VISUALIZATION ON THE WEB

D3 Data-Driven Documents

VISUAL ANALYTICS
D3.JS

Exercise #1

- Represent a set of numbers with a sequence of lines with length proportional to the corresponding number

Web Page Preparation

- Create a file HTML
- Create content for the page
- Include an empty DIV for the visualization
- Install and link D3
- Construct SVG element within the DIV element

Step 1

DATA TO ELEMENTS

Selection should correspond to data

```
var numbers =
[5,10,15,20,25];
var lines =
svg.selectAll(”line")
   .data(numbers)
   .enter().append(”line");
```

Data

SVG

Selection should correspond to data

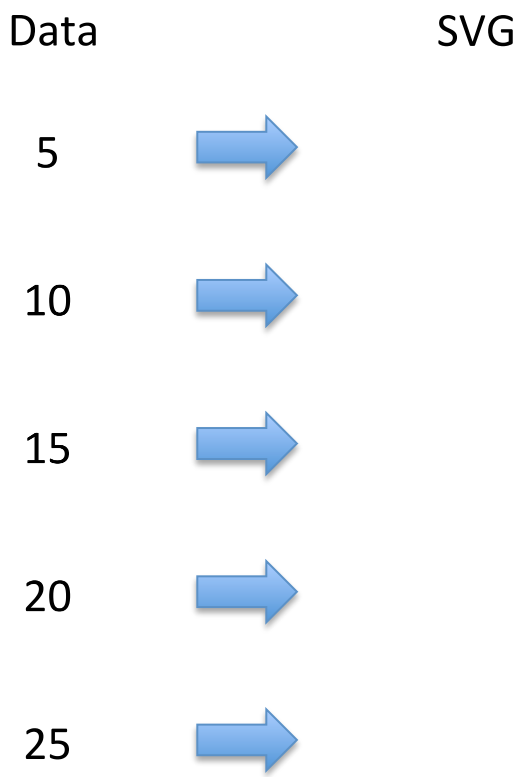
```
var numbers =
[5,10,15,20,25];
var lines =
svg.selectAll(”line")
    .data(numbers)
    .enter().append(”line");
```

Data	SVG
5	
10	
15	
20	
25	

Method data joins data with document elements

Selection should correspond to data

```
var numbers =
[5,10,15,20,25];
var lines =
svg.selectAll(”line")
   .data(numbers)
   .enter().append(”line");
```



