GENERAL ASSEMBLY

INTERMEDIATE D3.JS

Sebastian Gutierrez
DashingD3js.com
@DashingD3js

May 31, 2014
General Assembly
Intermediate Level D3.js Weekend Bootcamp

SEBASTIAN GUTIERREZ

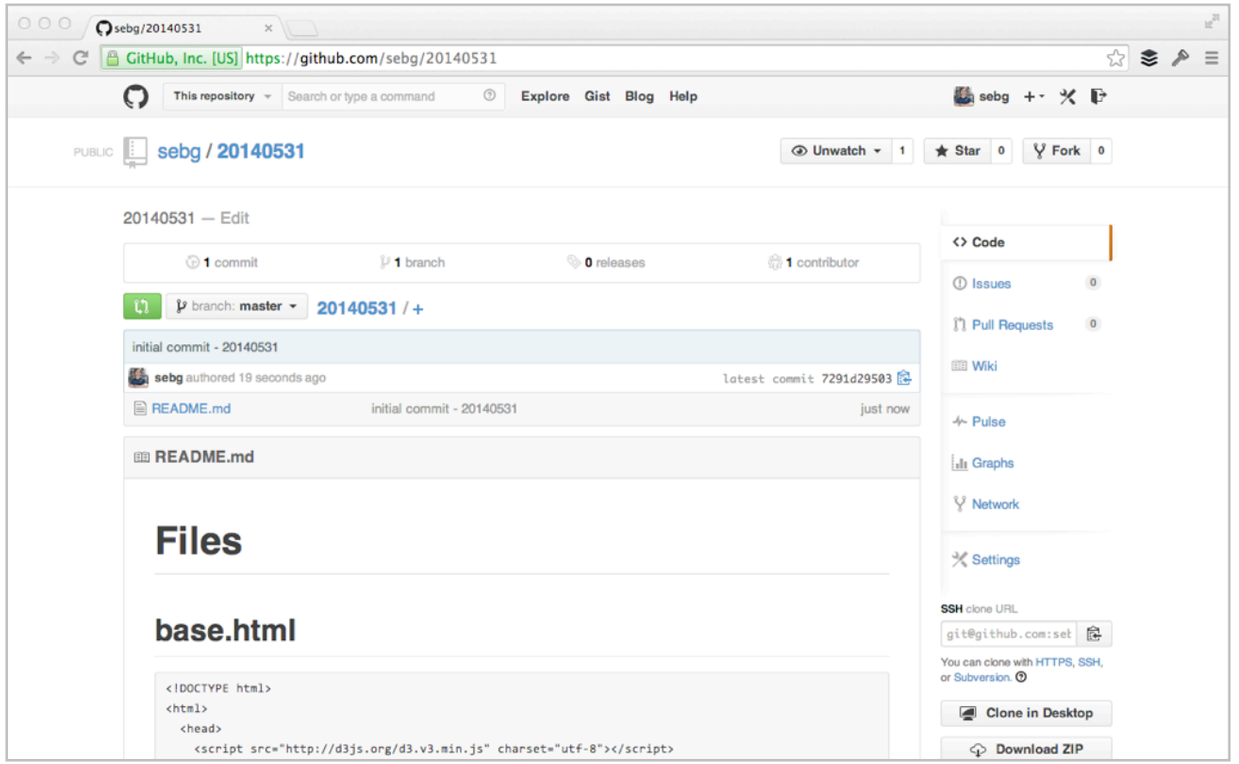
- Career in start-ups and Wall Street
- Specializes in Actionable Data Visualizations
- Runs DashingD3js.com (a D3.js tutorial website)
- Organizes NYC D3.js Meetup
- BS in Math, MA in Economics
- Teaches GA Intro to D3 Workshop

HELLO!

- Who you are
- What you do
- Learning goal for today

WEBSITE FOR LINKS, FILES & EXERCISE CODE

https://github.com/sebg/20140531



Today's Agenda

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

Sections

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

a) Four Seasons

b) BarChart.js

c) Pie Chart

d) Zoomable Circle

e) How D3.js Maps Work

Sections

- 1) The Goals
- 2) D3.js General Update Pattern
  - a) In-Depth Data Join
  - b) Basic General Update Pattern
  - c) Data Join With Key Function
  - d) Dynamic Data Join
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

Sections

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

a) DIY D3.js Framework

b) Functions Are Objects

c) Method Chaining, Getters & Setters

d) Simple Example

Sections

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

a) D3.js Layouts

b) D3.js Shape / Path Generator Function

c) D3.js Pie Layout

Sections

Sections

Sections

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

a) Key Take Aways

b) Next Steps

c) Resources

d) Class Closing / Q&A

SECTION 1
THE GOALS

Sections

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

a) Four Seasons

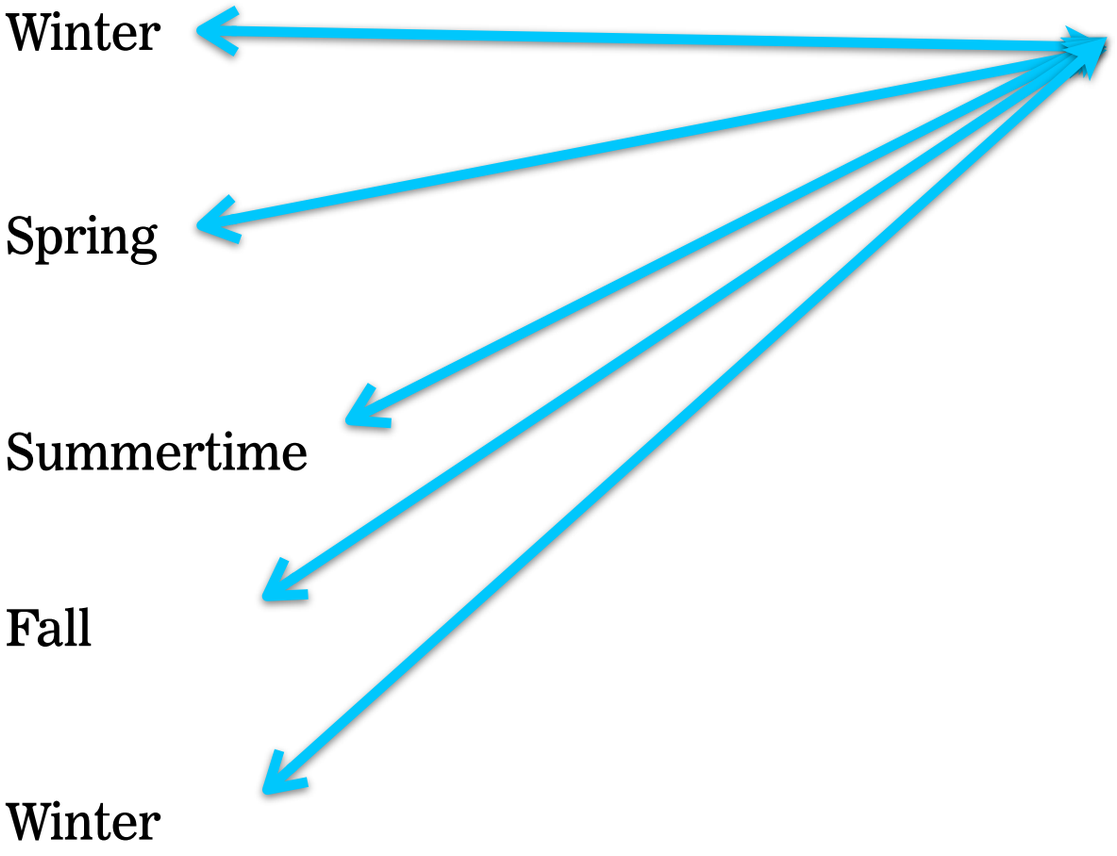
b) BarChart.js

c) Pie Chart

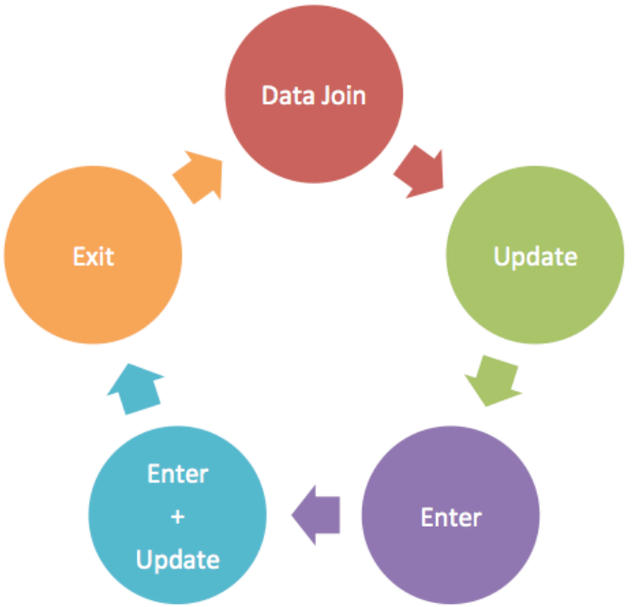
d) Zoomable Circle

e) How D3.js Maps Work

FOUR SEASONS



General Update Pattern Function



Sections

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

a) Four Seasons

b) BarChart.js

c) Pie Chart

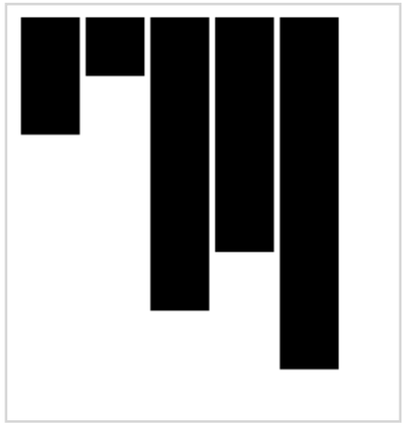
d) Zoomable Circle

e) How D3.js Maps Work

BARCHART.JS

```
var dataset = [2, 1, 5, 4, 6];

var myBarChart = barChart();

myBarChart
    .svgWidth(200)
    .svgHeight(200);

d3.select("body").append("div")
    .datum(dataset)
    .call(myBarChart);
```

```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<div>
      ▼<svg width="200" height="200">
          <rect x="0" y="0" width="38" height="66"></rect>
          <rect x="40" y="0" width="38" height="33"></rect>
          <rect x="80" y="0" width="38" height="165"></rect>
          <rect x="120" y="0" width="38" height="132"></rect>
          <rect x="160" y="0" width="38" height="198"></rect>
        </svg>
      </div>
      " "
  </body>
</html>
```



Sections

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

a) Four Seasons

b) BarChart.js

c) Pie Chart

d) Zoomable Circle

e) How D3.js Maps Work

PIE CHART

```
var data = [{amount: 1, color: "red"  },
            {amount: 2, color: "green"},
            {amount: 3, color: "blue" }];
```

Sections

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

a) Four Seasons

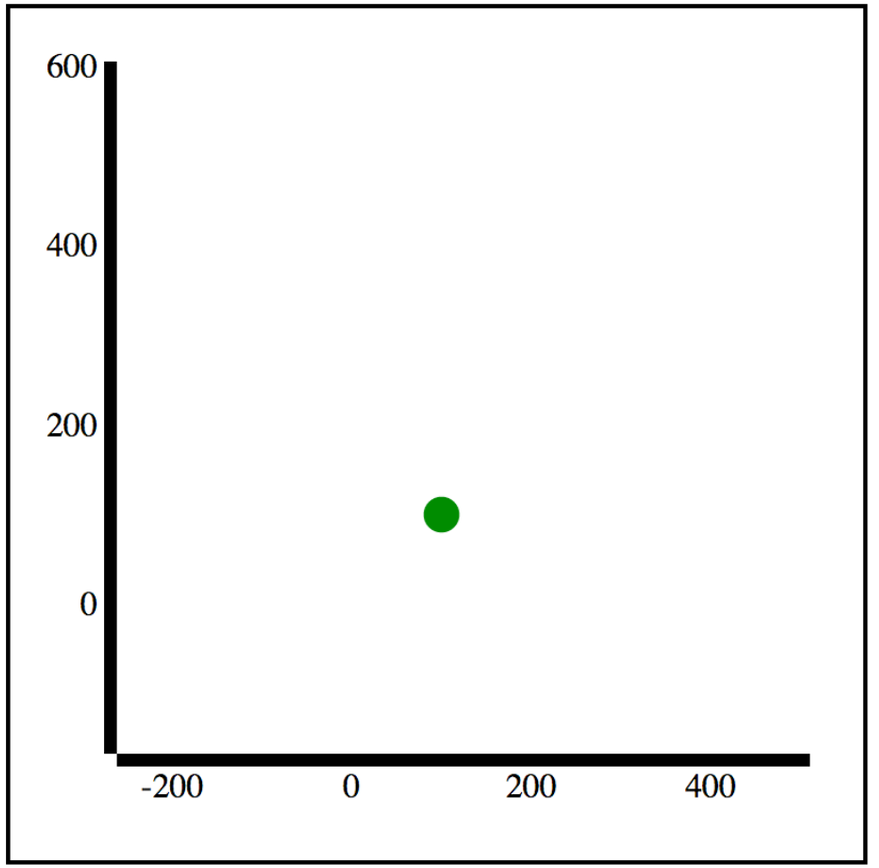
b) BarChart.js

c) Pie Chart

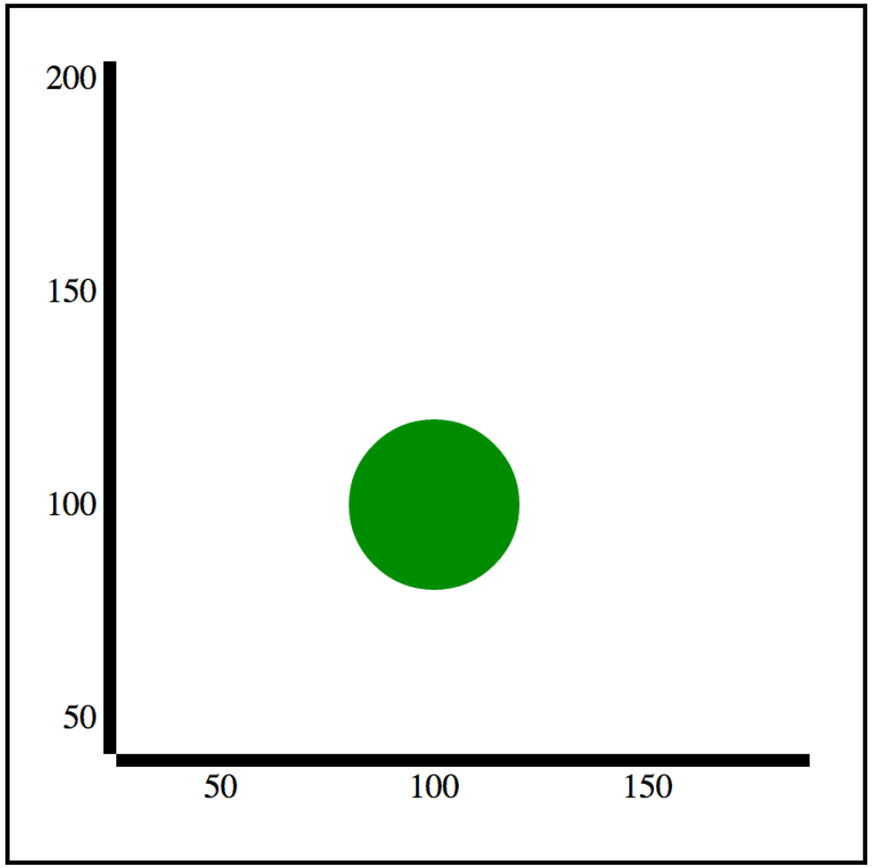
d) Zoomable Circle

e) How D3.js Maps Work

ZOOMABLE CIRCLE



Pan X Translate: 110.15190437698337
Pan Y Translate: 118.0139524438228
D3 Zoom Scale: 0.4209914366499001



Pan X Translate: -51
Pan Y Translate: -242
D3 Zoom Scale: 2

Sections

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

a) Four Seasons

b) BarChart.js

c) Pie Chart

d) Zoomable Circle

e) How D3.js Maps Work

HOW D3.JS MAPS WORK

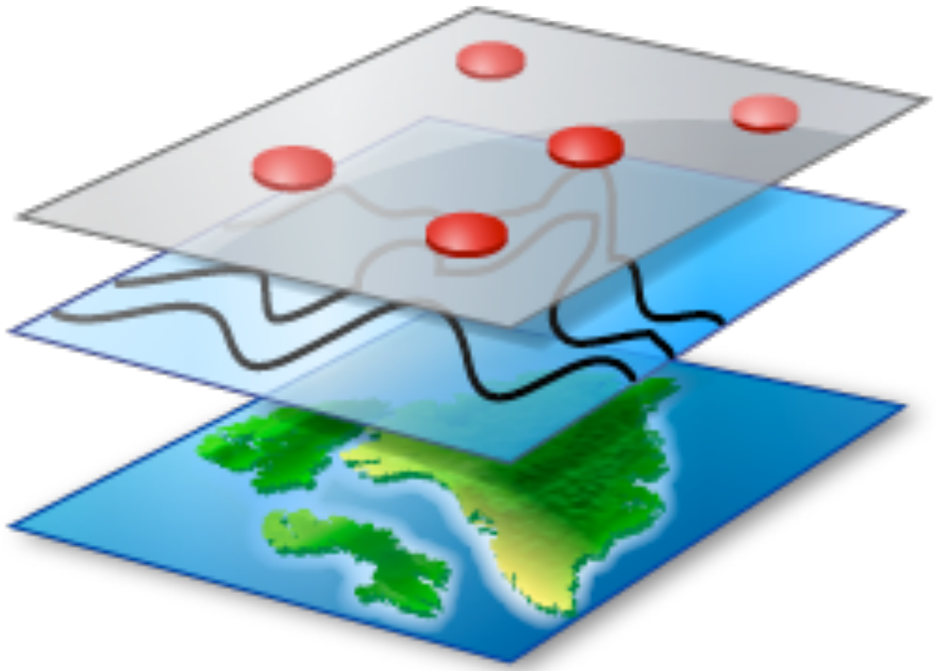


Image Source:
http://www.truenorthgeospatial.com

SECTION 2

D3.JS GENERAL UPDATE PATTERN

Sections

- 1) The Goals
- 2) D3.js General Update Pattern
- 3) D3.js Reusable Chart Pattern
- 4) D3.js Layouts
- 5) D3.js Behaviors
- 6) D3.js Mapping
- 7) Conclusion

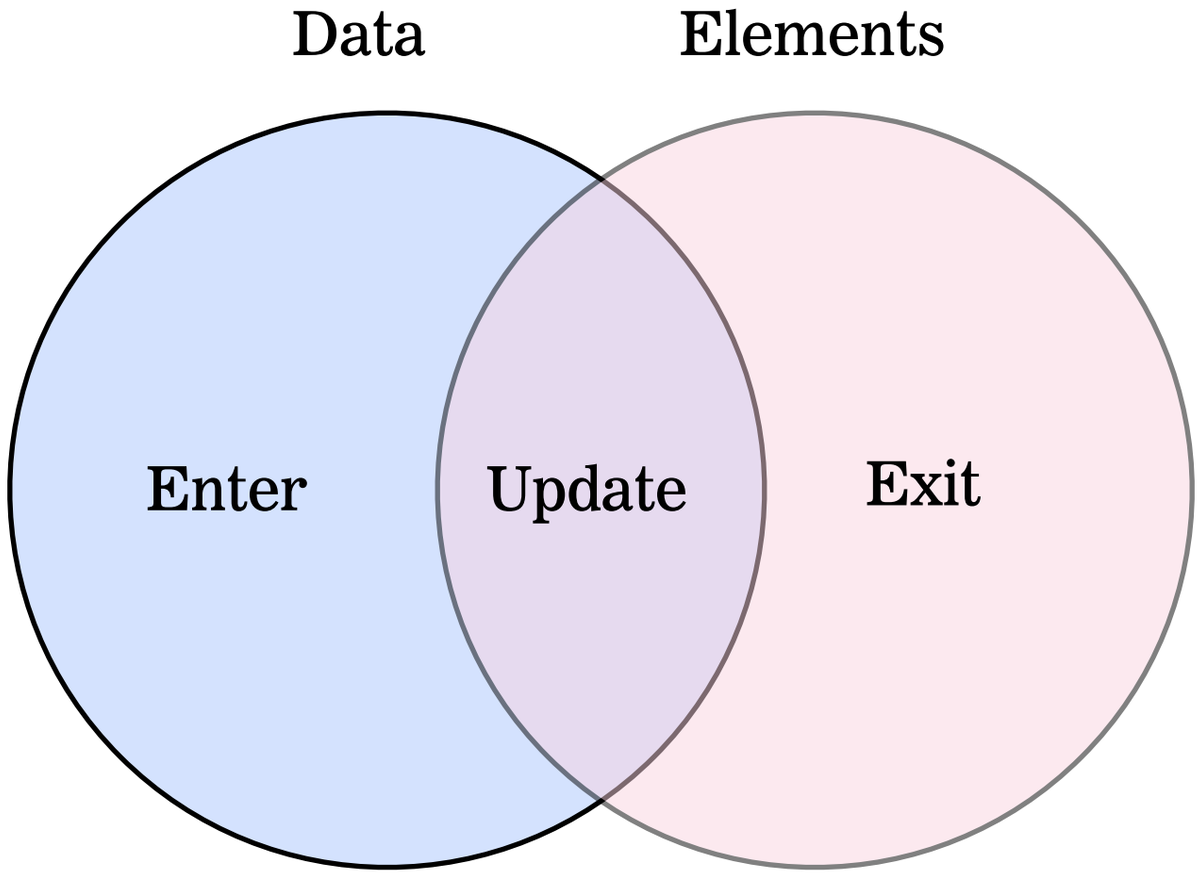
a) In-Depth Data Join

b) Basic General Update Pattern

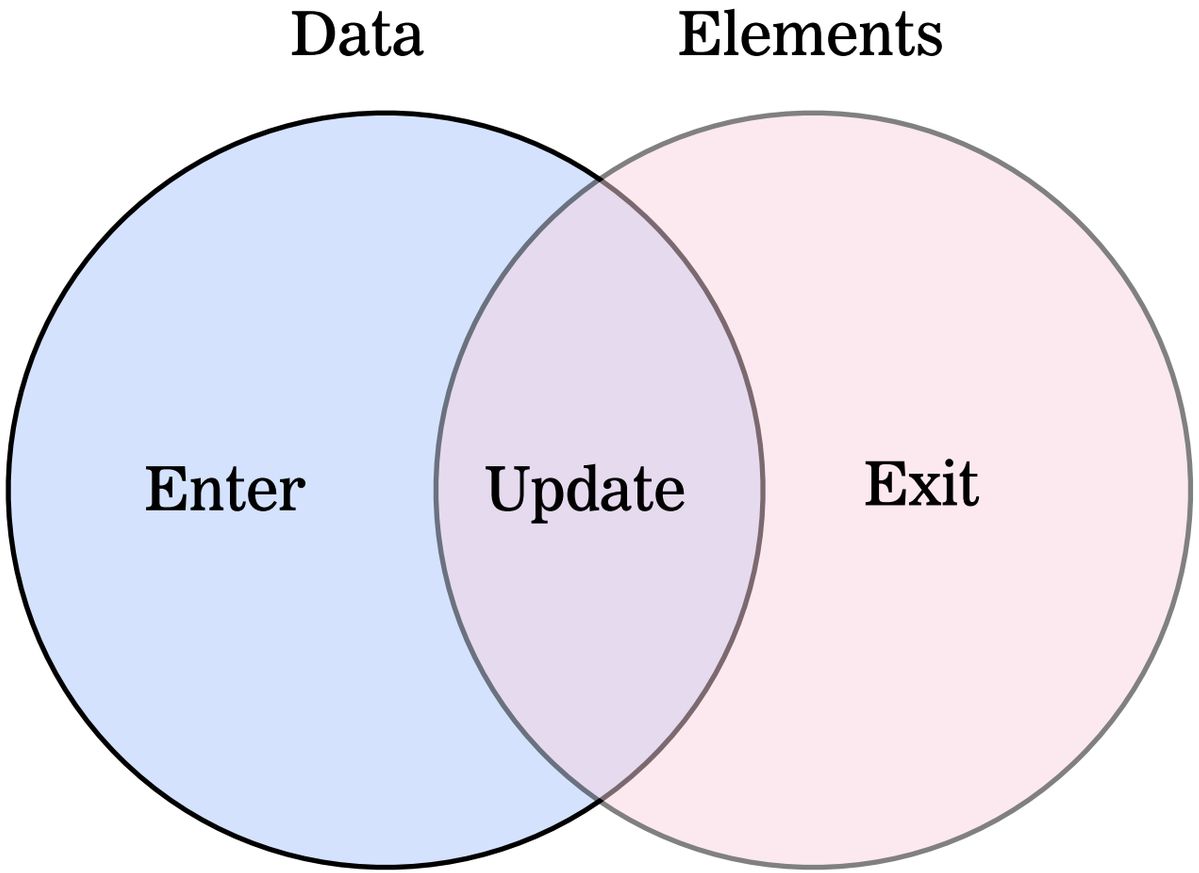
c) Data Join With Key Function

d) Dynamic Data Join

D3.JS DATA JOIN – VIRTUAL SELECTIONS



D3.JS DATA JOIN – SCENARIOS



Scenario 1:

Element # < Data #

Element # = 0 is a special case

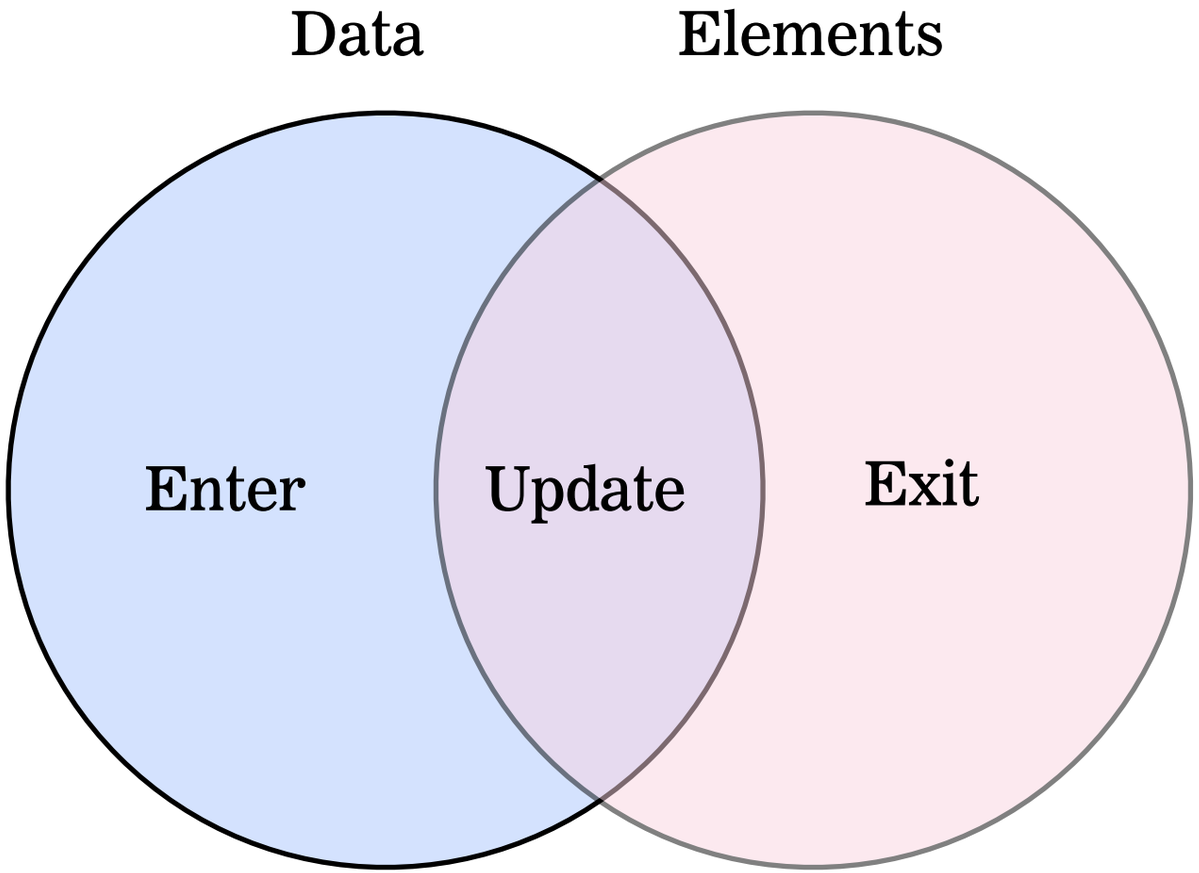
Scenario 2:

