

D3.JS

DATA DRIVEN DOCUMENTS.

@toonketels

WHAT IS THIS ABOUT

- Insert SVG into DOM
- Data joins
- Scales
- Axis & labels
- Axis & ticks
- Animations
- Update chart when data changes

SVG

SCALABLE VECTOR GRAPHICS

WHAT ARE SVG?

Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics that has support for interactivity and animation. The SVG specification is an open standard developed by the World Wide Web Consortium (W3C) since 1999.

[Wikipedia](#)

WHAT WE NEED TO KNOW

- XML based - "tags and attributes"
- optimized for images - better for visualization

SVG ELEMENTS

```
<svg></svg>  
<rect></rect>  
<path></path>  
<circle></circle>  
<text></text>
```

SVG ATTRIBUTES

```
<rect x="0" width="5" y="0" height="50"></rect>
```

```
<circle cx="100" cy="250" r="40"></circle>
```

SVG IN THE SOURCE

Hardcoded into the HTML source


```
<svg width="700" height="500">
  <rect x="0" width="5" y="0" height="50"></rect>
  <rect x="0" width="31" y="70" height="50"></rect>
  <rect x="0" width="86" y="140" height="50"></rect>
  <rect x="0" width="474" y="210" height="50"></rect>
  <rect x="0" width="308" y="280" height="50"></rect>
  <rect x="0" width="700" y="350" height="50"></rect>
  <rect x="0" width="630" y="420" height="50"></rect>
</svg>
```

GOT IT?

- Hardcode SVG into the DOM

INSERT SVG INTO DOM

How to use d3 to create SVG element?

HOW TO USE D3 TO CREATE SVG ELEMENT?

```
var canvas_d = {width: 700, height: 500},  
  
canvas = d3.select('#canvas').append('svg')  
    .attr('width', canvas_d.width)  
    .attr('height', canvas_d.height);
```

HOW TO USE D3 TO CREATE SVG ELEMENT?

```
<div id="canvas">  
  <svg width="700" height="500">  
    </svg>  
</div>
```

GOT IT?

- Search for existing DOM node
- Use `.append('svg')`
- Set attributes with `.attr('name', 'value')`

INSERT ELEMENTS INTO SVG

How to display content in SVG?

ATTEMPT ONE

ITERATE OVER EACH ITEM IN THE DATA AND APPEND A `RECT` ELEMENT

SOURCE WE WANT TO GENERATE

```
<rect x="0" width="5" y="0" height="50"></rect>  
<rect x="0" width="31" y="70" height="50"></rect>  
<rect x="0" width="86" y="140" height="50"></rect>  
<rect x="0" width="474" y="210" height="50"></rect>  
<rect x="0" width="308" y="280" height="50"></rect>  
<rect x="0" width="700" y="350" height="50"></rect>  
<rect x="0" width="630" y="420" height="50"></rect>
```

WHAT CHANGES?

- width
- y

WHAT MATH DO WE NEED?

```
y = 70 * i
```

```
width = canvas_d.width * (d/max_overall)
```

```
50          // our value  
100         // maximum value  
700 px      // width  
  
=> width = 700 px * (50 / 100)    // 350
```

USE D3 TO CALCULATE THE OVERALL MAX

```
var data = [26009896, 179804755, 494478797, 2718505888, 17656864  
var max_overall = d3.max(data);
```

ALL TOGETHER NOW

```
data.map(function(d, i) {  
  
    canvas.append('rect')  
      .attr('x', 0 )  
      .attr('width', canvas_d.width * (d/max_overall))  
      .attr('y', 70 * i)  
      .attr('height', 50)  
  
});
```

THE SAME, WRITTEN DIFFERENTLY

```
data.map(function(d, i) {  
  
  canvas.append('rect')  
    .attr('x', 0 )  
    .attr('width', (function(d, i) {  
      return canvas_d.width * (d/max_overall)  
    })(d, i))  
    .attr('y', (function(d, i) {  
      return 70 * i;  
    })(d, i))  
    .attr('height', 50);  
  
});
```

END RESULT

```
<svg width="700" height="500">  
  <rect x="0" width="5" y="0" height="50"></rect>  
  <rect x="0" width="31" y="70" height="50"></rect>  
  <rect x="0" width="86" y="140" height="50"></rect>  
  <rect x="0" width="474" y="210" height="50"></rect>  
  <rect x="0" width="308" y="280" height="50"></rect>  
  <rect x="0" width="700" y="350" height="50"></rect>  
  <rect x="0" width="630" y="420" height="50"></rect>  
</svg>
```