Data

SVG

5

10

15

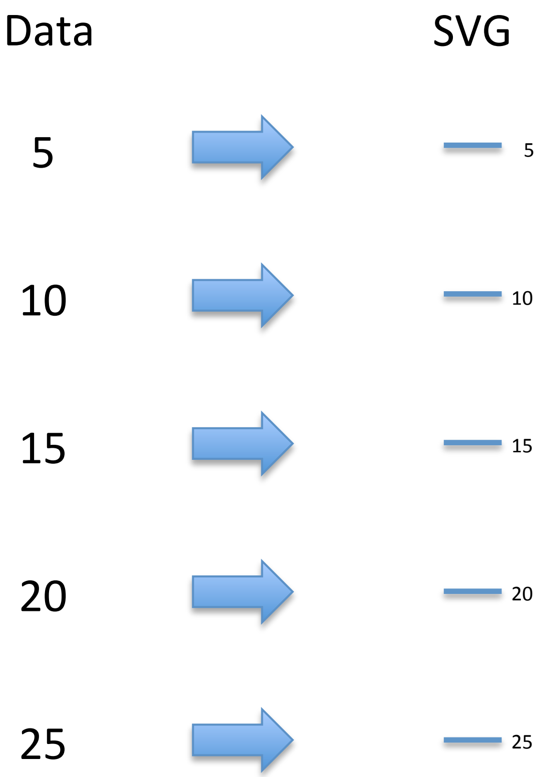
20

25

Method enter specifies the action for missing elements

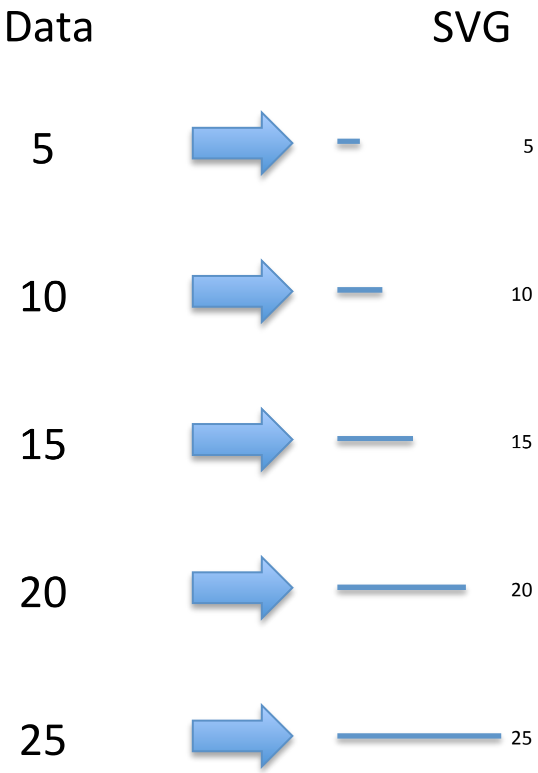
Selection should correspond to data

```
var numbers =
[5,10,15,20,25];
var lines =
svg.selectAll(”line")
    .data(numbers)
    .enter().append(”line");
```



Selection should correspond to data

```
var numbers =
[5,10,15,20,25];
var lines =
svg.selectAll(”line")
    .data(numbers)
    .enter().append(”line");

lines.attr(“x1”,10)
    .attr(“y1”,posy(d,i))
    .attr(“x2”,posx(d,i))
    .attr(“y2”,posy(d,i))
```



Data

SVG

5

10

15

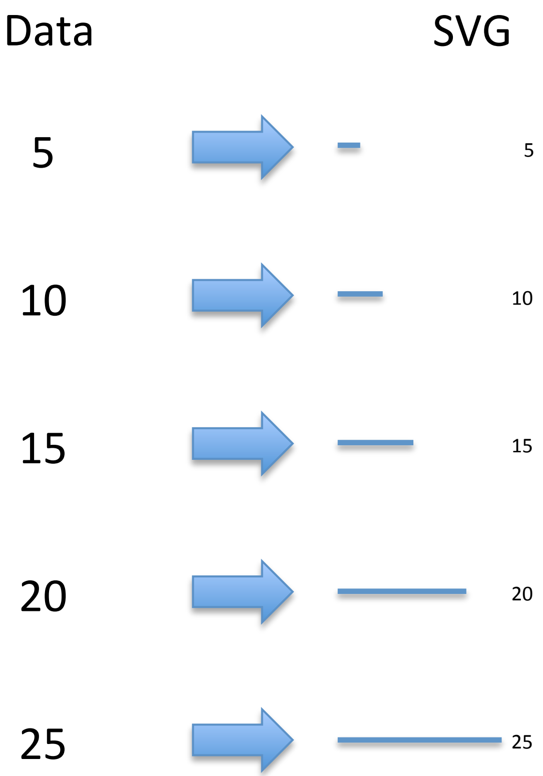
20

25

The new elements are bound to data. Data can be used to compute attributes

Selection should correspond to data

```
lines.attr(“x1”,10)
   .attr(“y1”,posy(d,i))
   .attr(“x2”,posx(d,i))
   .attr(“y2”,posy(d,i));

var posy = function(d,i){
   return i*10;
}

var posx = function(d,i){
   return d * 10;
}
```



