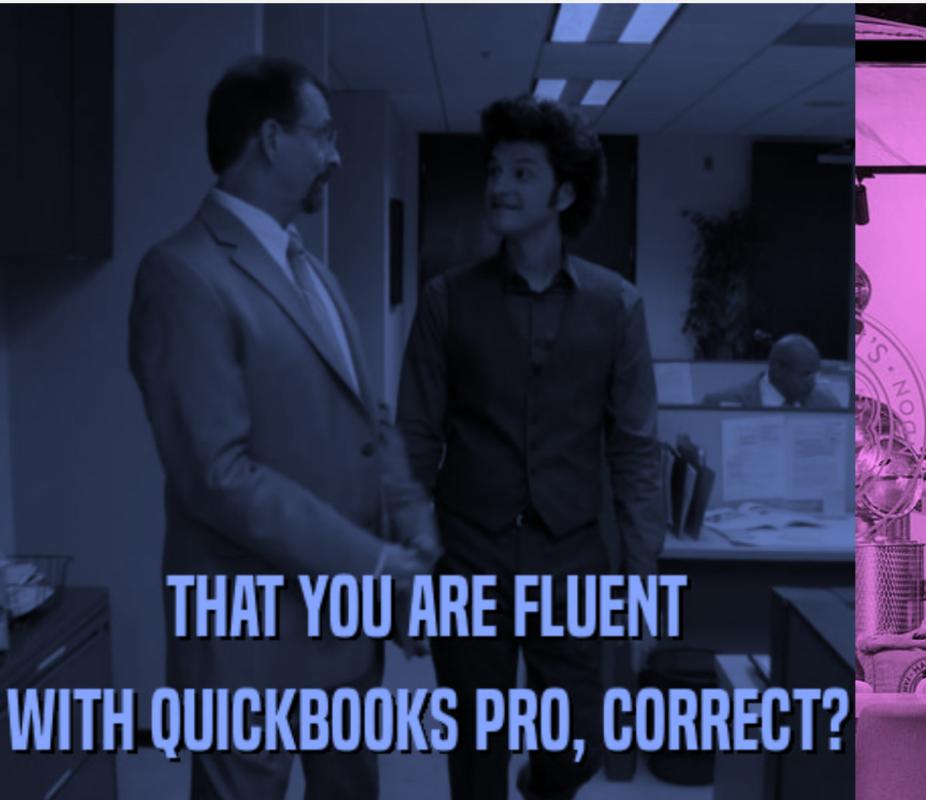
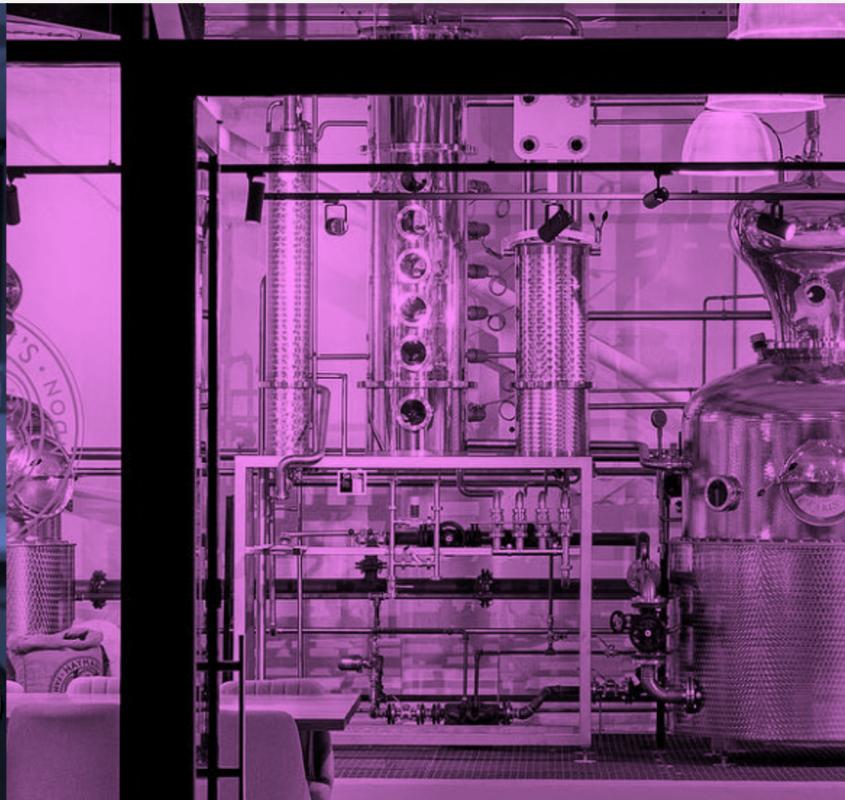