ATTEMPT TWO

DO IT THE D3 WAY WITH DATA-JOINS

HOW D3 WANTS US TO DO IT

```
canvas.selectAll('rect')  
  .data( data )  
.enter().append('rect')  
  .attr('x', 0 )  
  .attr('width', function(d, i){  
    return canvas_d.width * (d/max_overall);  
  })  
  .attr('y', function(d, i) {  
    return 70 * i;  
  })  
  .attr('height', 50)
```

ONE BY ONE

```
canvas.selectAll('rect')
```

```
.data( data )
```

```
.enter()
```

```
.append('rect')  
  .attr('x', 0 )  
  .attr('width', function(d, i){  
    return canvas_d.width * (d/max_overall);  
  })  
  .attr('y', function(d, i) {  
    return 70 * i;  
  })  
  .attr('height', 50)
```

HOW D3 WANTS US TO DO IT

```
canvas.selectAll('rect')  
  .data( data )  
.enter().append('rect')  
  .attr('x', 0)  
  .attr('width', function(d, i){  
    return canvas_d.width * (d/max_overall)  
  })  
  .attr('y', function(d, i) { return 70 * i })  
  .attr('height', 50);
```

GOT IT?

- Select elements with `.selectAll('rect')`
- Create data join with `.data(data)`
- Get the enter subcollection via `.enter()`
- Append the elements via `.append('rect')`
- Set attributes with `.attr('name', 'value')`

WHAT WE HAVE SO FAR

- Insert SVG container (canvas)
- Create new elements via data binding

DITCH THE MATH, PART 1

*I hate the math in our code, can't we use
scales for that?*

A SCALE CONVERT SOME INPUT INTO SOME OUTPUT

```
attr('width', function(d, i){ return chart_d.width * (d/max_overall) })
```

```
width: data => pixels
```

```
var width = scaleFunc(179804755) // 31
```


INPUT DOMAIN

```
var data = [26009896,  
            179804755,  
            494478797,  
2718505888,  
            1765686465,  
            4015692380,  
            3611612096];
```

OUTPUT

```
<rect x="0" width="5" y="0" height="50"></rect>  
<rect x="0" width="31" y="70" height="50"></rect>  
<rect x="0" width="86" y="140" height="50"></rect>  
<rect x="0" width="474" y="210" height="50"></rect>  
<rect x="0" width="308" y="280" height="50"></rect>  
<rect x="0" width="700" y="350" height="50"></rect>  
<rect x="0" width="630" y="420" height="50"></rect>
```

CONVERSION

26009896	=>	5
179804755	=>	31
494478797	=>	86
2718505888	=>	474
1765686465	=>	308
4015692380	=>	700
3611612096	=>	630

CONVERSION

0	=>	0
4015692380	=>	700

0	=>	0
overall_max	=>	chart_d.width

HOW TO CREATE A SCALE FUNCTION?

```
x = d3.scale.linear()  
  .domain([0, max_overall])  
  .range([0, chart_d.width]);
```

USE IT

```
chart.selectAll('rect')  
  .data( data )  
.enter().append('rect')  
  .attr('x', 0)  
  .attr('width', function(d, i){ return x(d) })  
  .attr('y', function(d, i) { return 60 * i })  
  .attr('height', 50);
```

IN SHORT

```
chart.selectAll('rect')  
  .data( data )  
.enter().append('rect')  
  .attr('x', 0)  
  .attr('width', x)  
  .attr('y', function(d, i) { return 60 * i })  
  .attr('height', 50);
```

GOT IT?

- Create a linear scale with `d3.scale.linear()`
- Set the input domain as the lowest/highest value
`.domain([0, max_overall])`
- Set the output range as the lowest/highest value
`.range([0, chart_d.width])`

WHAT WE HAVE SO FAR

- Insert SVG container (canvas)
- Create new elements via data binding
- Use linear scale to convert width

DITCH THE MATH, PART 2

I hate the math in our code, can't we use scales for that?

CONVERT INPUT TO OUTPUT

```
y: data-item => pixels
```

```
var y = someOtherScaleFunc(179804755) // 70
```

INPUT DOMAIN

```
var data = [26009896,  
            179804755,  
            494478797,  
2718505888,  
            1765686465,  
            4015692380,  
            3611612096];
```

OUTPUT

```
<rect x="0" width="5" y="0" height="50"></rect>  
<rect x="0" width="31" y="70" height="50"></rect>  
<rect x="0" width="86" y="140" height="50"></rect>  
<rect x="0" width="474" y="210" height="50"></rect>  
<rect x="0" width="308" y="280" height="50"></rect>  
<rect x="0" width="700" y="350" height="50"></rect>  
<rect x="0" width="630" y="420" height="50"></rect>
```

