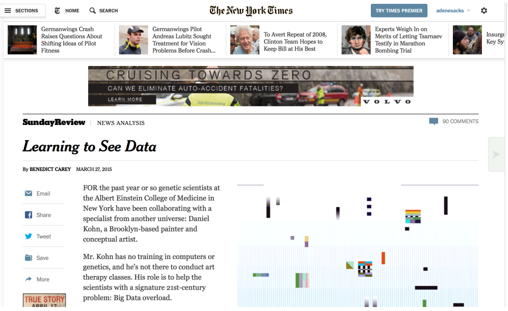
Data Visualization

A Broad and Burgeoning Field

- Graduate course at Berkeley
  - CS294-10
- Many mature commercial products & startups
  - Tableau, Qliktech, etc.
- Popular open-source packages
  - D3, Processing
- Academic Conferences
  - InfoVis, EuroVis, VAST…
- Popular coffee-table books
  - Edward Tufte, Stephen Few



Germanwings Crash Raises Questions About Shifting Ideas of Pilot Fitness

Germanwings Pilot Andreas Lubitz Sought Treatment for Vision Problems Before Crash...

To Avert Repeat of 2008, Clinton Team Hopes to Keep Bill at His Best

Experts Weigh In on Merits of Letting Tsarnaev Testify in Marathon Bombing Trial

Insurge Key Sy

SundayReview | NEWS ANALYSIS

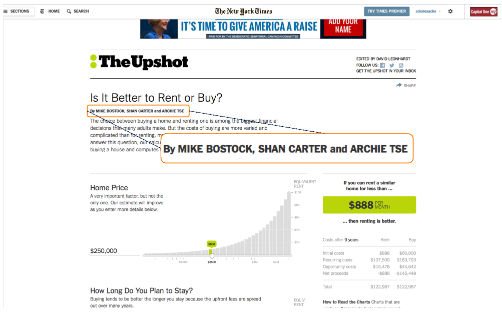
90 COMMENTS

Learning to See Data

By BENEDICT CAREY MARCH 27, 2015

FOR the past year or so genetic scientists at the Albert Einstein College of Medicine in New York have been collaborating with a specialist from another universe: Daniel Kohn, a Brooklyn-based painter and conceptual artist.

Mr. Kohn has no training in computers or genetics, and he's not there to conduct art therapy classes. His role is to help the scientists with a signature 21st-century problem: Big Data overload.



TheUpshot

EDITED BY DAVID LEONHARDT
FOLLOW US:
GET THE UPSHOT IN YOUR INBOX

SHARE

Is It Better to Rent or Buy?

By MIKE BOSTOCK, SHAN CARTER and ARCHIE TSE

The choice between buying a home and renting one is among the biggest financial decisions that many adults make. But the costs of buying are more varied and complicated than for renting, m… answer this question, our calcu… buying a house and computes

By MIKE BOSTOCK, SHAN CARTER and ARCHIE TSE

Home Price

A very important factor, but not the only one. Our estimate will improve as you enter more details below.

If you can rent a similar home for less than …

$888 PER MONTH

… then renting is better.

Costs after 9 years	Rent	Buy
Initial costs	$888	$60,000
Recurring costs	$107,509	$163,793
Opportunity costs	$15,478	$44,642
Net proceeds	-$888	-$145,448
Total	$122,987	$122,987

How Long Do You Plan to Stay?

Buying tends to be better the longer you stay because the upfront fees are spread out over many years.

EQUIV. RENT

How to Read the Charts Charts that are

Our Slice in CS186

- We'll focus more on *how* than *what*
- Particularly: *specifying* data visualizations

Key Observation #1

Q: What is visualization?
A: a *mapping* from *data to graphical objects*

Key Observation #2

Data mappings *are* queries.

- Vis languages evolved from imperative to declarative
  - With the attendant benefits
    - Concise, fit to the task, easy to analyze, resuable
- Much like databases!
  - But more recently and (hence) quickly

We'll look at 2 flavors of languages

- D3.js – an algebraic approach in JavaScript
  - Wildly popular with JS programmers
- Vega.js – a more declarative, restricted approach
  - Higher level but less expressive
  - Typical of a number of other languages

D3: Data-Driven Documents

Q: What is visualization?
A: a *mapping* from *data to graphical objects*

Web standards define graphical objects in browser
- Declarative languages: HTML (DOM), SVG

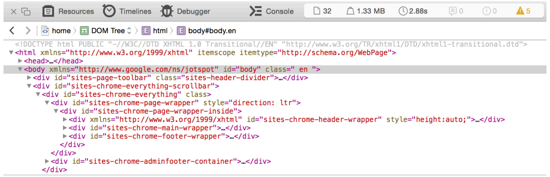
So let's map from data to those standards

D3: Key ideas

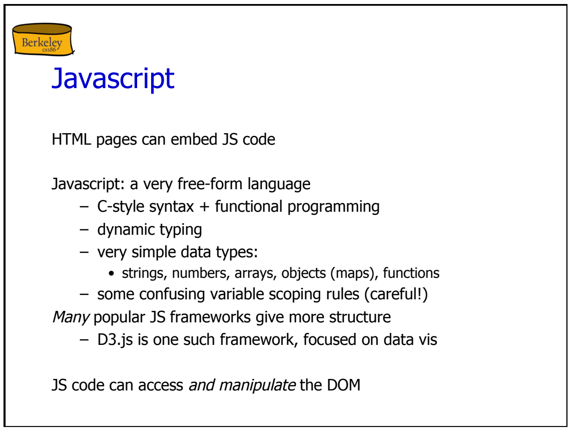
Key ideas:
- declarative *selections* of objects on a web page
- mapping as a *join* of data with page elements
- *outer join* to model evolution of data and page

