D3 Tutorial

Data Binding and Loading

Edit by Jiayi Xu and Han-Wei Shen, The Ohio State University

D3 - **Data**-Driven Documents

- D3 can map data to HTML/SVG elements
  - We can construct the DOM from Data
- Each data value has a corresponding HTML/SVG element (graphical marks)
  - D3 helps you maintain this mapping!

Data Binding - A simple example

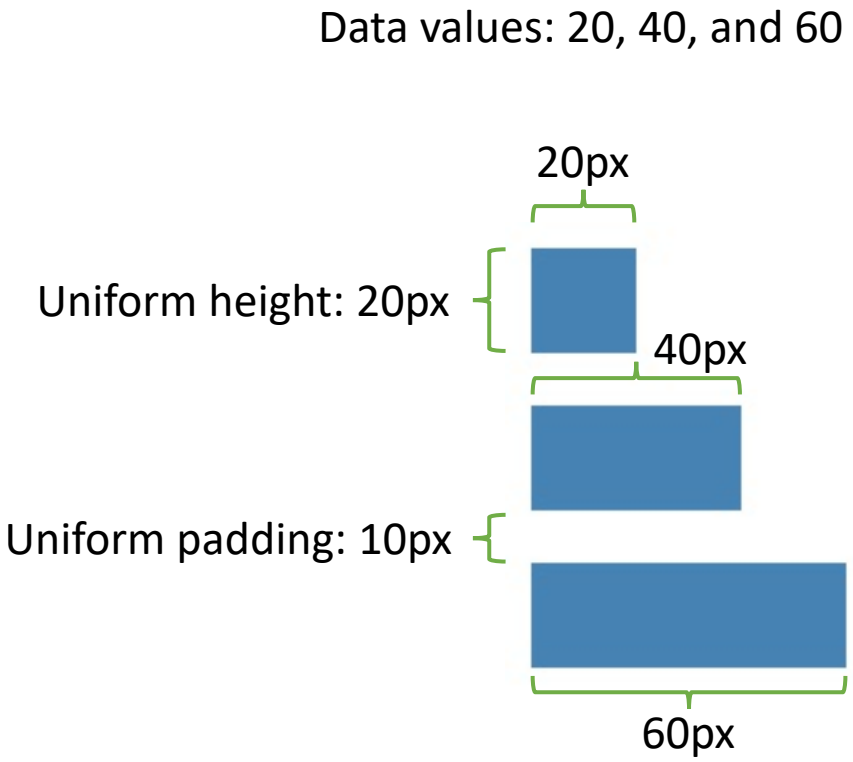
- What we have
  - three bars scattering on the screen
  - three data values: 20, 40, and 60
- Goal: we want to encode data values into the widths of bars

```
<svg>
    <rect x="10" y="20" width="25" height="90"></rect>
    <rect x="100" y="20" width="30" height="50"></rect>
    <rect x="200" y="100" width="40" height="50"></rect>
</svg>
```

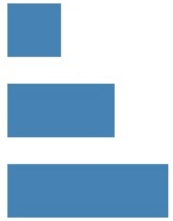
Data values: 20, 40, and 60

Data Binding - A simple example

- Design
  - Uniform height of bars
    - 20 pixels
  - Uniform padding between bars
    - 10 pixels
  - Varying width of bars
    - Proportional to the data values



Data Binding - A simple example

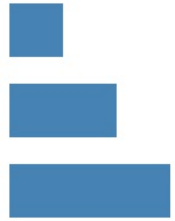


- Initialize variables

```
<script type="text/javascript">
    var padding = 10;
    var barHeight = 20;
    var data = [20, 40, 60];
    var rects =
        d3.selectAll("rect")
        .data(data)
        .attr("x", 0)
        .attr("y", function(d, i) {
            return padding + i * (barHeight + padding);
        })
        .attr("width", function(d, i) {
            return d;
        })
        .attr("height", barHeight)
        .style("fill", "steelblue");
</script>
```

- First, we create variables to store basic information.