Data

SVG

5

10

15

20

25

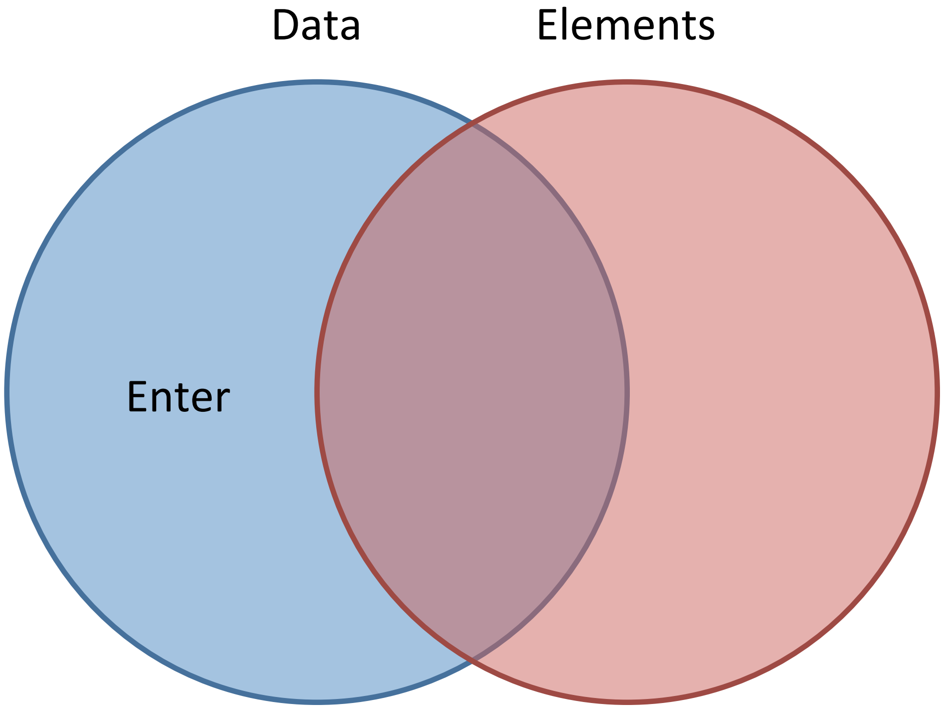
The `attr` functions takes in input a constant value or a function. The function is called automatically by d3, passing the data (`__data__`) bound to the element and a progressive counter

Thinking with Joins

ENTER, EXIT, AND UPDATE

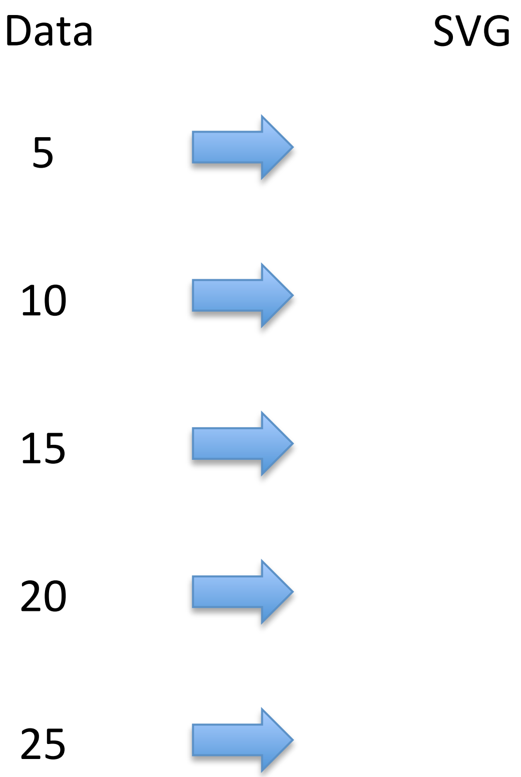
Enter

- New data, for which there were no existing elements.



Entering new elements

```
var numbers =
[5,10,15,20,25];
var lines =
svg.selectAll(”line")
    .data(numbers)
    .enter().append(”line");
```