Element # = Data #

Scenario 3:

Element # > Data #

D3.JS DATA JOIN – SCENARIO 1 (SPECIAL CASE)



Scenario 1:

Element # = 0

Virtual Selections:

update - empty

enter - non-empty

exit - empty

EXERCISE – DATA JOIN SCENARIO 1 (SPECIAL CASE)

Scenario 1: Element # = 0 is a special case

Save & Use base.html

```
var virtualSelection = d3.select("body").selectAll("p").data([1]);
d3.selectAll("p");
virtualSelection;
virtualSelection.enter();
virtualSelection.exit();
```

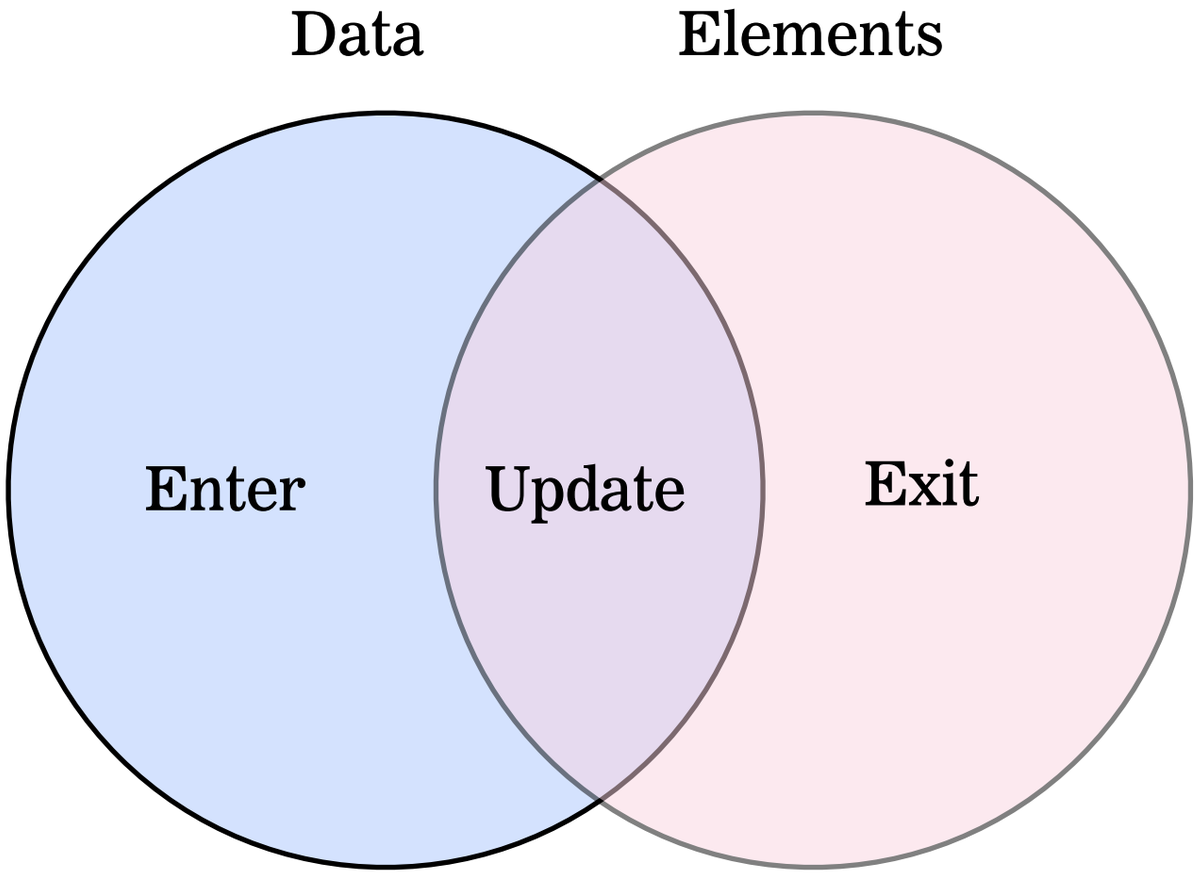
Discuss Virtual Selections

update - empty

enter - non-empty

exit - empty

D3.JS DATA JOIN – SCENARIO 1



Scenario 1:

Element # < Data #

Virtual Selections:

update - non-empty

enter - non-empty

exit - empty

\

EXERCISE – DATA JOIN SCENARIO 1

Scenario 1: Element # < Data

Use base.html

```
d3.select("body").append("p");

var virtualSelection = d3.select("body").selectAll("p").data([1, 2, 3]);

d3.selectAll("p");

virtualSelection;

virtualSelection.enter();

virtualSelection.exit();
```

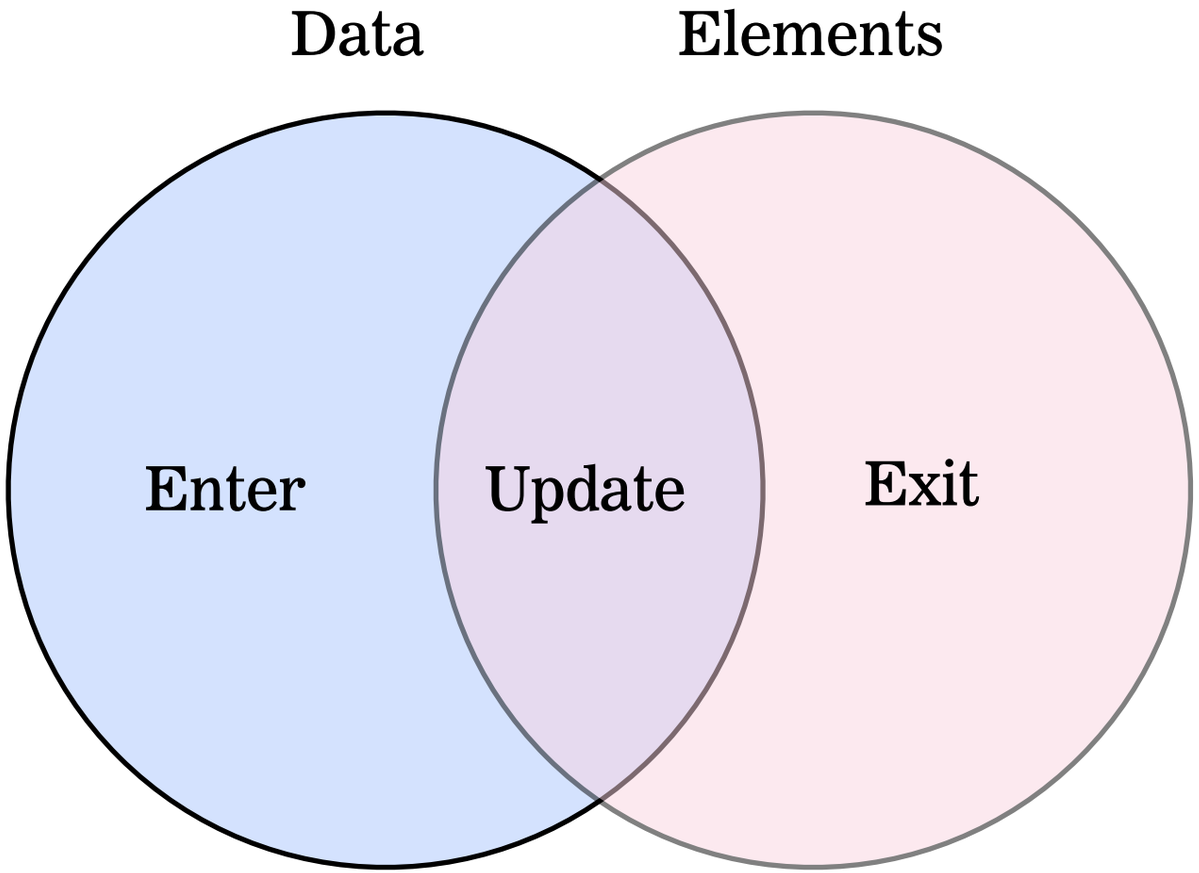
Discuss Virtual Selections

update - non-empty

enter - non-empty

exit - empty

D3.JS DATA JOIN – SCENARIOS



Scenario 2:

Element # = Data #

Virtual Selections:

update - non-empty

enter - empty

exit - empty

EXERCISE – DATA JOIN SCENARIO 2

Scenario 2: Element # = Data

Use base.html

```
d3.select("body").selectAll("p").data([0,0,0]).enter().append("p");
var virtualSelection = d3.select("body").selectAll("p").data([1, 2, 3]);
d3.selectAll("p");
virtualSelection;
virtualSelection.enter();
virtualSelection.exit();
```

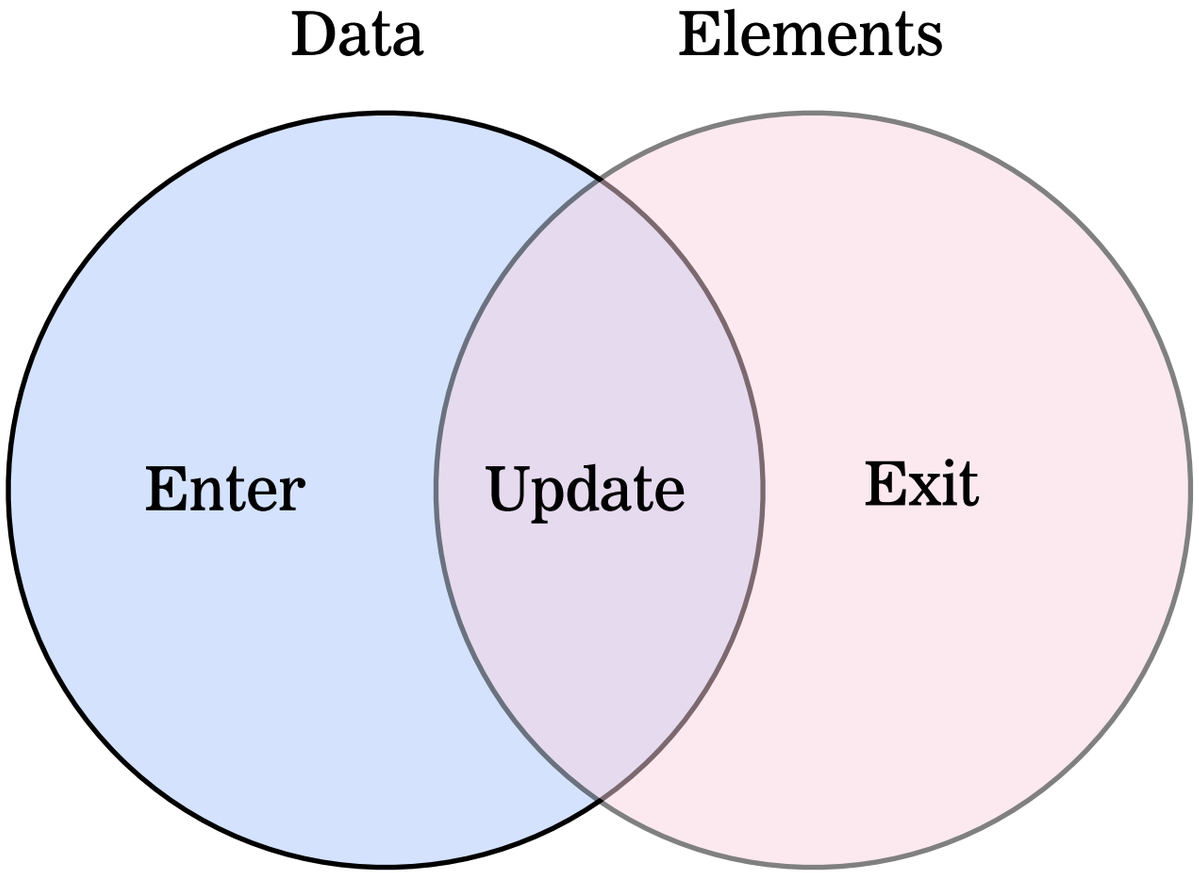
Discuss Virtual Selections

update - non-empty

enter - empty

exit - empty

D3.JS DATA JOIN – SCENARIOS



Source:
http://bost.ocks.org/mike/join/

Scenario 3:

Element # > Data #

Virtual Selections:

update - non-empty

enter - empty

exit - non-empty

EXERCISE – DATA JOIN SCENARIO 3

Scenario 3: Element # > Data

Use base.html

```
d3.select("body").selectAll("p").data([0,0,0]).enter().append("p");
var virtualSelection = d3.select("body").selectAll("p").data([1]);
d3.selectAll("p");
virtualSelection;
virtualSelection.enter();
virtualSelection.exit();
```

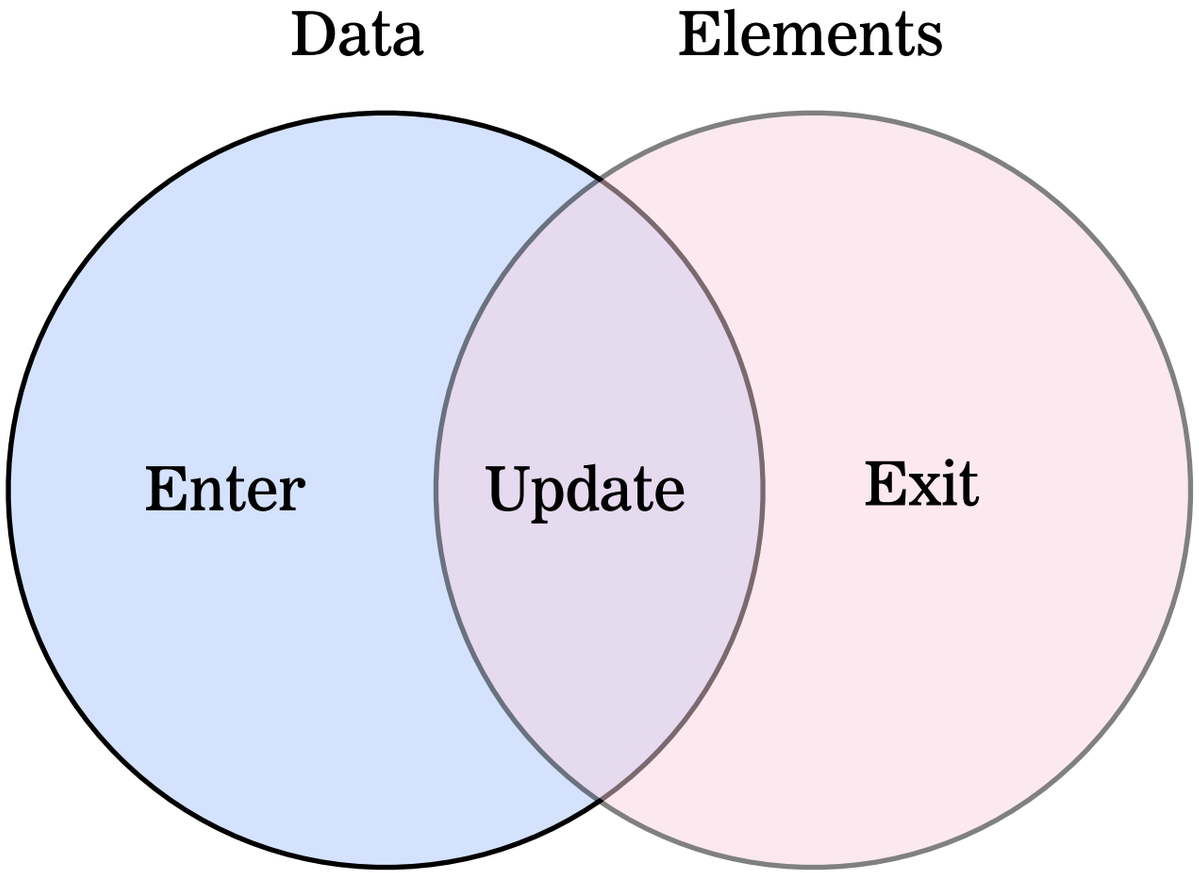
Discuss Virtual Selections

update - non-empty

enter - empty

exit - non-empty

D3.JS DATA JOIN – PROCESS REPEATS AD INFINITUM



Source:
http://bost.ocks.org/mike/join/

Scenario 1:

Element # < Data #

Element # = 0 is a special case

Scenario 2:

Element # = Data #

Scenario 3:

Element # > Data #

Sections

D3.JS DATA JOIN – PROCESS REPEATS HOW?

Data Comes In
-> Data Join
-> Elements Created

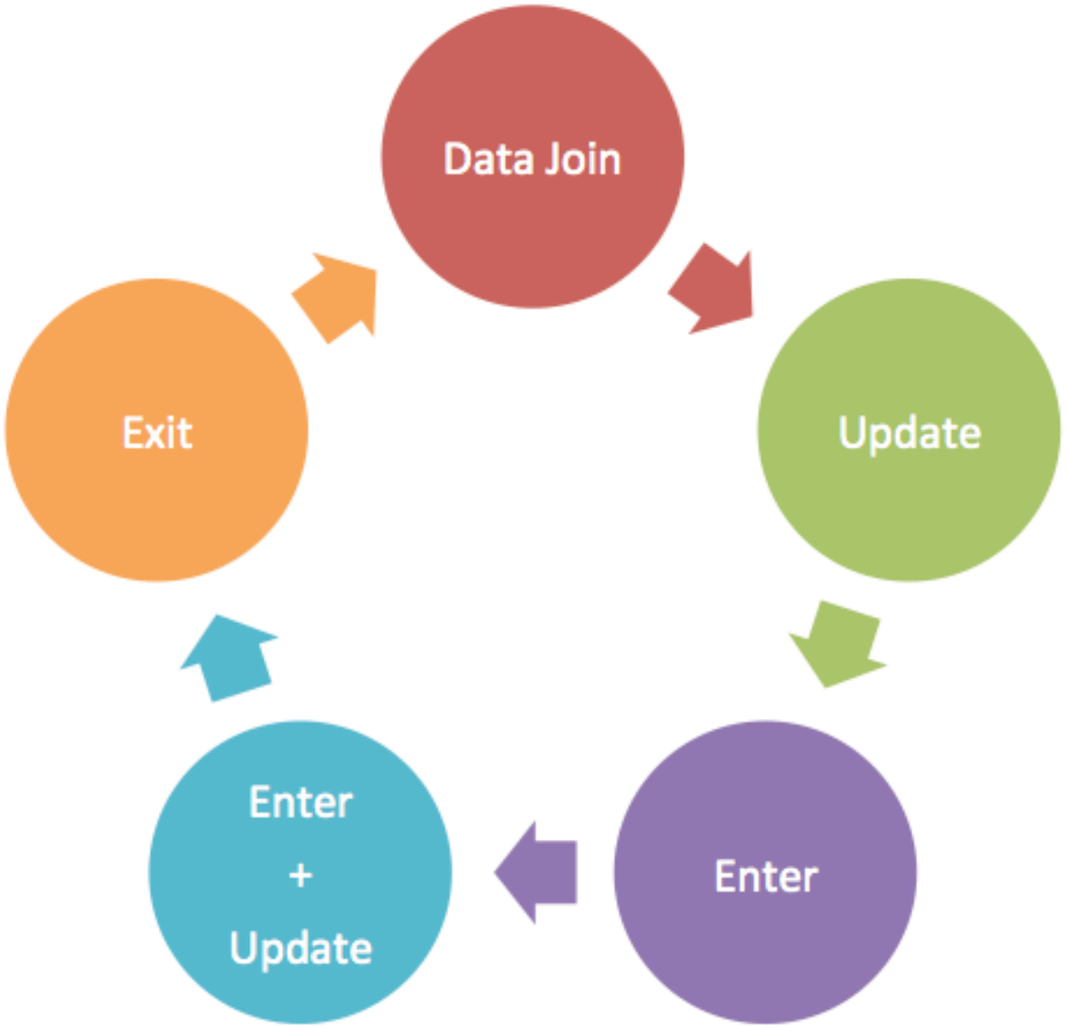
Data Comes In
-> Data Join
-> Update Selection?
-> Enter Selection?
-> Exit Selection?

Data Comes in.....

} What happens here is the general update pattern. Has to work for all scenarios

GENERAL UPDATE PATTERN FOR ALL SCENARIOS

Data Comes In
-> Data Join
-> Update Selection
-> Enter Selection
-> Enter + Update Selection
-> Exit Selection

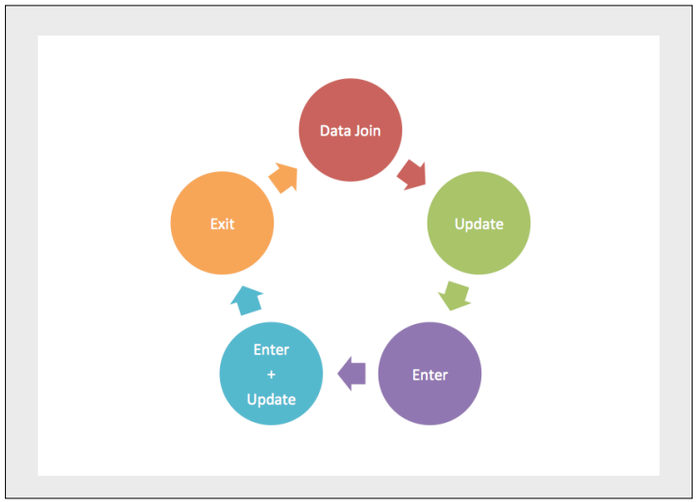


EXERCISE – ENTER + UPDATE SELECTION

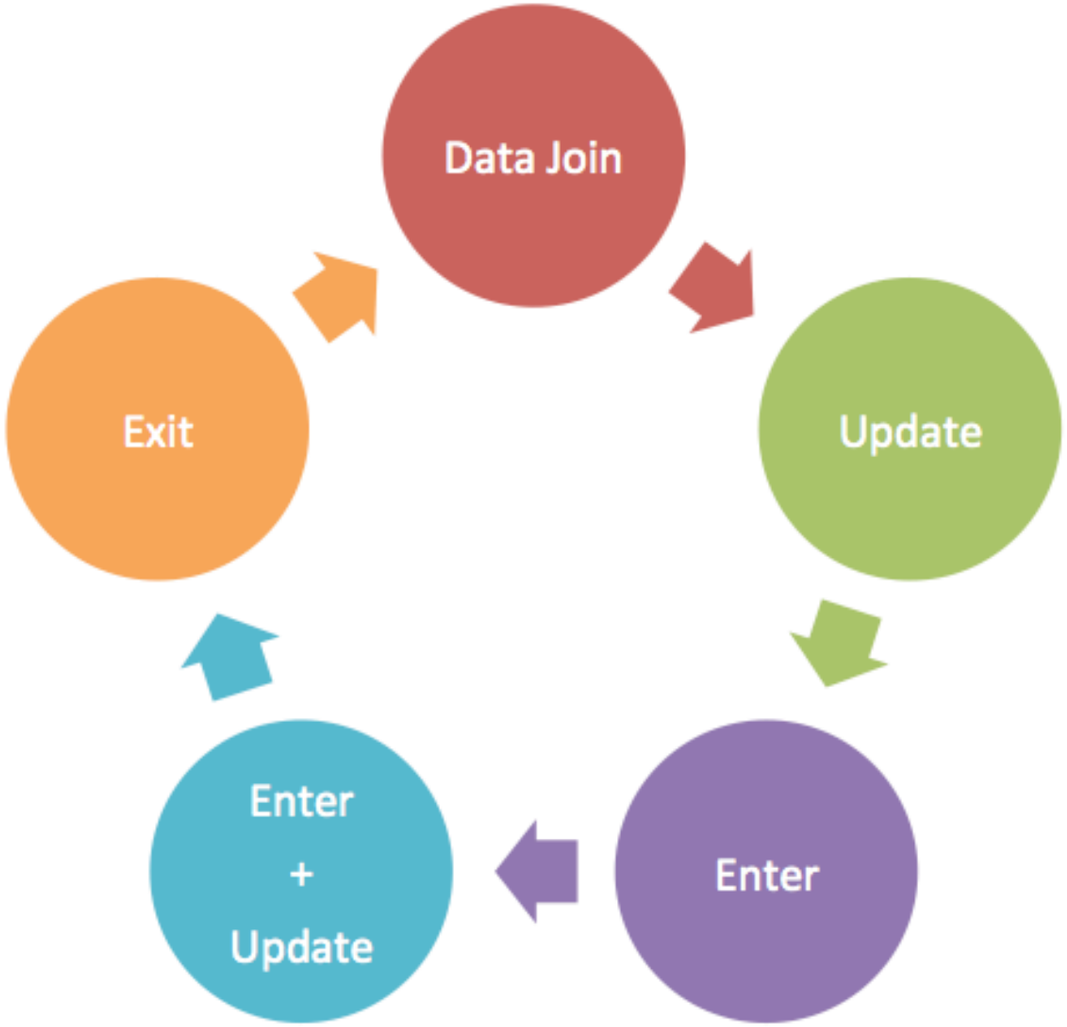
Enter + Update Selection - What actually happens?

