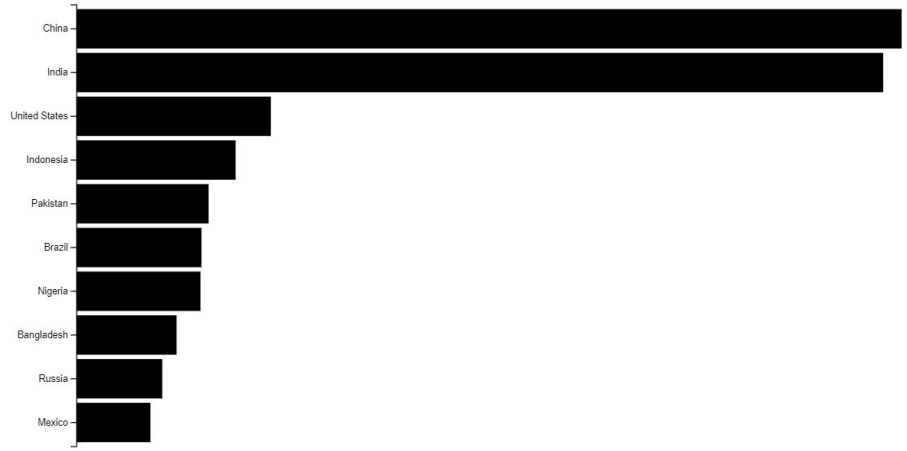
- We can add the country names using an “axisLeft”

```
const yAxis = d3.axisLeft(yScale);
```

- Then, we create a new group inside the previous one to append the created axis to it:

```
g.append('g').call(yAxis);
```

- Now we increase the left margin to fit it all



https://github.com/d3/d3-axis

Codepen

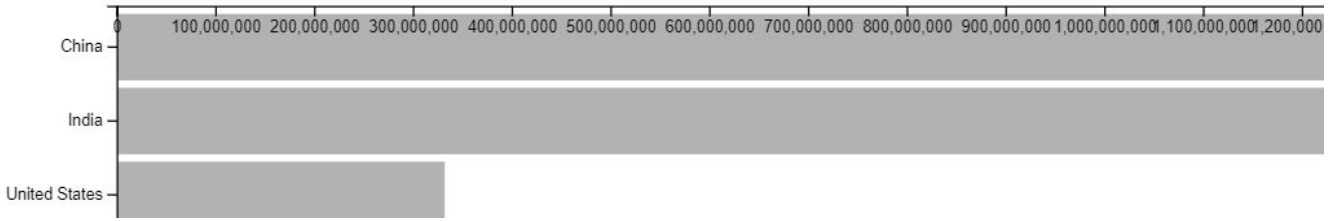
D3 bottom axis

- We can add the population using a “axisBottom”

```
const xAxis = d3.axisBottom(xScale);
```

```
g.append('g').call(xAxis);
```