Data Binding - A simple example

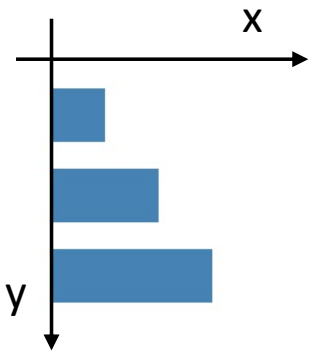


- *selection*.data(*dataArray*)
  - Bind the specified array of *data* with the selected elements

```
<script type="text/javascript">
    var padding = 10;
    var barHeight = 20;
    var data = [20, 40, 60];
    var rects =
        d3.selectAll("rect")
        .data(data)
        .attr("x", 0)
        .attr("y", function(d, i) {
            return padding + i * (barHeight + padding);
        })
        .attr("width", function(d, i) {
            return d;
        })
        .attr("height", barHeight)
        .style("fill", "steelblue");
</script>
```

- Select all the three *rect* tags and bind data with them.

Data Binding - A simple example

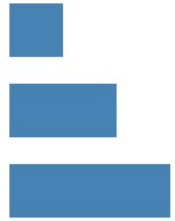


- Set the start point *(x, y)* of each bar.

```
<script type="text/javascript">
    var padding = 10;
    var barHeight = 20;
    var data = [20, 40, 60];
    var rects =
        d3.selectAll("rect")
        .data(data)
        .attr("x", 0)
        .attr("y", function(d, i) {
            return padding + i * (barHeight + padding);
        })
        .attr("width", function(d, i) {
            return d;
        })
        .attr("height", barHeight)
        .style("fill", "steelblue");
</script>
```

- *x* is always 0
- Pass a *function(d, i)* to modify *y* values
  - The variable *d* represents each data value;
  - *i* represents the index of each data value and starts from 0.

Data Binding - A simple example



- Set the *width* and *height* of each bar

```
<script type="text/javascript">
    var padding = 10;
    var barHeight = 20;
    var data = [20, 40, 60];
    var rects =
        d3.selectAll("rect")
        .data(data)
        .attr("x", 0)
        .attr("y", function(d, i) {
            return padding + i * (barHeight + padding);
        })
        .attr("width", function(d, i) {
            return d;
        })
        .attr("height", barHeight)
        .style("fill", "steelblue");
</script>
```

- The *width* of each bar is proportional to the corresponding data value.
- The *height* of each bar is fixed.

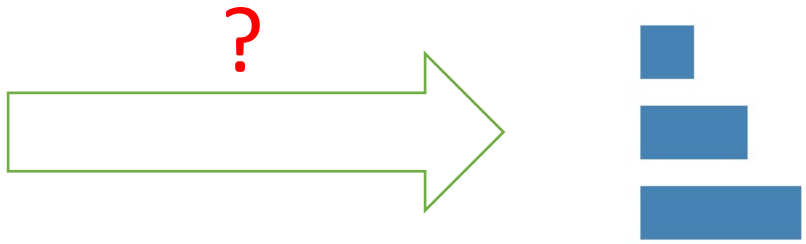
Data Binding - If we don’t have bars initially

- We just have three data values: 20, 40, and 60
  - No bars are on the screen initially
- Goal
  - We want to create three bars and encode data values into the widths of bars

Data values: 20, 40, and 60

```
<svg>
    <!-- The screen is empty! -->
</svg>
```

Empty Screen



Data Binding - If we don’t have bars initially

- What’s wrong with the previous codes?

```
<script type="text/javascript">
    var padding = 10;
    var barHeight = 20;
    var data = [20, 40, 60];
    var rects =
        d3.selectAll("rect")
        .data(data)
        .attr("x", 0)
        .attr("y", function(d, i) {
            return padding + i * (barHeight + padding);
        })
        .attr("width", function(d, i) {
            return d;
        })
        .attr("height", barHeight)
        .style("fill", "steelblue");
</script>
```

- We have no bars on the screen initially so nothing will be selected! We bind data with “nothing”!
- We need **a method** to create bars

Data Binding - If we don’t have bars initially

- A straightforward solution
  - We can create three *rect* tags first by the *append* function

```
var svg = d3.select("svg");
for (var i = 0; i < data.length; ++i) {
    svg.append("rect");
}
```

  - Then, use the same method
- D3.js also supports a more concise way by using *selection*.*enter*().