Use base.html and input lines one at a time

```
var updateSelection = d3.select("body").selectAll("p").data([0,0,0]);
updateSelection;
var enterSelection = updateSelection.enter();
updateSelection;
enterSelection;
var enterUpdateSelection = enterSelection.append("p");
enterSelection;
updateSelection;
enterUpdateSelection;
```



GENERAL UPDATE PATTERN IN D3.JS PSEUDO CODE

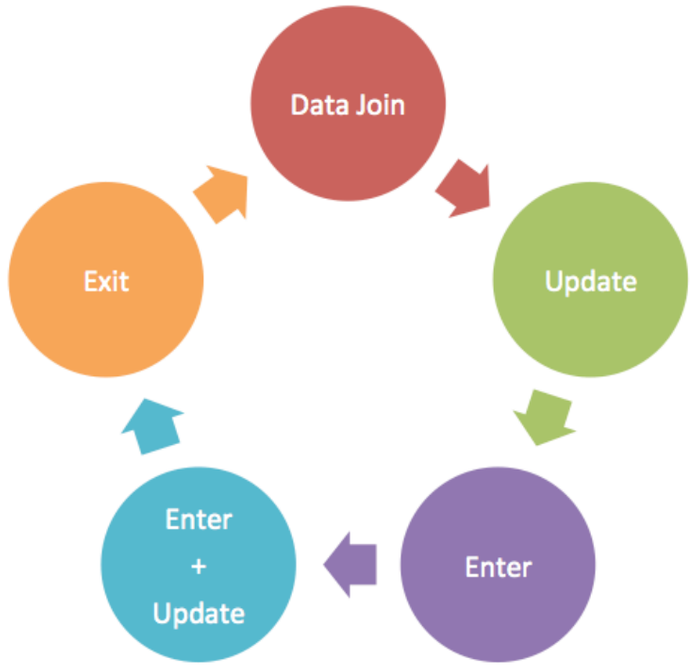



```
var newSel = selection.data(....)

newSel
   .attr( ... , function(d, i) { .... });

newSel.enter().append( ... )
   .attr( ... , function(d, i) { .... });

newSel
   .attr( ... , function(d, i) { .... });

newSel.exit().remove();
```

EXERCISE – GENERAL UPDATE PATTERN (1 OF 5)

Elements < # Data (special case where # Elements is 0)

```
var text = d3.select("body").selectAll("p")
    .data("winter".split(""))
  .enter().append("p")
    .attr("class", "enter_selection_winter")
    .text(function(d) { return d; });
```



EXERCISE – GENERAL UPDATE PATTERN (2 OF 5)

Elements = # Data

```
var springText = d3.select("body").selectAll("p")
    .data("spring".split(""));

springText.attr("class", "update_selection_spring");

springText.enter().append("p")
    .attr("class", "enter_selection_spring");

springText.text(function(d) { return d; });

springText.exit()
    .attr("class", "exit_selection_spring");

springText.exit().remove();
```

EXERCISE – GENERAL UPDATE PATTERN (3 OF 5)

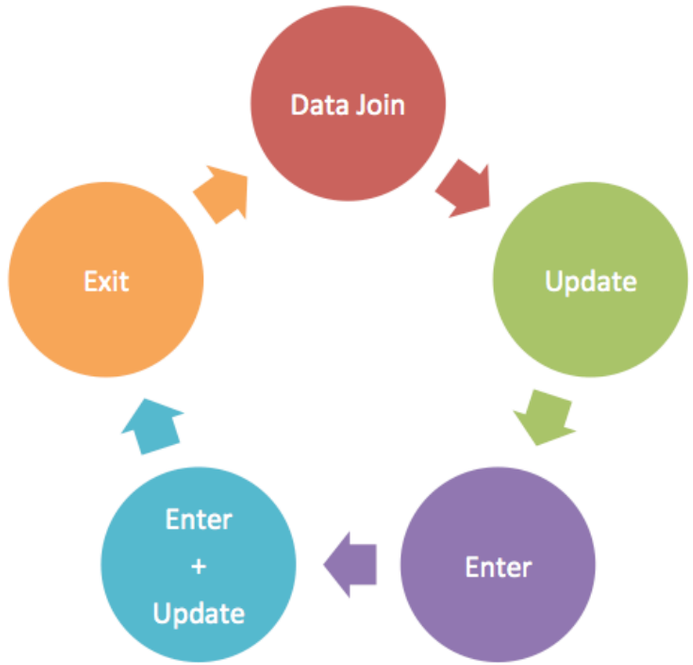
Elements < # Data

```
var summerText = d3.select("body").selectAll("p")
    .data("summertime".split(""));

summerText.attr("class", "update_selection_summer");

summerText.enter().append("p")
    .attr("class", "enter_selection_summer");

summerText.text(function(d) { return d; });

summerText.exit()
    .attr("class", "exit_selection_summer");

summerText.exit().remove();
```

EXERCISE – GENERAL UPDATE PATTERN (4 OF 5)

Elements > # Data

```
var fallText = d3.select("body").selectAll("p")
    .data("fall".split(""));

fallText.attr("class", "update_selection_fall");

fallText.enter().append("p")
    .attr("class", "enter_selection_fall");

fallText.text(function(d) { return d; });

fallText.exit()
    .attr("class", "exit_selection_fall");

fallText.exit().remove();
```

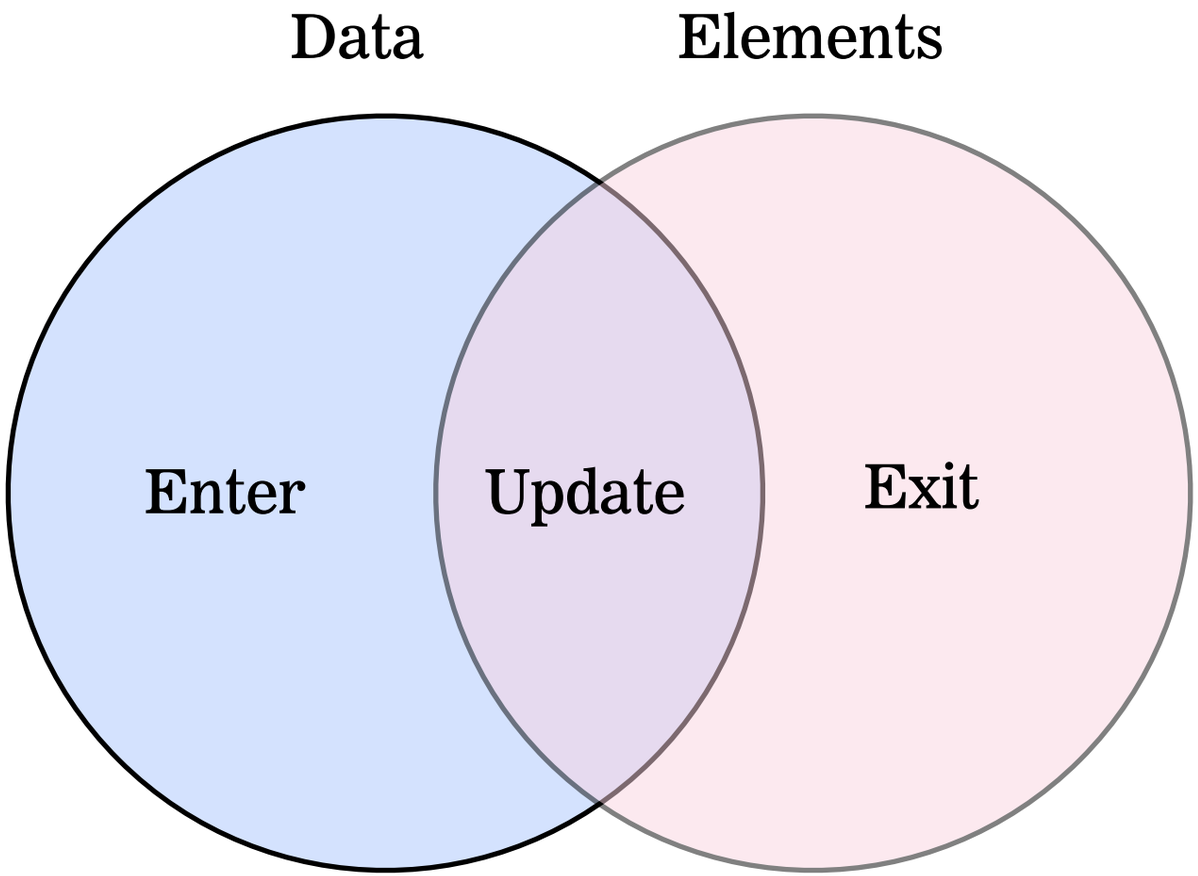


EXERCISE – GENERAL UPDATE PATTERN (5 OF 5)

Elements > # Data

```
var winterText = d3.select("body").selectAll("p")
    .data("winter".split(""));

winterText.attr("class", "update_selection_winter");

winterText.enter().append("p")
    .attr("class", "enter_selection_winter");

winterText.text(function(d) { return d; });

winterText.exit()
    .attr("class", "exit_selection_winter");

winterText.exit().remove();
```

D3.JS DATA JOINS DATA LEFT TO RIGHT



summertime

=> fallertime

=> fall

Data Join depends on element & data index.

Source:
http://bost.ocks.org/mike/join/

GENERAL UPDATE PATTERN TAKE AWAYS

All Steps Must Happen

-> Data Join

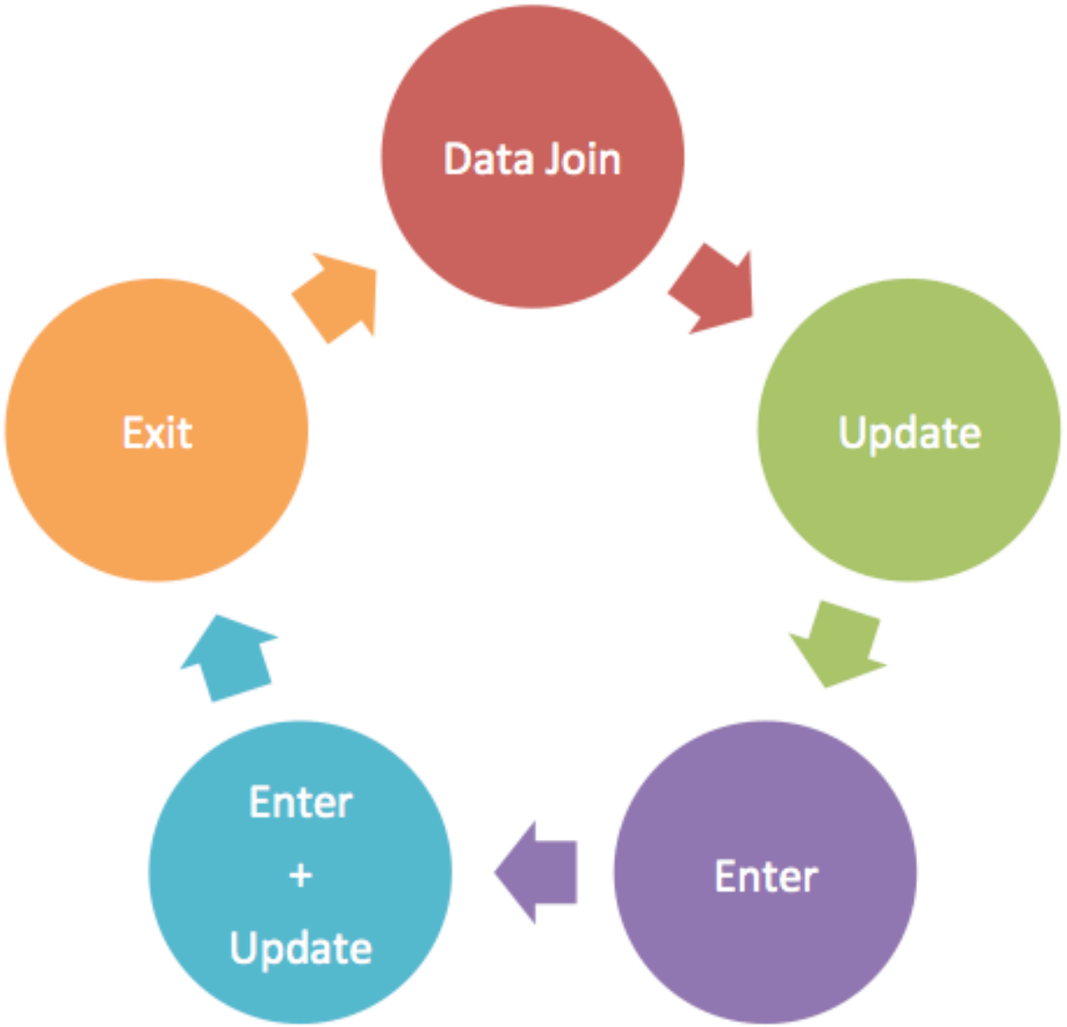
-> Update Selection

-> Enter Selection

-> Enter + Update Selection

-> Exit Selection

Data is joined in a first come first served way.



Sections

D3.JS DATA JOINS DATA LEFT TO RIGHT UNLESS

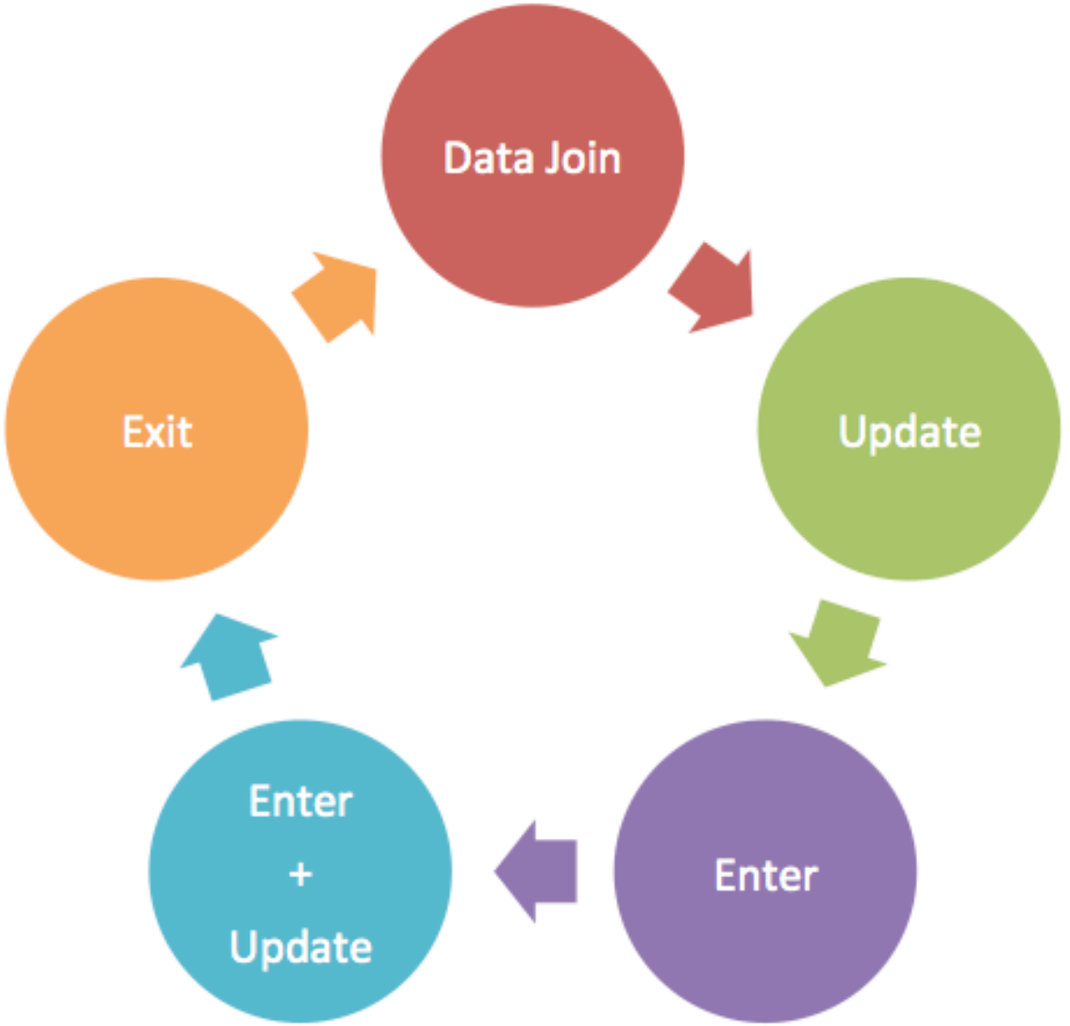
A key-function is specified

- This replaces the default by-index behavior of the Data Join.
- You programmatically choose how elements and data are joined

```
d3.selection.data(x)
d3.selection.data(x, function(d, i) { ... });
```

SAME GENERAL UPDATE PATTERN FOR ALL SCENARIOS

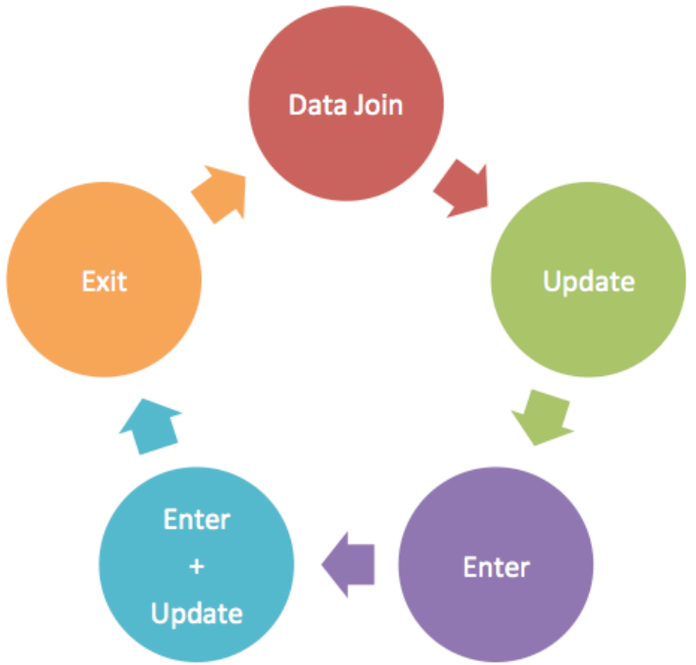
Data Comes In
-> Data Join with Key-Function
-> Update Selection
-> Enter Selection
-> Enter + Update Selection
-> Exit Selection



EXERCISE – KEY FUNCTION UPDATE PATTERN (1 OF 3)

Elements < # Data (special case where # Elements is 0)

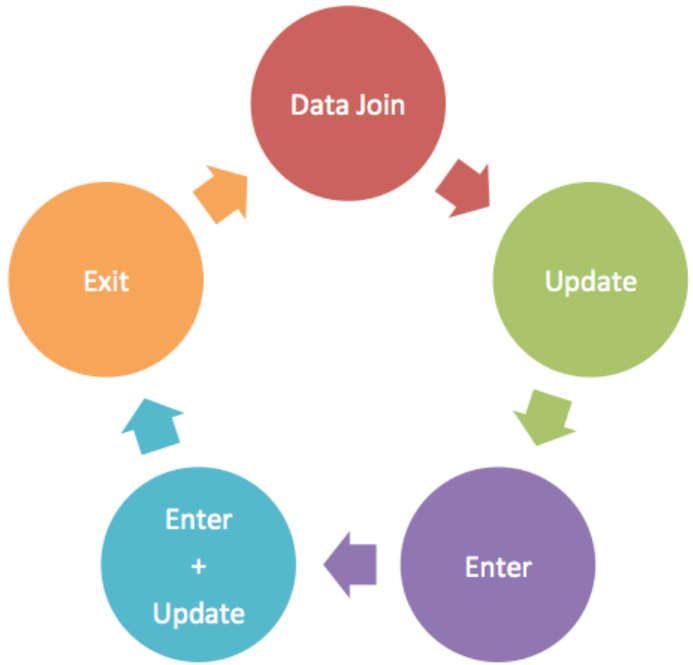
```
var dataSet = [{"color": "red", "r": 5},
               {"color": "blue", "r": 15},
               {"color": "green", "r": 25}];

var svg = d3.select("body").append("svg");

var circles = svg.selectAll("circle")
   .data(dataSet)
  .enter().append("circle")
   .attr("cx", function(d, i) { return i * 50 + 25; })
   .attr("cy", 25)
   .attr("r", function(d, i) { return d.r; })
   .style("fill", function(d, i) { return d.color; });
```



EXERCISE – KEY FUNCTION UPDATE PATTERN (2 OF 3)

New Data, Unfortunately Re-arranged (# element = # data)

```
var dataSet = [{"color": "blue", "r": 35},
               {"color": "green", "r": 25},
               {"color": "red", "r": 5}];

// Update Circle Data Properties
svg.selectAll("circle")
   .data(dataSet);

// No enter or Exit, so just focus on Update selection

d3.selectAll("circle")
   .attr("cx", function(d, i) { return i * 50 + 25; })
   .attr("cy", 25)
   .attr("r", function(d, i) { return d.r; })
   .style("fill", function(d, i) { return d.color; });
```

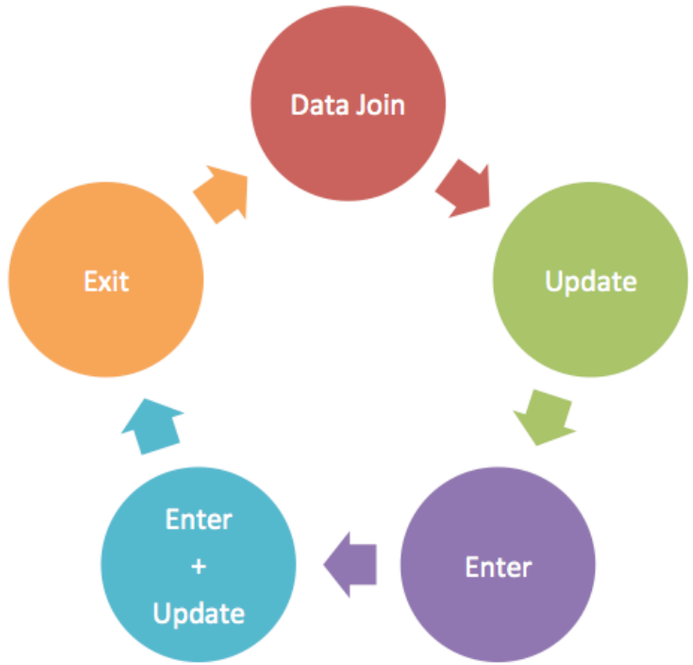


Left to right data join means wrong data for wrong circle.

EXERCISE – KEY FUNCTION UPDATE PATTERN (3 OF 3)

New Data, Unfortunately Re-arranged (# element = # data)

```
// Reload Slide 49, then run below
var dataSet = [{"color": "blue", "r": 35},
               {"color": "green", "r": 25},
               {"color": "red", "r": 5}];

// Update Circle Data Properties
svg.selectAll("circle")
   .data(dataSet, function(d, i) { return d.color; });

// No enter or Exit, so just focus on Update selection
d3.selectAll("circle")
   .attr("cx", function(d, i) { return i * 50 + 25; })
   .attr("cy", 25)
   .attr("r", function(d, i) { return d.r; })
   .style("fill", function(d, i) { return d.color; });
```



Key-function data join means right data for right circle.

KEY FUNCTION UPDATE PATTERN

Works when data moves around (# elements = # data)

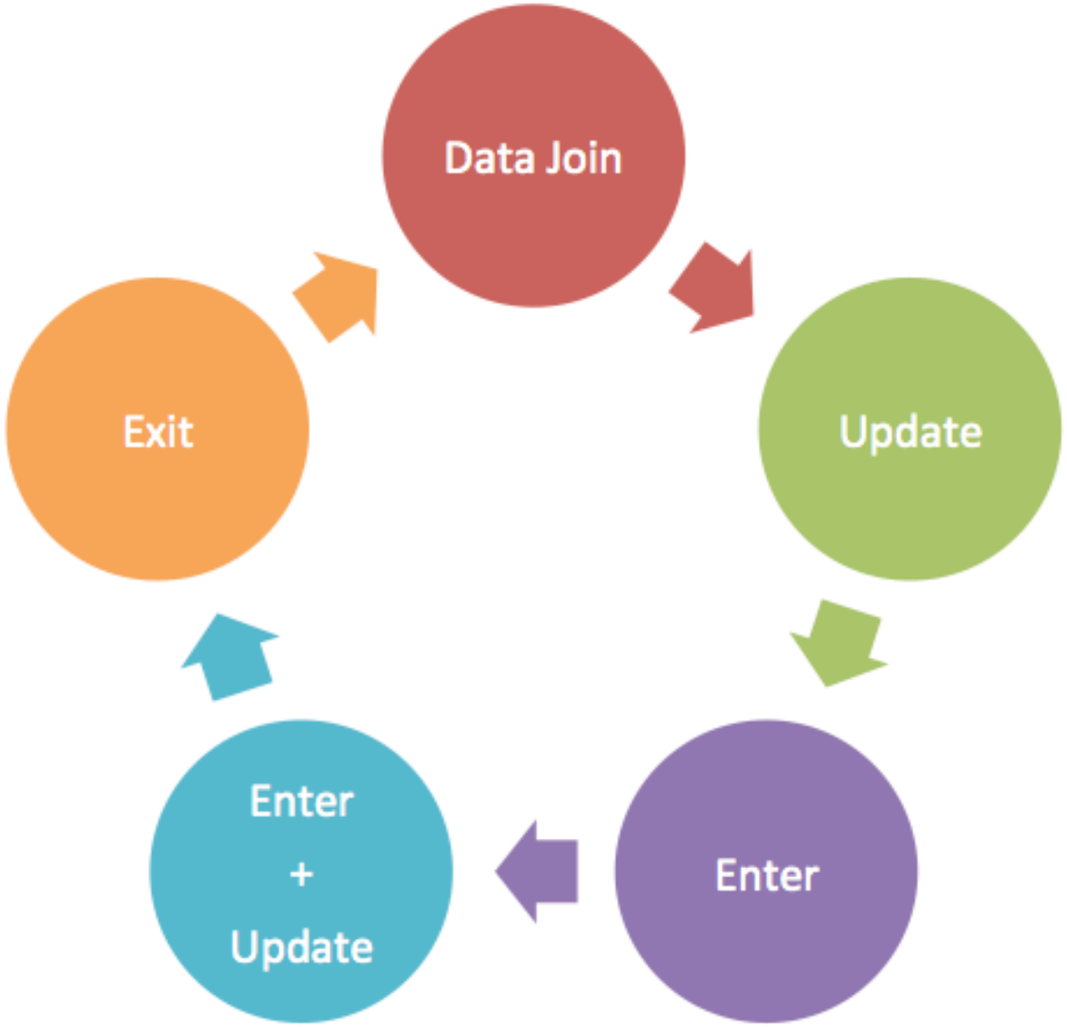
or

When there is a mismatch between # elements and # data.

- In cases of mismatch, have to do the full general update pattern

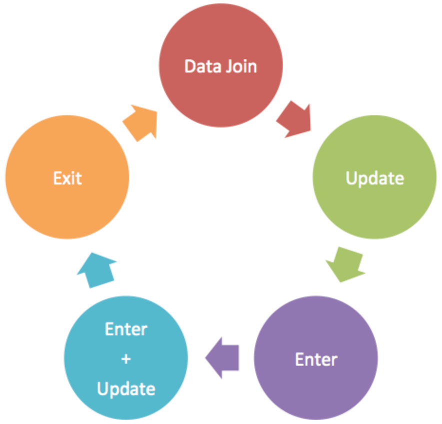
Sections

GENERAL UPDATE PATTERN IN D3.JS PSEUDO CODE




```
var newSel = selection.data(....)

newSel
   .attr( ... , function(d, i) { .... });

newSel.enter().append( ... )
   .attr( ... , function(d, i) { .... });

newSel
   .attr( ... , function(d, i) { .... });

newSel.exit().remove();
```

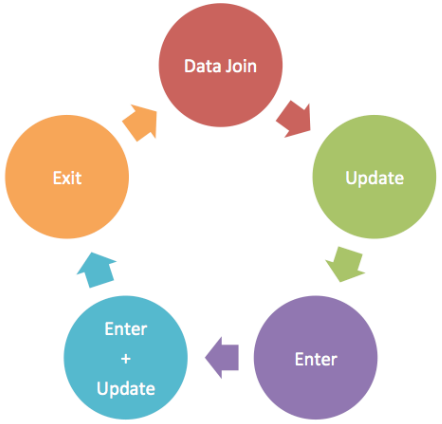
DYNAMIC DATA IMPLIES MANY DATA JOINS



New Data

New Data

New Data



New Data

New Data

New Data

New Data

New Data

ABSTRACT GENERAL UPDATE PATTERN AS FUNCTION

New Data

New Data

New Data

General Update Pattern Function

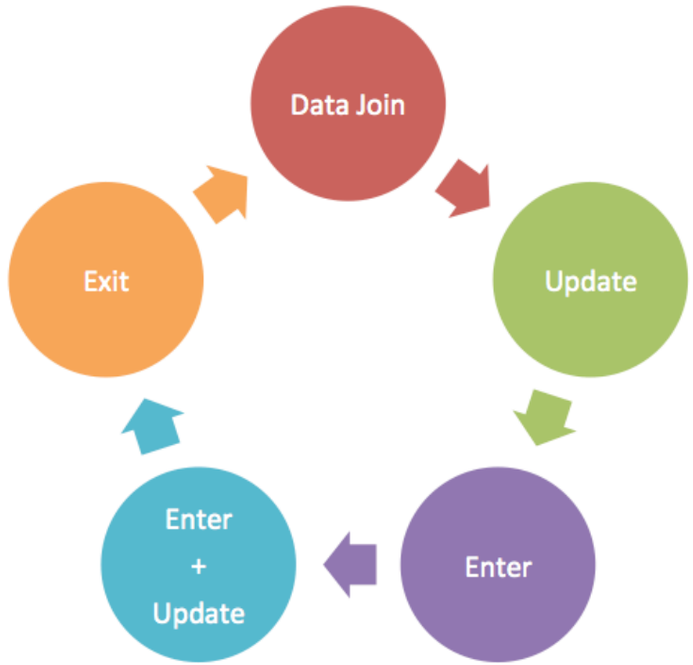
ABSTRACT TO FUNCTION GENERAL UPDATE PATTERN

```
function update(data) {

  // DATA JOIN
  var text = d3.select("body").selectAll("p")
      .data(data);

  // UPDATE
  text.attr("class", "update");

  // ENTER
  text.enter().append("p")
      .attr("class", "enter");

  // ENTER + UPDATE
  text.text(function(d) { return d; });

  // EXIT
  text.exit().attr("class", "exit").remove()
}
```

EXERCISE – DYNAMIC GENERAL UPDATE (0 OF 5)

Define Function

```
function update(data) {

  // DATA JOIN
  var text = d3.select("body").selectAll("p")
      .data(data);

  // UPDATE
  text.attr("class", "update");

  // ENTER
  text.enter().append("p")
      .attr("class", "enter");

  // ENTER + UPDATE
  text.text(function(d) { return d; });

  // EXIT
  text.exit().attr("class", "exit").remove()
}
```



EXERCISE – DYNAMIC GENERAL UPDATE (1 OF 5)

Elements < # Data (special case where # Elements is 0)

```
var data =  "winter".split("");

update(data);
```

EXERCISE – DYNAMIC GENERAL UPDATE (2 OF 5)

Elements = # Data

```
var data =  "spring".split("");

update(data);
```

EXERCISE – DYNAMIC GENERAL UPDATE (3 OF 5)

Elements < # Data

```
var data =  "summertime".split("");

update(data);
```

EXERCISE – DYNAMIC GENERAL UPDATE (4 OF 5)

Elements > # Data

```
var data =  "fall".split("");

update(data);
```

EXERCISE – DYNAMIC GENERAL UPDATE (5 OF 5)

Elements > # Data

```
var data =  "winter".split("");

update(data);
```

EXERCISE – SEASON LOOP

Elements ??? # Data

```
// Generate Data Set
var data = ["spring", "summertime", "fall", "winter"].map(function(d) { return d.split("")});

// General Update Pattern Function
function update(data) {

  var text = d3.select("body").selectAll("p")
      .data(data);

  text.attr("class", "update");

  text.enter().append("p").attr("class", "enter");

  text.text(function(d) { return d; });

  text.exit().attr("class", "exit").remove()
}

// Initial Display of Spring
update(data[0]);

// Grab a random season and update display
setInterval(function() {
  update(data[Math.floor(Math.random() * 4)]);
}, 1500);
```

EXERCISE – CIRCLE LOOP

Use circle_loop.js and then run it in base.html

```
// Take a look at it to understand it

// Add a .remove() after
// .attr("r", 0);
```

Discuss

- forEach
- d3.shuffle
- General Update Pattern Data Join with Key Function
- .exit() w/o .remove()
- .exit() w .remove()
- setInterval

SECTION 3

D3.JS REUSABLE CHART PATTERN

Sections

OTHER DATA VISUALIZATION LIBRARIES / TOOLS

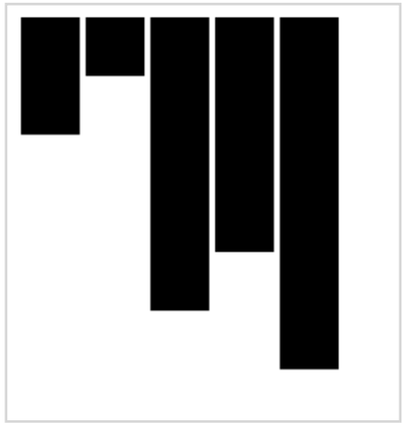
Cubism.js	n3-charts	Bokeh
NVD3.js	C3.js	Dimple.js
d3.chart	Visual Sedimentation	Vega
Graphene	Raw	Dance.js
Rickshaw	rCharts	DC.js
Crossfilter.js	d4.js	Many Others...