First some HTML basics: DOM

Document Object Model (DOM)
- A web page is a *tree* of elements (a.k.a. tags)
- Each element has a name and optional attributes:
  - <foo> (tag name)
  - <any id="foo"> (id)
  - <any class="foo”> (class)
  - ...
  - <any foo="bar"> (custom attribute)
- Open your browser's Web Inspector and have a look!



```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
▼<html xmlns="http://www.w3.org/1999/xhtml" itemscope itemtype="http://schema.org/WebPage">
  ▶<head>…</head>
  ▼<body xmlns="http://www.google.com/ns/jotspot" id="body" class=" en ">
    ▶<div id="sites-page-toolbar" class="sites-header-divider">…</div>
    ▼<div id="sites-chrome-everything-scrollbar">
      ▼<div id="sites-chrome-everything" class>
        ▼<div id="sites-chrome-page-wrapper" style="direction: ltr">
          ▼<div id="sites-chrome-page-wrapper-inside">
            ▶<div xmlns="http://www.w3.org/1999/xhtml" id="sites-chrome-header-wrapper" style="height:auto;">…</div>
            ▶<div id="sites-chrome-main-wrapper">…</div>
            ▶<div id="sites-chrome-footer-wrapper">…</div>
           </div>
         </div>
        ▶<div id="sites-chrome-adminfooter-container">…</div>
       </div>
```



Javascript

HTML pages can embed JS code

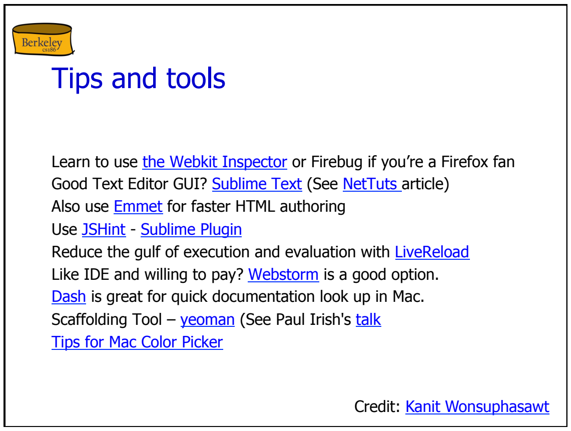
Javascript: a very free-form language

- C-style syntax + functional programming
- dynamic typing
- very simple data types:
  - strings, numbers, arrays, objects (maps), functions
- some confusing variable scoping rules (careful!)

Many popular JS frameworks give more structure

- D3.js is one such framework, focused on data vis

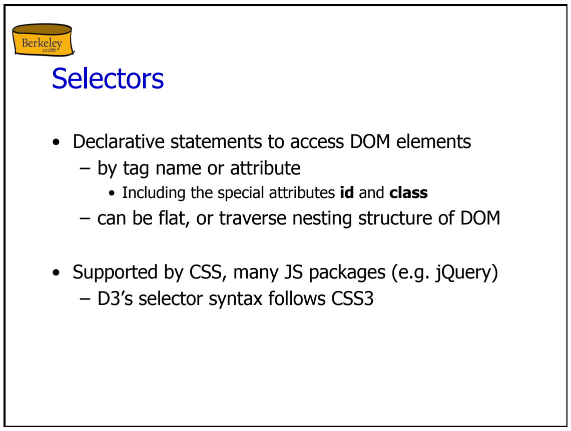
JS code can access *and manipulate* the DOM



Tips and tools

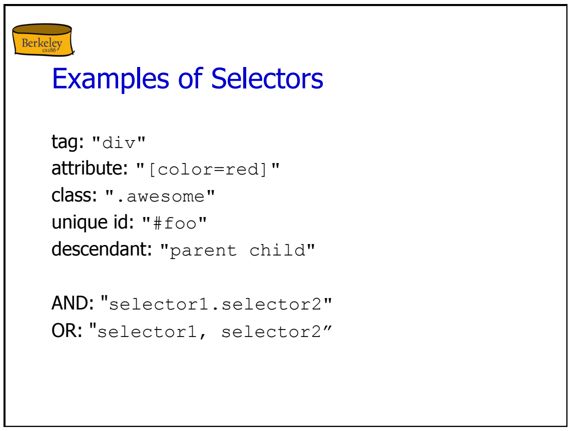
Learn to use the Webkit Inspector or Firebug if you're a Firefox fan
Good Text Editor GUI? Sublime Text (See NetTuts article)
Also use Emmet for faster HTML authoring
Use JSHint - Sublime Plugin
Reduce the gulf of execution and evaluation with LiveReload
Like IDE and willing to pay? Webstorm is a good option.
Dash is great for quick documentation look up in Mac.
Scaffolding Tool – yeoman (See Paul Irish's talk
Tips for Mac Color Picker