Data Binding - If we don't have bars initially

- A more concise solution

```
var svg = d3.select("svg");
var rects =
    svg.selectAll("rect")
    .data(data)
    .enter()
    .append("rect")
    .attr("x", 0)
    .attr("y", function(d, i) {
        return padding + i * (barHeight + padding);
    })
    .attr("width", function(d, i) {
        return d;
    })
    .attr("height", barHeight)
    .style("fill", "steelblue");
```

- We will insert *rect*s into *svg*.
- Declare that we **intend** to bind data with *rect* tags, although we don't have *rect*s now.

Data Binding - If we don’t have bars initially

- *selection*.enter()
  - Create placeholder nodes for data values that has **NO** corresponding DOM element in the selection.

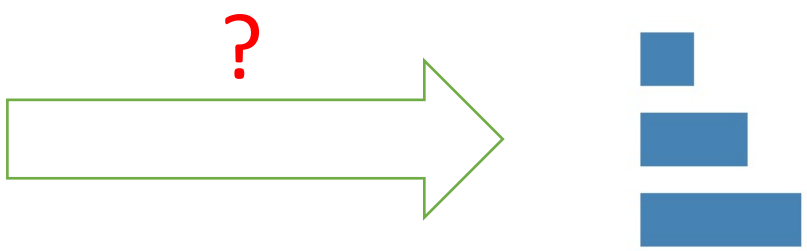
```
var svg = d3.select("svg");
var rects =
    svg.selectAll("rect")
    .data(data)
    .enter()
    .append("rect")
    .attr("x", 0)
    .attr("y", function(d, i) {
        return padding + i * (barHeight + padding);
    })
    .attr("width", function(d, i) {
        return d;
    })
    .attr("height", barHeight)
    .style("fill", "steelblue");
```

- Create placeholders for data values that have **NO** corresponding bars
- Then, each placeholder will be replaced by a *rect* tag
- Finally, set the attributes

Data Binding - Any number of initial bars

- Next, we will deal with any number of initial bars.
  - If the number of bars is larger than the number of data values
    - We can remove needless bars by *selection*.remove() and *selection*.exit()

Any number of bars



Data Binding - Any number of initial bars

```
var svg = d3.select("svg");
var padding = 10;
var barHeight = 20;
var data = [20, 40, 60];

var initialRects = svg.selectAll("rect").data(data);
var removedRects = initialRects.exit().remove();
var newRects = initialRects.enter().append("rect");
var allRects = initialRects.merge(newRects);
// Or, var allRects = svg.selectAll("rect");

allRects
    .attr("x", 0)
    .attr("y", function(d, i) {
        return padding + i * (barHeight + padding);
    })
    .attr("width", function(d, i) {
        return d;
    })
    .attr("height", barHeight)
    .style("fill", "steelblue");
```

- Select all initial bars and declare the intention that we will bind data with rect tags

Data Binding - Any number of initial bars

- *selection*.exit()
  - Return existing elements in the selection for which no new datum was found.
  - Get needless bars

```
var svg = d3.select("svg");
var padding = 10;
var barHeight = 20;
var data = [20, 40, 60];

var initialRects = svg.selectAll("rect").data(data);
var removedRects = initialRects.exit().remove();
var newRects = initialRects.enter().append("rect");
var allRects = initialRects.merge(newRects);
// Or, var allRects = svg.selectAll("rect");

allRects
    .attr("x", 0)
    .attr("y", function(d, i) {
        return padding + i * (barHeight + padding);
    })
    .attr("width", function(d, i) {
        return d;
    })
    .attr("height", barHeight)
    .style("fill", "steelblue");
```

- If we have excessive bars, remove needless bars

Data Binding - Any number of initial bars

- *selection*.enter()
  - Create placeholder nodes for data values that has **NO** corresponding DOM element in the selection.

```
var svg = d3.select("svg");
var padding = 10;
var barHeight = 20;
var data = [20, 40, 60];

var initialRects = svg.selectAll("rect").data(data);
var removedRects = initialRects.exit().remove();
var newRects = initialRects.enter().append("rect");
var allRects = initialRects.merge(newRects);
// Or, var allRects = svg.selectAll("rect");

allRects
    .attr("x", 0)
    .attr("y", function(d, i) {
        return padding + i * (barHeight + padding);
    })
    .attr("width", function(d, i) {
        return d;
    })
    .attr("height", barHeight)
    .style("fill", "steelblue");
```