- The axis ticks are below the bars

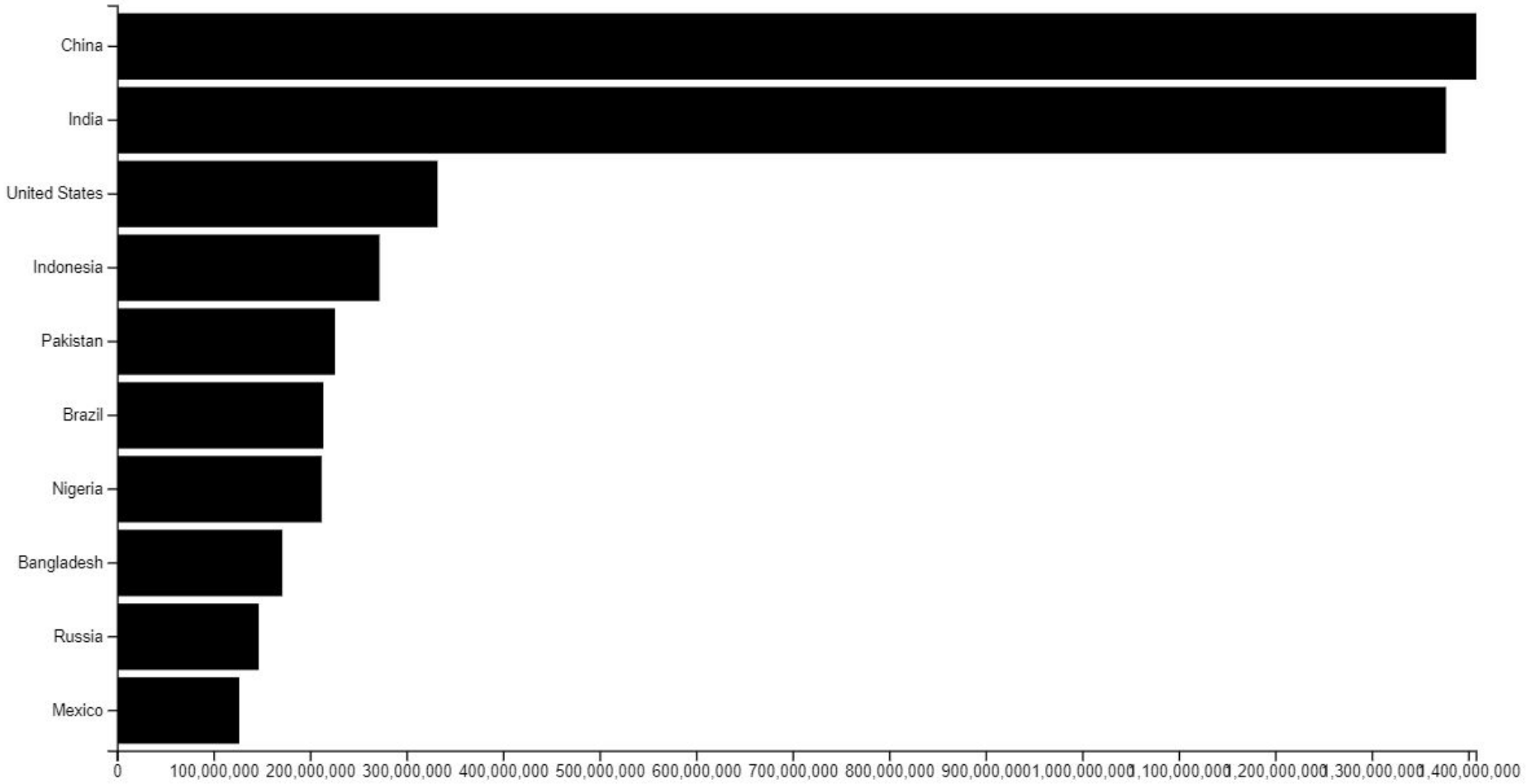


- The reference is (0,0), so we need to translate the axis:

```
g.append('g').call(xAxis)
  .attr('transform', `translate(0,${innerHeight})`);
```

Codepen

D3 bottom axis



Customizing Axes

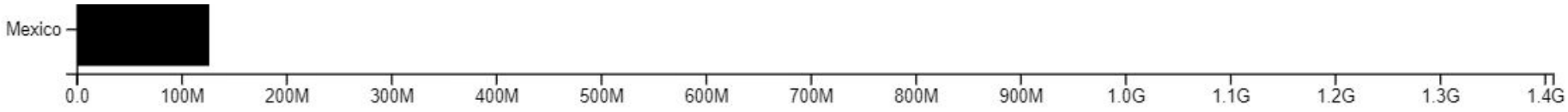
Formatting the ticks

- The axes has a tickFormat method that receives a custom formatting function:

```
const xAxis = d3.axisBottom(xScale)
  .tickFormat(myCustomFormattingFunction);
```

- D3 also offers a good helper to format numbers:
  - **d3.format('s')** returns a formatting function that transforms 1000000 in 1M. You can see other patterns.

```
const myCustomFormattingFunction = number =>
  d3.format('.2s')(number);
```