CONVERSION

26009896	=>	0
179804755	=>	70
494478797	=>	140
2718505888	=>	210
1765686465	=>	280
4015692380	=>	350
3611612096	=>	420

CONVERSION

26009896	=>	0
179804755	=>	?
494478797	=>	?
2718505888	=>	?
1765686465	=>	?
4015692380	=>	?
3611612096	=>	500 - 60

26009896	=>	0
179804755	=>	?
494478797	=>	?
2718505888	=>	?
1765686465	=>	?
4015692380	=>	?
3611612096	=>	height - rangeBand

HOW TO CREATE A SCALE FUNCTION?

```
y = d3.scale.ordinal()  
  .domain(data)  
  .rangeBands([0, chart_d.height]);
```

USE IT

```
chart.selectAll('rect')  
  .data( data )  
.enter().append('rect')  
  .attr('x', 0)  
  .attr('width', x)  
  .attr('y', y)  
  .attr('height', 50);
```

FOR THE HEIGHT

```
chart.selectAll('rect')  
  .data( data )  
.enter().append('rect')  
  .attr('x', 0)  
  .attr('width', x)  
  .attr('y', y)  
  .attr('height', y.rangeBand);
```

GOT IT?

- Create an ordinal scale with `d3.scale.ordinal()` for individual items
- Pass all the values as input `domain.domain(data)`
- Use `rangeBands` as output `.rangeBands([0, chart_d.height])`
- Use `y.rangeBand` to get the height of a bar

WHAT WE HAVE SO FAR

- Insert SVG container (canvas)
- Create new elements via data binding
- Use linear scale to convert width
- Use ordinal scale to convert Y value

LABELS AND AXIS

How do we create labels?

LABELS ARE AUTOMATICALLY CREATED BY THE AXIS.

CREATING AXIS

CREATING AXIS IS A TWO STEP PROCESS

- axis creator function
- draw axis

AXIS CREATOR FUNCTION

```
axis_y_f = d3.svg.axis()  
    .scale(y)  
    .orient('left')
```

DRAW IT

```
axis_y = chart.append('g')  
        .attr('class', 'axis y')  
        .call(axis_y_f);
```

MAKE IT PRETTIER

```
axis_y_f = d3.svg.axis()  
    .scale(y)  
    .orient('left')  
    .tickPadding(30)  
    .tickSize(0, 0, 0);
```

GOT IT?

- First we create an axis with `d3.svg.axis()`
- We use `toScale` so it knows what to draw
- We can set other attributes to "tune" its display
- Finally, draw it by appending group
`chart.append('g')`
- And call the creator function `.call(axis_y_f)`

WHAT WE HAVE SO FAR

- Insert SVG container (canvas)
- Create new elements via data binding
- Use linear scale to convert width
- Use ordinal scale to convert Y value
- Display labels through axis

TICKS

How do we create those vertical lines?

TICKS ARE AUTOMATICALLY CREATED BY THE AXIS.

SAME STORY

```
axis_x_f = d3.svg.axis()  
    .scale(x)  
    .orient('top')
```

```
    .tickPadding(10)  
    .ticks(5)  
    .tickSize(chart_d.height, 0, 0)
```

```
axis_x = chart.append('g')  
    .attr('class', 'axis x')  
    .attr('transform', 'translate('+ 0 +', '+ chart_d.height +')')  
    .call(axis_x_f);
```


ADD SUBTICKS

```
axis_x_f = d3.svg.axis()  
    .scale(x)  
    .orient('top')  
    .tickPadding(10)  
    .ticks(5)  
    .tickSize(chart_d.height, chart_d.height, 0)  
    .tickSubdivide(2)
```

GOT IT?

- We create axis just like before (create/draw)
- Just be sure we set a the tick size to charts height
`tickSize(chart_d.height, chart_d.height, 0)`

WHAT WE HAVE SO FAR

- Insert SVG container (canvas)
- Create new elements via data binding
- Use linear scale to convert width
- Use ordinal scale to convert Y value
- Display labels through axis
- Display ticks through axis

REAL LABELS

*Should we display what the values represent
as label?*

CHANGE THE DATA DATA SOURCE

```
var data = [{value: 26009896, name: "angular.js"},  
            {value: 179804755, name: "backbone.js"},  
            {value: 494478797, name: "batman.js"},  
            {value: 2718505888, name: "ember.js"},  
            {value: 1765686465, name: "knockout.js"},  
            {value: 4015692380, name: "sammy.js"},  
            {value: 3611612096, name: "spine.js"}];
```

UPDATE WITH DATA ACCESSOR FUNCTIONS

SAME FOR OUR LABELS

```
.domain(data)
```

```
.domain(data.map(function(d, i){ return d.name })))
```


GOT IT?