Presentation Options



Accounting Software

😔 too dull



Gin Making

🤖 🍷 need to share gin



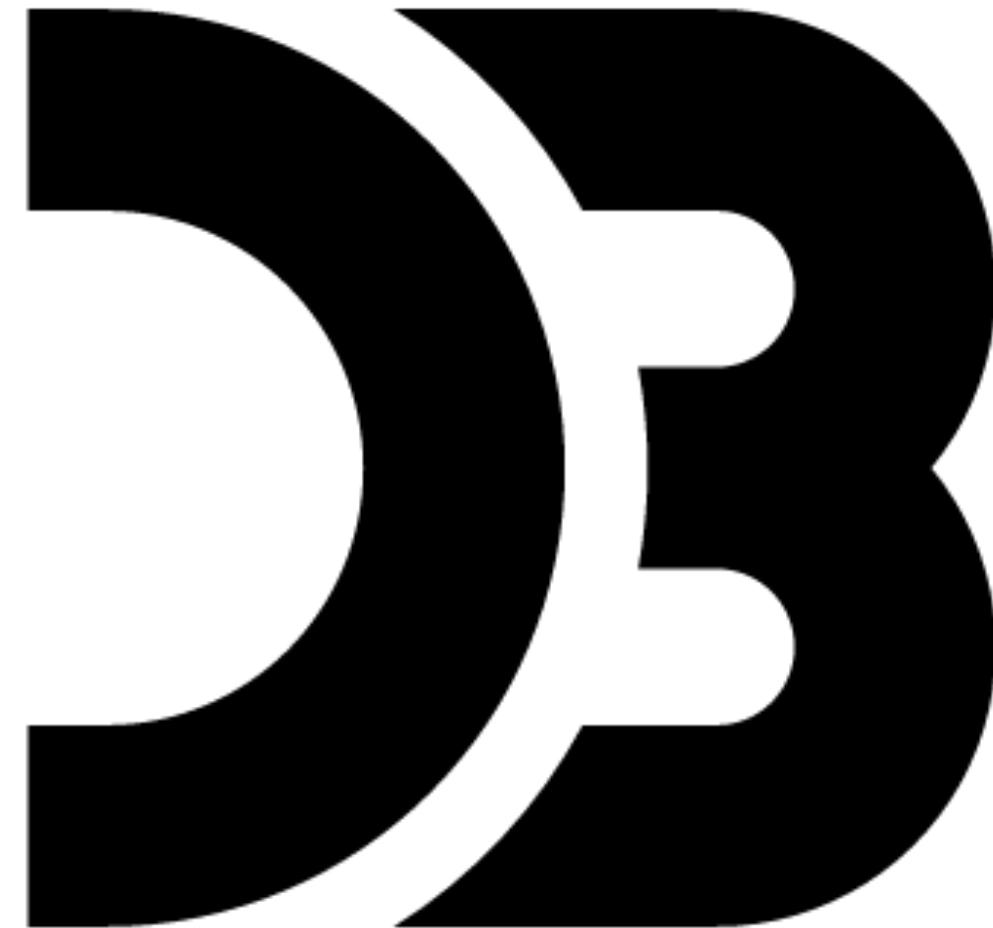
Mountain Sports

🚫 🏔 no mountains in London

An Intro to D3.js

by someone who doesn't
really know D3 or JS

 Fullstack d3, Amelia Wattenberger

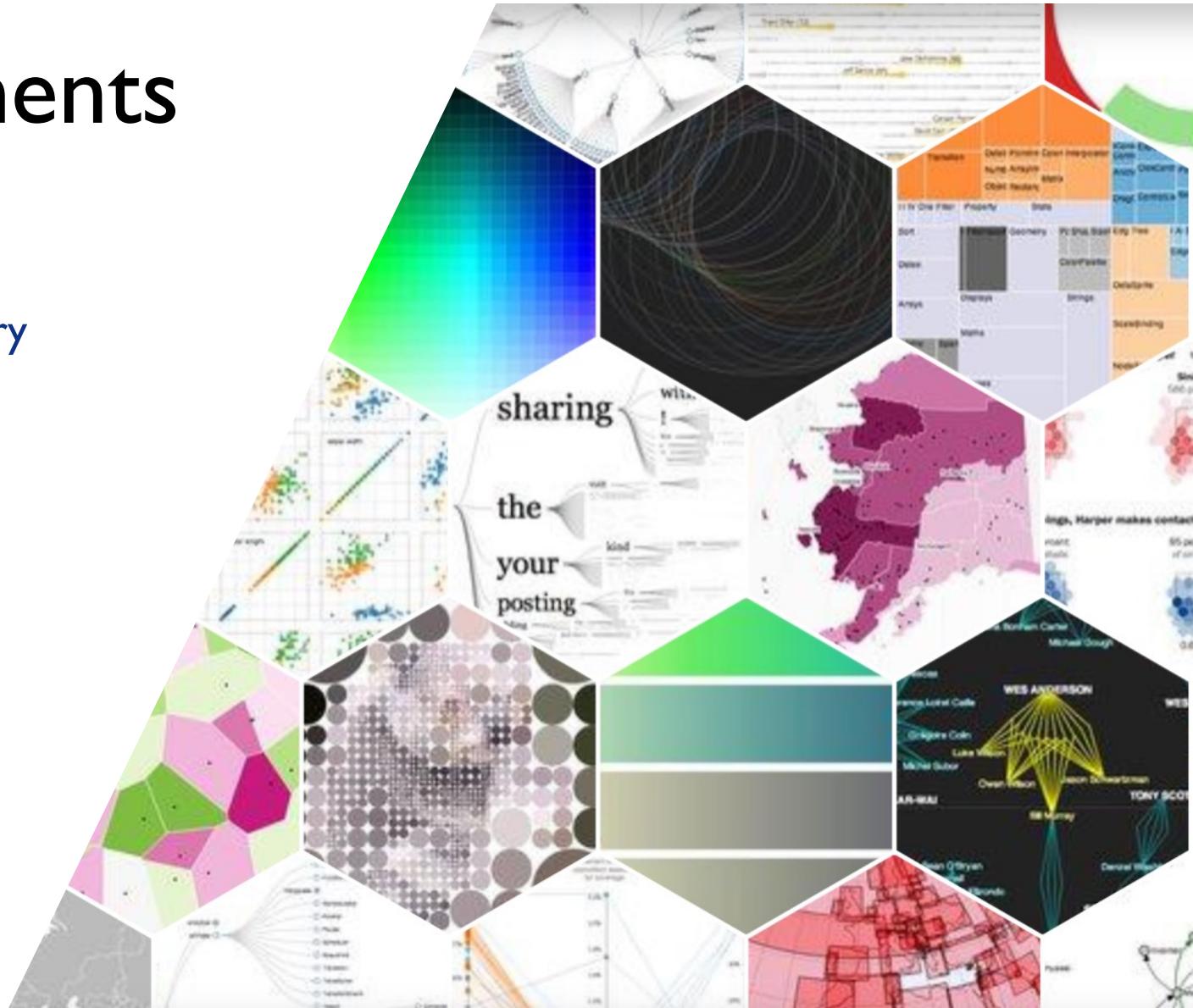


Data Driven Documents

- What**
- Advanced, interactive data viz
 - Low level; not a simple charting library

- How**
- **Bind** data to DOM elements
 - **Transform** elements based on data
 - Setup **transitions** and animations