Data

SVG

5

10

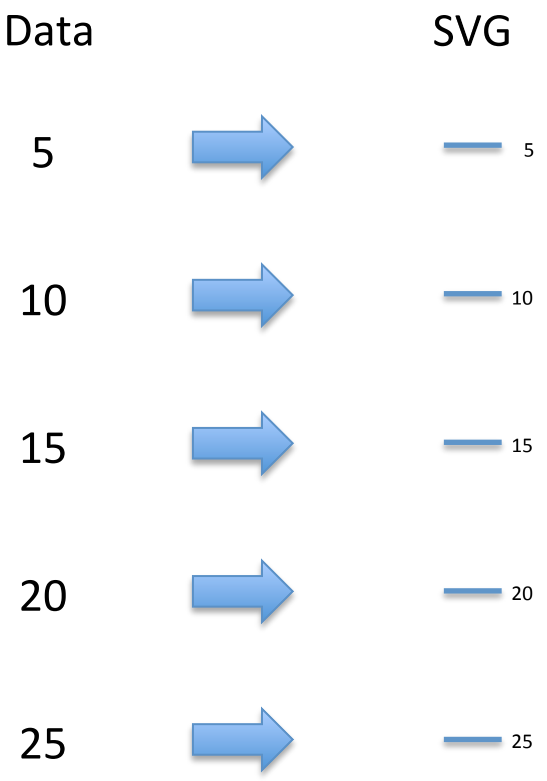
15

20

25

Entering new elements

```
var numbers =
[5,10,15,20,25];
var lines =
svg.selectAll(”line")
    .data(numbers)
    .enter().append(”line");
```