Source:
http://mikemcdearmon.com/portfolio/techposts/data-visualization-base-on-d3-js

BASIC D3.JS BAR CHART EXAMPLE

```
var dataset = [2, 1, 5, 4, 6];

var svg = d3.select("body").append("svg")
    .attr("width", 200)
    .attr("height", 200);

svg.selectAll("rect")
    .data(dataset)
  .enter().append("rect")
    .attr("x", function(d, i) { return i * 33; })
    .attr("y", 0)
    .attr("width", 30)
    .attr("height", function(d) { return d * 30; });
```

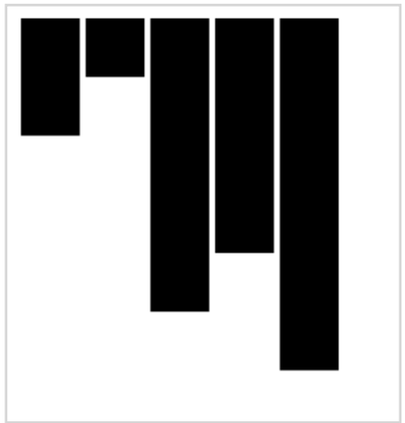
```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<svg width="200" height="200">
       <rect x="0" y="0" width="30" height="60"></rect>
       <rect x="33" y="0" width="30" height="30"></rect>
       <rect x="66" y="0" width="30" height="150"></rect>
       <rect x="99" y="0" width="30" height="120"></rect>
       <rect x="132" y="0" width="30" height="180"></rect>
     </svg>
   </body>
</html>
```



D3.JS FRAMEWORK - BARCHART.JS

```
var dataset = [2, 1, 5, 4, 6];

var myBarChart = barChart();

myBarChart
    .svgWidth(200)
    .svgHeight(200);

d3.select("body").append("div")
    .datum(dataset)
    .call(myBarChart);
```

```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<div>
      ▼<svg width="200" height="200">
          <rect x="0" y="0" width="38" height="66"></rect>
          <rect x="40" y="0" width="38" height="33"></rect>
          <rect x="80" y="0" width="38" height="165"></rect>
          <rect x="120" y="0" width="38" height="132"></rect>
          <rect x="160" y="0" width="38" height="198"></rect>
        </svg>
      </div>
      " "
  </body>
</html>
```



Sections

THREE WAYS TO DEFINE JAVASCRIPT FUNCTIONS

Named Function:	`function plusOne (d) { return d + 1; }`
Anonymous Function:	`var plusTwo = function (d) { return d + 2; };`
Function Constructor:	`var plusThree = new Function('d', 'return d+3;');`

JAVASCRIPT TYPEOF

typeOf returns a string indicating the type of the unevaluated operand

Type	Result
Undefined	`"undefined"`
Null	`"object"`
Boolean	`"boolean"`
Number	`"number"`
String	`"string"`
Host object (provided by the JS environment)	*Implementation-dependent*
Function object (implements [[Call]] in ECMA-262 terms)	`"function"`
E4X XML object	`"xml"`
E4X XMLList object	`"xml"`
Any other object	`"object"`

```
function plusOne (d) { return d + 1; }
undefined
var plusTwo = function (d) { return d + 2; };
undefined
var plusThree = new Function('d', 'return d+3;');
undefined

typeof plusOne;
"function"
typeof plusTwo;
"function"
typeof plusThree;
"function"
```

Chart Source:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof

JAVASCRIPT FUNCTIONS OBJECTS PROPERTIES

JavaScript Functions are Objects

- We can set properties

- We can get properties back

```
function plusOne (d) { return d + 1; }
undefined

plusOne.foo = "bar";
"bar"

plusOne;
function plusOne(d) { return d + 1; }

plusOne.foo;
"bar"
```

EXERCISE - JS FUNCTION OBJECT PROPERTIES

Use base.html

```
function plusOne (d) { return d + 1; }

plusOne(10);

plusOne.ten = 10;

plusOne.ten;

plusOne.twenty = function() { return 20; };

plusOne.twenty();
```

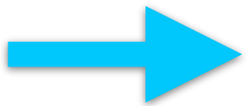
Discuss

Javascript Objects & Properties

Sections

D3.JS METHOD CHAINING

```
var para = d3.select("body")
      .append("p");

para.text("DataViz!");

para.style("font-size", "24px");

para.attr("class", "body-paragraph");

para.attr("id", "first-paragraph");
```



```
var para = d3.select("body").append("p")
      .text("DataViz!")
      .style("font-size", "24px")
      .attr("class", "body-paragraph")
      .attr("id", "first-paragraph");
```

WANT METHOD CHAINING FOR JS FUNCTION OBJECT

Method chaining allows for easy to read definition of properties.

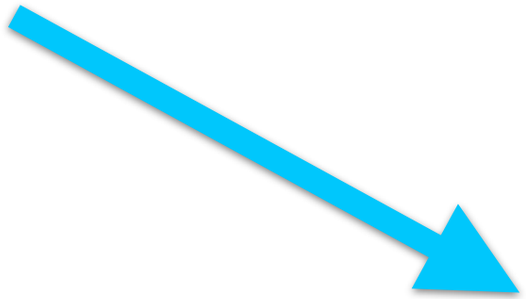
```
function plusOne (d) { return d + 1; }
undefined

plusOne.fooOne = "bar_1";
"bar_1"

plusOne.fooTwo = "bar_2";
"bar_2"

plusOne.fooThree = "bar_3";
"bar_3"
```


```
function plusOne (d) { return d + 1; }
undefined

plusOne
    .fooOne("bar_1")
    .fooTwo("bar_2")
    .fooThree("bar_3");

plusOne.fooOne();
"bar_1"

plusOne.fooTwo();
"bar_2"

plusOne.fooThree();
"bar_3"
```

JS FUNCTION METHOD CHAINING

Method Chaining relies on Function returning function object

```
function barChart() {

    // create chart
    function chart() { return; }

    // chart properties
    chart.foo = "bar";

    // return chart
    return chart;

};
```



```
var myBarChart = barChart();
undefined

myBarChart;
function chart() { return; }

typeof myBarChart;
"function"

myBarChart.foo;
"bar"
```

JS FUNCTION PROPERTY SETTING

We can set function properties of a returned function

```
var myBarChart = barChart();
undefined

myBarChart.foo;
"bar"

myBarChart.foo = "cupcake";
"cupcake"

myBarChart.foo;
"cupcake"
```

DISCUSS: HOWEVER, CANNOT YET CHAIN

GETTERS & SETTERS

We can set/get properties of the returned function object with a function

```
function barChart() {

    var foo;

    function chart() { return; }

    chart.foo = function(value) {
      if (!arguments.length) return foo;
      foo = value;
      return chart;
     };

    return chart;

};
```

```
var myBarChart = barChart();
undefined

myBarChart.foo("bar");
function chart() { return; }

myBarChart.foo();
"bar"

myBarChart.foo("cupcake");
function chart() { return; }

myBarChart.foo();
"cupcake"
```

GETTERS & SETTERS FOR METHOD CHAINING

We can set/get properties of the returned function object with a function

```
function barChart() {

    var svgHeight, svgWidth;

    function chart() { return; }

    chart.svgHeight = function(value) {
        if (!arguments.length) return svgHeight;
        svgHeight = value;
        return chart;
     };

    chart.svgWidth = function(value) {
        if (!arguments.length) return svgWidth;
        svgWidth = value;
        return chart;
     };

    return chart;

};
```

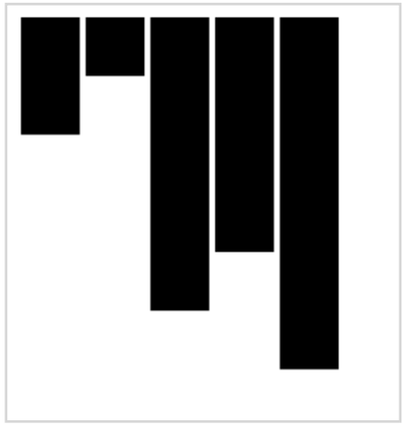
```
var myBarChart = barChart();
undefined

myBarChart.svgHeight(200).svgWidth(200);
function chart() { return; }

myBarChart.svgHeight();
200

myBarChart.svgWidth();
200
```

Sections

D3.JS FRAMEWORK - BARCHART.JS: 1/2 WAY THERE

```
var dataset = [2, 1, 5, 4, 6];

var myBarChart = barChart();

myBarChart
    .svgWidth(200)
    .svgHeight(200);

d3.select("body").append("div")
    .datum(dataset)
    .call(myBarChart);
```

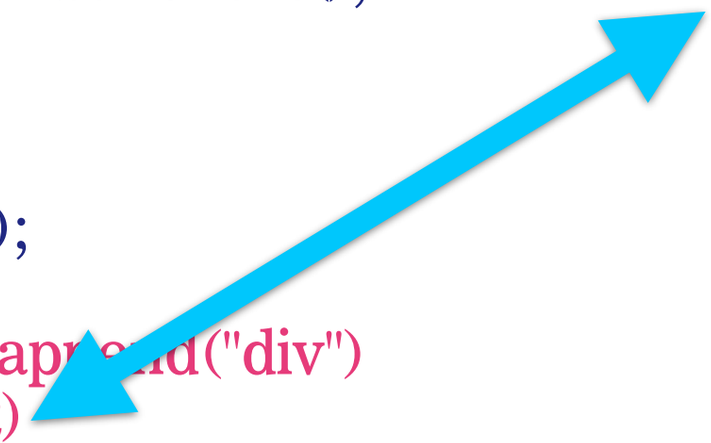
```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<div>
      ▼<svg width="200" height="200">
          <rect x="0" y="0" width="38" height="66"></rect>
          <rect x="40" y="0" width="38" height="33"></rect>
          <rect x="80" y="0" width="38" height="165"></rect>
          <rect x="120" y="0" width="38" height="132"></rect>
          <rect x="160" y="0" width="38" height="198"></rect>
        </svg>
      </div>
      " "
  </body>
</html>
```



DISCUSS: D3 BINDING PATTERN

BARCHART.JS: REST OF THE WAY

```
var dataset = [2, 1, 5, 4, 6];

var myBarChart = barChart();

myBarChart
    .svgWidth(200)
    .svgHeight(200);

d3.select("body").append("div")
    .datum(dataset)
    .call(myBarChart);
```



```
function barChart() {

    var svgHeight, svgWidth;

    function chart() { return; }

    chart.svgHeight = function(value) {
        if (!arguments.length) return svgHeight;
        svgHeight = value;
        return chart;
    };

    chart.svgWidth = function(value) {
        if (!arguments.length) return svgWidth;
        svgWidth = value;
        return chart;
    };

    return chart;

};
```

DISCUSS: GET DATA INTO CHART() AND BUILD CHART

USE DATA BOUND TO EACH ELEMENT IN SELECTION

```
function chart() { return; }
```

```
function chart(selection) { return; }
```

```
var myBarChart = barChart();
undefined

myBarChart(d3.selection);

// or

d3.selection.call(myBarChart);
```

DISCUSS: BARCHART() -> CHART FUNCTION -> VAR

SELECTION CAN CONTAIN 1 OR MORE ELEMENTS

```
function chart(selection) {
  selection.each(function(d, i) {
    // Build Chart Here For Each Element In The Selection
  }
}
```

BARCHART.JS - CHART FUNCTION

```
function chart(selection) {

    selection.each(function(d, i) {

        var barXSpacing    = svgWidth / d.length;
        var barUnitHeight = Math.floor(svgHeight / d3.max(d));

        var barChartSVG = d3.select(this).append("svg")
            .attr( "width", svgWidth)
            .attr("height", svgHeight);

        barChartSVG.selectAll("rect")
              .data(d)
          .enter().append("rect")
              .attr("x", function (d, i) { return i * barXSpacing; })
              .attr("y", 0)
              .attr("width", function(d, i) { return barXSpacing - 2; })
              .attr("height", function(d) { return d * barUnitHeight; })

    });
}
```

DISCUSS:
BAR SPACING
BAR UNIT HEIGHT
D3 PATTERN
RECT __DATA__

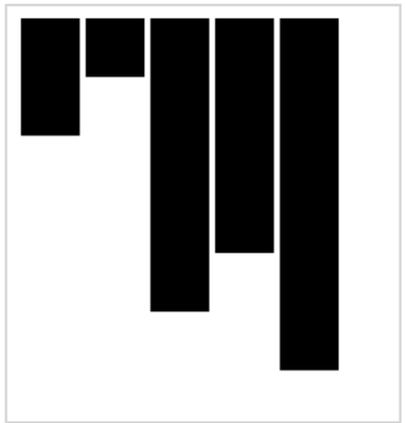
BARCHART.JS - FULL CODE

```
function barChart() {

  var svgHeight, svgWidth;

  function chart(selection) {

    selection.each(function(d, i) {

      var barXSpacing = svgWidth / d.length;
      var barUnitHeight = Math.floor(svgHeight / d3.max(d));

      var barChartSVG = d3.select(this).append("svg")
          .attr( "width", svgWidth)
          .attr("height", svgHeight);

      barChartSVG.selectAll("rect")
          .data(d)
        .enter().append("rect")
          .attr("x", function (d, i) { return i * barXSpacing; })
          .attr("y", 0)
          .attr("width", function(d, i) { return barXSpacing - 2; })
          .attr("height", function(d) { return d * barUnitHeight; })
    });}

    chart.svgWidth = function(_) {
      if (!arguments.length) return svgWidth;
      svgWidth = _;
      return chart;
    };

    chart.svgHeight = function(_) {
      if (!arguments.length) return svgHeight;
      svgHeight = _;
      return chart;
    };

    return chart;
  }
```

D3.JS FRAMEWORK - BARCHART.JS IN ACTION

```
var dataset = [2, 1, 5, 4, 6];

var myBarChart = barChart();

myBarChart
    .svgWidth(200)
    .svgHeight(200);

d3.select("body").append("div")
    .datum(dataset)
    .call(myBarChart);
```

```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<div>
      ▼<svg width="200" height="200">
          <rect x="0" y="0" width="38" height="66"></rect>
          <rect x="40" y="0" width="38" height="33"></rect>
          <rect x="80" y="0" width="38" height="165"></rect>
          <rect x="120" y="0" width="38" height="132"></rect>
          <rect x="160" y="0" width="38" height="198"></rect>
        </svg>
      </div>
      " "
  </body>
</html>
```



EXERCISE - BAR CHART WITH ONE DATA SET

Save & Use barchart.html

```
var dataset = [1, 1, 2, 3, 5, 8, 13, 8, 5, 3, 2, 1, 1];
var myBarChart = barChart();
myBarChart.svgWidth(200).svgHeight(200);
d3.select("body").append("div").datum(dataset);
d3.select("div").call(myBarChart);
```

Discuss

Datum

Explore Div, Rect Elements

EXERCISE - BAR CHART WITH MULTIPLE DATA SETS

Use barchart.html

```
var dataset = [[1, 1, 2, 3], [5, 8, 13, 8], [5, 3, 2, 1, 1]];
var myBarChart = barChart();
myBarChart.svgWidth(200).svgHeight(200);
d3.select("body").selectAll("div").data(dataset).enter().append("div");
d3.selectAll("div");
d3.selectAll("div").call(myBarChart);
```

Discuss
Datum vs Data
Explore Div, Rect Elements

EXERCISE - EXTERNAL BARCHART.JS

Save & Use barchart_external.html and barchart.js

```
var dataset = [[1, 1], [2, 3, 5, 8, 13, 8], [5, 3, 2, 1]];
var myBarChart = barChart();
myBarChart.svgWidth(200).svgHeight(200);
d3.select("body").selectAll("div").data(dataset).enter().append("div");
d3.selectAll("div");
d3.selectAll("div").call(myBarChart);
```

Discuss

Explore Div, Rect Elements

EXERCISE – D3 REUSABLE BARCHART EXAMPLE

Save & Use citybike.html and citybike_data_20131001_20131101.csv

```
// csv: http://bit.ly/barchart-csv

// Save both to the same location
// Start web server at command line
python -m SimpleHTTPServer

// Visit URL in browser
0.0.0.0:8000/citybike.html

d3.selectAll(".chartDiv");

d3.selectAll("rect");
```

Discuss
- Margins
- X & Y Scale
- X & Y Axis
- HTML Text Labels
- Transitions

Do html div creation on the fly?

SECTION 4
D3.JS LAYOUTS

Sections

DATA VISUALIZATION EXPERTS

2005: Prefuse (Java, Heer @ Berkeley)

2007: Flare (ActionScript, Heer @ Berkeley)

2009: Protovis (JavaScript, Heer & Bostock @ Stanford)

2011: D3 (JavaScript, Heer & Bostock @ Stanford)

TLDR:
very smart people thought very hard
about data visualization for a very long time

Source:
http://anna.ps/talks/fel/#/4

D3 LAYOUTS

Layouts

> Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

D3 LAYOUTS – BUNDLE

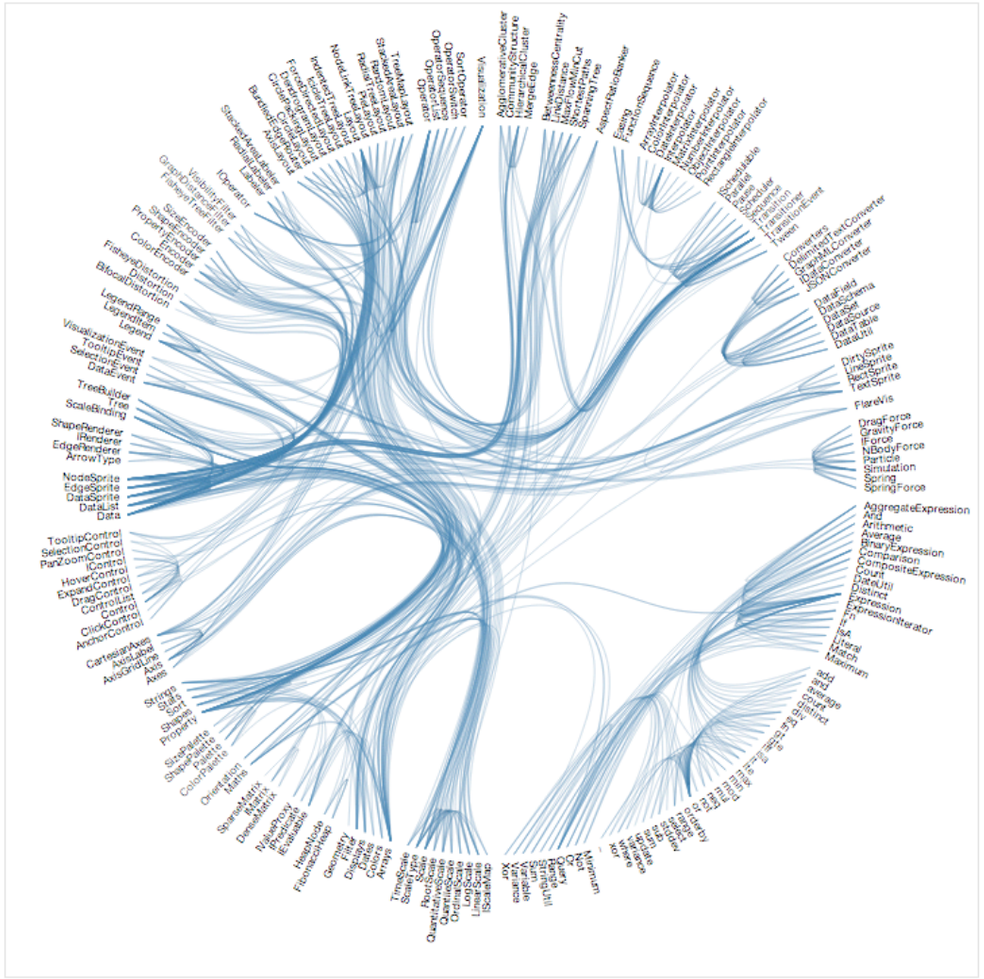
Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

Hierarchical Edge Bundling



Source:
http://bl.ocks.org/mbostock/1044242

D3 LAYOUTS – CHORD

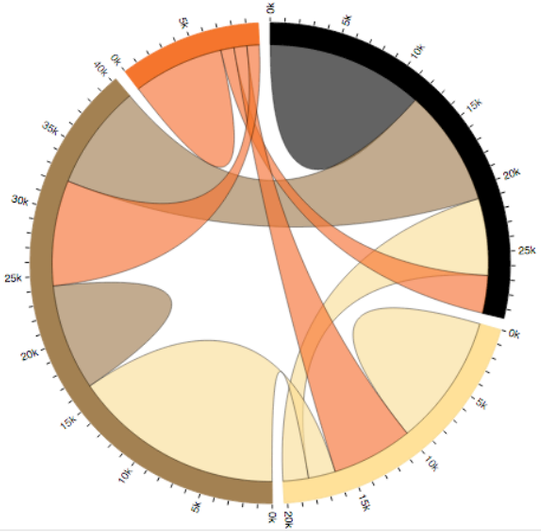
Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

Chord Diagram



Source:
http://bl.ocks.org/mbostock/4062006

D3 LAYOUTS – CLUSTER

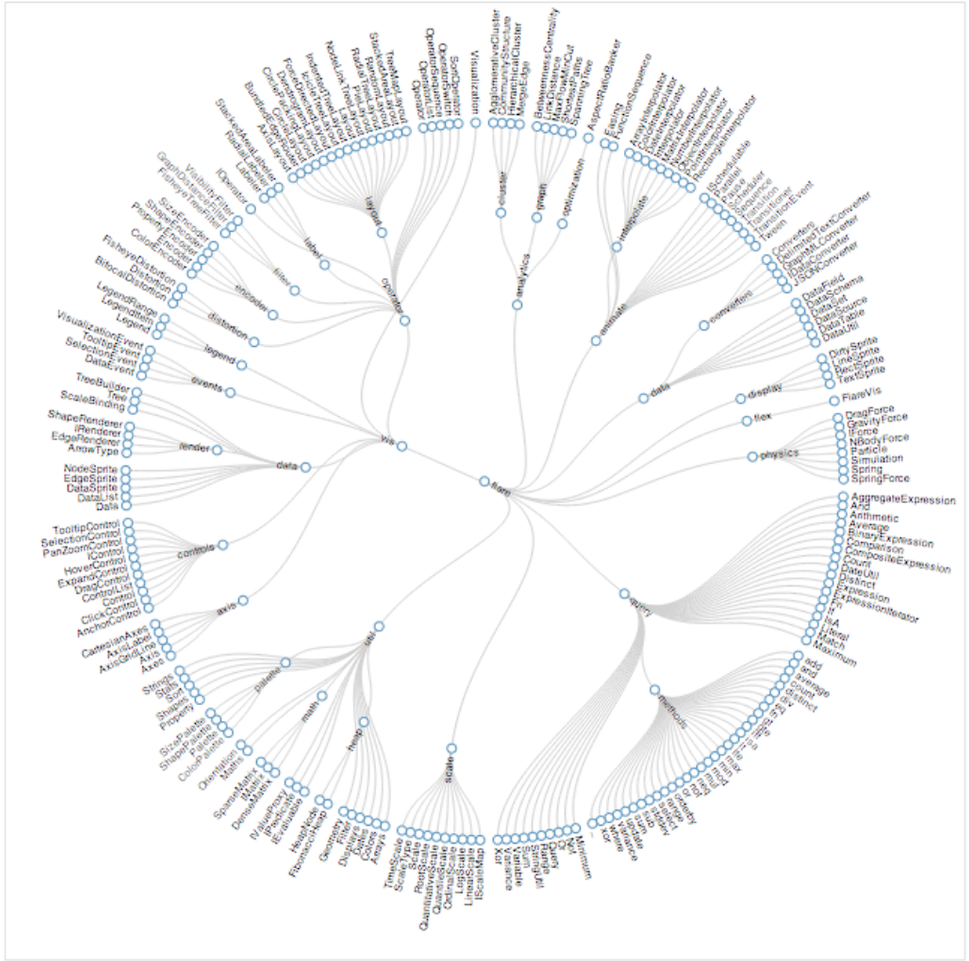
Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

Cluster Dendrogram



Source:
http://bl.ocks.org/mbostock/4339607

D3 LAYOUTS – FORCE

Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

Force-Directed Graph

Source:
http://bl.ocks.org/mbostock/4062045

D3 LAYOUTS – HIERARCHY

Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

- Cluster
- Pack
- Partition
- Tree
- Treemap

Source:
https://github.com/mbostock/d3/wiki/Hierarchy-Layout

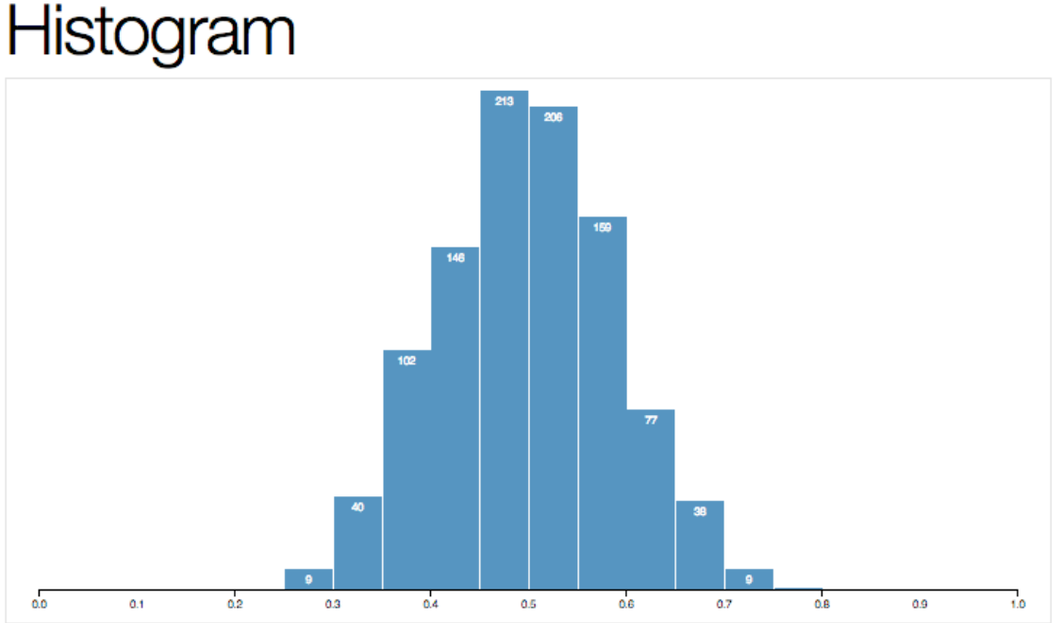
D3 LAYOUTS – HISTOGRAM

Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.