Credit: Kanit Wonsuphasawt



Selectors

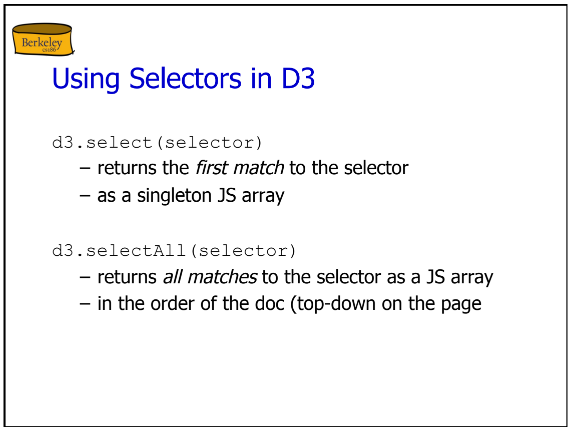
- Declarative statements to access DOM elements
  - by tag name or attribute
    - Including the special attributes **id** and **class**
  - can be flat, or traverse nesting structure of DOM
- Supported by CSS, many JS packages (e.g. jQuery)
  - D3's selector syntax follows CSS3



Examples of Selectors

tag: `"div"`
attribute: `"[color=red]"`
class: `".awesome"`
unique id: `"#foo"`
descendant: `"parent child"`

AND: `"selector1.selector2"`
OR: `"selector1, selector2”`



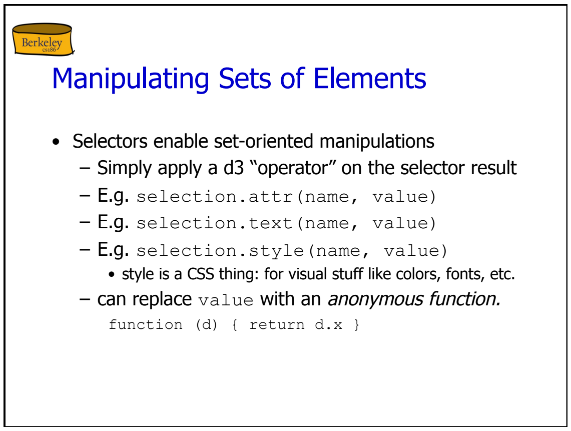
Using Selectors in D3

`d3.select(selector)`

- returns the *first match* to the selector
- as a singleton JS array

`d3.selectAll(selector)`

- returns *all matches* to the selector as a JS array
- in the order of the doc (top-down on the page



Manipulating Sets of Elements

- Selectors enable set-oriented manipulations
  - Simply apply a d3 "operator" on the selector result
  - E.g. `selection.attr(name, value)`
  - E.g. `selection.text(name, value)`
  - E.g. `selection.style(name, value)`
    - style is a CSS thing: for visual stuff like colors, fonts, etc.
  - can replace `value` with an *anonymous function.*

```
function (d) { return d.x }
```

Examples

- All examples are in the course github repo
  - https://github.com/cs186-spring15/course

Simple Example: Styling Text

```
<script type="text/javascript">
     var paras = d3.selectAll("p");
     paras.style("font-size", 12);
     paras.style("font-family","Courier");
     paras.style("color", "red");
</script>
```

Method Chaining for Ease of Reading

```
<script type="text/javascript">
     d3.selectAll("p")
          .style("font-size", 12)
          .style("font-family", "Courier")
          .style("color", "red");
</script>
```

Data-Driven Manipulation

- Note that function arguments are just data
  - e.g. *f(x)*, where *x* is an integer
- A set of data can lead to a set of function calls
  - In SQL: `SELECT f(x) from T`
- Note: in JS, functions can manipulate the DOM!
  - Change attributes
  - Change page content!

Data in D3

- Arrays of numbers

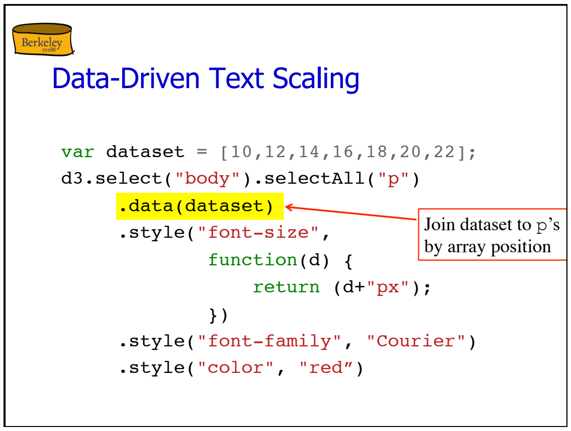
```
[2, 5, 6, 4, 2]
```

- Arrays of objects (i.e. tuples)

```
[{x:1, y:2},
 {x:2, y:5},
 {x:3, y:6},
 {x:4, y:4},
 {x:5, y:2}]
```

Data-Driven Text Scaling

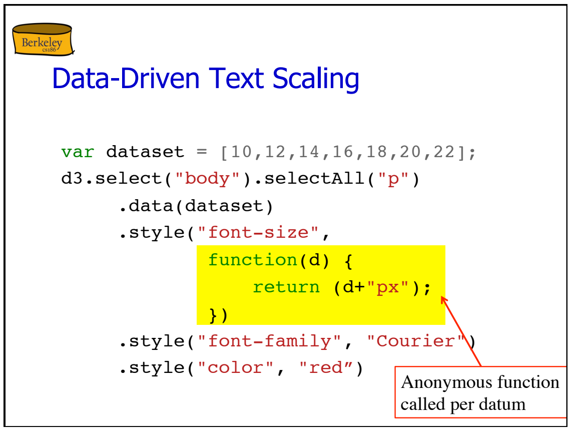
```
var dataset = [10,12,14,16,18,20,22];
d3.select("body").selectAll("p")
     .data(dataset)
     .style("font-size",
             function(d) {
                 return (d+"px");
             })
     .style("font-family", "Courier")
     .style("color", "red”)
```