- Who**
- Mike Bostock , NYT (2011)
 - Industry Standard (191M downloads)



D3 Modules

D3 contains ~30 modules, all written in a functional style, just install the packages you need:

Prepare Data

d3-fetch
d3-array
d3-time-formats
d3-time-intervals

DOM Manipulation

d3-selection

Draw SVG shapes

d3-shapes
d3-path
d3-axes

Convert data to pixels

d3-scales
d3-colors
d3-hsv

Animate & add interactions

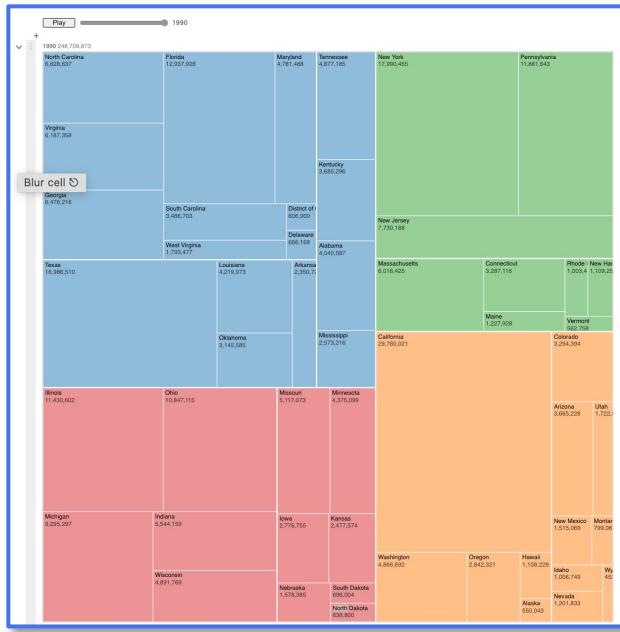
d3-transitions
d3-easings
d3-brushes
d3-dragging
d3-zooming
d3-timers