Source:
http://bl.ocks.org/mbostock/3048450

D3 LAYOUTS – PACK

Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

Circle Packing

Source:
http://bl.ocks.org/mbostock/4063530

D3 LAYOUTS – PARTITION

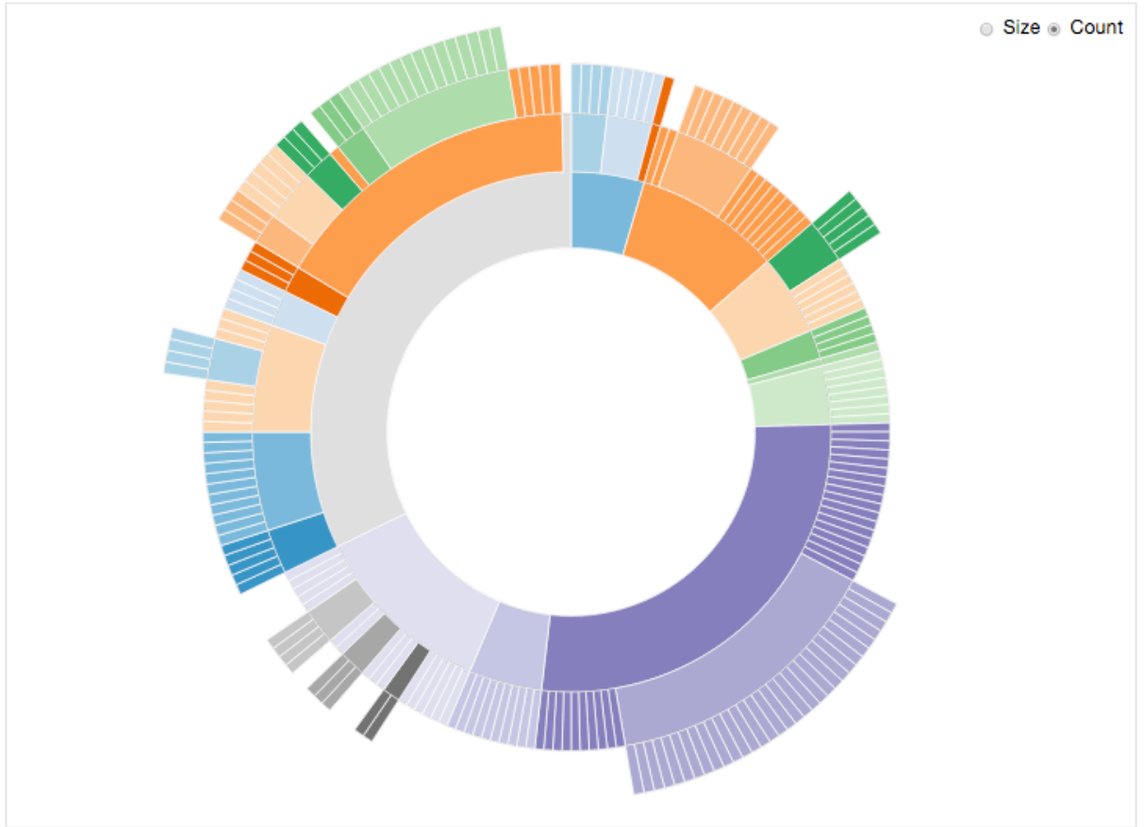
Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

Sunburst Partition



Source:
http://bl.ocks.org/mbostock/4063423

D3 LAYOUTS – PIE

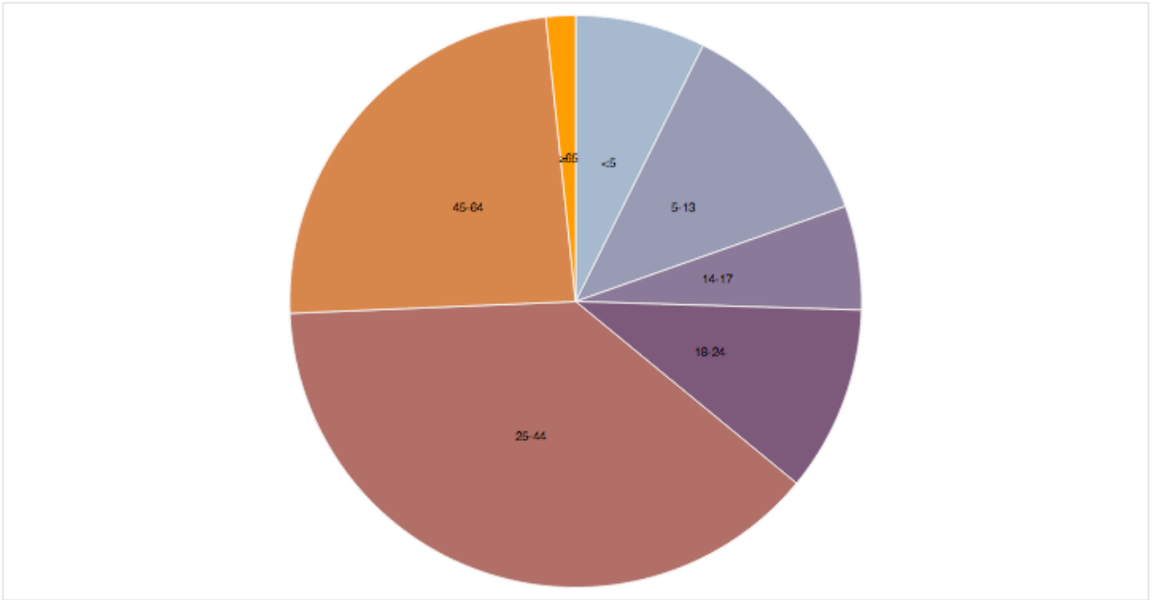
Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- **Pie - compute the start and end angles for arcs in a pie or donut chart.**
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

Pie Chart



Source:
http://bl.ocks.org/mbostock/3887235

D3 LAYOUTS – STACK

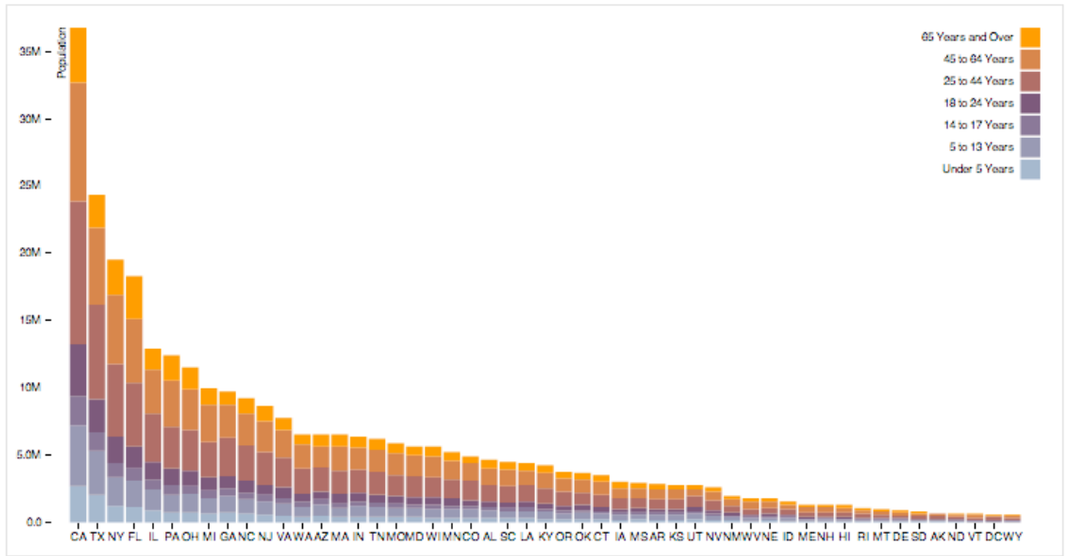
Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

Stacked Bar Chart



Source:
http://bl.ocks.org/mbostock/3886208

D3 LAYOUTS – TREE

Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

Radial Reingold–Tilford Tree

Source:
http://bl.ocks.org/mbostock/4063550

D3 LAYOUTS – TREEMAP

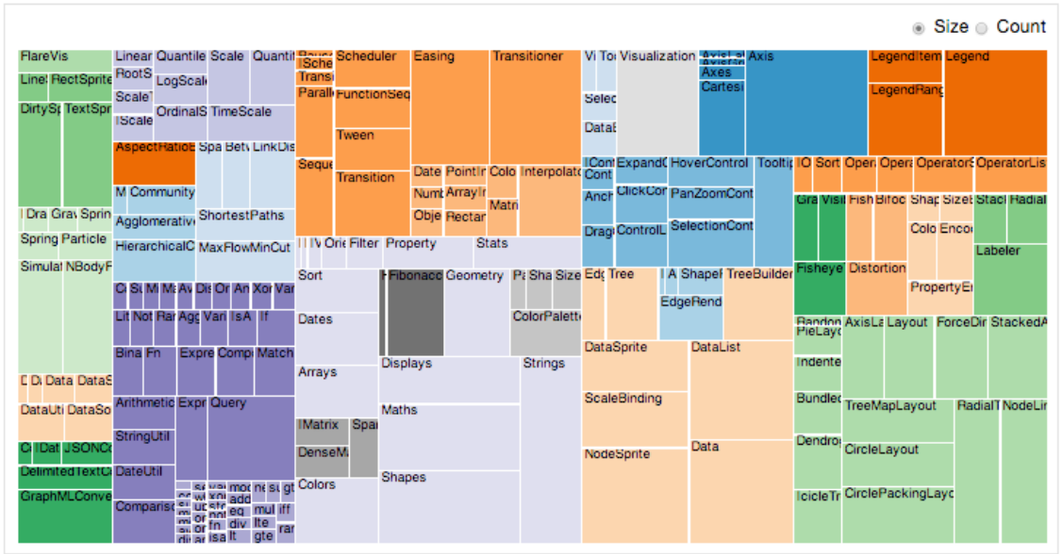
Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

Treemap



Source:
http://bl.ocks.org/mbostock/4063582

D3 PLUGINS

box	Add missing semicolons.
bullet	Link to vertical demo.
chernoff	Remove redundant svg: prefixes.
cubehelix	Restore example.
fisheye	Remove a few unused variables.
force_labels	Add missing semicolons.
geo	Slightly prettier link.
geodesic	Fix bug with flipped triangles.
geom	Update geom/contour/README.md
graph	removed traverse
hexbin	Include hexbin grid coordinates.
hive	Move to top-level directory.
horizon	Remove horizon.duration.
interpolate-zoom	Add deprecation notice.
jsonp	Fix token replacement in jsonp plugin
keybinding	Update keybinding with improvements from iD project
longscroll	Shorten.
qq	Add qq plugin.
rollup	Create README.md
sankey	Add demo links.
simplify	Remove obsolete d3.simplify plugin.
superformula	Add superformula plugin.
urlencode	Add missing semicolons.
LICENSE	Update copyright year.
README.md	Add an example (d3.svg.hive).

Sections

D3.JS AND SVG BASIC SHAPES

Shape	SVG Construction	D3 Construction
	<svg width="100" height="100"> <circle cx="35" cy="35" r="25" /> </svg>	d3.select("svg").append("circle") .attr("cx", "35").attr("cy", "35").attr("r", "25");
	<svg width="100" height="100"> <rect x="10" y="10" width="50" height="50" /> </svg>	d3.select("svg").append("rect") .attr("x", "10").attr("y", "10").attr("height", "50").attr("width", "50");
	<svg width="100" height="100"> <ellipse cx="25" cy="25" rx="15" ry="10" /> </svg>	d3.select("svg").append("ellipse") .attr("cx", "35").attr("cy", "35").attr("rx", "15").attr("ry", "25");
	<svg width="50" height="50"> <line x1="10" y1="10" x2="50" y2="50" stroke-width="3" stroke="black" /> </svg>	d3.select("svg").append("line") .attr("x1", 10).attr("y1", 10).attr("x2", 50).attr("y2", 50) .attr("stroke-width", 3).attr("stroke", "black");
Cupcake	<svg width="50" height="50"> <text x="25" y="25">Cupcake</text> </svg>	d3.select("svg").append("text") .attr("x","25").attr("y","25").text("Cupcake");
	<svg width="50" height="50"> <path d="M10,10L50,50" stroke-width="3" stroke="black"> </path> </svg>	d3.select("svg").append("path") .attr("d", "M10,10L50,50") .attr("stroke-width", 3).attr("stroke", "black");

GIVEN A SET OF POINTS -> GENERATE A PATH

Shape	SVG Construction	D3 Construction
	<svg width="50" height="50"> <path d="M10,10L50,50" stroke-width="3" stroke="black"> </path> </svg>	d3.select("svg").append("path") .attr("d", "M10,10L50,50") .attr("stroke-width", 3).attr("stroke", "black");

```
var lineData = [ { "x": 10,   "y": 10},  { "x": 50,  "y": 50}];

// "M10,10L50,50"
```

SVG PATH MINI-LANGUAGE INSTRUCTIONS

Command	Parameters	Repeatable	Explanation
Pen Command			
M (m)	x, y	Yes	**moveto** Move the pen to a new location. No line is drawn. All path data must begin with a 'moveto' command.
Line Commands			
L (l)	x, y	Yes	**lineto** Draw a line from the current point to the point (x,y).
H (h)	x	Yes	**horizontal lineto** Draw a horizontal line from the current point to x.
V (v)	y	Yes	**vertical lineto** Draw a horizontal line from the current point to y.
Cubic Bezier Curve Commands			
C (c)	x1 y1 x2 y2 x y	Yes	**curveto** Draw a cubic Bézier curve from the current point to the point (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve.
S (s)	x2 y2 x y	Yes	**shorthand/smooth curveto** Draw a cubic Bézier curve from the current point to (x,y). The first control point is assumed to be the reflection of the last control point on the previous command relative to the current point. (x2,y2) is the second control point (i.e., the control point at the end of the curve).

Command	Parameters	Repeatable	Explanation
Quadratic Bezier Curve Commands			
Q (q)	x1 y1 x y	Yes	**quadratic Bézier curveto** Draw a quadratic Bézier curve from the current point to (x,y) using (x1,y1) as the control point.
T (t)	x y	Yes	**Shorthand/smooth quadratic Bézier curveto** Draw a quadratic Bézier curve from the current point to (x,y). The control point is assumed to be the reflection of the control point on the previous command relative to the current point.
Elliptical Arc Curve Command			
A (a)	rx ry x-axis-rotation large-arc-flag sweep-flag x y	Yes	**elliptical arc** Draws an elliptical arc from the current point to (x, y). The size and orientation of the ellipse are defined by two radii (rx, ry) and an x-axis-rotation, which indicate how the ellipse as a whole is rotated relative to the current SVG coordinate system. The center (cx, cy) of the ellipse is calculated automatically to satisfy the constraints imposed by the other parameters. large-arc-flag and sweep-flag contribute to the automatic calculations and help determine how the arc is drawn.
End Path Command			
Z (z)	none	No	**closepath** Closes the path. A line is drawn from the last point to the first point drawn.

SVG LINE DATA GENERATOR & ACCESSOR FUNCTIONS

```
var lineFunction = d3.svg.line()
        .x(function(d) { return d.x; })
        .y(function(d) { return d.y; })
        .interpolate("linear");

var lineData = [ { "x": 10,   "y": 10},  { "x": 50,  "y": 50}];

lineFunction(lineData);
// returns "M10,10L50,50"
```

SVG PATH DATA GENERATOR INTERPOLATION OPTIONS

- linear - piecewise linear segments, as in a polyline.
- linear-closed - close the linear segments to form a polygon.
- step - alternate between horizontal and vertical segments, as in a step function.
- step-before - alternate between vertical and horizontal segments, as in a step function.
- step-after - alternate between horizontal and vertical segments, as in a step function.
- basis - a B-spline, with control point duplication on the ends.
- basis-open - an open B-spline; may not intersect the start or end.
- basis-closed - a closed B-spline, as in a loop.
- bundle - equivalent to *basis*, except the *tension* parameter is used to straighten the spline.
- cardinal - a Cardinal spline, with control point duplication on the ends.
- cardinal-open - an open Cardinal spline; may not intersect the start or end, but will intersect other control points.
- cardinal-closed - a closed Cardinal spline, as in a loop.
- monotone - cubic interpolation that preserves monotonicity in *y*.

Source:
https://github.com/mbostock/d3/wiki/SVG-Shapes#wiki-line

SVG LINE PATH DATA GENERATOR

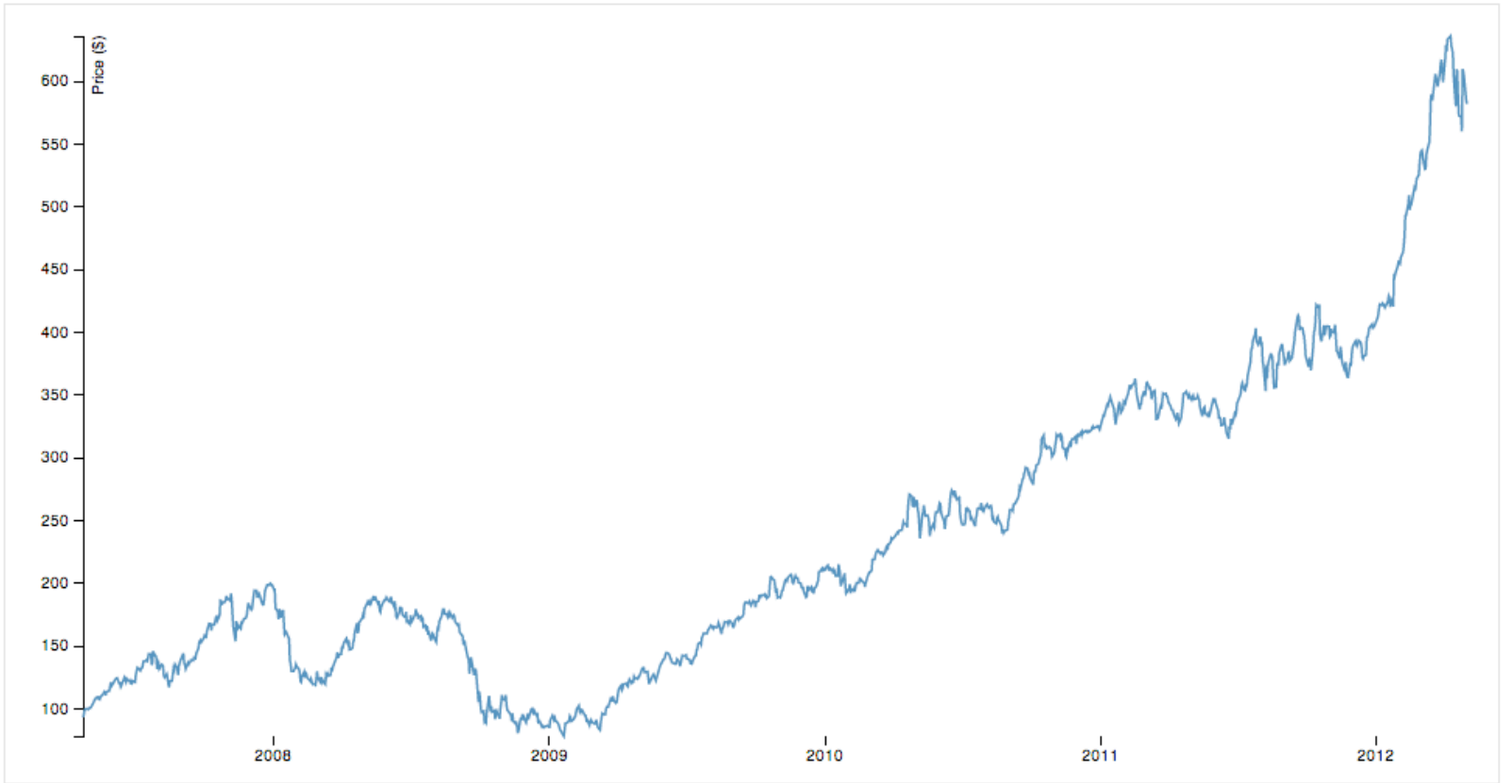
Shape	SVG Construction	D3 Construction
	<svg width="50" height="50"> <path d="M10,10L50,50" stroke-width="3" stroke="black"> </path> </svg>	d3.select("svg").append("path") .attr("d", "M10,10L50,50") .attr("stroke-width", 3).attr("stroke", "black");

```
var lineFunction = d3.svg.line()
      .x(function(d) { return d.x; })
      .y(function(d) { return d.y; })
      .interpolate("linear");

var lineData = [ { "x": 10,   "y": 10},  { "x": 50,  "y": 50}];

lineFunction(lineData);
// returns "M10,10L50,50"

d3.select("svg").append("path")
    .attr("d", lineFunction(lineData))
    .attr("stroke-width", 3).attr("stroke", "black");
```

SVG PATH DATA GENERATOR EXAMPLE

mbostock's block #3883245 October 13, 2012

Line Chart



1280
Point Objects
And Connecting
Lines
Graphed Easily

Source:
http://bl.ocks.org/mbostock/3883245

SVG PATH DATA GENERATOR EXAMPLE

```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body style>
    <script src="http://d3js.org/d3.v3.js"></script>
    ▶<script>…</script>
    ▼<svg width="960" height="500">
      ▼<g transform="translate(50,20)">
        ▶<g class="x axis" transform="translate(0,450)">…</g>
        ▶<g class="y axis">…</g>
          <path class="line" d=
          "M890,43.6266867372721 66L889.5147219193021,42.1348314
        </g>
      </svg>
    </body>
  </html>
```

Instead of 1279 SVG <line ... /> elements
It is One SVG <path ... /> element

Source:
http://bl.ocks.org/mbostock/3883245

EXERCISE – PATH GENERATOR – 2 POINTS

Use base.html

```
var lineFunction = d3.svg.line()
        .x(function(d) { return d.x; })
        .y(function(d) { return d.y; })
        .interpolate("linear");

var lineData = [{x: 5, y: 5}, {x: 55, y: 40}];

d3.select("body").append("svg").append("path")
    .attr("d", lineFunction(lineData))
    .attr("stroke-width", 3).attr("stroke", "black");
```

- linear
- linear-closed
- step
- step-before
- step-after
- basis
- basis-open
- basis-closed
- bundle
- cardinal
- cardinal-open
- cardinal-closed
- monotone

Discuss

Interpolations & Path

EXERCISE – MANY POINTS PATH GENERATOR

Use base.html

```
var lineFunction = d3.svg.line()
        .x(function(d) { return d.x; })
        .y(function(d) { return d.y; })
        .interpolate("linear");

var lineData = [{x:10, y: 10}, {x:50, y: 210}, { x:80, y:20}];

d3.select("body").append("svg").append("path")
    .attr("d", lineFunction(lineData))
    .attr("stroke-width", 5)
    .attr("stroke", "blue").attr("fill","none");
```

- linear
- linear-closed
- step
- step-before
- step-after
- basis
- basis-open
- basis-closed
- bundle
- cardinal
- cardinal-open
- cardinal-closed
- monotone

Discuss

Interpolations & Path

D3.JS SVG SHAPE GENERATOR – ARC

```
var arc = d3.svg.arc()
    .innerRadius(20)
    .outerRadius(100)
    .startAngle(function(d, i) { return d.start; })
    .endAngle(function(d, i) { return d.start + d.size; });

var data = [{start: 0, size: 1.57079633}];

arc(data[0]);
// returns "M0,-100A100,100 0 0,1 100,3.2051035159241787e-7L20,6.410207031848357e-8A20,20 0 0,0 0,-20Z"
```

Pixels from Center of "Circle"

Radians From Top of "Circle"

Source: https://github.com/mbostock/d3/wiki/SVG-Shapes

SVG ARC MINI-LANGUAGE INSTRUCTIONS

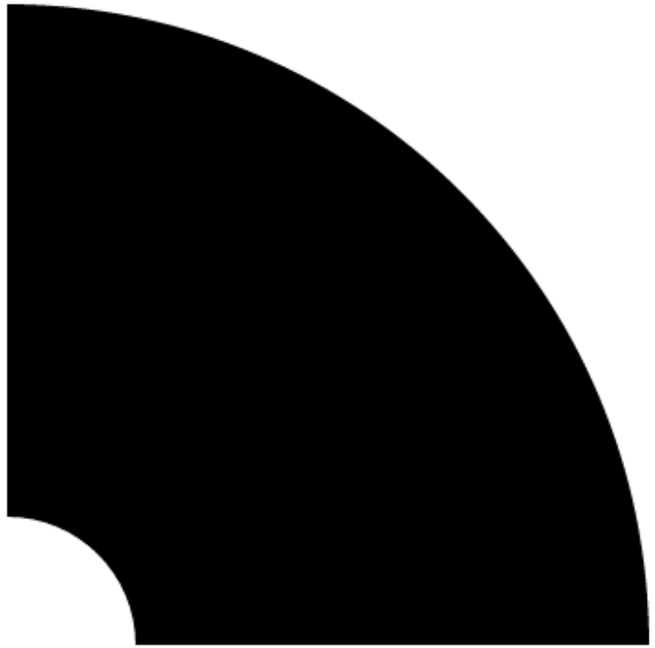
"M0,-100A100,100 0 0,1 100,3.205103515924178 7e-7L20,6.41020703184835 7e-8A20,20 0 0,0 0,-20Z"

Command	Parameters	Repeatable	Explanation
Pen Command			
M (m)	x, y	Yes	**moveto** Move the pen to a new location. No line is drawn. All path data must begin with a 'moveto' command.
Line Commands			
L (l)	x, y	Yes	**lineto** Draw a line from the current point to the point (x,y).
H (h)	x	Yes	**horizontal lineto** Draw a horizontal line from the current point to x.
V (v)	y	Yes	**vertical lineto** Draw a horizontal line from the current point to y.
Cubic Bezier Curve Commands			
C (c)	x1 y1 x2 y2 x y	Yes	**curveto** Draw a cubic Bézier curve from the current point to the point (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve.
S (s)	x2 y2 x y	Yes	**shorthand/smooth curveto** Draw a cubic Bézier curve from the current point to (x,y). The first control point is assumed to be the reflection of the last control point on the previous command relative to the current point. (x2,y2) is the second control point (i.e., the control point at the end of the curve).

Command	Parameters	Repeatable	Explanation
Quadratic Bezier Curve Commands			
Q (q)	x1 y1 x y	Yes	**quadratic Bézier curveto** Draw a quadratic Bézier curve from the current point to (x,y) using (x1,y1) as the control point.
T (t)	x y	Yes	**Shorthand/smooth quadratic Bézier curveto** Draw a quadratic Bézier curve from the current point to (x,y). The control point is assumed to be the reflection of the control point on the previous command relative to the current point.
Elliptical Arc Curve Command			
A (a)	rx ry x-axis-rotation large-arc-flag sweep-flag x y	Yes	**elliptical arc** Draws an elliptical arc from the current point to (x, y). The size and orientation of the ellipse are defined by two radii (rx, ry) and an x-axis-rotation, which indicate how the ellipse as a whole is rotated relative to the current SVG coordinate system. The center (cx, cy) of the ellipse is calculated automatically to satisfy the constraints imposed by the other parameters. large-arc-flag and sweep-flag contribute to the automatic calculations and help determine how the arc is drawn.
End Path Command			
Z (z)	none	No	**closepath** Closes the path. A line is drawn from the last point to the first point drawn.

D3.JS SVG ARC SIMPLE EXAMPLE

```
var data = [{start: 0, size: 1.57079633}];

var arc = d3.svg.arc()
    .innerRadius(20)
    .outerRadius(100)
    .startAngle(function(d, i) { return d.start; })
    .endAngle(function(d, i) { return d.start + d.size; });

var chart = d3.select("body").append("svg")
    .attr("class", "chart")
  .append("g")
    .attr("transform", "translate(0,100)");

chart.selectAll("path")
    .data(data)
  .enter().append("path")
    .attr("d", arc);
```



Discuss

Why transform, translate?