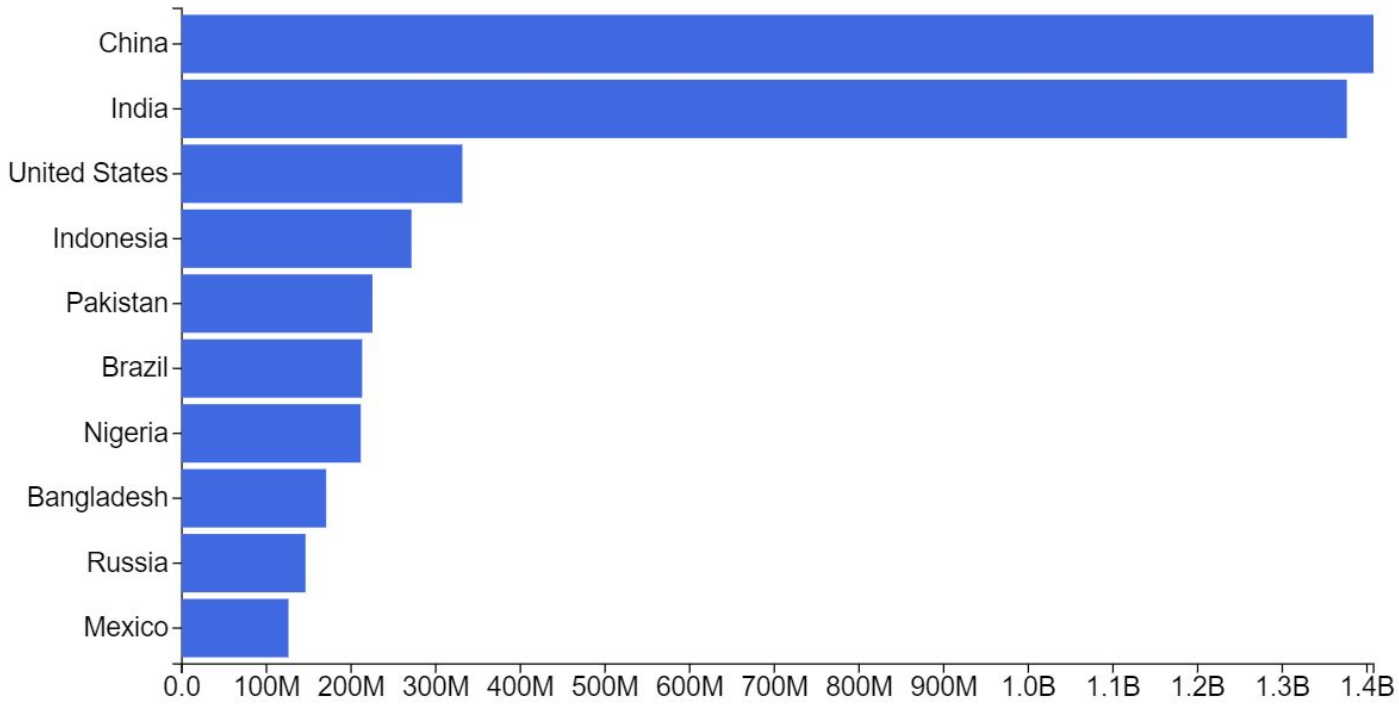

```
const myCustomFormattingFunction = number =>
  d3.format('.2s')(number)
    .replace('G', 'B');
```

Codepen

Formatting with CSS

- We can use CSS to completely change the appearance of our chart:

```
text {
  font-size: 1.8em;
}
rect {
  fill: royalblue;
}
```