Data-Driven Text Scaling

```
var dataset = [10,12,14,16,18,20,22];
d3.select("body").selectAll("p")
     .data(dataset)
     .style("font-size",
             function(d) {
                 return (d+"px");
             })
     .style("font-family", "Courier")
     .style("color", "red”)
```

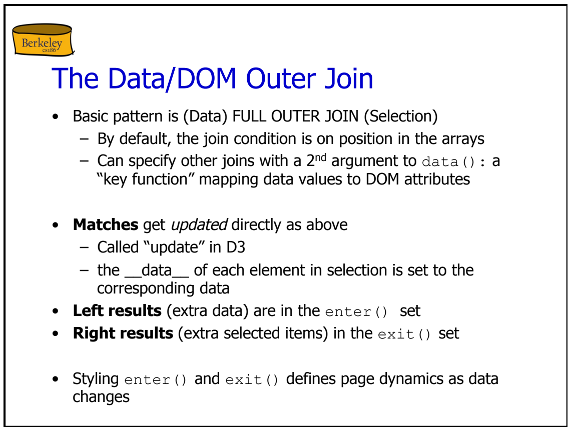
Join dataset to p’s by array position



Data-Driven Text Scaling

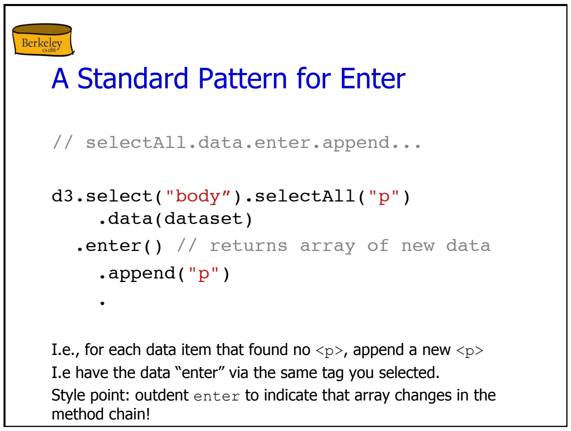
```
var dataset = [10,12,14,16,18,20,22];
d3.select("body").selectAll("p")
     .data(dataset)
     .style("font-size",
             function(d) {
                 return (d+"px");
             })
     .style("font-family", "Courier")
     .style("color", "red”)
```

Anonymous function called per datum



The Data/DOM Outer Join

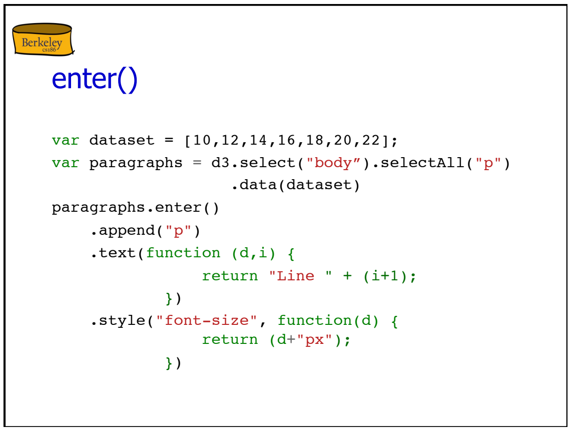
- Basic pattern is (Data) FULL OUTER JOIN (Selection)
  - By default, the join condition is on position in the arrays
  - Can specify other joins with a 2nd argument to `data()`: a “key function” mapping data values to DOM attributes
- **Matches** get *updated* directly as above
  - Called “update” in D3
  - the __data__ of each element in selection is set to the corresponding data
- **Left results** (extra data) are in the `enter()` set
- **Right results** (extra selected items) in the `exit()` set
- Styling `enter()` and `exit()` defines page dynamics as data changes



A Standard Pattern for Enter

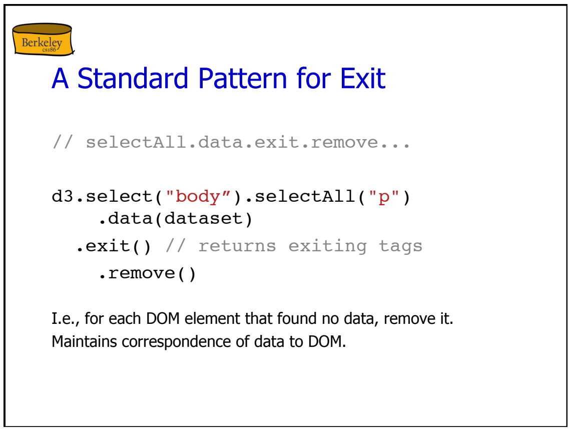
```
// selectAll.data.enter.append...

d3.select("body”).selectAll("p")
    .data(dataset)
  .enter() // returns array of new data
    .append("p")
    .
```