EXERCISE – SVG ARC GENERATOR – SINGLE ARC

Use base.html

```
var data = [{start: 0, size: 3.14159}];

var arc = d3.svg.arc()
    .innerRadius(80)
    .outerRadius(100)
    .startAngle(function(d, i) { return d.start; })
    .endAngle(function(d, i) { return d.start + d.size; });

var chart = d3.select("body").append("svg")
    .attr("class", "chart")
  .append("g")
    .attr("transform", "translate(50,100)");

chart.selectAll("path")
    .data(data)
  .enter().append("path")
    .attr("d", arc);
```

- 80, 100
- 100, 80
- 80, 80
- 0, 80
- -80, 0
- -80, 80

Discuss

Radius, Angles, Size, arc

EXERCISE – SVG ARC GENERATOR – MULTI ARC

Use base.html

```
var data = [{start: 0, size: 3.14159}, {start: 3.92694, size: 1.57079}];

var arc = d3.svg.arc()
    .innerRadius(80)
    .outerRadius(100)
    .startAngle(function(d, i) { return d.start; })
    .endAngle(function(d, i) { return d.start + d.size; });

var chart = d3.select("body").append("svg")
    .attr("class", "chart")
  .append("g")
    .attr("transform", "translate(150,100)");

chart.selectAll("path")
    .data(data)
  .enter().append("path")
    .attr("d", arc);
```

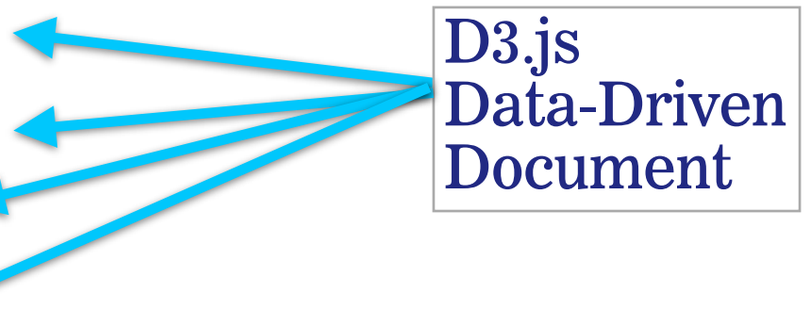
- 80, 100
- 100, 80
- 80, 80
- 0, 80
- -80, 0
- -80, 80

Discuss

Radius, Angles, Size, arc

D3.JS SVG SHAPE GENERATOR – DATA-DRIVEN

```
var arc = d3.svg.arc()
    .innerRadius(function(d, i) { return ...; })
    .outerRadius(function(d, i) { return ...; })
    .startAngle(function(d, i) { return ...; })
    .endAngle(function(d, i) { return ...; });
```



Source:
https://github.com/mbostock/d3/wiki/SVG-Shapes

EXERCISE – SVG ARC GENERATOR – RADII

Use base.html

```
var data = [ {start: 0, size: 2,     inner: 50, outer: 100, color: "green"},
             {start: 2, size: 2,     inner: 25, outer: 150, color: "blue"},
             {start: 4, size: 2.28, inner: 75, outer: 125, color: "red"}];

var arc = d3.svg.arc()
   .innerRadius(function(d, i) { return d.inner; })
   .outerRadius(function(d, i) { return d.outer; })
   .startAngle(function(d, i) { return d.start; })
   .endAngle(function(d, i) { return d.start + d.size; });

var chart = d3.select("body").append("svg")
   .attr("class", "chart")
  .append("g")
   .attr("transform", "translate(150,200)");

chart.selectAll("path")
   .data(data)
  .enter().append("path")
   .style("fill", function(d, i) { return d.color; })
   .attr("d", arc);
```

Change
- inner
- outer

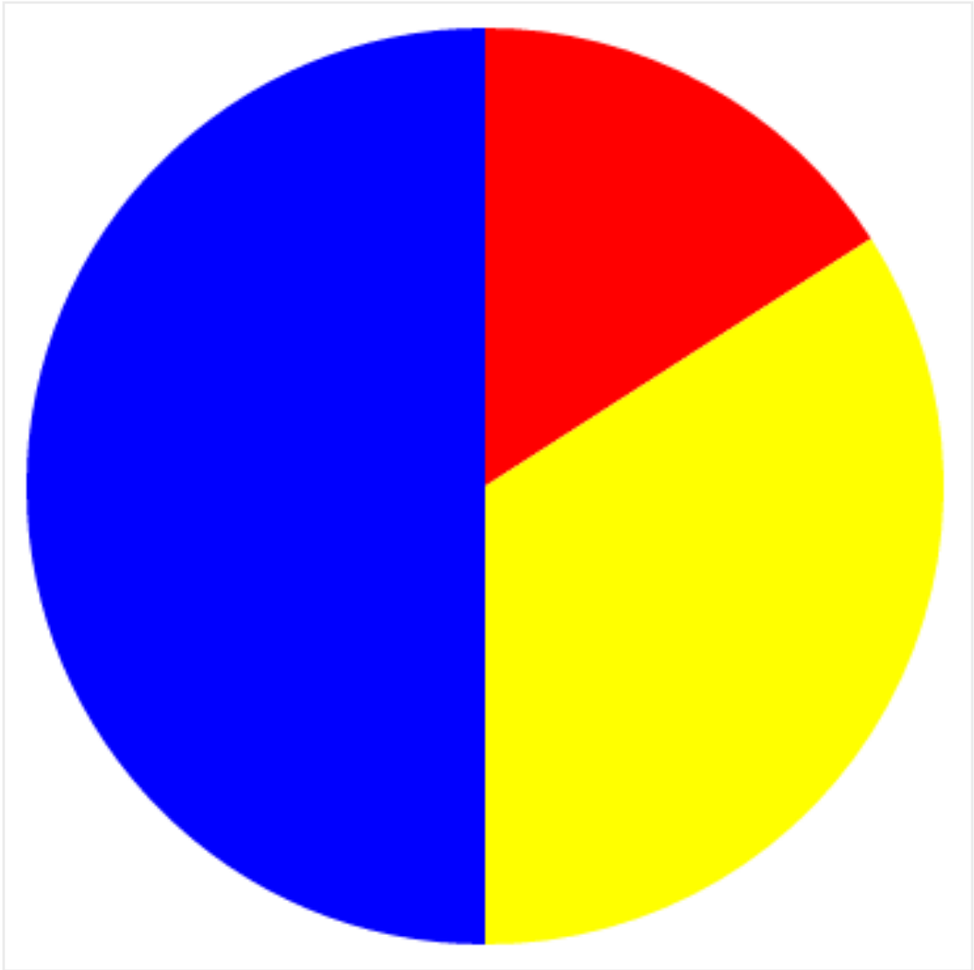
Discuss

D3 Operators, inner, outer

Sections

A PIE CHART IS MADE UP OF ARCS

```
var data = [
    {start: 0, size: 1, color: "red"},
    {start: 1, size: 2.14159265, color: "yellow"},
    {start: 3.14159265, size: 3.14159266, color: "blue"}];

var arc = d3.svg.arc()
      .innerRadius(0)
      .outerRadius(100)
      .startAngle(function(d, i) { return d.start; })
      .endAngle(function(d, i) { return d.start + d.size; });

var chart = d3.select("body").append("svg")
      .append("g")
      .attr("transform", "translate(100,100)");

chart.selectAll("path")
      .data(data)
    .enter().append("path")
      .style("fill", function(d, i) { return d.color; })
      .attr("d", arc);
```



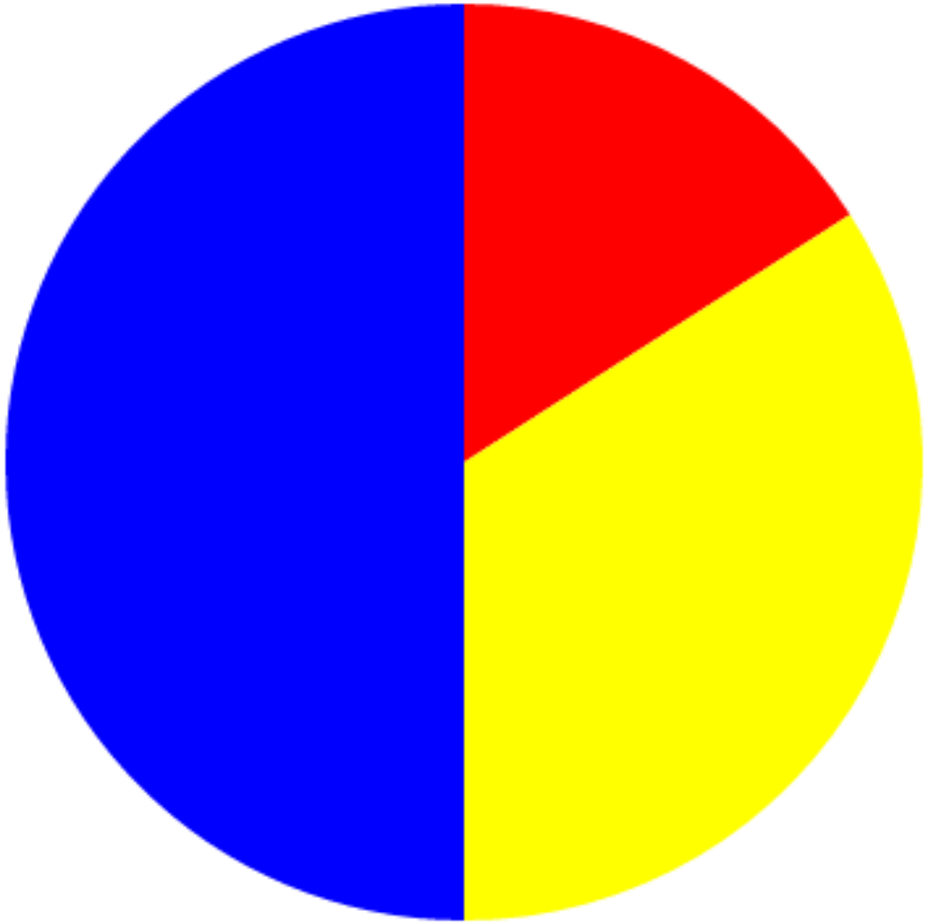
PIE CHART – LET D3.JS DO THE MATH FOR US

Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.



PIE LAYOUT – COMPUTES ARC ANGLES FROM DATA

```
> var data = [ {amount: 1, color: "red"  },
               {amount: 2, color: "green"},
               {amount: 3, color: "blue" }];
  undefined
> var pie = d3.layout.pie()
      .sort(null)
      .value(function(d) { return d.amount; });
  undefined
> pie(data)
  [▼Object i                                , ▼Object i                                , ▼Object i                                ]
     ▶data: Object                             ▶data: Object                             ▶data: Object
      endAngle: 1.0471975511965976              endAngle: 3.141592653589793               endAngle: 6.283185307179586
      startAngle: 0                             startAngle: 1.0471975511965976            startAngle: 3.141592653589793
      value: 1                                  value: 2                                  value: 3
     ▶__proto__: Object                        ▶__proto__: Object                        ▶__proto__: Object
>
```

EXERCISE – PIE LAYOUT ARC GENERATOR

Use base.html

```
var data = [
    {amount: 0.75},
    {amount: 2.5},
    {amount: 3.75}
];

var pie = d3.layout.pie()
      .sort(null)
      .value(function(d) { return d.amount; });

console.table(pie(data));
```

- 0.75, 2.5, 3.75
- 1, 2, 3
- 0, 3, 0
- 1, 20, 50
- 108, 109, 110

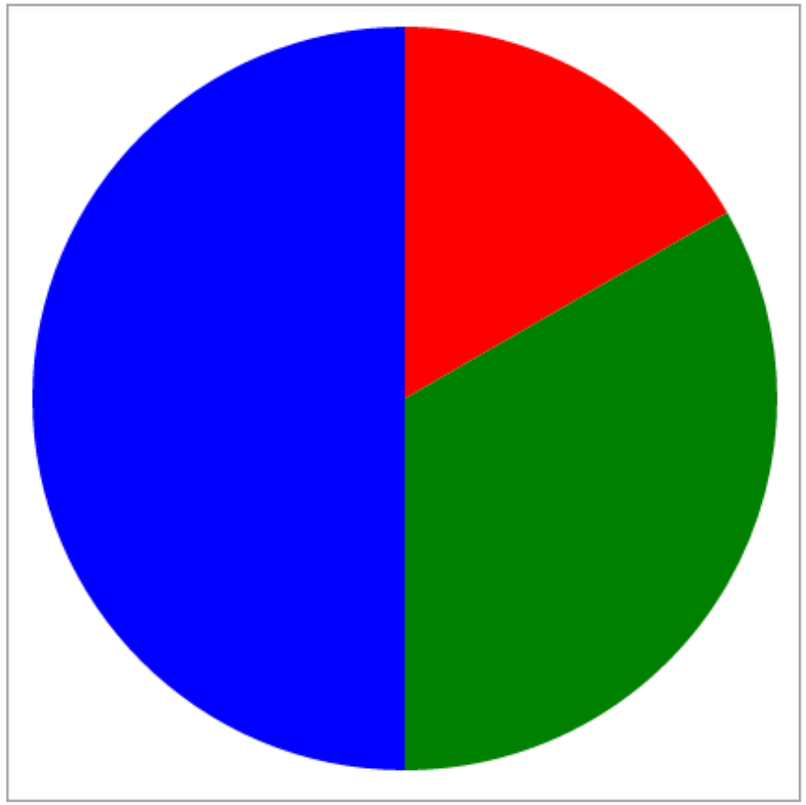
null,
function(d) { return d.amount; }

Discuss

sort, sort fn, console.table

PIE CHART = PIE LAYOUT + SVG ARC GENERATOR

```
var data = [ {amount: 1, color: "red"  },
             {amount: 2, color: "green"},
             {amount: 3, color: "blue" }];

var arc = d3.svg.arc()
    .innerRadius(0)
    .outerRadius(100);

var pie = d3.layout.pie()
    .sort(null)
    .value(function(d) { return d.amount; });

var chart = d3.select("body").append("svg")
  .append("g")
    .attr("transform", "translate(100,100)");

var arcGs = chart.selectAll(".arc")
    .data(pie(data))
  .enter().append("g")
    .attr("class", "arc");

arcGs.append("path")
    .attr("d", arc)
    .style("fill", function(d, i) { return d.data.color; });
```



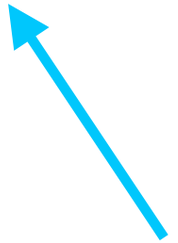
Discuss

arc vs d.data.color

EXERCISE – PIE LAYOUT PIE CHART GENERATION

Use base.html

```
var data = [ {amount: 10, color: "red"  }, {amount: 20, color: "green"}, {amount: 5, color: "blue" }];

var arc = d3.svg.arc()
    .innerRadius(0)
    .outerRadius(100);

var pie = d3.layout.pie()
    .sort(null)
    .value(function(d) { return d.amount; });

var chart = d3.select("body").append("svg").append("g")
    .attr("transform", "translate(100,100)");

var arcGs = chart.selectAll(".arc")
    .data(pie(data))
  .enter().append("g");

arcGs.append("path")
    .attr("d", arc)
    .style("fill", function(d, i) { return d.data.color; });
```



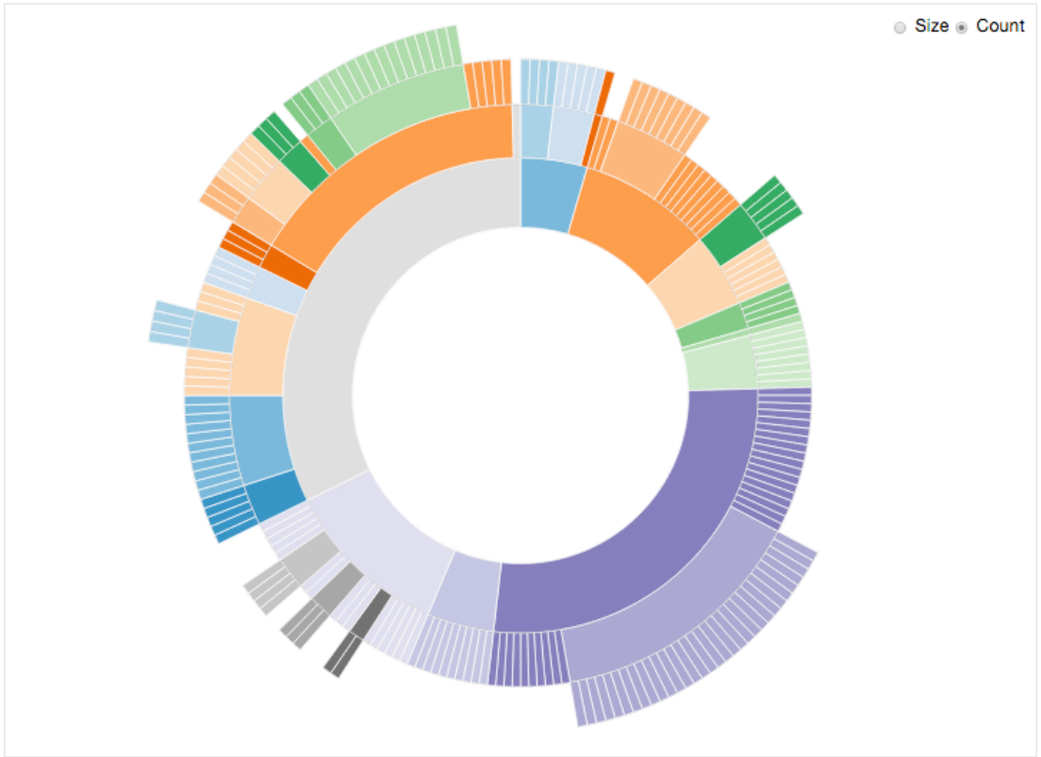
- 0.75, 2.5, 3.75
- 1, 2, 3
- 0, 3, 0
- 1, 20, 50
- 108, 109, 110

Discuss

Radius, Angles, Size, arc

(THOUGHT) EXERCISE – SUNBURST PARTITION

Sunburst Partition



Source:
http://bl.ocks.org/mbostock/4063423

Discuss

Pie Chart Generator?

SECTION 5
D3.JS BEHAVIORS

Sections

D3.JS DRAG BEHAVIOR

This behavior creates event listeners to handle drag gestures
Both mouse and touch events are supported.

```
d3.behavior.drag()
    .on("dragstart", function(){ ... } )
    .on("drag",      function(){ ... } )
    .on("dragend",   function(){ ... } );
```

Discuss
dragstart, dragend optional

D3.JS DRAG BEHAVIOR – D3.JS GIVES US D3.EVENT

Drag Event (not dragstart nor dragend) give us:

d3.event.x
Returns horizontal coordinate of the event relative to whole document

d3.event.y
Returns vertical coordinate of the event relative to whole document

D3.JS DRAG BEHAVIOR – WHAT ARE WE DRAGGING?

d3.behavior.drag() returns a function.
To use it we have to call it on some D3.js Selection.

```
var drag = d3.behavior.drag()
    .on("drag", function(){ ... } );

svg.append("circle")
    .attr("cx", 25)
    .attr("cy", 25)
    .attr("r", 25)
    .call(drag);
```

EXERCISE – BASIC DRAG

Use base.html

```
var svg = d3.select("body").append("svg");

function dragmove(d) {
    d3.select(this)
        .attr("cx", d.x = d3.event.x)
        .attr("cy", d.y = d3.event.y);
}

var drag = d3.behavior.drag()
    .origin(Object)
    .on("drag", dragmove);

svg.selectAll("circle")
    .data([{x: 25, y: 25}])
  .enter().append("circle")
    .attr("r", 25)
    .attr("cx", function(d, i) { return d.x; })
    .attr("cy", function(d, i) { return d.y; })
    .call(drag);
```

Try removing
- .origin(Object)
- Then drag circle from the edge

Discuss

.origin(Object)

EXERCISE – DRAG THE CIRCLE

Save & Use drag.html

// Read code & change things to make sure you understand it

// How would you do it with SVG Rectangle?

// How would you do it with SVG Line?

// How would you do it if you wanted to move several SVG Shapes at once?

Discuss
Questions

EXERCISE – DRAG THE CIRCLE WITH BOUNDARIES

Save & Use drag_with_boundaries.html

// Read code & change things to make sure you understand it

// How would you do it with SVG Rectangle?

// How would you do it with SVG Line?

// How would you do it if you wanted to move several SVG Shapes at once?

Discuss
Questions

EXERCISE – DRAG THE CIRCLE WITH 3 DRAG EVENTS

Save & Use drag_with_three_events.html

// Read code & change things to make sure you understand it

// How would you do it with SVG Rectangle?

// How would you do it with SVG Line?

// How would you do it if you wanted to move several SVG Shapes at once?

Discuss
Questions

Sections

D3.JS ZOOM BEHAVIOR

This behavior creates event listeners to handle zooming and panning. Both mouse and touch events are supported.

```
d3.behavior.zoom()
    .on("zoomstart", function(){ ... } )
    .on("zoom",      function(){ ... } )
    .on("zoomend",   function(){ ... } );
```

D3.JS ZOOM BEHAVIOR – WHAT ARE WE ZOOMING?

d3.behavior.zoom() returns a function.
To use it we have to call it on some D3.js Selection.

```
var zoom = d3.behavior.zoom()
    .on("zoom", function() { ... });

svgViewport.call(zoom);
```

D3.JS ZOOM BEHAVIOR – D3.JS GIVES US D3.EVENT

Zoom Event (not dragstart nor dragend) give us:

d3.event.translate
Returns a 2-element array representing current translation vector

d3.event.scale
Returns a number which represents the current zoom scale
[0 - n]
# is multiplicative in scale.
2 = everything is twice as big
0.2 = everything is 5 times smaller

D3.JS ZOOM BEHAVIOR – SCALES X & Y AXIS FOR US

Zoom Event can automatically change the domain of the X & Y scales

```
var zoom = d3.behavior.zoom()
    .x(xAxisScale)
    .y(yAxisScale)
    .on("zoom", function () { ... });
```

Note that we have to redraw the axis again in the zoom event function:

```
innerSpace.call(xAxis);
innerSpace.call(yAxis);
```

D3.JS ZOOM BEHAVIOR – LETS US CLAMP ZOOMING

Zoom Event lets us define how far out and in we can zoom in...

```
var zoom = d3.behavior.zoom()
    .scaleExtent([0.2, 10])
    .on("zoom", function () { ... });
```

1st number is the lower bound
2nd number is the upper bound
# is multiplicative in scale.
2 = everything is twice as big
0.2 = everything is 5 times smaller

D3.JS ZOOM BEHAVIOR – REDRAW OBJECTS

Zoom Event

- Updates X scale for us
- Updates Y scale for us
- Tell us Zoom scale

Using this, make sure to redraw objects:
- axis
- any visual element you want zoomed

EXERCISE – CIRCLE ZOOM

Save & Use zoom.html

// Read code & change things to make sure you understand it

// How would you do it with SVG Rectangle?

// How would you do it with SVG Line?

// How would you do it if you wanted to move several SVG Shapes at once?

Discuss
Questions

D3.JS ZOOM BEHAVIOR – MOST FAQ – EMPTY SPACE

We attach a zoom event to an SVG element selection...

How do we account for someone trying to zoom in on an empty space?

Secret Hidden Rectangle to Fully Capture ALL Zoom Events

```
var zoom = d3.behavior.zoom()
    .on("zoom", function() { ... });

svgViewport.call(zoom);

svgViewport.append( .... HIDDEN RECTANGLE ... )
```

EXERCISE – CIRCLE ZOOM WITH HIDDEN RECTANGLE

Save & Use zoom_secret.html

Look for // Hidden Rectangle to Fully Capture Zoom Events

// Read code & change things to make sure you understand it

Discuss
Questions

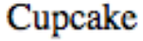
SECTION 6
D3.JS MAPPING

Sections

SVG SHAPES

Shape	SVG Construction	D3 Construction
	<svg width="100" height="100"> <circle cx="35" cy="35" r="25" /> </svg>	d3.select("svg").append("circle") .attr("cx", "35").attr("cy", "35").attr("r", "25");
	<svg width="100" height="100"> <rect x="10" y="10" width="50" height="50" /> </svg>	d3.select("svg").append("rect") .attr("x", "10").attr("y", "10").attr("height", "50").attr("width", "50");
	<svg width="100" height="100"> <ellipse cx="25" cy="25" rx="15" ry="10" /> </svg>	d3.select("svg").append("ellipse") .attr("cx", "35").attr("cy", "35").attr("rx", "15").attr("ry", "25");

	<svg width="50" height="50">
	  <line x1="10" y1="10" x2="50" y2="50"
	  stroke-width="3" stroke="black" />
	</svg>
		d3.select("svg").append("line")
		  .attr("x1", 10).attr("y1", 10).attr("x2", 50).attr("y2", 50)
		  .attr("stroke-width", 3).attr("stroke", "black");

	<svg width="50" height="50"> <text x="25" y="25">Cupcake</text> </svg>	d3.select("svg").append("text") .attr("x","25").attr("y","25").text("Cupcake");

	<svg width="50" height="50">
	  <path d="M10,10L50,50" stroke-width="3" stroke="black">
	  </path>
	</svg>
		d3.select("svg").append("path")
		  .attr("d", "M10,10L50,50")
		  .attr("stroke-width", 3).attr("stroke", "black");

SVG MAP IN A SINGLE PATH VARIABLE DECLARATION

```
// Draw Mexico Path
var mexicoPath = svg.append("path")
    .attr("d", mexicoPath)
    .attr("fill", "gray");
```

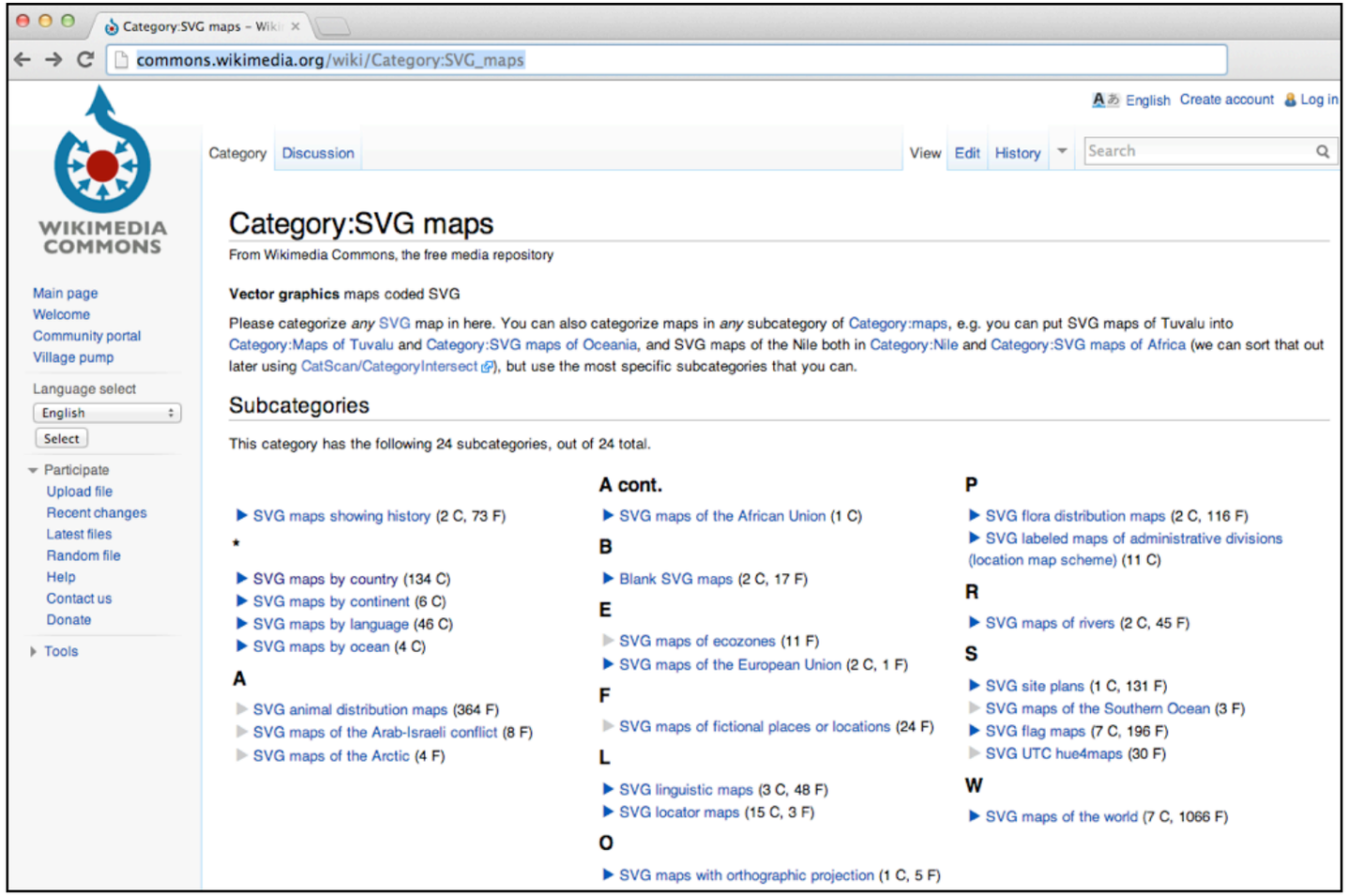
SVG Path Source:
http://upload.wikimedia.org/wikipedia/commons/c/c5/Blank_Mexico_map%2C_no_States.svg