- If existing bars are not enough, we have to create more *rect* tags for data values

Data Binding - Any number of initial bars

- *selection*.merge(*otherSelection*)
  - merging this selection with the specified *other selection*

```
var svg = d3.select("svg");
var padding = 10;
var barHeight = 20;
var data = [20, 40, 60];

var initialRects = svg.selectAll("rect").data(data);
var removedRects = initialRects.exit().remove();
var newRects = initialRects.enter().append("rect");
var allRects = initialRects.merge(newRects);
// Or, var allRects = svg.selectAll("rect");

allRects
    .attr("x", 0)
    .attr("y", function(d, i) {
        return padding + i * (barHeight + padding);
    })
    .attr("width", function(d, i) {
        return d;
    })
    .attr("height", barHeight)
    .style("fill", "steelblue");
```

- Merge the selection of initial bars with the selection of newly created bars to get all existing bars now.
- Or, we can directly use the *selectAll* function.
- Finally, set the attributes

Data Binding - Try a real data

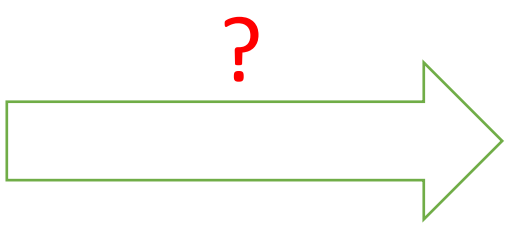
- Populations of cities

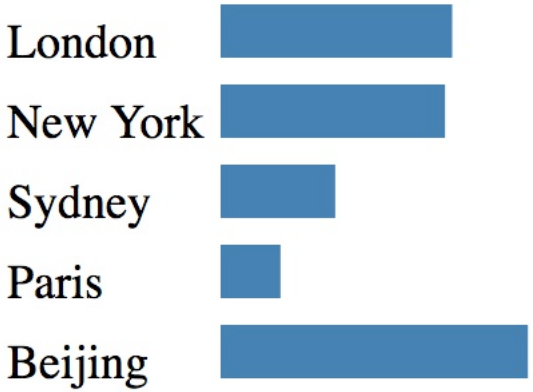
```
var cities = [
  { name: 'London', population: 8674000},
  { name: 'New York', population: 8406000},
  { name: 'Sydney', population: 4293000},
  { name: 'Paris', population: 2244000},
  { name: 'Beijing', population: 11510000}
];
```

Populations of cities

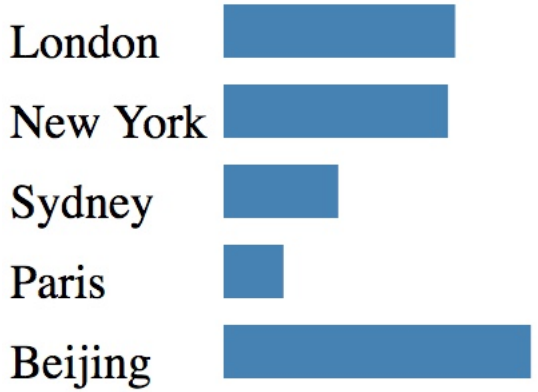
- Represent populations by width of bars

```
var cities = [
  { name: 'London', population: 8674000},
  { name: 'New York', population: 8406000},
  { name: 'Sydney', population: 4293000},
  { name: 'Paris', population: 2244000},
  { name: 'Beijing', population: 11510000}
];
```