I.e., for each data item that found no `<p>`, append a new `<p>`
I.e have the data “enter” via the same tag you selected.
Style point: outdent `enter` to indicate that array changes in the method chain!



enter()

```
var dataset = [10,12,14,16,18,20,22];
var paragraphs = d3.select("body”).selectAll("p")
                   .data(dataset)
paragraphs.enter()
    .append("p")
    .text(function (d,i) {
            return "Line " + (i+1);
        })
    .style("font-size", function(d) {
            return (d+"px");
        })
```



A Standard Pattern for Exit

```
// selectAll.data.exit.remove...

d3.select("body”).selectAll("p")
    .data(dataset)
  .exit() // returns exiting tags
    .remove()
```

I.e., for each DOM element that found no data, remove it.
Maintains correspondence of data to DOM.

exit()

```
paragraphs.exit()
     .text(“null”)
     .style(“color”,”blue”)
     .style(“font-style”, “italic”)
```

An example of exit that doesn’t remove.
More typically might put some animation prior to exit.
E.g. Via d3’s `transition()` operator.

A Bar Chart Made of HTML Divs

```
d3.select("body").selectAll("div")
     .data(dataset)
.enter()
  .append("div")
  .style({"display": "inline-block",
           "background-color": "teal",
           "width": "20px",
           "margin-right": "2px"})
  .style("height",function(d) {
            return ((d*5)+"px");
            })
```

SVG

- A markup language for shapes
- Compiles down to efficient, smooth vector graphics

```
<svg width="700", height="100">
     <circle fill="red" cx="30" cy="50"
r="10"></circle>
</svg>
```

- Right in the DOM, so can be manipulated with D3!

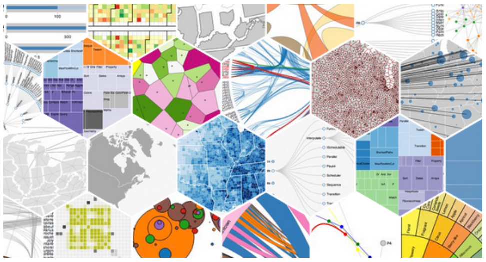
A Bar Chart Made out of SVG

```
var bars = d3.select("body").select("svg")
     .selectAll("rect").data(dataset)
var w = 500
var h = 150
bars.enter()
     .append("rect")
       .attr("width",20)
       .attr("fill", "teal")
       .attr("y", function(d){return h-(d*5)})
       .attr("x", function(d,i){return i*22;})
       .attr("height",function(d)
                      {return ((d*5));})
```

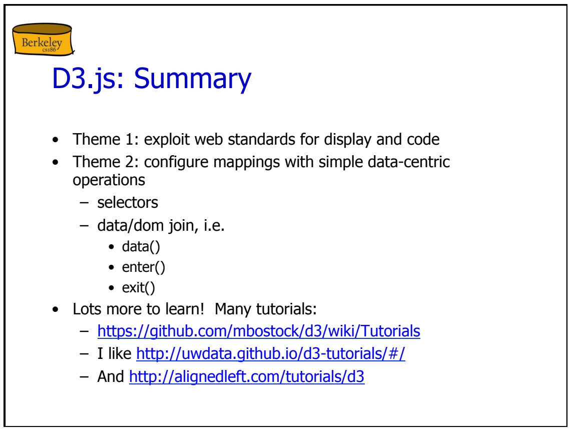
Circles Instead of Bars

```
// this returns a function that assigns colors to
indexes 0..19
var colors = d3.scale.category20()

bars.enter()
     .append("circle")
       .attr("fill", "teal")
       .attr("cx", function(d,i){return 30 + i*50;})
       .attr("cy", 50)
       .attr("r", function(d) {return ((d));})
       .attr("fill", function(d,i) {return colors(i);})
```

So Much More is Possible

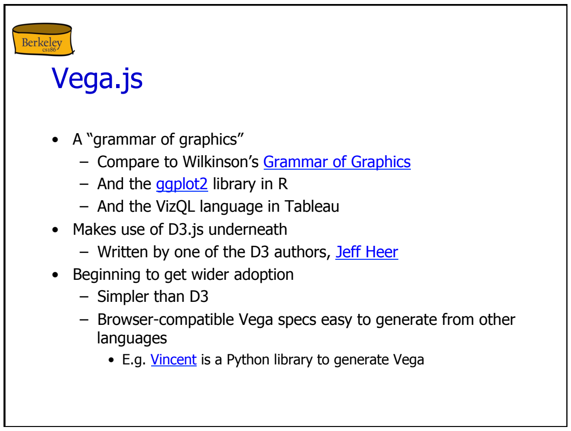


https://github.com/mbostock/d3/wiki/Gallery



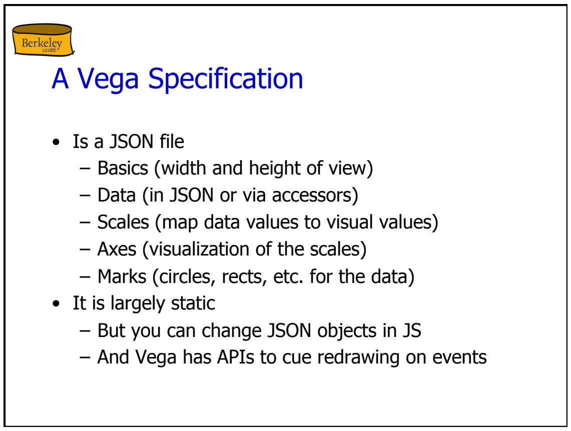
D3.js: Summary

- Theme 1: exploit web standards for display and code
- Theme 2: configure mappings with simple data-centric operations
  - selectors
  - data/dom join, i.e.
    - data()
    - enter()
    - exit()
- Lots more to learn! Many tutorials:
  - https://github.com/mbostock/d3/wiki/Tutorials
  - I like http://uwdata.github.io/d3-tutorials/#/
  - And http://alignedleft.com/tutorials/d3