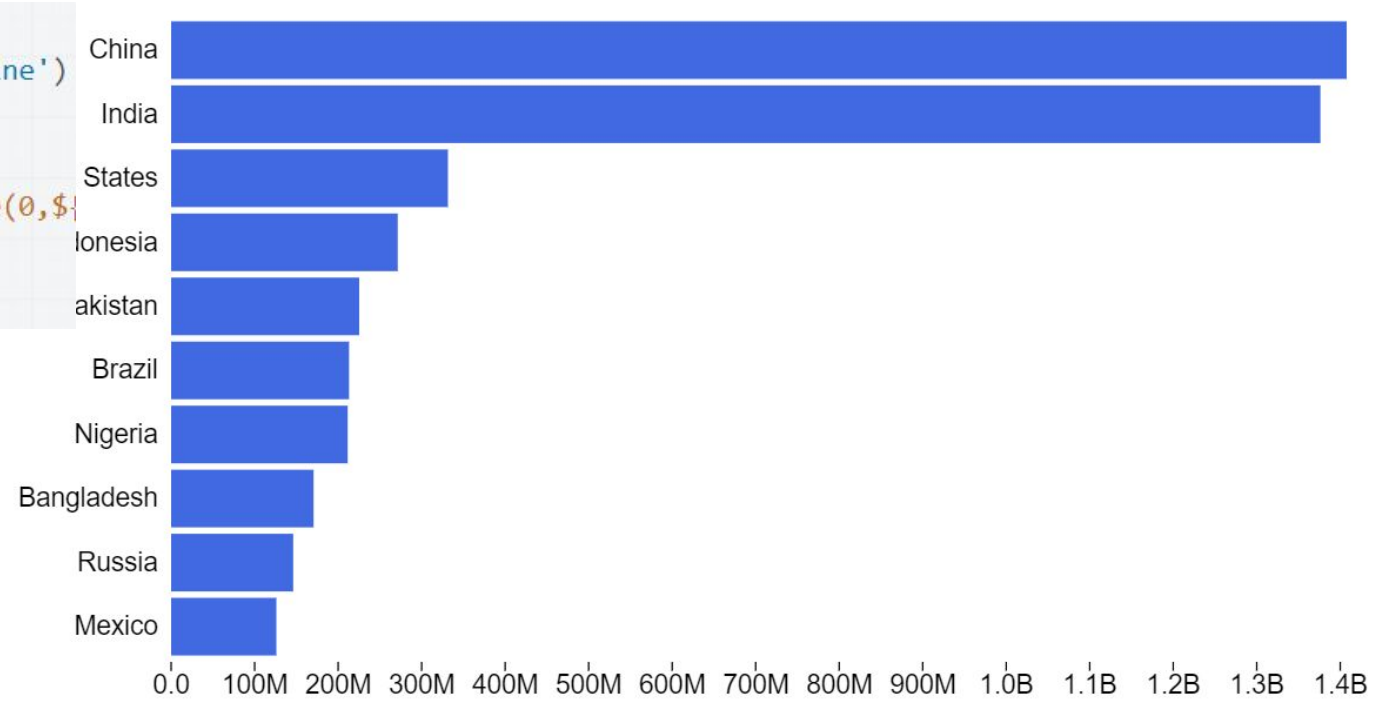


Codepen

Removing unnecessary lines

- You can select elements by their query selector

```
g.append('g').call(yAxis)
  .selectAll('.domain, .tick line')
    .remove();
g.append('g').call(xAxis)
  .attr('transform', `translate(0,${
  .select('.domain')
    .remove();
```