Data

SVG

5

5

10

10

15

15

20

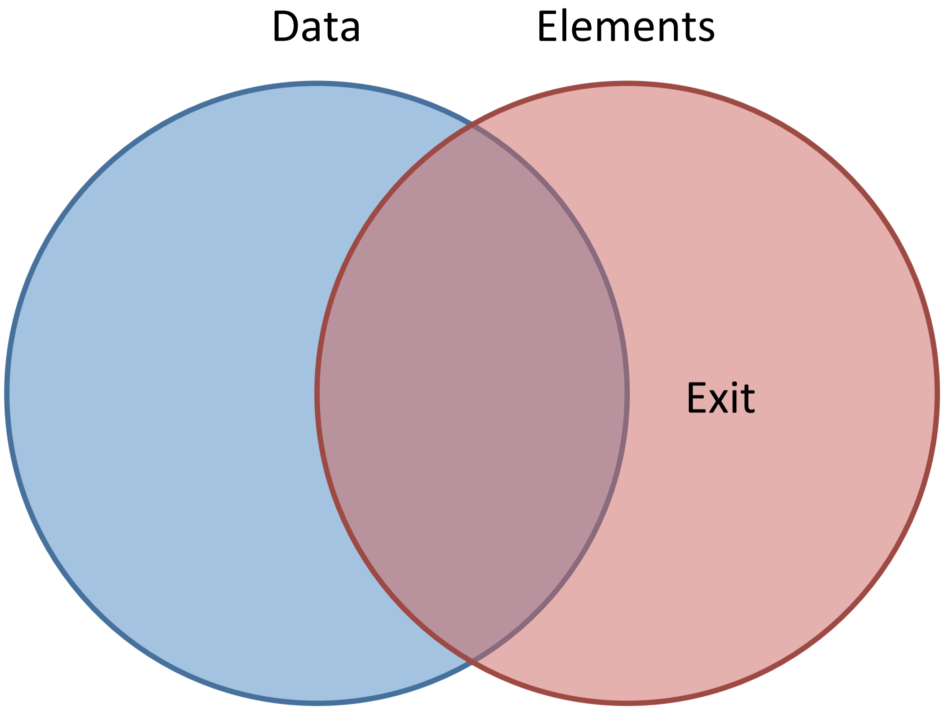
20

25

25

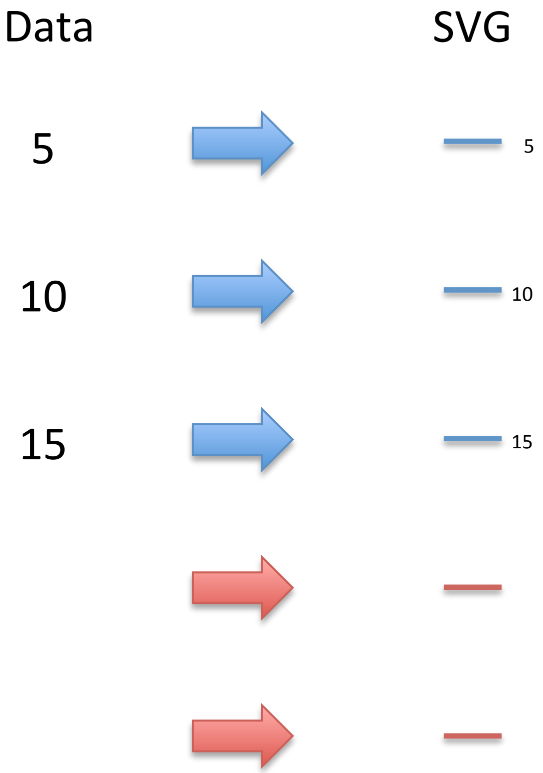
Exit

- Elements that are associated with no data



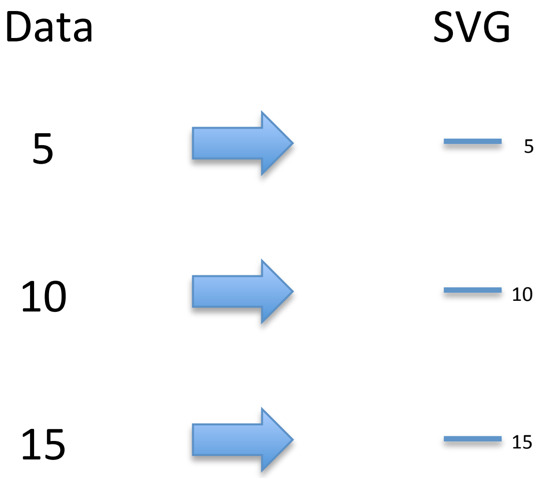
Entering new elements

```
var numbers =
[5,10,15,20,25];
var lines =
svg.selectAll(”line")
    .data(numbers)
    .exit().remove();
```



Entering new elements

```
var numbers =
[5,10,15,20,25];
var lines =
svg.selectAll(”line")
    .data(numbers)
    .enter().remove();
```