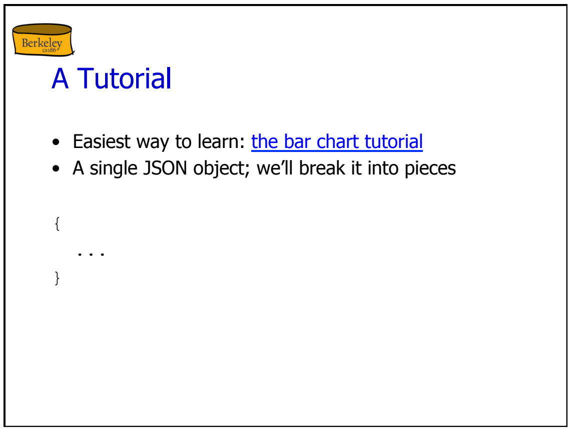
Vega.js

- A "grammar of graphics"
  - Compare to Wilkinson's Grammar of Graphics
  - And the ggplot2 library in R
  - And the VizQL language in Tableau
- Makes use of D3.js underneath
  - Written by one of the D3 authors, Jeff Heer
- Beginning to get wider adoption
  - Simpler than D3
  - Browser-compatible Vega specs easy to generate from other languages
    - E.g. Vincent is a Python library to generate Vega



A Vega Specification

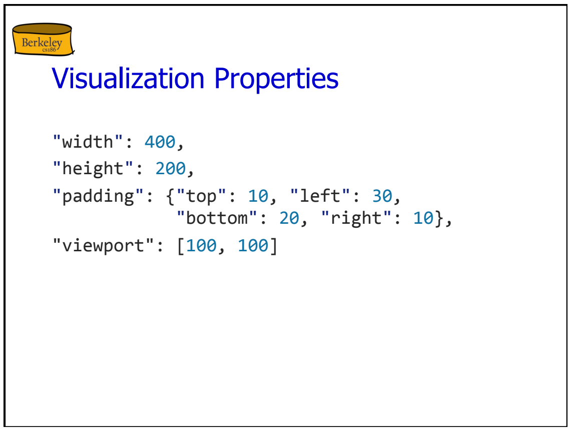
- Is a JSON file
  - Basics (width and height of view)
  - Data (in JSON or via accessors)
  - Scales (map data values to visual values)
  - Axes (visualization of the scales)
  - Marks (circles, rects, etc. for the data)
- It is largely static
  - But you can change JSON objects in JS
  - And Vega has APIs to cue redrawing on events



A Tutorial

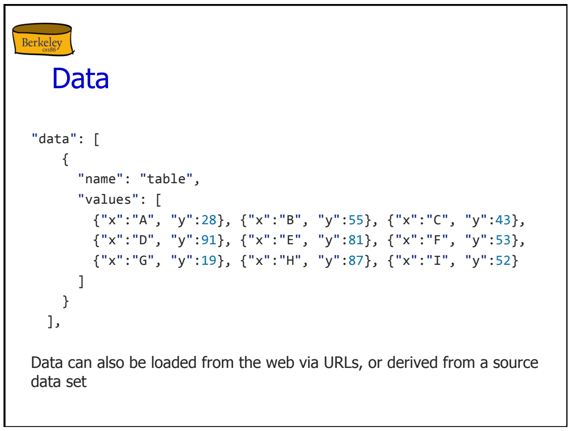
- Easiest way to learn: the bar chart tutorial
- A single JSON object; we'll break it into pieces

```
{
   ...
}
```



Visualization Properties

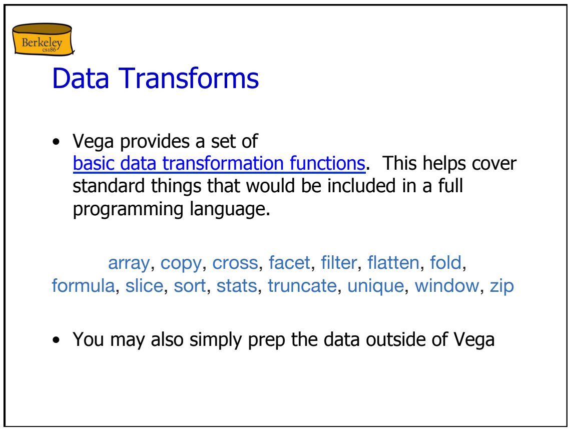
```
"width": 400,
"height": 200,
"padding": {"top": 10, "left": 30,
            "bottom": 20, "right": 10},
"viewport": [100, 100]
```



Data

```
"data": [
    {
      "name": "table",
      "values": [
        {"x":"A", "y":28}, {"x":"B", "y":55}, {"x":"C", "y":43},
        {"x":"D", "y":91}, {"x":"E", "y":81}, {"x":"F", "y":53},
        {"x":"G", "y":19}, {"x":"H", "y":87}, {"x":"I", "y":52}
      ]
    }
  ],
```