- To use real labels they need to be in the data source
- We use *data accessor functions* to tell d3 what attributes to use

WHAT WE HAVE SO FAR

- Insert SVG container (canvas)
- Create new elements via data binding
- Use linear scale to convert width
- Use ordinal scale to convert Y value
- Display labels through axis
- Display ticks through axis

ANIMATION

Why is stuff not moving?

ANIMATE BARS' WIDTH

```
chart.selectAll('rect')  
  .data( data )  
.enter().append('rect')  
  .attr('x', 0)  
  .attr('width', 0)  
  .attr('y', function(d, i) { return y(d.name) })  
  .attr('height', y.rangeBand)  
  .style('fill', '#333')  
.transition()  
  .duration(600)  
  .attr('width', function(d, i) { return x(d.value) })
```

CHAIN A COLOR TRANSITION

```
chart.selectAll('rect')  
  .data( data )  
.enter().append('rect')  
  .attr('x', 0)  
  .attr('width', 0)  
  .attr('y', function(d, i) { return y(d.name) })  
  .attr('height', y.rangeBand)  
  .style('fill', '#333')  
.transition()  
  .duration(600)  
  .attr('width', function(d, i) { return x(d.value) })  
.transition()  
  .duration(400)  
  .style('fill', 'black')
```

ANIMATION CAN BE DONE ON ABOUT ANYTHING AXIS TOO...

```
axis_y = chart.append('g')  
    .attr('class', 'axis y')  
    .transition()  
    .delay(800)  
    .duration(800)  
    .call(axis_y_f);
```


GOT IT?

- To animate use `.transition()`
- Transitions can have `.delay(500)` and `duration(500)`
- Change a attribute/style... before and after transition, d3 will do the rest

WHAT WE HAVE SO FAR

- Insert SVG container (canvas)
- Create new elements via data binding
- Use linear scale to convert width
- Use ordinal scale to convert Y value
- Display labels through axis
- Display ticks through axis
- Animate the bars

TIMELY UPDATES

How could we easily update our numbers?

**OUR DATA CHANGES EVERY X SECONDS... WE BETTER UPDATE
THE CHART.**

We've changed the data source so that:

- drawChart gets called once, the first time
- every x seconds, updateChart gets called with new data

NOTHING CHANGES IN DRAWCHART

```
bars = chart.selectAll('rect')
    .data( data )
    .enter().append('rect')
    .attr('x', 0)
    .attr('width', 0)
    .attr('y', function(d, i) { return y(d.name) })
    .attr('height', y.rangeBand)
    .style('fill', '#333');

bars
    .transition()
    .duration(600)
    .attr('width', function(d, i) { return x(d.value) })
    .transition()
    .duration(400)
    .style('fill', 'black')
```

UPDATECHART..

```
function updateChart(data) {  
  
  bars  
    .data(data)  
    .transition()  
    .duration(600)  
    .attr('width', function(d, i) { return x(d.value) });  
  
}
```

GOT IT?

- To update the chart, rebind the data via `.data(updated_data)`
- To redraw, actually call a method that will update the display like `.attr('width' , ...)`

WHAT WE HAVE SO FAR

- Insert SVG container (canvas)
- Create new elements via data binding
- Use linear scale to convert width
- Use ordinal scale to convert Y value
- Display labels through axis
- Display ticks through axis
- Animate the bars
- Update the bars when data changes

UPDATE THE ENTIRE AXIS

What's with all that unused whitespace?

Ask ourself: "What would need to change to update our axis?"

- axis creator function
- scale function
- scale's input domain

To update the axis, just *update the scale* and call something on the axis again.

```
// Set the current overall max;  
max_overall = d3.max(data, function(d, i) { return d.value });
```

```
// Update the x scale  
x.domain([0, max_overall])
```

```
// Update x axis  
axis_x  
  .transition()  
    .duration(600)  
    .call(axis_x_f);
```

```
// Update the bars with new data  
bars  
  .data(data)  
  .transition()  
    .delay(800)  
    .duration(600)  
    .attr('width', function(d, i) { return x(d.value) });
```

GOT IT?

- We can update anything
- Always ask: "For it to update, what needs to change?"
- Change that value and redraw

WHAT WE HAVE SO FAR

- Insert SVG container (canvas)
- Create new elements via data binding
- Use linear scale to convert width
- Use ordinal scale to convert Y value
- Display labels through axis
- Display ticks through axis
- Animate the bars
- Update the bars when data changes
- Update the entire chart when data changes

THAT'S IT.

THANKS

TOON KETELS

@toonketels