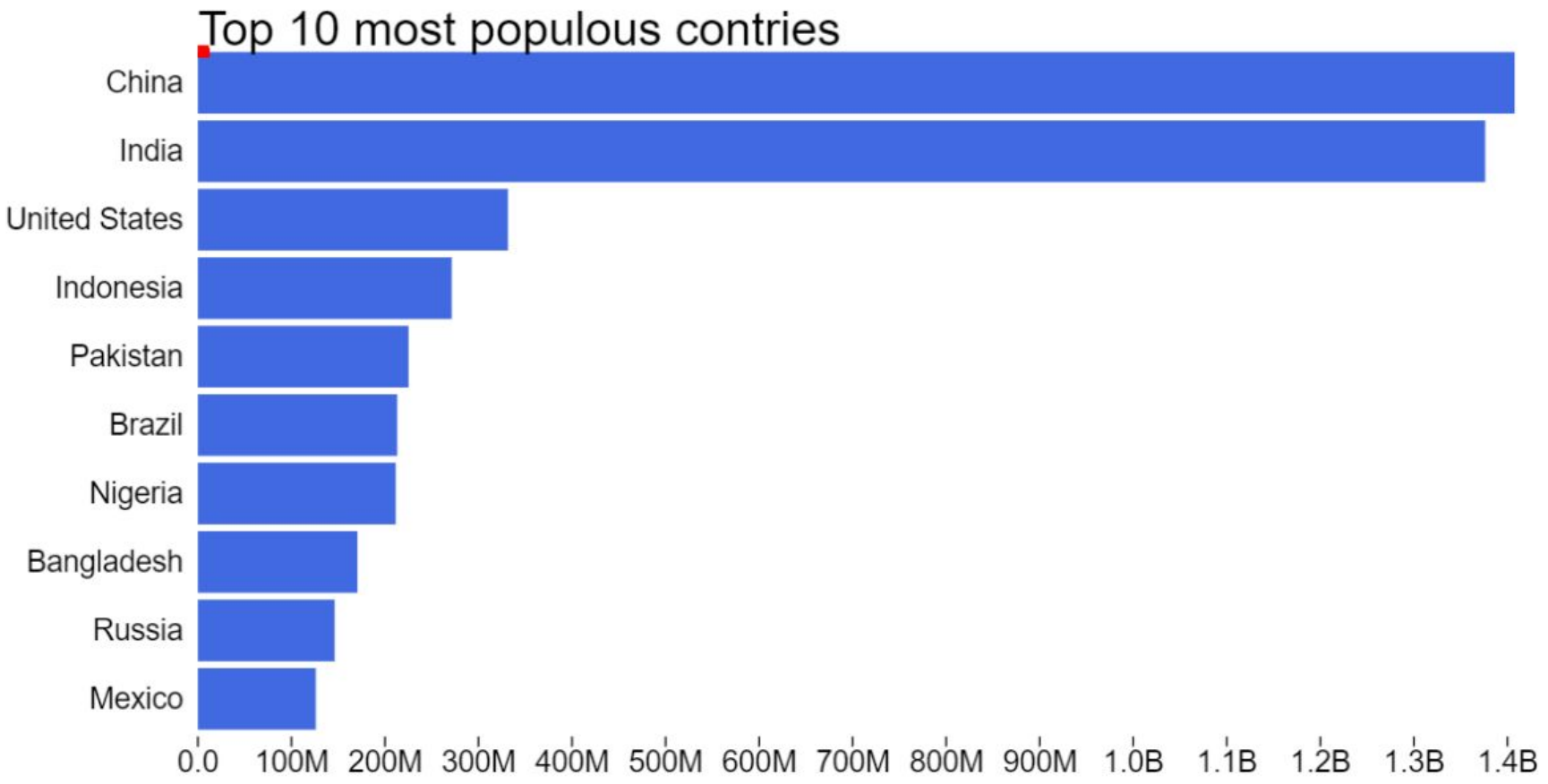
Codepen

Adding a title

- You can add a text element to the svg:

[Codepen]

```
g.append('text')
   .text('Top 10 most populous contries');
```

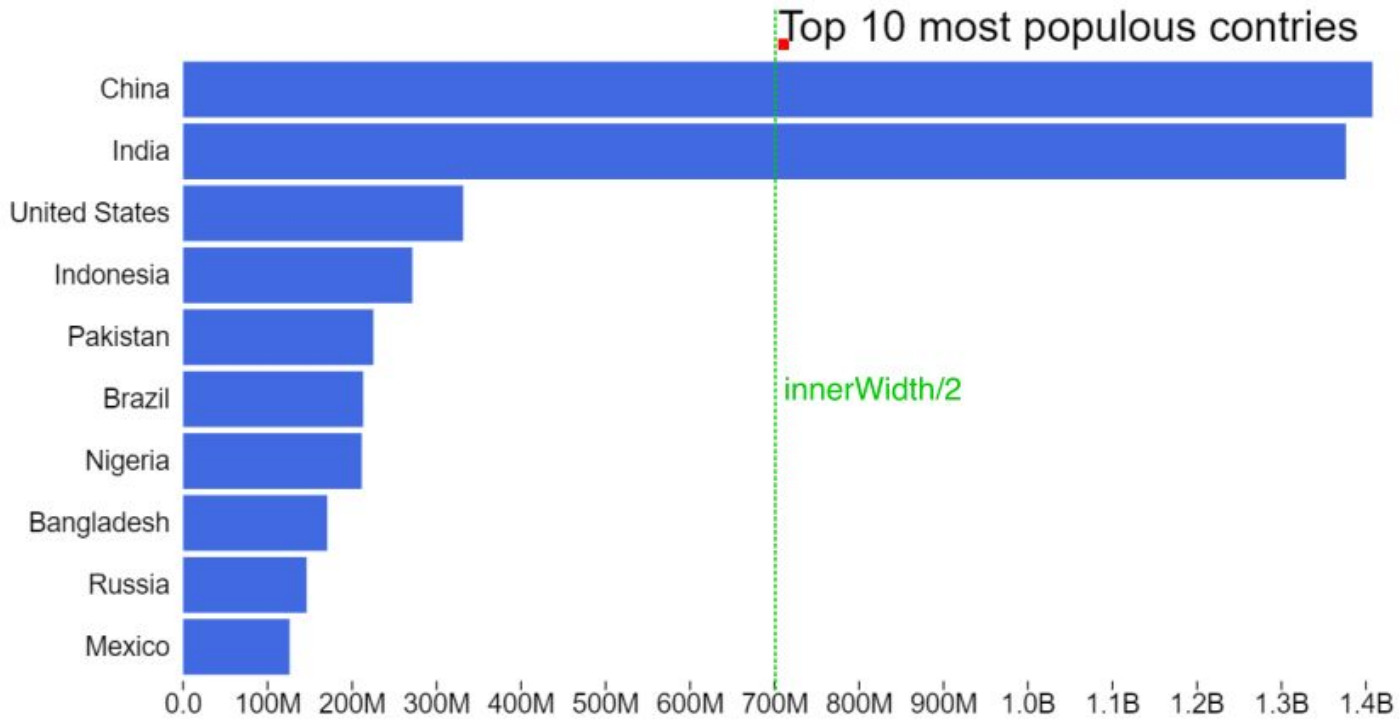


Adding a title

- First try to centralize the title

Codepen

```
g.append('text')
    .text('Top 10 most populous contries')
    .attr('y', -10)
    .attr('x', innerWidth/2);
```