Data can also be loaded from the web via URLs, or derived from a source data set

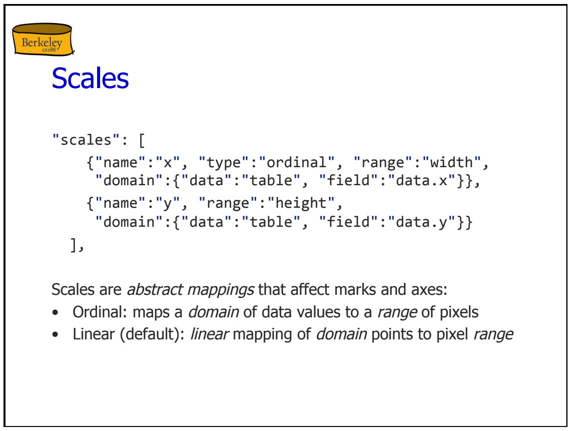


Data Transforms

- Vega provides a set of basic data transformation functions. This helps cover standard things that would be included in a full programming language.

array, copy, cross, facet, filter, flatten, fold, formula, slice, sort, stats, truncate, unique, window, zip

- You may also simply prep the data outside of Vega

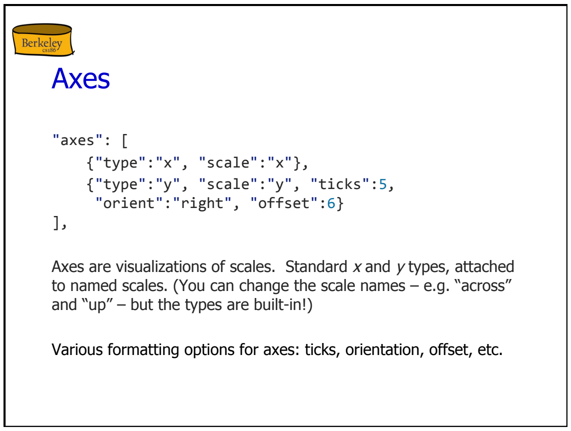


Scales

```
"scales": [
    {"name":"x", "type":"ordinal", "range":"width",
     "domain":{"data":"table", "field":"data.x"}},
    {"name":"y", "range":"height",
     "domain":{"data":"table", "field":"data.y"}}
  ],
```

Scales are *abstract mappings* that affect marks and axes:

- Ordinal: maps a *domain* of data values to a *range* of pixels
- Linear (default): *linear* mapping of *domain* points to pixel *range*

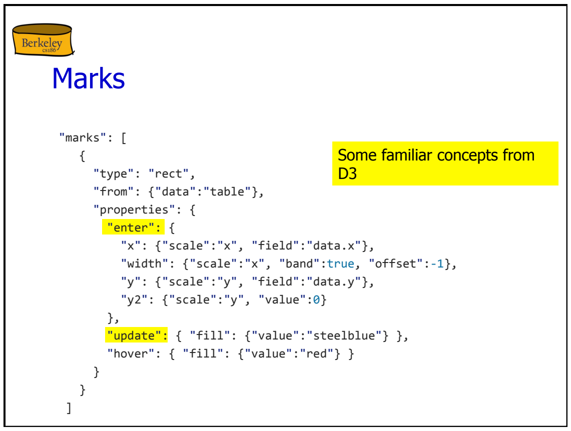


Axes

```
"axes": [
    {"type":"x", "scale":"x"},
    {"type":"y", "scale":"y", "ticks":5,
     "orient":"right", "offset":6}
],
```

Axes are visualizations of scales. Standard *x* and *y* types, attached to named scales. (You can change the scale names – e.g. "across" and "up" – but the types are built-in!)

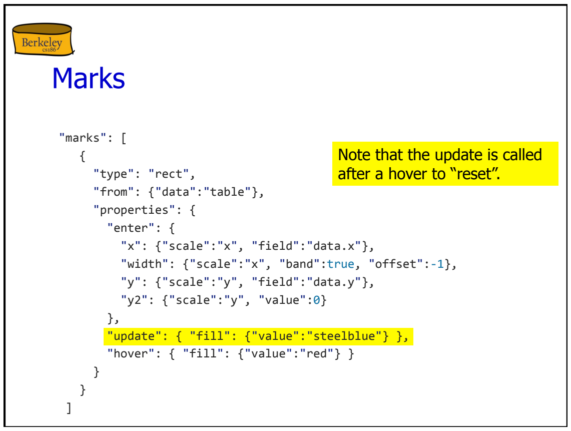
Various formatting options for axes: ticks, orientation, offset, etc.