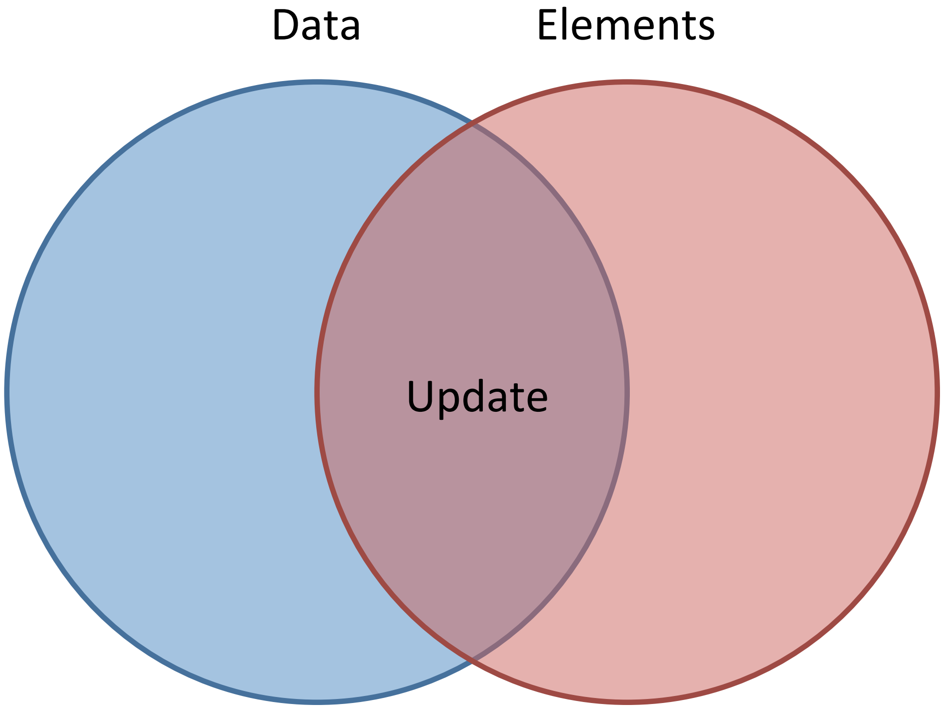


Step 2

DATA ATTRIBUTES TO ELEMENTS ATTRIBUTES

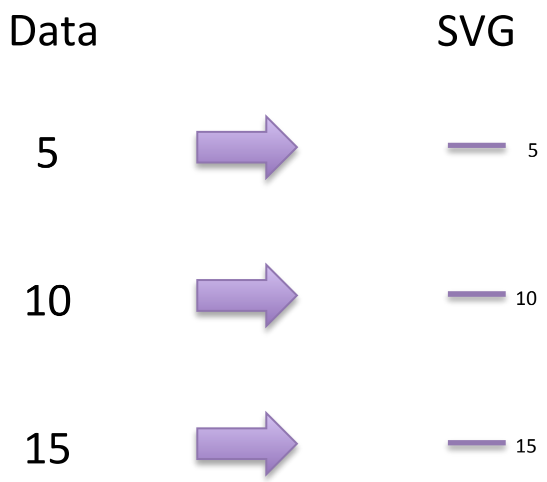
Update

- Data already joined with previous elements



Update existing elements with new data

```
var numbers =
[5,10,15,20,25];
var lines =
svg.selectAll(”line")
    .data(numbers);

lines.attr(“x1”,10)
    .attr(“y1”,posy(d,i))
    .attr(“x2”,posx(d,i))
    .attr(“y2”,posy(d,i));
```



Exercise #2

- Same visualization of Ex#1, using rectangles

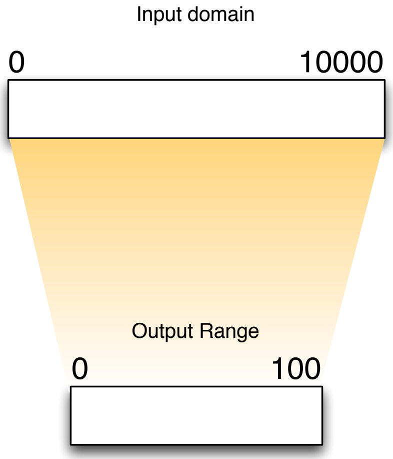
SCALES FUNCTION

Scales

- Data values do not correspond to pixel coordinates
- We need to map data values to new values to meet visualization constraints
- Scales are **functions** that map from an input domain to an output range
- More details available at D3.js documentation: https://github.com/mbostock/d3/wiki/Quantitative-Scales

Manual Mapping

1. For input domain
   1. Select the largest number in original interval (10000)
   2. Select the smallest number in original interval (0)
   3. Select the difference of the two values (10000)
2. For output range
   1. Select the largest number in the new interval (100)
   2. Select the minimum number in the new interval (0)
   3. Select the difference of the two values (100)
3. Compute the ratio of the two intervals'range (10000/100 = 100)



- This is an example of a linear scaling
  - y = mx +b, where b=0 and m=1/100
  - 100 units in the original interval correspond to 1 unit in the destination interval

An example – an alternative solution

```
join.enter()
    .append("rect")
    .attr("x", function(d,i){
        return i*barw;
    })
    .attr("y",function(d){
        return height - d*4;
    })
    .attr("width", barw)
    .attr("height",function(d){
        return d*4;
    });
```

```
join.enter()
    .append("rect")
    .attr("x", function(d,i){
        return x(i);
    })
    .attr("y",function(d){
        return y(d);
    })
    .attr("width", barw)
    .attr("height",function(d){
        return h(d);
    });
function x(d){return m*d + b};
function y(d){return m’*d + b’};
function h(d){return m’’*d + b’’};
```

D3.js Scales generator

- D3 provides several scale types
  - Quantitative
    - Continuous
      - Identity
      - Linear (y=mx+b)
      - Power (y=mx^k+b)
      - Log (y=m log(x) + b)
    - Discrete
      - Quantize
      - Quantile
      - Threshold
  - Ordinal
  - Time