MEXICO SVG PATH D VARIABLE

var mexicoPath = "m 62.249981,20.911255 c -0.03654,0.191337 -0.07479,0.361557 -0.125,0.5625 -0.328474,0.109414 -0.01563,0.07809 -0.343751,0.1875 l -0.15625,0.34375 -0.5,0 -0.34375,0.3125 0,3.71875 c 0.267173,0.200416 0.166212,0.15625 0.5,0.15625 l [illegible] -0.031,-0.54282 -7.81e-4,-2.71494 -0.5625,-2.6875 z"

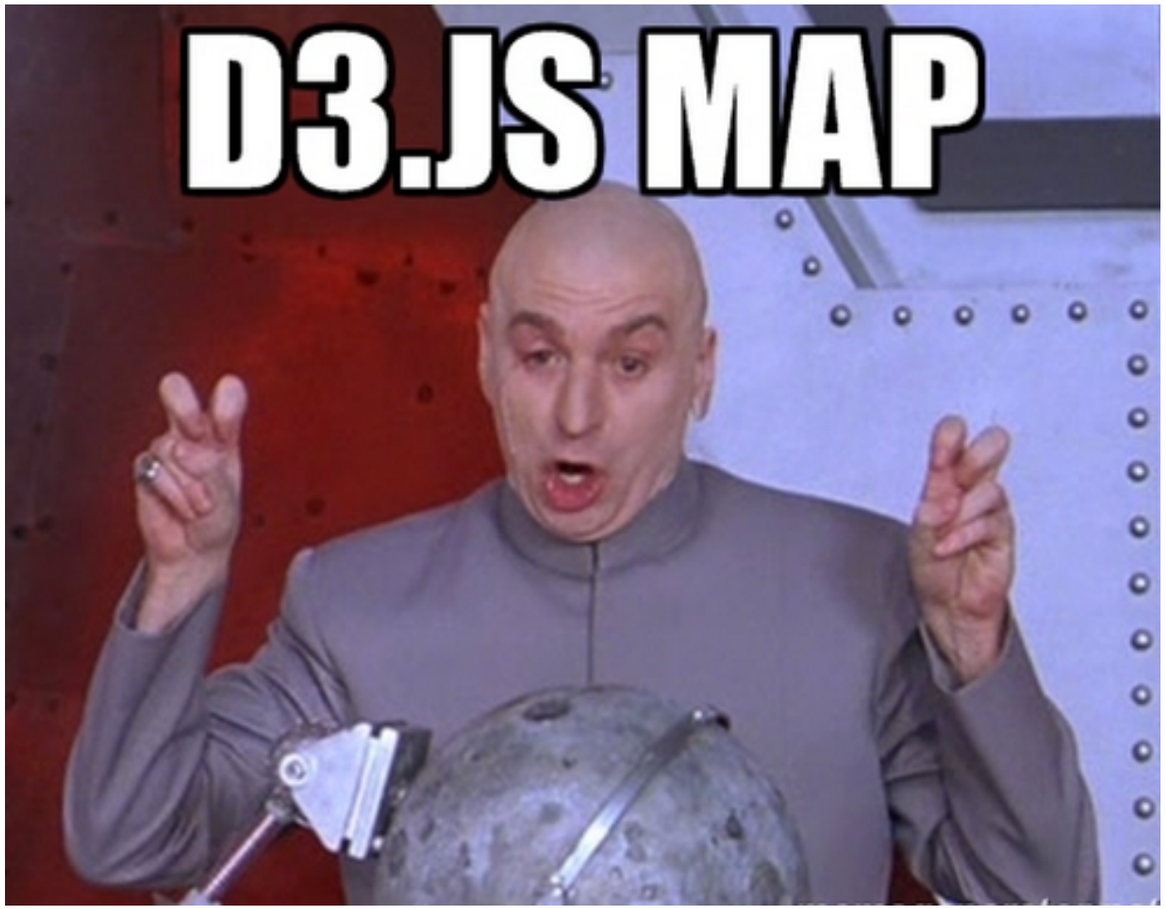
SVG Path Source: http://upload.wikimedia.org/wikipedia/commons/c/c5/Blank_Mexico_map%2C_no_States.svg

FIND SVG MAPS ON WIKIMEDIA (SEARCH SVG MAPS)



SVG Wikimedia Maps: http://commons.wikimedia.org/wiki/Category:SVG_maps

SIMPLEST D3.JS MAP POSSIBLE



Dr. Evil Meme: http://memegenerator.net

Used D3.js,
but
it was really SVG

EXERCISE – SEARCH FOR SVG MAP ON WIKIPEDIA

Go to google.com

In search field type: site:wikimedia.org [country name] SVG

Keep clicking into link & picture until you see the actual country

Right click and view source - then look for SVG Path with d attribute.

Discuss

Did you find it?

Sections

GEOJSON IS A FORMAT FOR ENCODING GEOGRAPHY

GeoJSON is a standard for defining Geographic Data Structures

```
// GeoJSON Format for a Latitude and Longitude Points
{
   "type": "Point",
   "coordinates": [ -105.01621, 39.57422]
}
```

Note: coordinates are used x, y coordinates -> so Latitude goes first and then Longitude.
Grid is based on 0,0 so positive X means moving East and positive Y means moving North.

SVG Path Source:
http://geojsonlint.com/

GEOJSON: ONE OF MANY GEOGRAPHIC DATA FORMATS

Popular Geo-Spatial Data Formats:

- Shapefiles
- GPX
- KML
- GeoJSON
- TopoJSON
- SVG
- GeoTIFF
- JPEG
- ASC

Ways To Convert Between Formats:

- GDAL
- OGR
- OGRE
- GPSBabel
- Kml2gpx
- TopoJSON

SVG Path Source:
http://geojsonlint.com/

GEOJSON DEFINES FOUR MAIN THINGS

GeoJSON Standard

GeoJSON Object always consists of a single object

GeoJSON Object represents one of four things:

1. Geometry
2. Feature
3. Feature Collection
4. Geometry Collection (not as prevalent as other three)

GeoJSON Standard:
http://geojson.org

GEOJSON GEOMETRY

GeoJSON Geometry Object

```
{
  "type": "YYYYY",
  "coordinates": [ ..... ]
}
```

where YYYYY
- => "Point" => array of x and y coordinates
- => "MultiPoint" => array of many coordinates
- => "LineString" => array of many coordinates
- => "MultiLineString" => array of arrays of coordinates
- => "Polygon" => array of arrays of coordinates
- => "MultiPolygon" => multidimensional array of coordinates
- => "GeometryCollection" => collection of geometry objects

GeoJSON Standard:
http://geojson.org

GEOJSON FEATURE

GeoJSON Feature Object

```
{
  "type": "Feature",
  "geometry": { GeoJSON Geometry Object },
  "properties": { JSON Object or JSON null value }
}
```

"properties" value for a Feature object is a JSON object with non-spatial attributes or a JSON null value.

GeoJSON Standard:
http://geojson.org

GEOJSON FEATURE COLLECTION

GeoJSON Feature Collection Object

```
{
  "type": "FeatureCollection",
  "features": [ GeoJSON Feature Object(s) ]
}
```

GeoJSON Standard:
http://geojson.org

GEOJSON GEOMETRY COLLECTION

GeoJSON Feature Collection Object

GeoJSON Geometry Collection Object

```
{
   "type": "GeometryCollection",
   "geometries": [
      GeoJSON Geometry Object,
      ...
      GeoJSON Geometry Object
   ]
}
```

Feature collections allow us to encode non-spatial information, where as Geometry collections are pure spatial information.

GeoJSON Standard:
http://geojson.org

EXERCISE – GEOJSONLINT.COM

Explore GeoJSONLint.com

1) Click on Point and explore format
2) Click on MultiPoint and explore format
3) Click on LineString and explore format
4) Click on MultiLineString and explore format
5) Click on Polygon and explore format
6) Click on MultiPolygon and explore format
7) Click on Feature and explore format
8) Click on FeatureCollection and explore format
9) Click on GeometryCollection and explore format

Discuss

Did it make sense?

GeoJSON Standard:
http://geojson.org

Sections

D3.JS GEO PATH GENERATOR

D3.js Geo Path Generator is similar to d3.svg.line & other SVG Shape Generators

d3.geo.path() does math & translation for you:
-> Takes in data in GeoJSON Data Format
-> returns SVG Path Mini-Language Instructions

Note: Assumes a default Map Projection (albersUSA) and a point radius of 4.5 pixels.

d3.geo.path():
https://github.com/mbostock/d3/wiki/Geo-Paths

SVG PATH DATA GENERATOR

Shape	SVG Construction	D3 Construction
	<svg width="50" height="50"> <path d="M10,10L50,50" stroke-width="3" stroke="black"> </path> </svg>	d3.select("svg").append("path") .attr("d", "M10,10L50,50") .attr("stroke-width", 3).attr("stroke", "black");

```
var lineFunction = d3.svg.line()
                         .x(function(d) { return d.x; })
                         .y(function(d) { return d.y; })
                         .interpolate("linear");

var lineData = [ { "x": 10,   "y": 10},  { "x": 50,  "y": 50}];

lineFunction(lineData);
// returns "M10,10L50,50"

d3.select("svg").append("path")
    .attr("d", lineFunction(lineData))
    .attr("stroke-width", 3).attr("stroke", "black");
```

SVG GEO PATH DATA GENERATOR

Shape	SVG Construction	D3 Construction
	<svg width="50" height="50"> <path d="M10,10L50,50" stroke-width="3" stroke="black"> </path> </svg>	d3.select("svg").append("path") .attr("d", "M10,10L50,50") .attr("stroke-width", 3).attr("stroke", "black");

```
var geoPathGen = d3.geo.path();

var GeoJSONPoint = {
   "type": "Point",
   "coordinates": [ -105.01621, 39.57422]
}

geoPathGen(GeoJSONPoint);
// returns "M360.27654376425505,227.45060942581244m0,4.5a4.5,4.5 0 1,1 0,-9a4.5,4.5 0 1,1 0,9z"

d3.select("svg").append("path")
   .attr("d", geoPathGen(GeoJSONPoint))
   .attr("stroke-width", 3).attr("stroke", "black");
```

SVG GEO PATH DATA GENERATOR THE D3 WAY

```
d3.select("body").append("svg");

var geoPathGen = d3.geo.path();

var GeoJSONPoint = {
   "type": "Point",
   "coordinates": [ -105.01621, 39.57422]
}

d3.select("svg").selectAll("path")
    .data([GeoJSONPoint])
  .enter().append("path")
    .attr("d", geoPathGen)
    .attr("stroke-width", 3).attr("stroke", "black");
```

GEO PATH DATA GENERATOR FOR GEOJSON OBJECTS

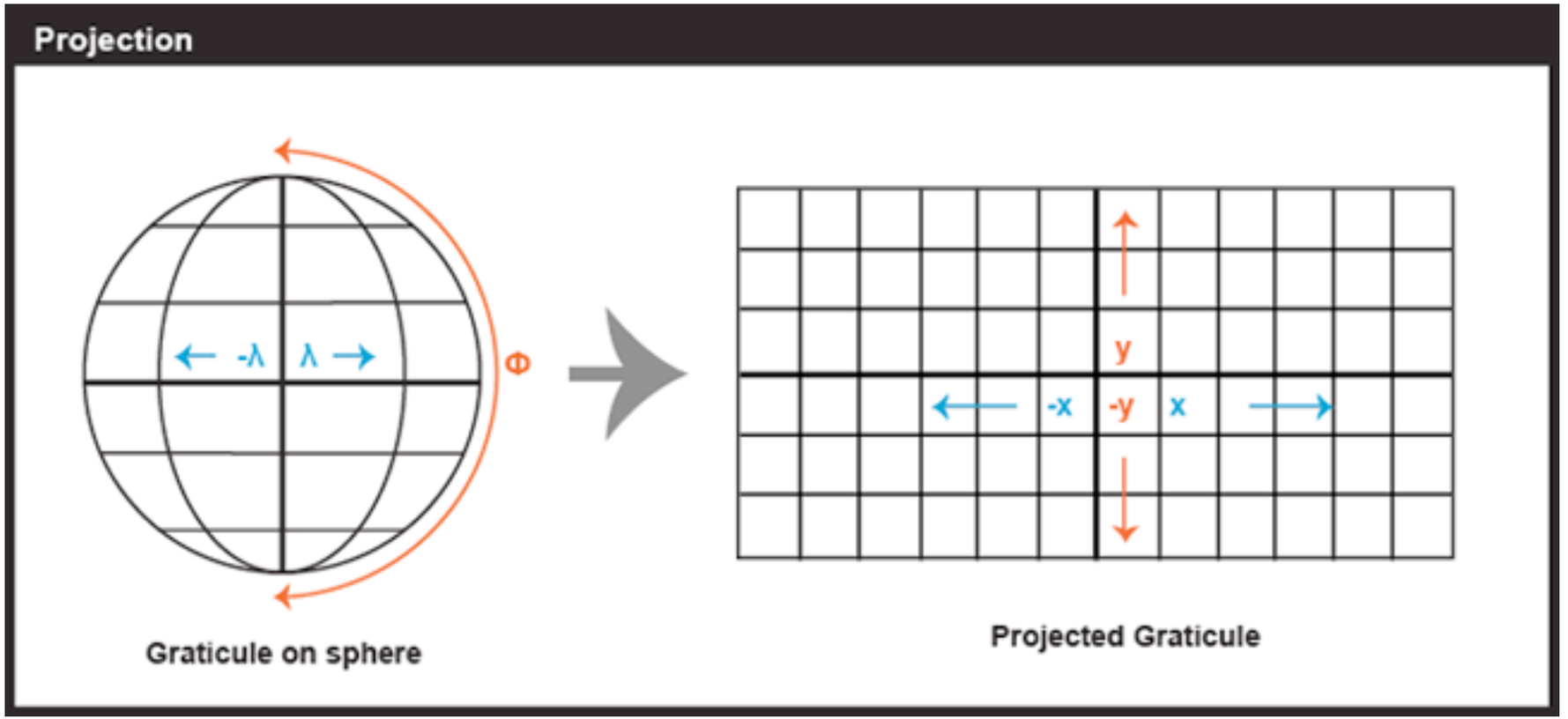
Create a distinct Path element for each object in the
- Feature Collection
- Geometry Collection

```
svg.selectAll("path")
    .data(features)
  .enter().append("path")
    .attr("d", d3.geo.path());
```

d3.geo.path():
https://github.com/mbostock/d3/wiki/Geo-Paths

Sections

D3.JS MAP PROJECTION



CHOOSING MAP PROJECTIONS IS A BIG DEAL

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page
Toolbox
Print/export
Languages
日本語
Русский
Edit links

Table of Projections [edit source | edit beta]

Projection	Images	Type	Properties	Creator	Year	Notes
Equirectangular = equidistant cylindrical = rectangular = la carte parallélogrammatique		Cylindrical	Compromise	Marinus of Tyre	120 (c.)	Simplest geometry; distances along meridians are conserved. Plate carrée: special case having the equator as the standard parallel.
Mercator = Wright		Cylindrical	Conformal	Gerardus Mercator	1569	Lines of constant bearing (rhumb lines) are straight, aiding navigation. Areas inflate with latitude, becoming so extreme that the map cannot show the poles.
Gauss–Krüger = Gauss conformal = (Ellipsoidal) Transverse Mercator		Cylindrical	Conformal	Carl Friedrich Gauss Johann Heinrich Louis Krüger	1822	This transverse, ellipsoidal form of the Mercator is finite, unlike the equatorial Mercator. Forms the basis of the Universal Transverse Mercator system.
Gall						Intended to resemble the Mercator while also displaying the poles.

57 Different Projections Listed on Wikipedia

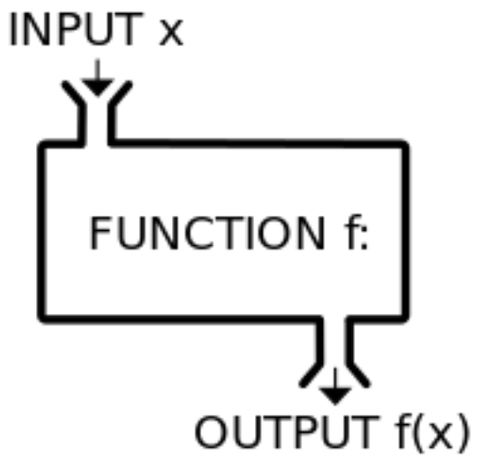
Some projections go back as far as 1569

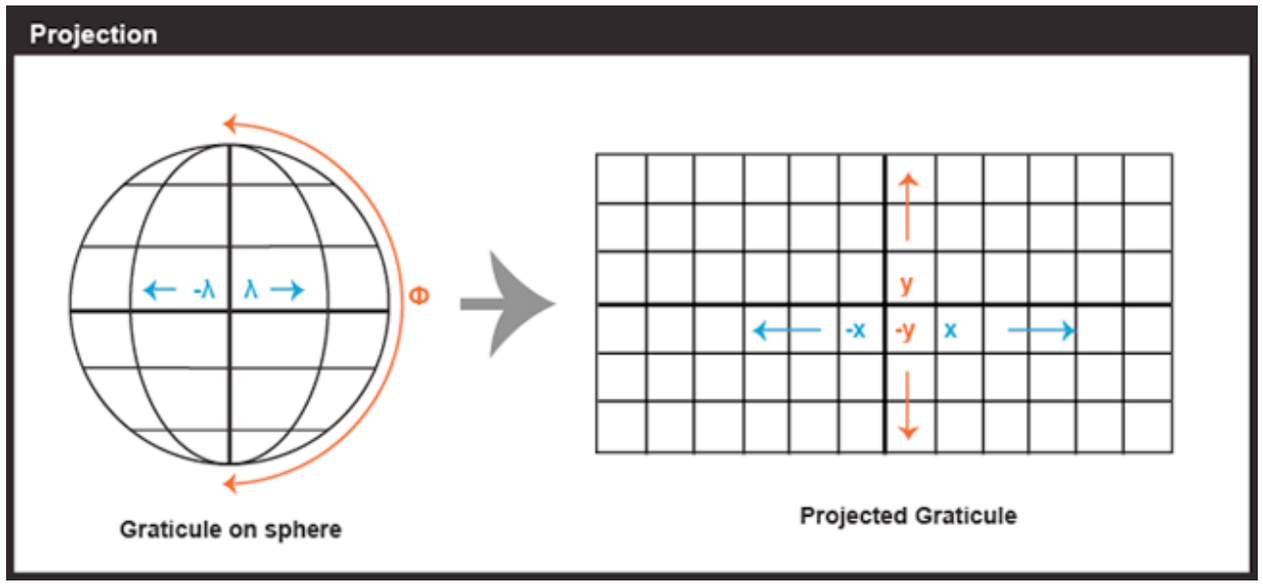
MAP PROJECTIONS PRESERVE DIFFERENT THINGS

Projection Classified by what properties of the map they preserve:

- azimuthal (preserves direction)
- conformal (preserves shape)
- equal-area (preserves area)
- equidistant (presents distance)
- gnomonic (preserves shortest route)

MAP PROJECTIONS ARE MATH FUNCTIONS





D3.JS GEO PATH GENERATOR + MAP PROJECTION

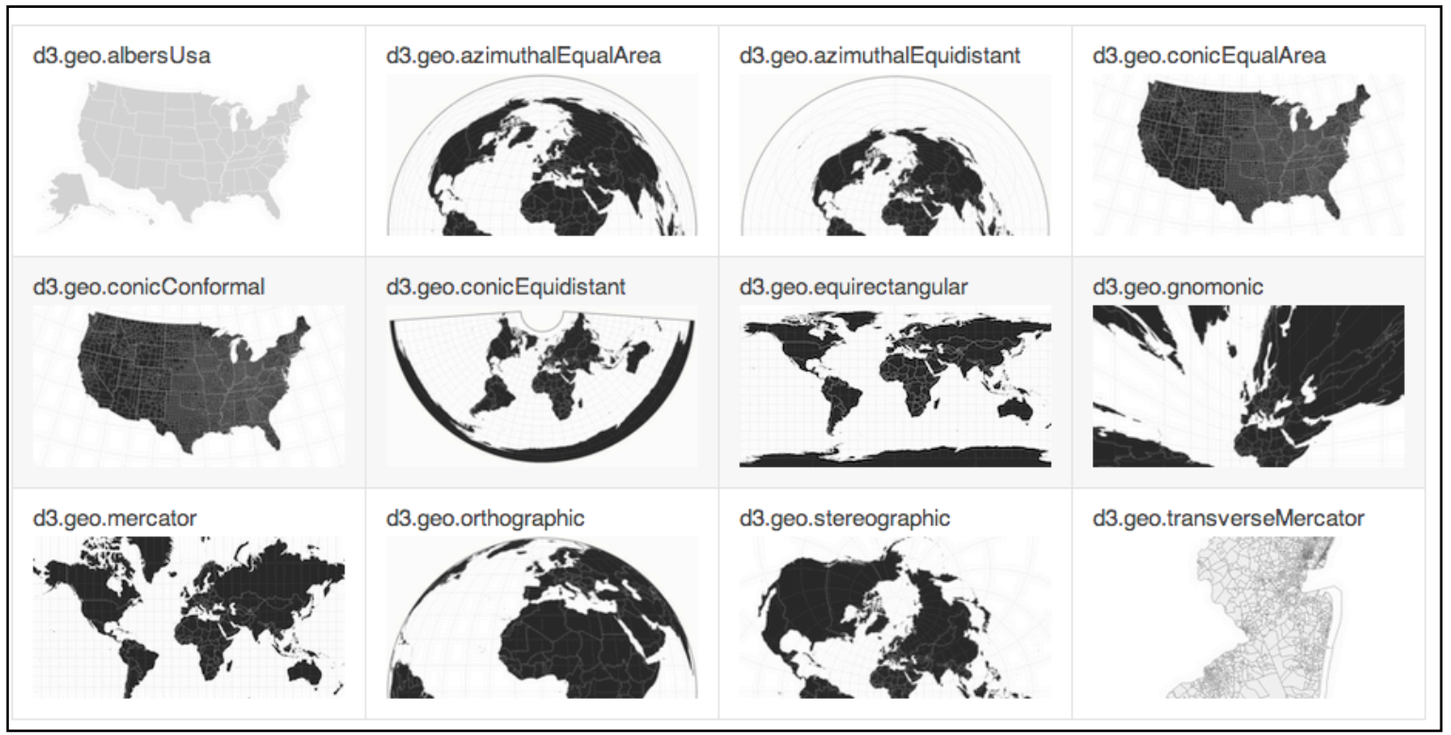
d3.geo.path() does math & translation for you:
-> Takes in data in GeoJSON Data Format
-> returns SVG Path Mini-Language Instructions

D3.js map projection tells d3.geo.path()
which Math Function to use for the math & translation

d3.geo.path():
https://github.com/mbostock/d3/wiki/Geo-Paths

BASIC D3.JS STANDARD MAP PROJECTIONS

d3.geo.albers()
d3.geo.albersUsa()
d3.geo.azimuthalEqualArea()
d3.geo.azimuthalEquidistant()
d3.geo.conicConformal()
d3.geo.conicEqualArea()
d3.geo.conicEquidistant()
d3.geo.equirectangular()
d3.geo.gnomonic()
d3.geo.mercator()
d3.geo.orthographic()
d3.geo.stereographic()
d3.geo.transverseMercator()



D3.js Geo Projections
https://github.com/mbostock/d3/wiki/Geo-Projections

USING A D3.JS MAP PROJECTION IS EASY

```
var geoPath = d3.geo.path()
    .projection(d3.geo.albersUsa());
```

Then use D3.js Geo Path Generator just like we covered before.