Specific Visualizations

d3-geo-projection
d3-geographies
d3-sankey
d3-chords
d3-voronoi

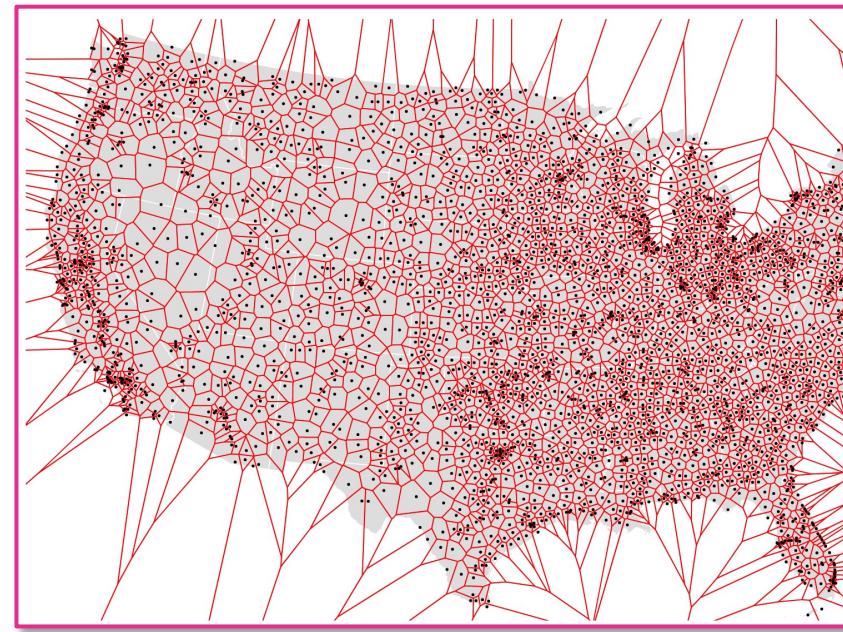
Examples

<https://observablehq.com/@d3/gallery>



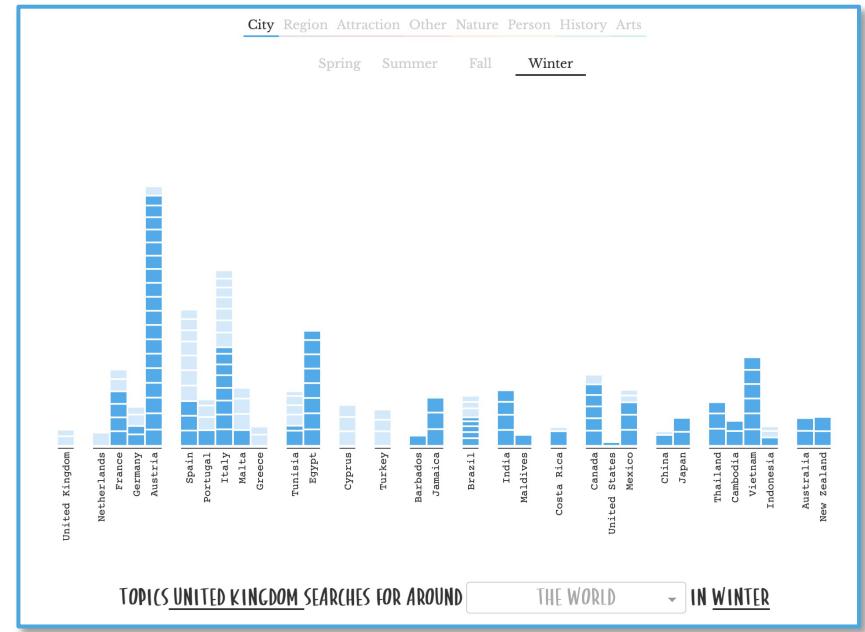
Population by state over time

Mike Bostock



Nearest airport by US city

Mike Bostock



Exploratory analysis of travel searches

Shirley Wu

D3 Basics

1. Fetch data

```
dataset = d3.json('example.json');
```

2. Select /create DOM elements

```
plot = d3.select('#plotDiv').append('svg');
```

*

3. Bind data to document elements

```
dataMap= plot.selectAll('.elements').data(dataset);
```

4. Create/remove elements as required

```
dataMap.enter().append('circle')
```

5. Transform elements based on data

```
.attr('cx', d => scale(d))
```

6. Transition (animate) between states

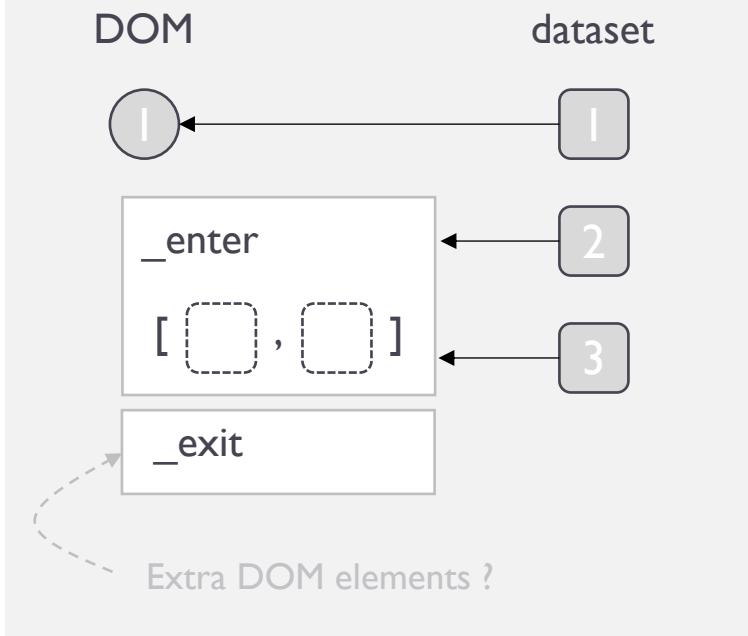
```
.transition().duration(1000)
```

* Will review data, enter and scales in more depth

d3.data(), enter(), exit()

data() →
Maps array of data to DOM elements

```
points =  
d3.selectAll('circle').data(dataset)
```



Once you have the mapping ...