Creating a scale

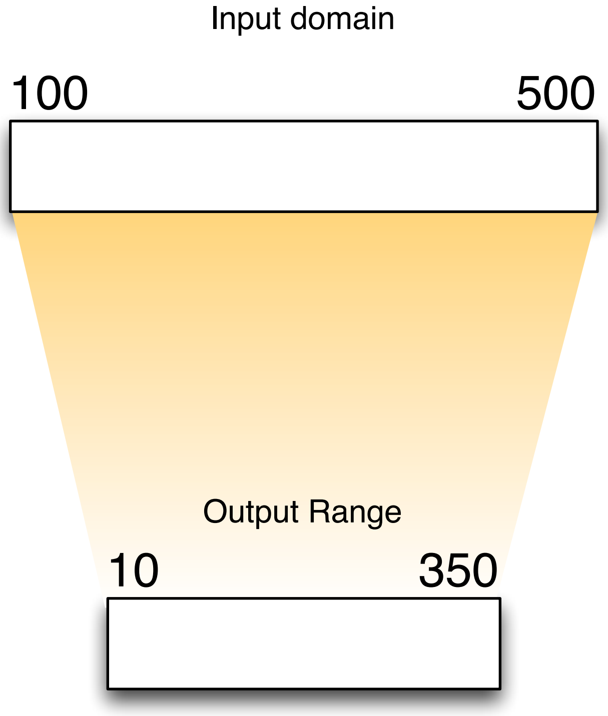
```
var scale = d3.scale.linear();
```

- Default scale uses
  - Domain is [0,1]
  - Range is [0,1]
  - Function is Identity
  - `scale(2.5); //returns 2.5`

Creating a scale – setting domain and range

```
var scale = d3.scale.linear()
    .domain([100,500])
    .range([10,350]);
```

- Default scale uses
  - `scale(100); //returns 10`
  - `scale(300); //returns 180`
  - `scale(500); //returns 350`



Quantitative power scale – circle radius

Previous example

```
g.append("circle")
    .attr("fill","pink")
    .attr("stroke","red")
    .attr("r",function(d){
        return Math.sqrt(d*100);
    })
```

Refined solution

```
var r = d3.scale.sqrt()
    .domain([0,20])
    .range([0,30];

g.append("circle")
    .attr("fill","pink")
    .attr("stroke","red")
    .attr("r",function(d){
        return r(d);
    })
```

Domains & Ranges

- Typically, domains are derived from data while ranges are constant.

```
var x = d3.scale.linear()
    .domain([0, d3.max(numbers)])
    .range([0, 720]);
```

```
var x = d3.scale.log()
    .domain(d3.extent(numbers))
    .range([0, 720]);
```

```
function value(d) { return d.value; }

var x = d3.scale.log()
    .domain(d3.extent(objects, value))
    .range([0, 720]);
```

Utility functions: d3.min, d3.max, d3.extent

- To determine the domain and range interval we should know min and max of the two intervals
- D3.js provides utility functions to access such values
  - `d3.min(array[,accessor])`
  - `d3.max(array[,accessor])`
  - `d3.extent(array[,accessor])`

Utility Functions: examples

```
d3.min([10,30,40,70,100]) //returns 10
d3.max([10,30,40,70,100]) //returns 100
d3.extent([10,30,40,70,100]) //returns [10,100]
```

Interpolators

```
var x = d3.scale.linear()
    .domain([12, 24])
    .range(["steelblue", "brown"]);

x(16); // #666586
```

```
var x = d3.scale.linear()
    .domain([12, 24])
    .range(["0px", "720px"]);

x(16); // 240px
```

```
var x = d3.scale.linear()
    .domain([12, 24])
    .range(["steelblue", "brown"])
    .interpolate(d3.interpolateHsl);

x(16); // #3cb05f
```

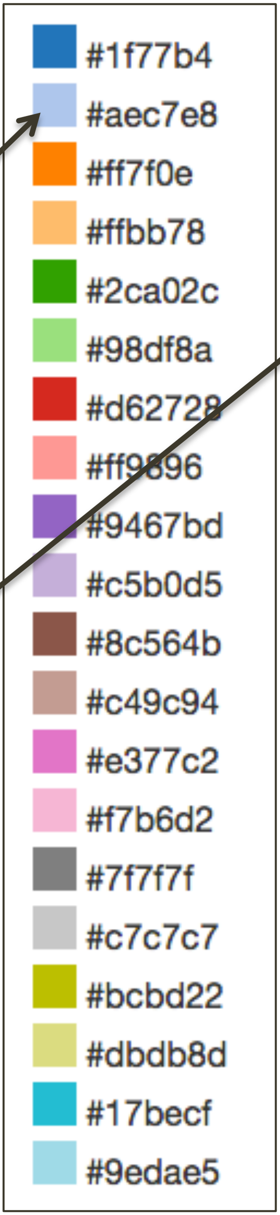
Ordinal Scales

```
var x = d3.scale.ordinal()
    .domain(["A", "B", "C", "D"])
    .range([0, 10, 20, 30]);

x("B"); // 10
```

```
var x = d3.scale.category20()
    .domain(["A", "B", "C", "D"]);

x("B"); // #aec7e8
```