Populations of cities - Texts



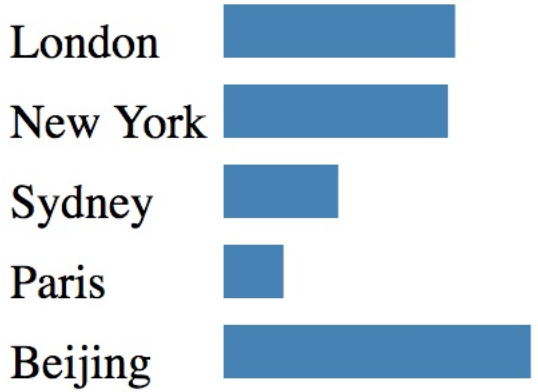
- Initialize variables

```
var padding = 10;
var barHeight = 20;
var fontSize = 18;
```

```
var svg = d3.select("svg");
var texts = svg.selectAll('text')
    .data(cities)
    .enter().append("text")
    .attr('x', 0)
    .attr('y', function(d, i) {
        return (i + 1) * (barHeight + padding);
    })
    .attr('font-size', fontSize)
    .text(function(d) {
        return d.name;
    });
```

- First, we create variables to store basic information

Populations of cities - Texts

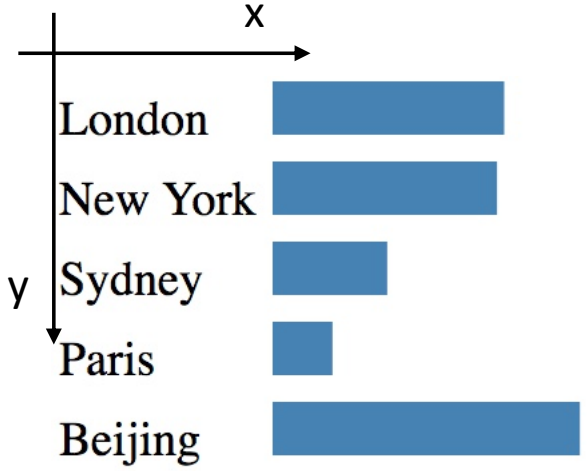


- Bind data with *text* tags

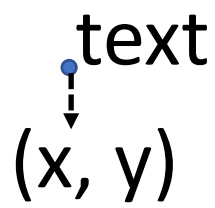
```
var padding = 10;
var barHeight = 20;
var fontSize = 18;

var svg = d3.select("svg");
var texts = svg.selectAll('text')
    .data(cities)
    .enter().append("text")
    .attr('x', 0)
    .attr('y', function(d, i) {
        return (i + 1) * (barHeight + padding);
    })
    .attr('font-size', fontSize)
    .text(function(d) {
        return d.name;
    });
```

- Create text tags to show names of cities

Populations of cities - Texts



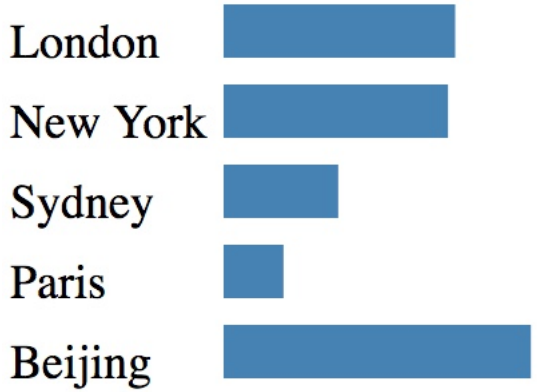
- (*x, y*) of a *text*
  - Bottom left-hand corner




```
var padding = 10;
var barHeight = 20;
var fontSize = 18;

var svg = d3.select("svg");
var texts = svg.selectAll('text')
    .data(cities)
    .enter().append("text")
    .attr('x', 0)
    .attr('y', function(d, i) {
        return (i + 1) * (barHeight + padding);
    })
    .attr('font-size', fontSize)
    .text(function(d) {
        return d.name;
    });
```

- Set coordinates of texts

Populations of cities - Texts

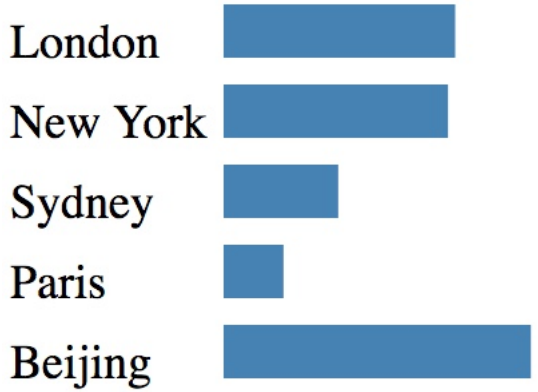


- Font size and text content

```
var padding = 10;
var barHeight = 20;
var fontSize = 18;

var svg = d3.select("svg");
var texts = svg.selectAll('text')
    .data(cities)
    .enter().append("text")
    .attr('x', 0)
    .attr('y', function(d, i) {
        return (i + 1) * (barHeight + padding);
    })
    .attr('font-size', fontSize)
    .text(function(d) {
        return d.name;
    });
```