Marks

```
"marks": [
   {
     "type": "rect",
     "from": {"data":"table"},
     "properties": {
       "enter": {
         "x": {"scale":"x", "field":"data.x"},
         "width": {"scale":"x", "band":true, "offset":-1},
         "y": {"scale":"y", "field":"data.y"},
         "y2": {"scale":"y", "value":0}
       },
       "update": { "fill": {"value":"steelblue"} },
       "hover": { "fill": {"value":"red"} }
     }
   }
 ]
```

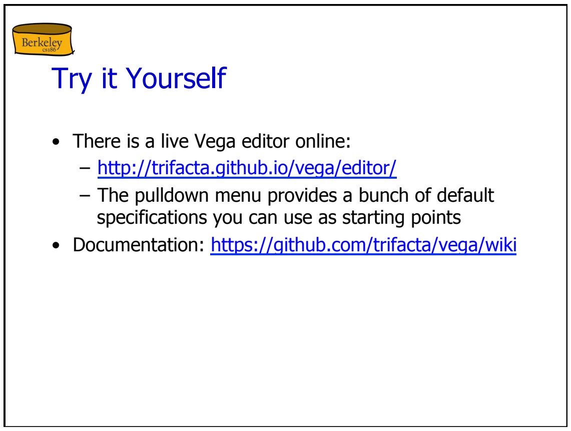
Some familiar concepts from D3



Marks

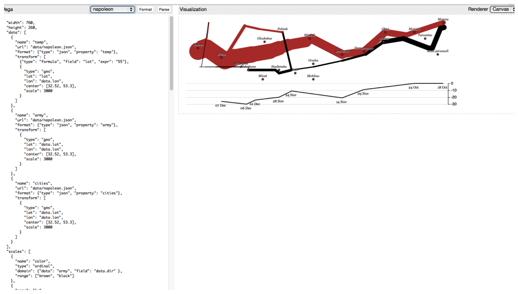
```
"marks": [
   {
     "type": "rect",
     "from": {"data":"table"},
     "properties": {
       "enter": {
         "x": {"scale":"x", "field":"data.x"},
         "width": {"scale":"x", "band":true, "offset":-1},
         "y": {"scale":"y", "field":"data.y"},
         "y2": {"scale":"y", "value":0}
       },
       "update": { "fill": {"value":"steelblue"} },
       "hover": { "fill": {"value":"red"} }
     }
   }
 ]
```

Note that the update is called after a hover to "reset".



Try it Yourself

- There is a live Vega editor online:
  - http://trifacta.github.io/vega/editor/
  - The pulldown menu provides a bunch of default specifications you can use as starting points
- Documentation: https://github.com/trifacta/vega/wiki



```
"width": 760,
"height": 260,
"data": [
  {
    "name": "temp",
    "url": "data/napoleon.json",
    "format": {"type": "json", "property": "temp"},
    "transform": [
      {"type": "formula", "field": "lat", "expr": "55"},
      {
        "type": "geo",
        "lat": "lat",
        "lon": "data.lon",
        "center": [32.52, 53.3],
        "scale": 3000
      }
    ]
  },
  {
    "name": "army",
    "url": "data/napoleon.json",
    "format": {"type": "json", "property": "army"},
    "transform": [
      {
        "type": "geo",
        "lat": "data.lat",
        "lon": "data.lon",
        "center": [32.52, 53.3],
        "scale": 3000
      }
    ]
  },
  {
    "name": "cities",
    "url": "data/napoleon.json",
    "format": {"type": "json", "property": "cities"},
    "transform": [
      {
        "type": "geo",
        "lat": "data.lat",
        "lon": "data.lon",
        "center": [32.52, 53.3],
        "scale": 3000
      }
    ]
  }
],
"scales": [
  {
    "name": "color",
    "type": "ordinal",
    "domain": {"data": "army", "field": "data.dir" },
    "range": ["brown", "black"]
  },
  {
```

Vega.js Summary

- Standard mapping of data:
  - scales and marks do most of the work
  - D3-like `enter`, `update`, `exit` and `hover`
    - But defined entirely on marks; no DOM involved
  - axes and other standard plotting properties available
- Surprisingly general
  - And much more flexible than, say, the Excel chart wizard!

Notes

- Many details were omitted here
  - Lots of visualization-specific issues
    - E.g. geo-data is quite complex and custom (projections)
    - E.g. graph layout is quite complex and custom
  - Some compositional issues in the languages
    - E.g. How to generate a host of related charts ("small multiples"), nest charts, etc.
  - Data access and transformation issues
    - Connecting to databases
    - Pushing big tasks down into the database
- Data prep for visualization is often a big effort
  - Many say 80% of the actual time

More Notes

- There are tons of tutorials, blogs, forums online
  - People love this stuff!
- Outside the JS ecosystem there is lots of other stuff
  - E.g. Tableau, Qlik, etc.
  - E.g. R+ggplot2, R+shiny
  - E.g. Python+{matplotlib, bokeh, or plotly}
  - E.g. Processing
  - Decent overview at http://www.fastcolabs.com/3029760/the-five-best-libraries-for-building-data-vizualizations
  - More comprehensive list at http://courses.cs.washington.edu/courses/cse512/15sp/resources.html

Takeaways

- Modern visualization specs based in declarative queries
  - Selections
  - Mapping data into attributes of marks
- Once again, bulk function invocation as join
  - As in HW4!
- Domain-Specific Languages win over time
  - Simple and composable
  - High-level/declarative, compile down to multiple targets
    - E.g. Vega compiles to SVG or Canvas
- Lots of quick change in this space
  - D3 is only 4 years old

Credits: resources I cribbed from

- The original D3 paper
- The D3 website
- The Vega website
- Jeff Heer's visualization course at Washington
- Kanit Wonsuphasawt's tutorials on D3 (part1 and part2) and fundamentals
- Scott Murray's D3 tutorial
- Mike Bostock's VizBI 2012 talk
- Ras Bodik's CS164 lecture on D3