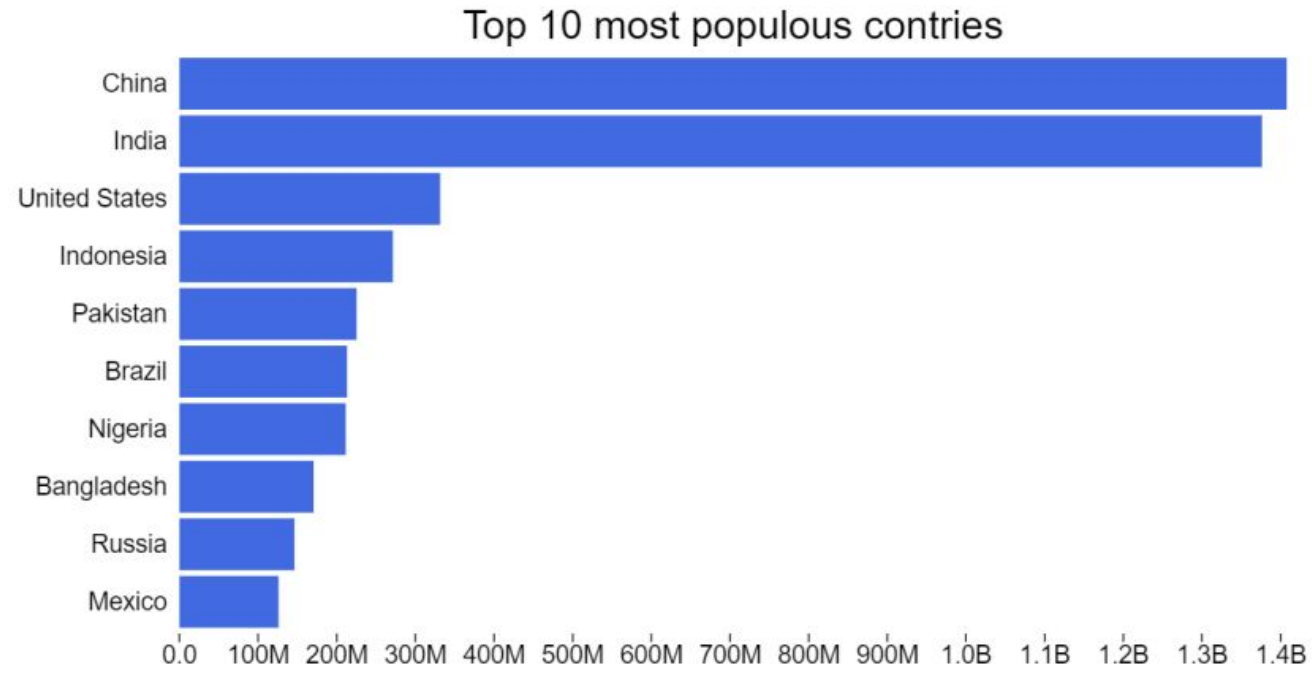


Adding a title

- Changing the text-anchor property

[Codepen]

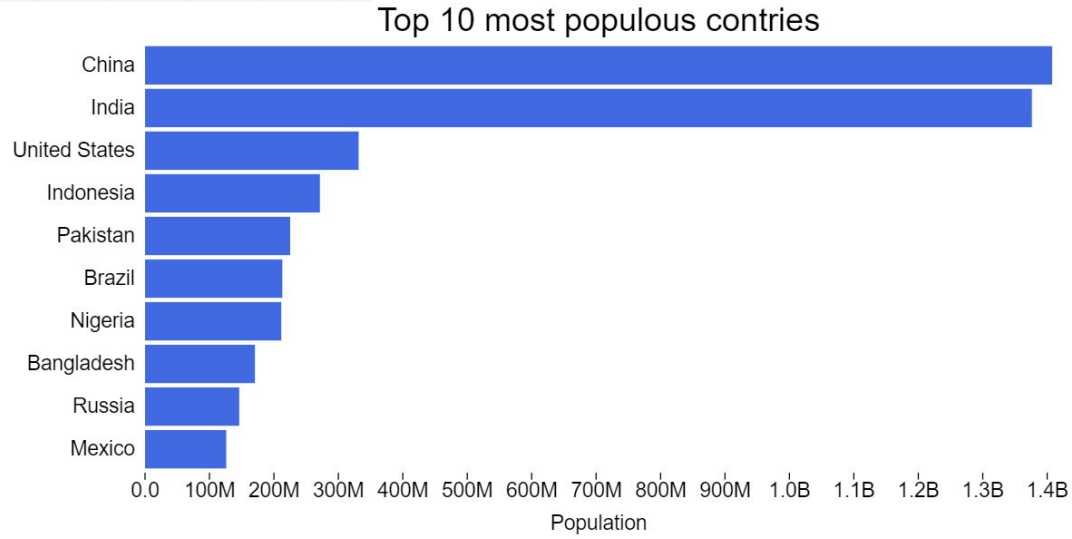
```
g.append('text')
    .text('Top 10 most populous contries')
    .attr('y', -10)
    .attr('x', innerWidth/2)
    .attr('text-anchor', 'middle');
```