⬅ **enter()**

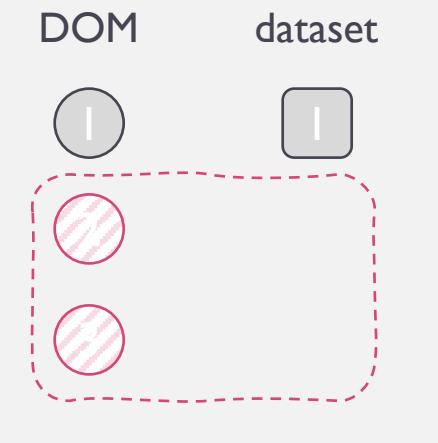
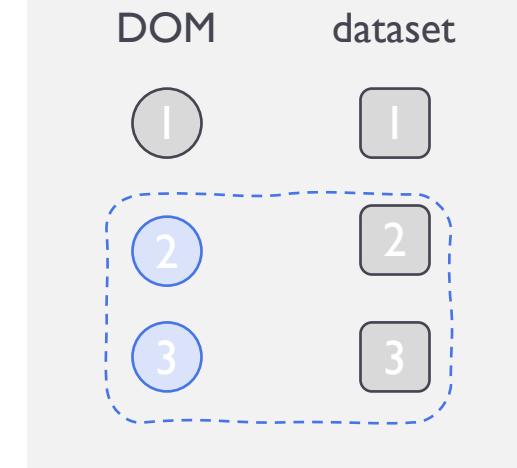
Select data missing elements

```
points.enter()  
.append('circle')  
.attr('fill', 'blue')
```

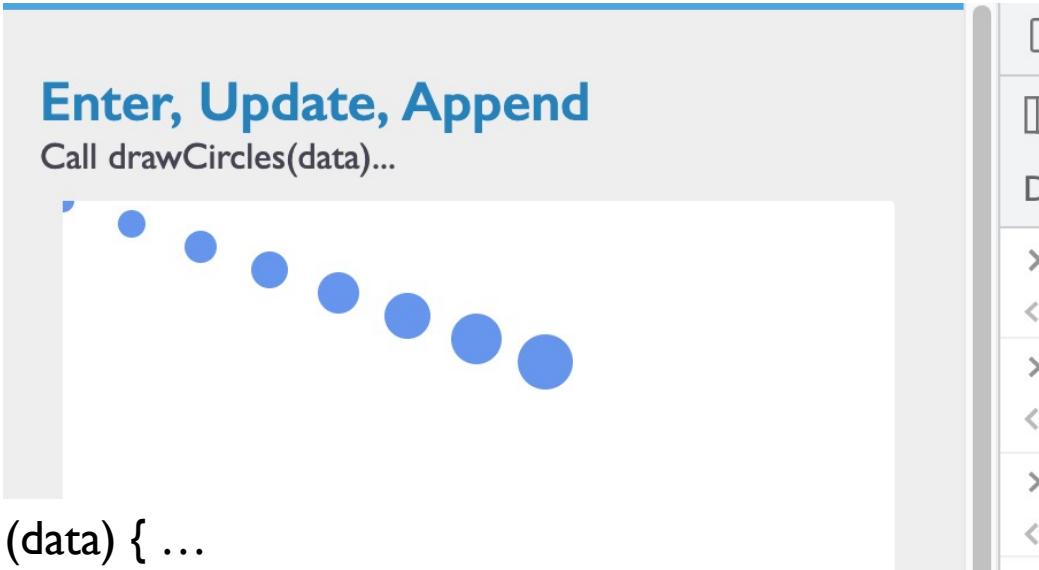
➡ **exit()**

Select elements missing data

```
points.exit()  
.attr('fill', 'red')  
.remove()
```

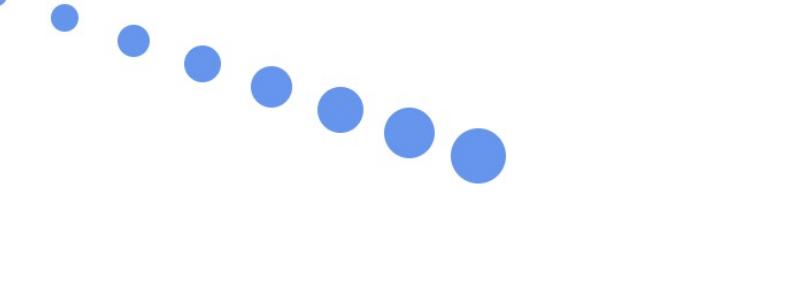


> drawCircles(dataTest)



```
function drawCircles (data) { ...  
  
// Bind data elements to any existing SVG circles  
const circles = plot.selectAll('circle').data(data);  
  
// Create new circles for unmapped data  
circles.enter().append('circle')  
  .attr('cx', d => d*30)  
  .attr('cy', d => d*10)
```