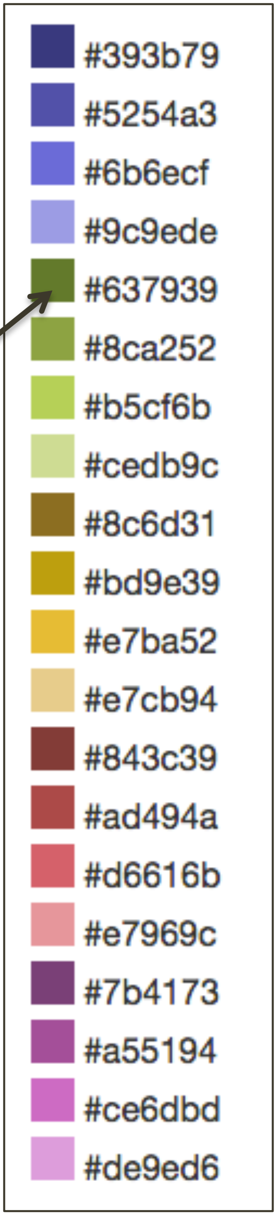
```
var x = d3.scale.category20b()
    .domain(["A", "B", "C", "D"]);

x("E"); // #637939
x.domain(); // A, B, C, D, E
```

category20



#1f77b4
#aec7e8
#ff7f0e
#ffbb78
#2ca02c
#98df8a
#d62728
#ff9896
#9467bd
#c5b0d5
#8c564b
#c49c94
#e377c2
#f7b6d2
#7f7f7f
#c7c7c7
#bcbd22
#dbdb8d
#17becf
#9edae5

category20b



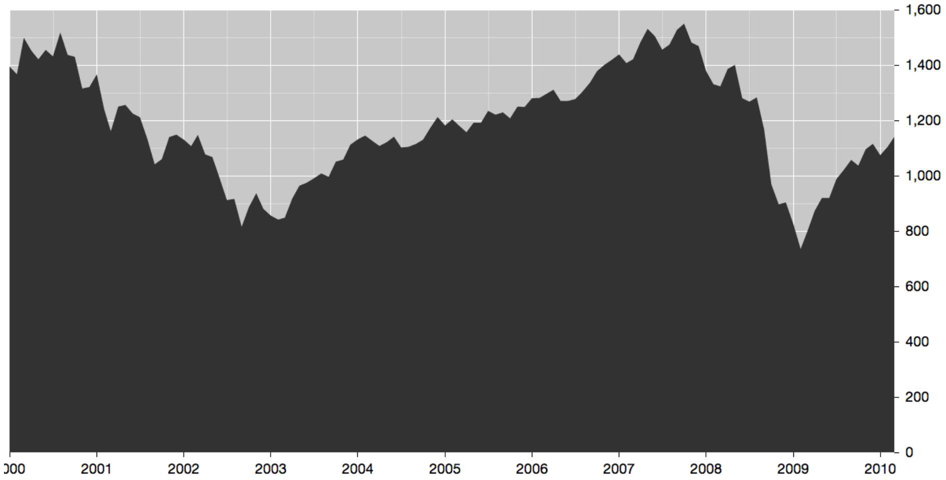
#393b79
#5254a3
#6b6ecf
#9c9ede
#637939
#8ca252
#b5cf6b
#cedb9c
#8c6d31
#bd9e39
#e7ba52
#e7cb94
#843c39
#ad494a
#d6616b
#e7969c
#7b4173
#a55194
#ce6dbd
#de9ed6

d3.svg.axis()

```
var y = d3.scale.linear()
        .domain([0,100])
        .range([0,100]);

var yAxis = d3.svg.axis()
    .scale(y)
    .orient("left");

svg.append("g")
    .attr("class", "y axis")
    .call(yAxis);
```

CSS

```
.axis path, .axis line {
  fill: none;
  stroke: #000;
  shape-rendering: crispEdges;
}
```


1,600
1,400
1,200
1,000
800
600
400
200
0
000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010