Adding axis labels

- You can add text elements to the axis groups:

```
const xAxisG = g.append('g').call(xAxis)
  .attr('transform', `translate(0,${innerHeight})`);

xAxisG.select('.domain')
  .remove();

xAxisG.append('text')
  .attr('fill', 'black')
  .attr('y', 50)
  .attr('x', innerWidth/2)
  .text('Population');
```



- Adding a text on a axis group (by call) requires setting the “fill” attr.
- The “text-anchor” is already set to “middle”
- Remember to adjust the margins!

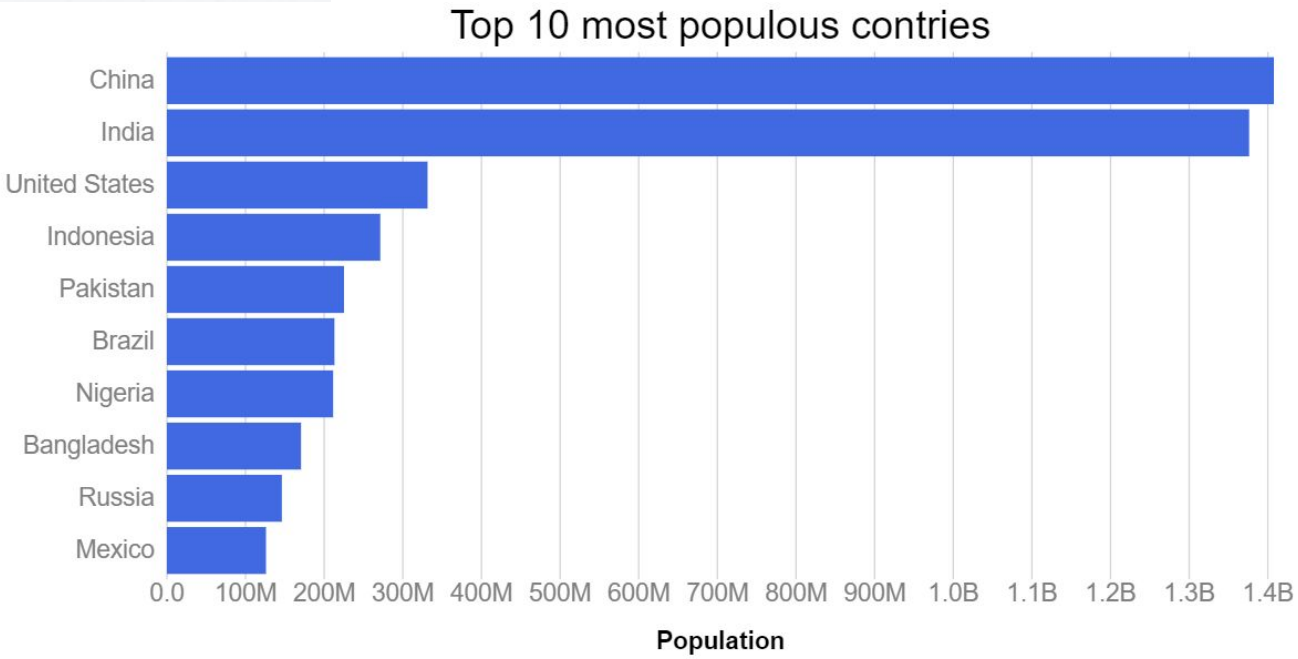
Codepen

Changing the tick size

- You can improve visualization changing the “tickSize”

```
const xAxis = d3.axisBottom(xScale)
  .tickFormat(myCustomFormattingFunction)
  .tickSize(-innerHeight);
```

```
12 .tick text {
13     color: gray;
14   }
15 .tick line {
16     stroke: lightgray;
17   }
```



Codepen