Enter, Update, Append
Call drawCircles(data)...



```
Elements Console > 1 | Filter  
Default levels ▾ | 1 Issue: 1  
▶ const dataTest = Array.from(Array(8).keys())  
◀ undefined  
▶ dataTest  
◀ ▶ (8) [0, 1, 2, 3, 4, 5, 6, 7]  
▶ drawCircles(dataTest)  
◀ undefined
```

This function draws circles for each element in an array, positioning them based on their value * 30 pixels

```
> drawCircles([...dataTest, ...[8,9,10]])
```

Enter, Update, Append
Call drawCircles(data)...

```
function drawCircles (data) { ...  
  // Update existing circles: color grey  
  circles.attr('fill', 'darkgrey');  
  
  // Create new circles for unmapped data (blue)  
  circles.enter().append('circle')  
    .attr('color', 'cornflowerblue')
```

Elements Console >

top Filter

Default levels 1 Issue: 1

> dataTest.push(8,9,10)
< 11
> drawCircles(dataTest)
< undefined
>

Newly added circles are blue
Existing circles are grey

> drawCircles(dataTest.slice(0, 5))

The screenshot shows a browser developer tools interface with the following elements:

- Elements Tab:** Shows a list of nodes with a "Skip" button.
- Console Tab:** Shows the command `> drawCircles(dataTest.slice(0, 5))` and its result `< undefined`. It also shows a message `6 addedMutatedNodes empty skip`.
- Code Block:** A dark box containing the code for the `drawCircles` function, specifically the `exit` and `remove` logic.
- Output Area:** A light gray area showing four small gray circles representing the drawn circles.

Code in the dark box:

```
function drawCircles (data) { ...  
  // Remove unmapped elements (red -> remove)  
  circles.exit()  
    .remove();
```

Text at the bottom right:

If circles already on the DOM are no longer needed they are removed

d3.scale

Scales are functions that map data to physical ranges

1. Choose function type
2. Input data domain
3. Output range

```
xScale = d3.scaleLinear()  
    .domain([-10.54, 34.56])  
    .range([0, 800])
```

Scale Types

- Continuous (Linear, Power, Log, Time, Radial)
- Sequential
- Diverging
- Quantize
- Quantile
- Threshold
- Ordinal (Band, Point)

Dataset domain example ...

[-7.56, -3.45, -2.04, -4.39, -9.24, -10.54, -7.64, -1.24, 6.15, 4.13, 9.84, ..]

Domain: `d3.extent(dataset) → [-10.54, 34.56]`

Physical range examples ...

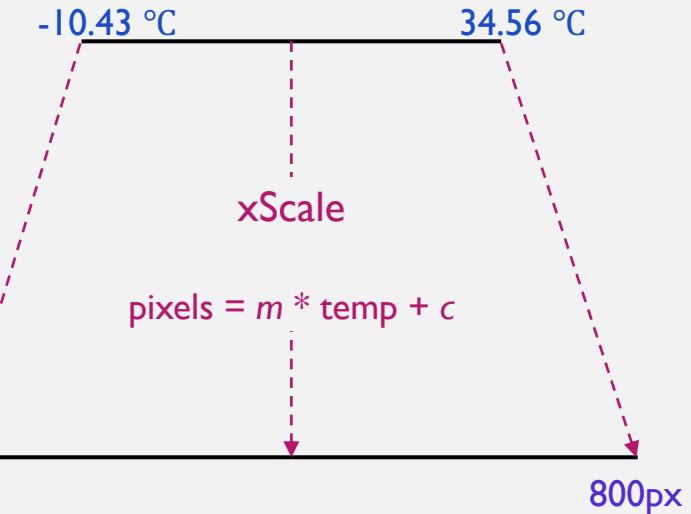
SVG area:

0px → 800px

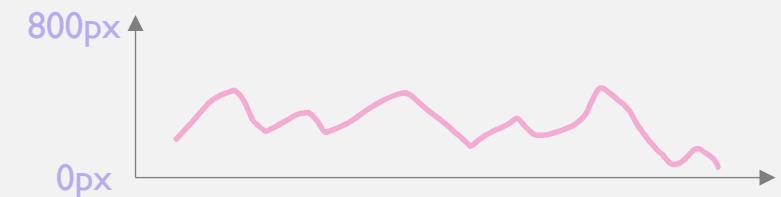
Colours:

#4775E7 → #D54773

scaleLinear

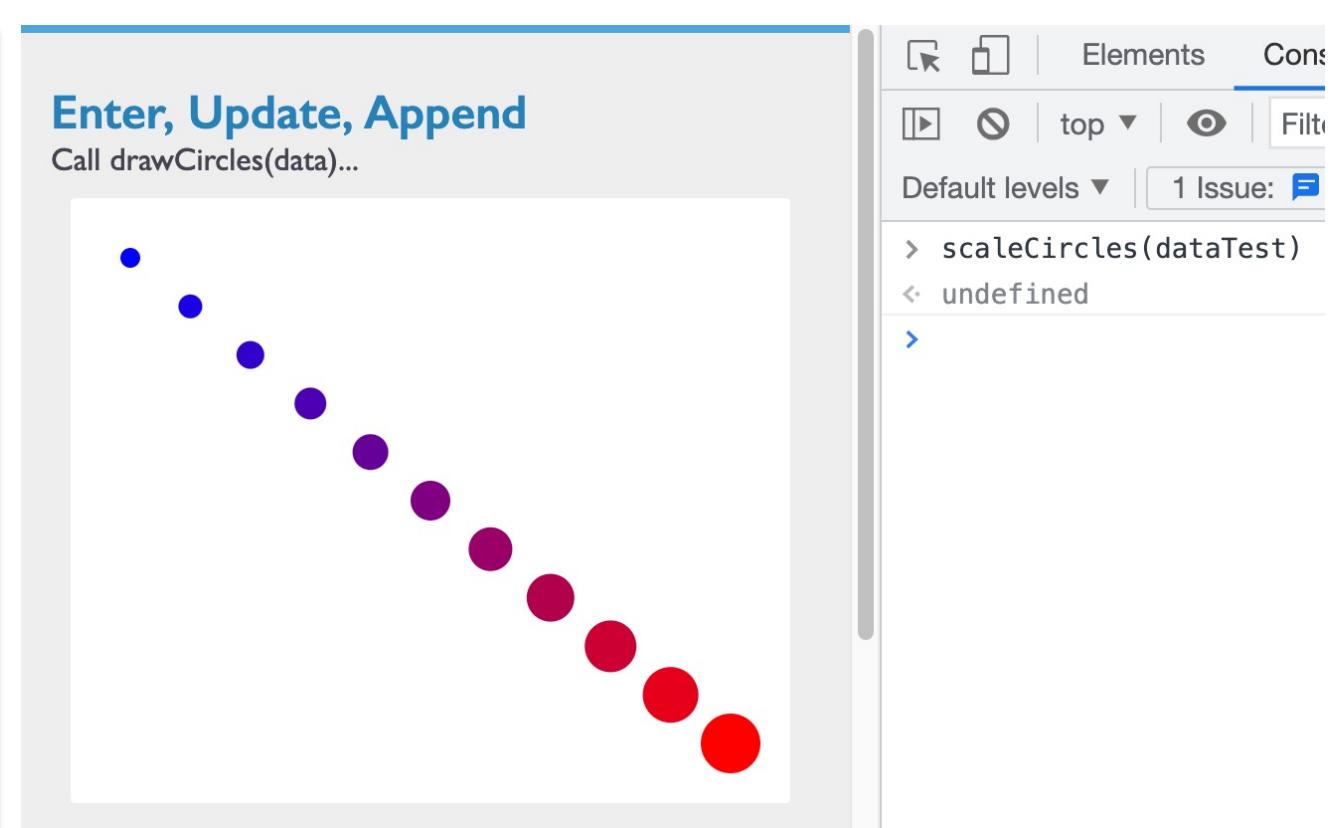


Use scale to help position elements on the DOM



> scaleCircles(dataTest)

```
function scaleCircles (data) {  
  
  const xScale = d3.scaleLinear()  
    .domain(d3.extent(data))  
    .range([30, width-30]);  
  
  const colourScale = d3.scaleLinear()  
    .domain(d3.extent(data))  
    .range(['blue', 'red']);  
  
  circles.attr('fill', 'darkgrey')  
    .attr('cx', d => xScale(d))  
    .attr('cy', d => yScale(d))  
    .attr('fill', d=>colourScale(d))
```



Now circles are positioned based on the width of the parent div and coloured from 'red' to 'blue'

Workflow For Charts

-  **Access Data** Import dataset, optionally declare 'accessor' functions for key fields
-  **Chart Dimensions** Declare chart size in pixels, helps to define scale outputs
-  **Draw Canvas** Render the chart area
-  **Create Scales** Create scales for every data-pixel mapping required in the chart e.g. y-pos, colour
-  **Draw Data** Render SVG or other DOM elements for the data
-  **Draw Peripherals** Render axis, labels, legends
-  **Setup Interactions** Initialize event listeners and create interaction behaviour

Demo

grid Access Data

```
async function drawScatter () {  
  
  let dataset = await d3.json('./data/my_weather_data.json');  
  
  const xAccessor = d => d.dewPoint;  
  const yAccessor = d => d.humidity;  
}
```

- D3.json parses json file and returns an array with the datapoints in the file
- xAcessor and yAccessor are functions to make accessing the relevant fields in the dataset easier.