D3.js Geo Projections
https://github.com/mbostock/d3/wiki/Geo-Projections

D3.JS MAP PROJECTIONS ARE HIGHLY CUSTOMIZABLE

projection.inter(point)
projection.rotate([rotation])
projection.center([location])
projection.translate([point])
projection.scale([scale])
projection.clipAngle(angle)
projection.clipExtent(extent)
projection.precision(precision)
projection.stream(listener)

D3.js Geo Projections
https://github.com/mbostock/d3/wiki/Geo-Projections

EXERCISE – NORTH AMERICA MAP PROJECTIONS

// https://www.dashingd3js.com/na-map-projections

// Try different projections
* Specifically, mercator vs albers

// Explore elements to see three SVG Paths

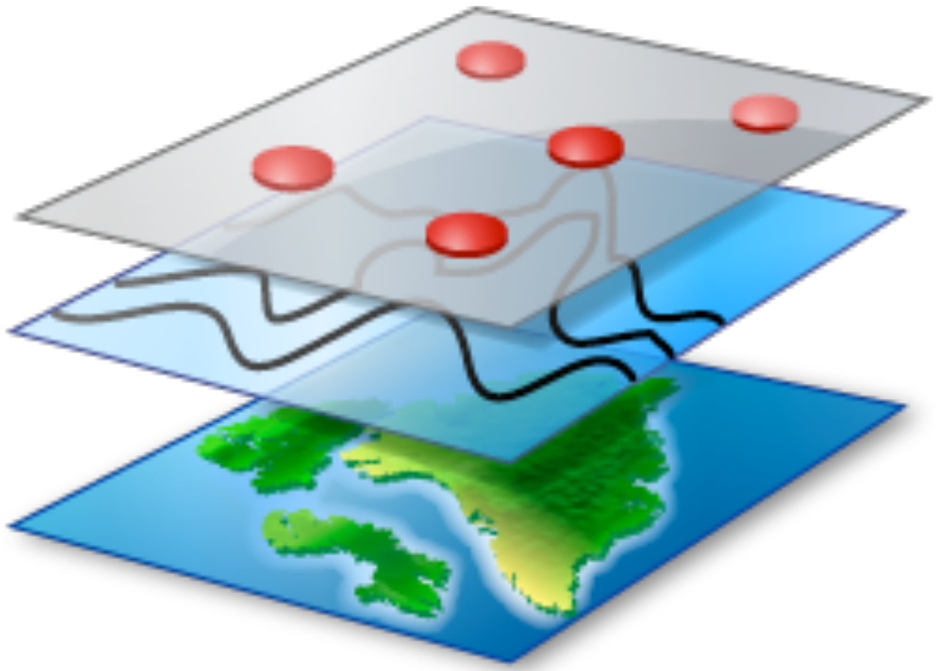
// Look at the source code of the page to see
* zoom
* projectionObject
* "Network" tab in Chrome Developer Tools
  - Preview mexgeojson

Discuss

Zoom, projection, GeoJSON

Sections

HOW TO THINK ABOUT IT – LAYERS

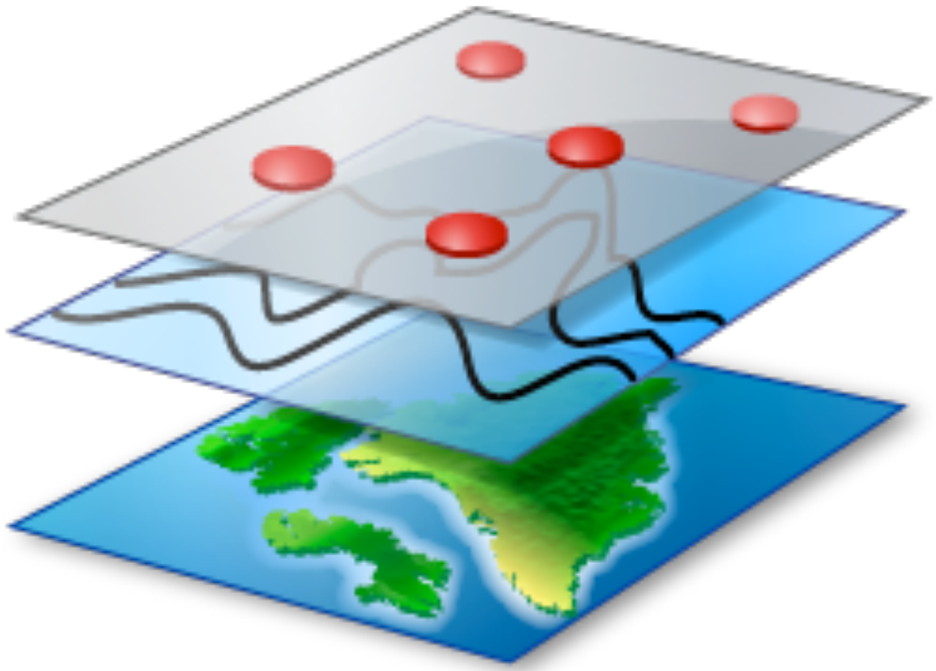


Base Layer - SVG Path

Mapped Objects - SVG Basic Shapes

Image Source:
http://www.truenorthgeospatial.com

1ST STEP – SETUP D3.JS GEO PATH & PROJECTION

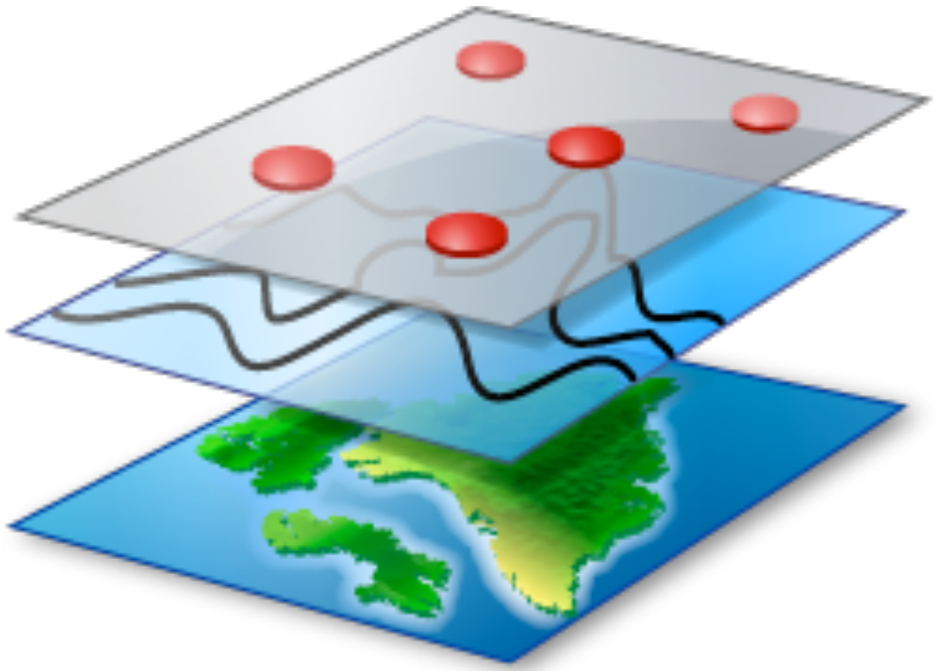


Base Layer - SVG Path

Mapped Objects - SVG Basic Shapes

Image Source:
http://www.truenorthgeospatial.com

2ND STEP – USE D3.JS GEO PATH TO CREATE SVG PATH

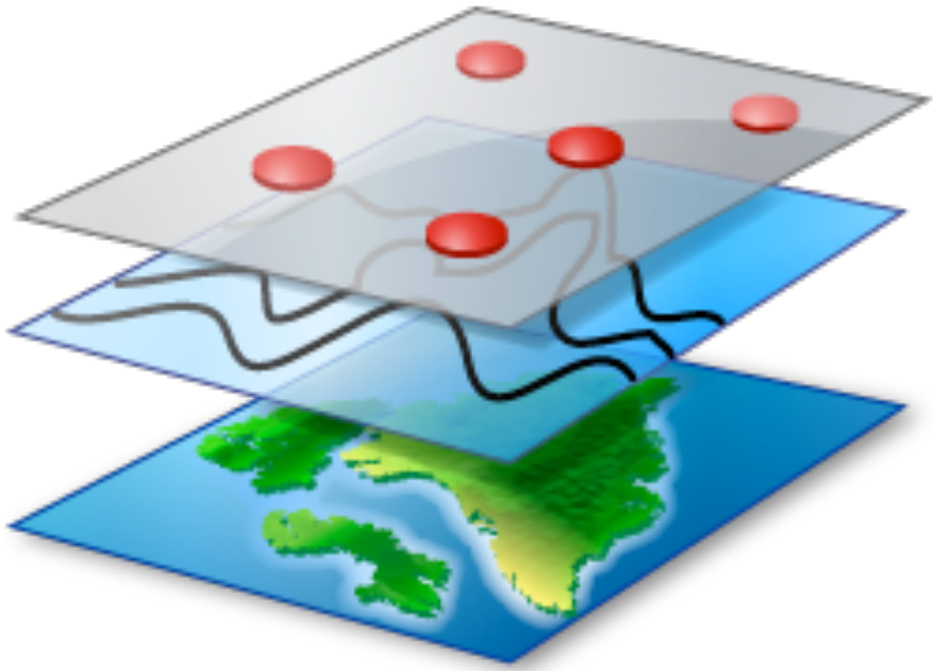


Base Layer - SVG Path

Mapped Objects - SVG Basic Shapes

Image Source:
http://www.truenorthgeospatial.com

3RD STEP – USE D3.JS GEO PATH TO PLACE OBJECTS

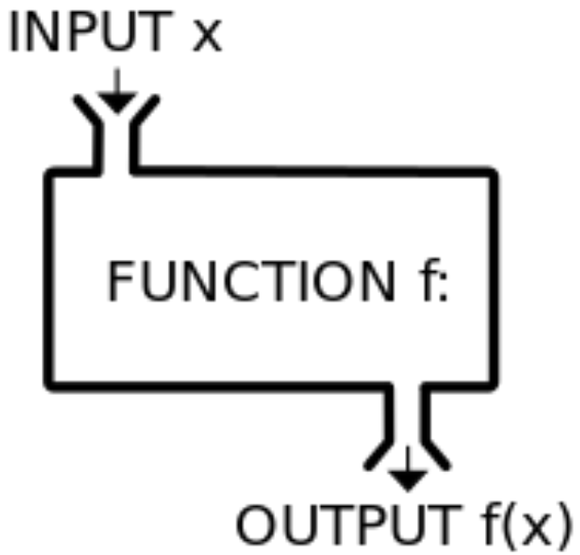


Base Layer - SVG Path

Mapped Objects - SVG Basic Shapes

Image Source:
http://www.truenorthgeospatial.com

MAPPING OBJECTS ON MAP PROJECTIONS



It is all math

a) GeoJSON gets converted to SVG Path Mini-Language
b) Path is used to create Map Shape
c) D3.js Geo Path Function takes in GeoJSON Point Geometry Objects and converts them to paths as well.
d) These are then places on the Map Shape created earlier.

EXERCISE – MAPPING OBJECTS ON NA PROJECTION

// https://www.dashingd3js.com/objects-on-na-map-projections

// Try different projections
* Specifically, mercator vs albers

// Explore elements to see three SVG Paths

// Look at the source code of the page to see
* zoom
* transitions
* mouse events

Discuss
Zoom, transitions,
mouse events

SECTION 6
CONCLUSION

Sections

- 1) The Goals
- 2) D3.js Events
- 3) D3.js Transitions & Animations
- 4) D3.js General Update Pattern
- 5) D3.js Mapping
- 6) Conclusion

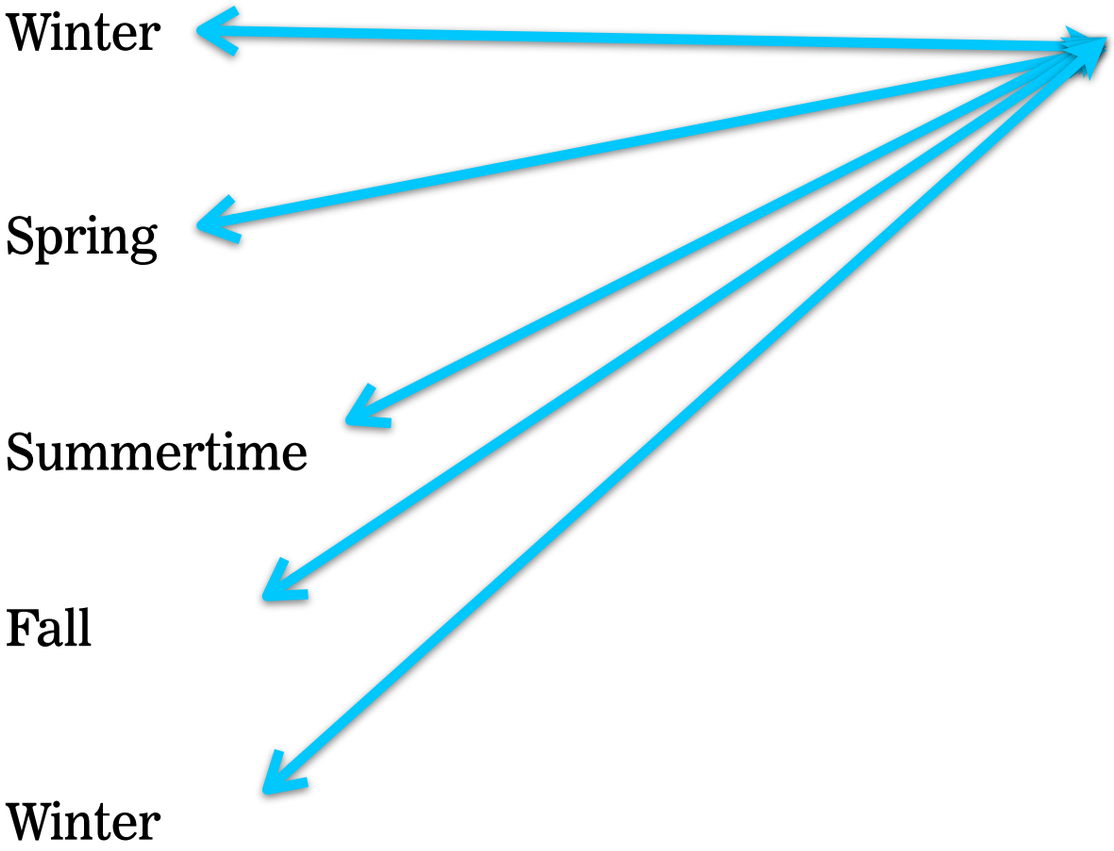
a) Key Take Aways

b) Next Steps

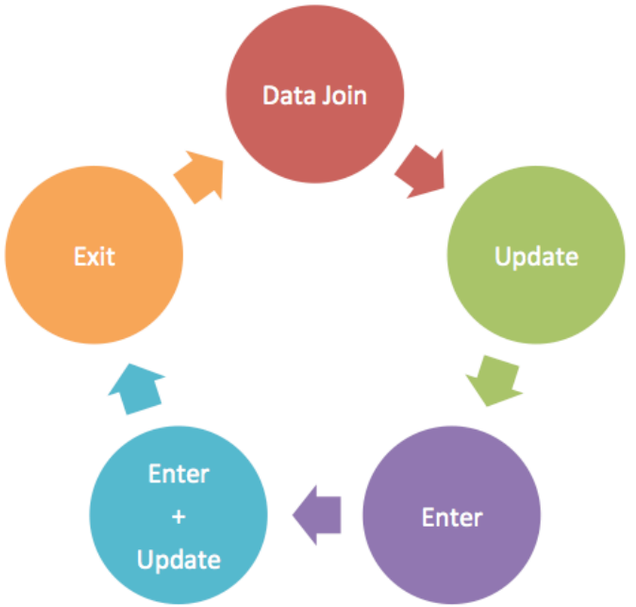
c) Resources

d) Class Closing / Q&A

ABSTRACT GENERAL UPDATE PATTERN AS FUNCTION



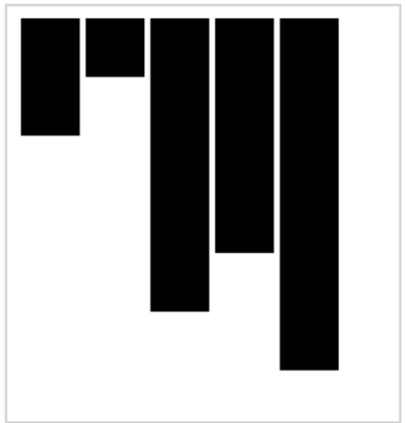
General Update Pattern Function



MAKE YOUR OWN D3 LIBRARY

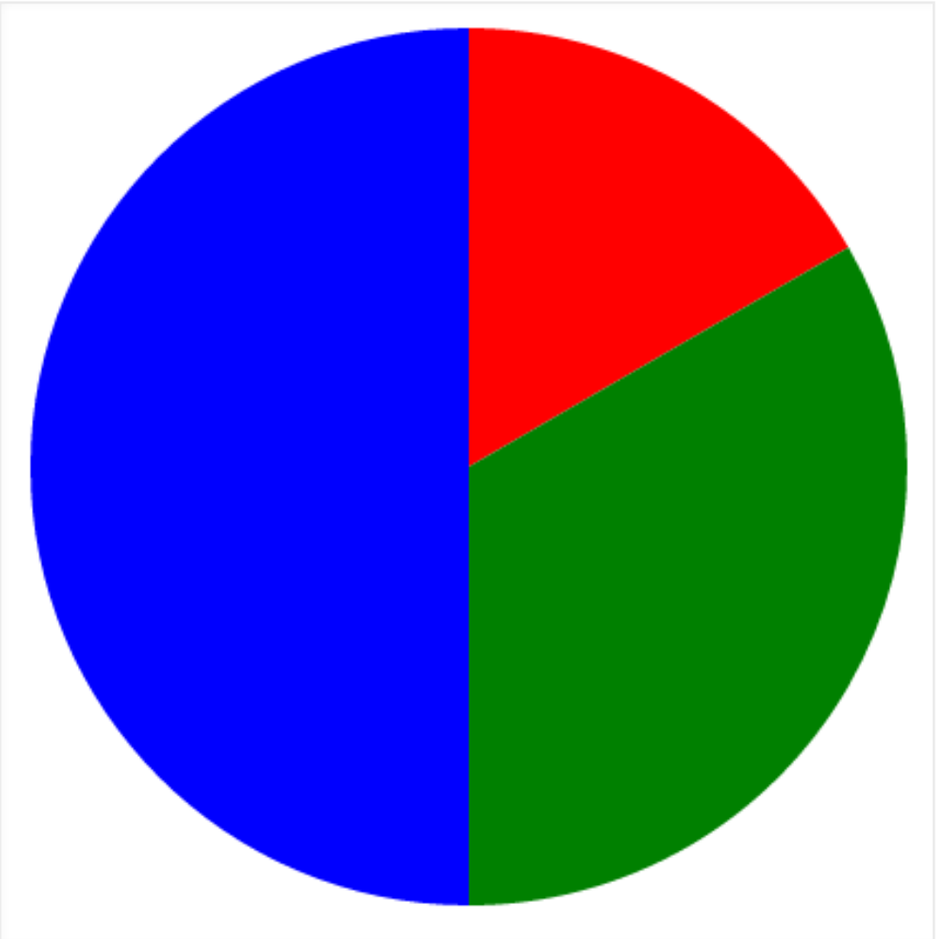
```
var dataset = [2, 1, 5, 4, 6];

var myBarChart = barChart();

myBarChart
    .svgWidth(200)
    .svgHeight(200);

d3.select("body").append("div")
    .datum(dataset)
    .call(myBarChart);
```

```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<div>
      ▼<svg width="200" height="200">
          <rect x="0" y="0" width="38" height="66"></rect>
          <rect x="40" y="0" width="38" height="33"></rect>
          <rect x="80" y="0" width="38" height="165"></rect>
          <rect x="120" y="0" width="38" height="132"></rect>
          <rect x="160" y="0" width="38" height="198"></rect>
        </svg>
      </div>
      " "
  </body>
</html>
```

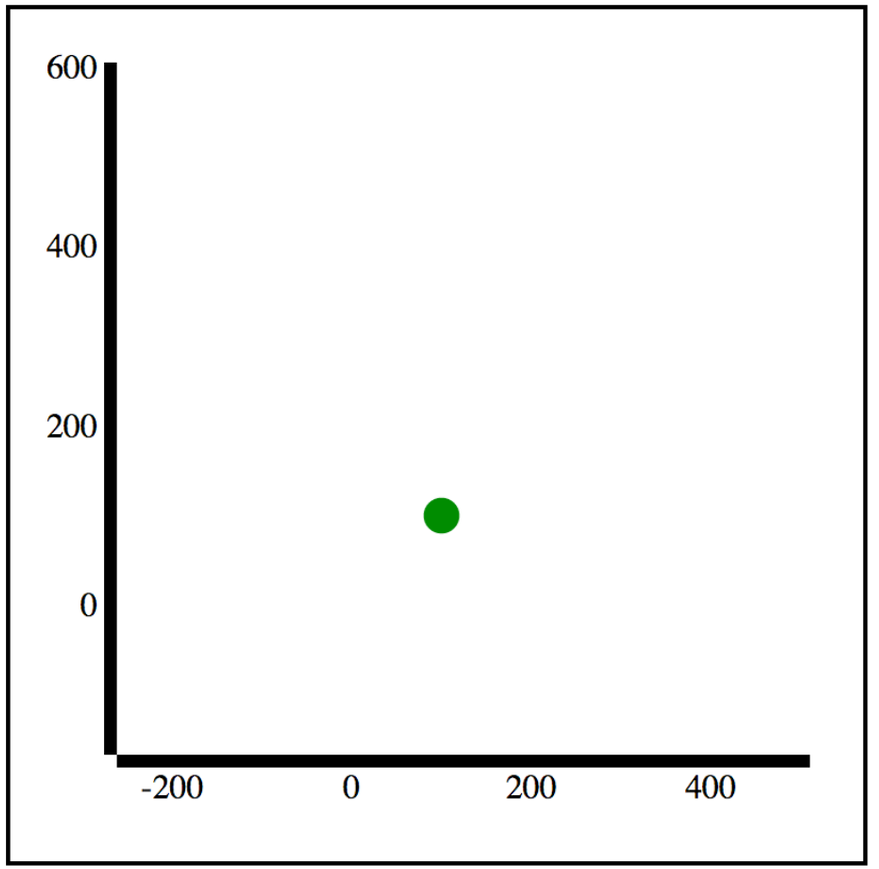


LET D3 DO THE MATH & HARD WORK FOR YOU

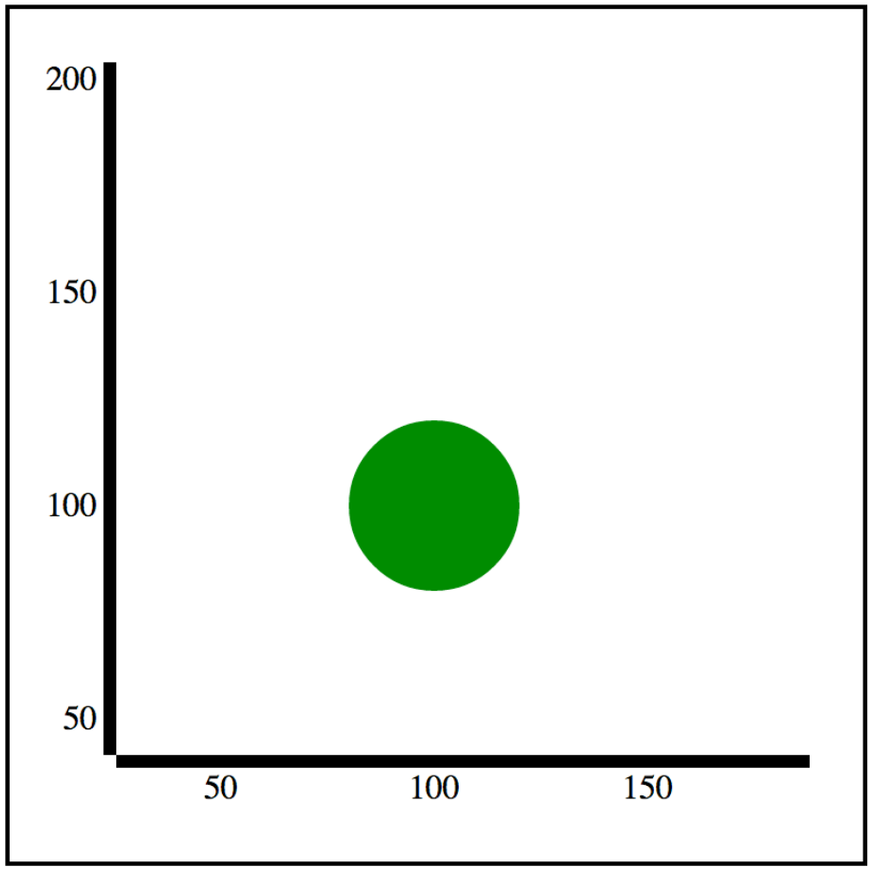
```
var data = [{amount: 1, color: "red"  },
            {amount: 2, color: "green"},
            {amount: 3, color: "blue" }];
```



ZOOMING REQUIRES THOUGHT, PROVIDES GREAT POWER



Pan X Translate: 110.15190437698337
Pan Y Translate: 118.0139524438228
D3 Zoom Scale: 0.4209914366499001



Pan X Translate: -51
Pan Y Translate: -242
D3 Zoom Scale: 2

D3.JS MAPS – MATH & LAYERS

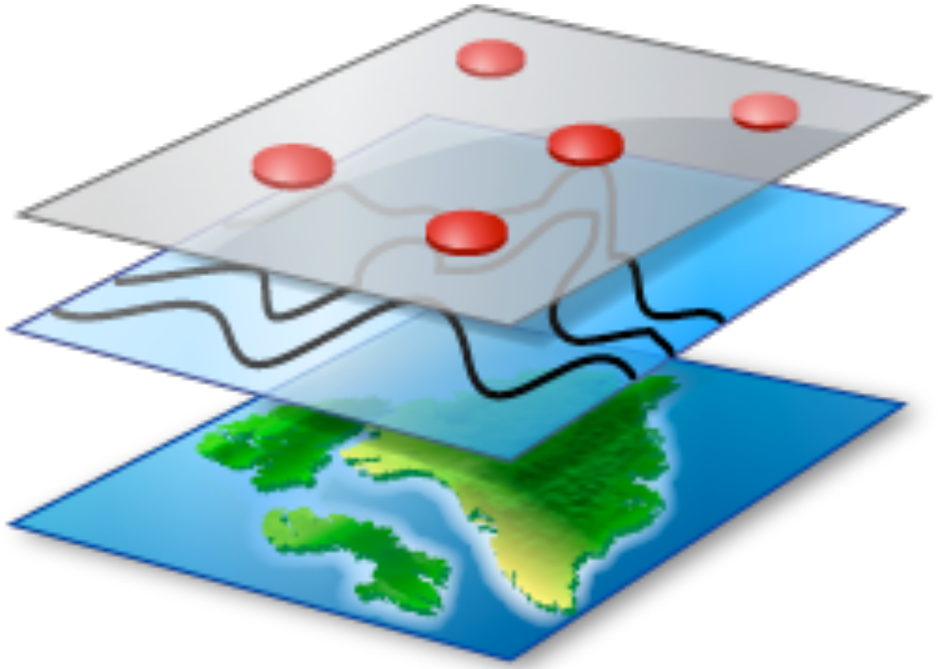


Image Source:
http://www.truenorthgeospatial.com

Sections

- 1) The Goals
- 2) D3.js Events
- 3) D3.js Transitions & Animations
- 4) D3.js General Update Pattern
- 5) D3.js Mapping
- 6) Conclusion

a) Key Take Aways

b) Next Steps

c) Resources

d) Class Closing / Q&A

D3 LAYOUTS & PLUGINS

Layouts

Wiki ▸ API Reference ▸ Layouts

See one of:

- Bundle - apply Holten's *hierarchical bundling algorithm* to edges.
- Chord - produce a chord diagram from a matrix of relationships.
- Cluster - cluster entities into a dendrogram.
- Force - position linked nodes using physical simulation.
- Hierarchy - derive a custom hierarchical layout implementation.
- Histogram - compute the distribution of data using quantized bins.
- Pack - produce a hierarchical layout using recursive circle-packing.
- Partition - recursively partition a node tree into a sunburst or icicle.
- Pie - compute the start and end angles for arcs in a pie or donut chart.
- Stack - compute the baseline for each series in a stacked bar or area chart.
- Tree - position a tree of nodes tidily.
- Treemap - use recursive spatial subdivision to display a tree of nodes.

box	Add missing semicolons.
bullet	Link to vertical demo.
chernoff	Remove redundant svg: prefixes.
cubehelix	Restore example.
fisheye	Remove a few unused variables.
force_labels	Add missing semicolons.
geo	Slightly prettier link.
geodesic	Fix bug with flipped triangles.
geom	Update geom/contour/README.md
graph	removed traverse
hexbin	Include hexbin grid coordinates.
hive	Move to top-level directory.
horizon	Remove horizon.duration.
interpolate-zoom	Add deprecation notice.
jsonp	Fix token replacement in jsonp plugin
keybinding	Update keybinding with improvements from iD project
longscroll	Shorten.
qq	Add qq plugin.
rollup	Create README.md
sankey	Add demo links.
simplify	Remove obsolete d3.simplify plugin.
superformula	Add superformula plugin.
urlencode	Add missing semicolons.
LICENSE	Update copyright year.
README.md	Add an example (d3.svg.hive).

OTHER DATA VISUALIZATION LIBRARIES / TOOLS

Cubism.js	Insights.js	Bokeh
NVD3.js	C3.js	Dimple.js
d3.chart	Visual Sedimentation	Vega
Graphene	Raw	Dance.js
Rickshaw	rCharts	DC.js
Crossfilter.js	d4.js	Many Others...

Source:
http://mikemcdearmon.com/portfolio/techposts/data-visualization-base-on-d3-js

MORE D3 & JAVASCRIPT MAPS

http://bost.ocks.org/mike/map/

TopoJSON

Hundreds of D3.js Map Examples - The world is your oyster!

D3 API

https://github.com/mbostock/d3/wiki/API-Reference

Sections

- 1) The Goals
- 2) D3.js Events
- 3) D3.js Transitions & Animations
- 4) D3.js General Update Pattern
- 5) D3.js Mapping
- 6) Conclusion

a) Key Take Aways

b) Next Steps

c) Resources

d) Class Closing / Q&A

D3.JS RESOURCES

http://dashingd3js.com

http://bost.ocks.org/mike/

http://www.jasondavies.com/

http://vallandingham.me/vis/

http://alignedleft.com/tutorials/d3/

http://christopheviau.com/d3list/gallery.html

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

http://www.jeromecukier.net/wp-content/uploads/2012/10/d3-cheat-sheet.pdf

DATA VISUALIZATION RESOURCES

http://wtfviz.net/

http://stamen.com/

http://vis.stanford.edu/

http://thumbsupviz.com/

https://twitter.com/DashingD3js

http://annkemery.com/dataviz-checklist/

http://www.visual-literacy.org/periodic_table/periodic_table.html

http://visualoop.com/13484/the-30-data-viz-blogs-you-cant-miss-in-one-place

Sections

- 1) The Goals
- 2) D3.js Events
- 3) D3.js Transitions & Animations
- 4) D3.js General Update Pattern
- 5) D3.js Mapping
- 6) Conclusion

a) Key Take Aways

b) Next Steps

c) Resources

d) Class Closing / Q&A

CONTACT:

SEBASTIAN GUTIERREZ

SEBASTIAN@DASHINGD3JS.COM

@DASHINGD3JS