Demo



Chart Dimensions

```
const width = d3.min([window.innerHeight * 0.9, window.innerWidth * 0.9]);  
  
let dimensions = {  
  width,  
  height: width,  
  margin: {top: 10, right: 10, bottom: 10, left: 10}  
};  
  
dimensions.plotWidth = dimensions.width - dimensions.margin.right -  
dimensions.margin.left;  
dimensions.plotHeight = dimensions.height - dimensions.margin.top -  
dimensions.margin.bottom;
```

- Save dimensions of chart in an object: setting `height = width = 90% window`
- Calculate dimensions of inner plot area by subtracting the margins

Demo



Draw Canvas

```
const wrapper = d3.select('#wrapper-scatter')
  .append('svg')
  .attr('width', dimensions.width)
  .attr('height', dimensions.height);

const plot = wrapper.append('g')
  .style('transform', `translate(${dimensions.margin.left}px,
${dimensions.margin.top}px)`);
```

- Similar approach to jQuery: select div by id, append svg and set attributes
- Plot is a svg group element and will maintain a margin between the plot elements and the outer svg

Demo



Create Scales

```
const xScale = d3.scaleLinear()  
  .domain(d3.extent(dataset, xAccessor))  
  .range([0, dimensions.plotWidth])  
  .nice();  
  
const yScale = d3.scaleLinear()  
  .domain(d3.extent(dataset, yAccessor))  
  .range([dimensions.plotHeight, 0])  
  .nice();
```

- Create scale functions that map datapoints to physical positions
- The scale functions are provided the domain of the data and the physical (pixel) range
- Nice is a helper method that makes ranges ‘nicer’ e.g. [-0.7823, 0.197] -> [-0.8, 0.2]

Demo



```
const points = plot.selectAll('circle').data(data);

points.join('circle') //shortcut for .enter().append('circle').merge(dataset)
  .attr('cx', d => xScale(xAccessor(d)))
  .attr('cy', d => yScale(yAccessor(d)))
  .attr('r', 5)
  .attr('fill', d => colourScale(colourAccessor(d)));
```

- Select all circle SVG elements on the plot and bind them to the datapoints
- Add new circles for each datapoint that cannot be bound to an existing circle
- Update the attributes (x and y position based on applying the scale function to each relevant datapoint)
- Set colour based on colour scale (not shown on earlier slides)

Demo



Draw Peripherals

```
const xAxisGenerator = d3.axisBottom(xScale);
const xAxis = plot.append('g').call(xAxisGenerator)
  .style('transform', `translateY(${dimensions.plotHeight}px)`);

const yAxisGenerator = d3.axisLeft(yScale);
const yAxis = plot.append('g').call(yAxisGenerator);
```

- Create an axis generator based on the scale functions
- Append a new g element to the plot, apply the axis generator
- Change the position of the elements as required

D3 and React

Both **React** and **D3** want control of the DOM but there are approaches to make them work together:

Option 1 **React** takes control of DOM, **D3** as a utility library

Option 2 **D3** takes control of DOM

Option 3 **D3** and **React** share control of DOM

▶ <https://www.youtube.com/watch?v=zXBdNDnqV2Q>

▶ <https://www.youtube.com/watch?v=l-PZ7rQWs04>

Further Resources

Category	Name	Link	Description
Data Visualization	Stephen Few	http://www.stephen-few.com/blog/	Focused on business information & dashboards
Data Visualization	Edward Tufte	https://www.edwardtufte.com/tufte/	Professor emeritus statistics Yale
Data Visualization	Alberto Cairo	http://albertocairo.com/	Chair visual journalism University of Miami
D3.js	Shirley Wu	https://shirleywu.studio/	Freelance data artist & tutor
D3.js	Amelia Wattenberger	https://wattenberger.com/	Engineer at Github
D3.js	Mike Bostock	https://observablehq.com/@mbostock	D3 creator

Appendix

SVG Elements Cheatsheet

svg

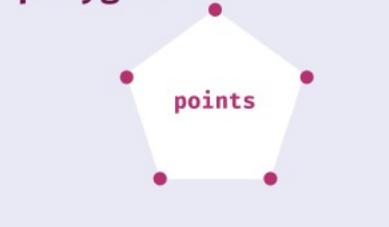
The Grand Poobah.

Use this to surround all other SVG elements or to create a new coordinate system.

rect



polygon



defs

The definitions element.

Use this to store elements to be used elsewhere. For example:

line



path

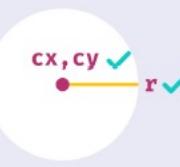


clipPath

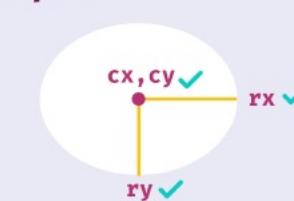
Store in defs. `reference: url(#id)`

Used to clip other elements outside of its children elements' shape.

circle



ellipse



linearGradient, radialGradient

Store in defs. `reference: url(#id)`

Used to define a gradient, using:

stop

`offset`
✓`stop-color`
✓`stop-opacity`

g

A container to group other SVG elements.

Similar to an HTML <div>.

text

The only way to create text within SVG.

`x, y`
`dx`
`dy`
`rotate`
`textLength`
`lengthAdjust`