- Set font size and text content

Populations of cities - Scaling



- Scale populations
  - so that we can display bars within the screen

```
var scaleFactor = 1e-5;
var rects =  svg.selectAll("rect")
    .data(cities)
    .enter().append("rect")
    .attr("x", 80)
    .attr("y", function(d, i) {
        return padding + i * (barHeight + padding);
    })
    .attr("width", function(d, i) {
        return d.population * scaleFactor;
    })
    .attr("height", barHeight)
    .style("fill", "steelblue");
```

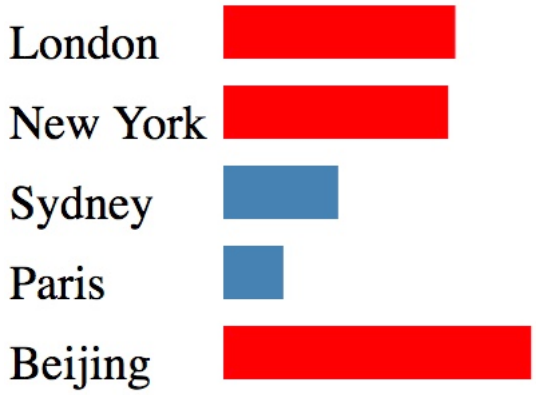
```
var cities = [
  { name: 'London', population: 8674000},
  { name: 'New York', population: 8406000},
  { name: 'Sydney', population: 4293000},
  { name: 'Paris', population: 2244000},
  { name: 'Beijing', population: 11510000}
];
```

- London: 86.74 pixels
- New York: 84.06 pixels
- Sydney: 42.93 pixels
- Paris: 22.44 pixels
- Beijing: 115.1 pixels

Populations of cities - Filter cities

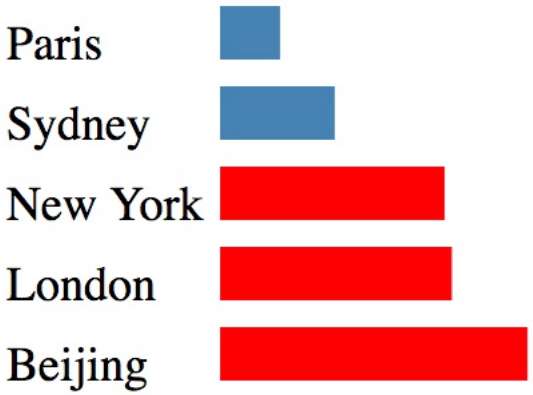
- *selection*.filter(*filter*)
  - Filters the selection, returning a new selection that contains only the elements for which the specified *filter* is true.
- Highlight cities that populations are larger than five million

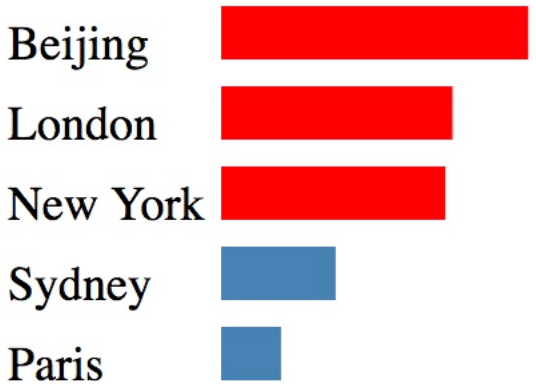
```
rects
.filter(function(d, i) {
    return d.population > 5 * 1e6;
})
.style("fill", "red");
```



Populations of cities - Sort cities

- Ascending/Descending order





Populations of cities - Sort cities

- *selection*.sort(*compare*)
  - Returns a new selection that contains a copy of each element in this selection sorted according to the *compare* function

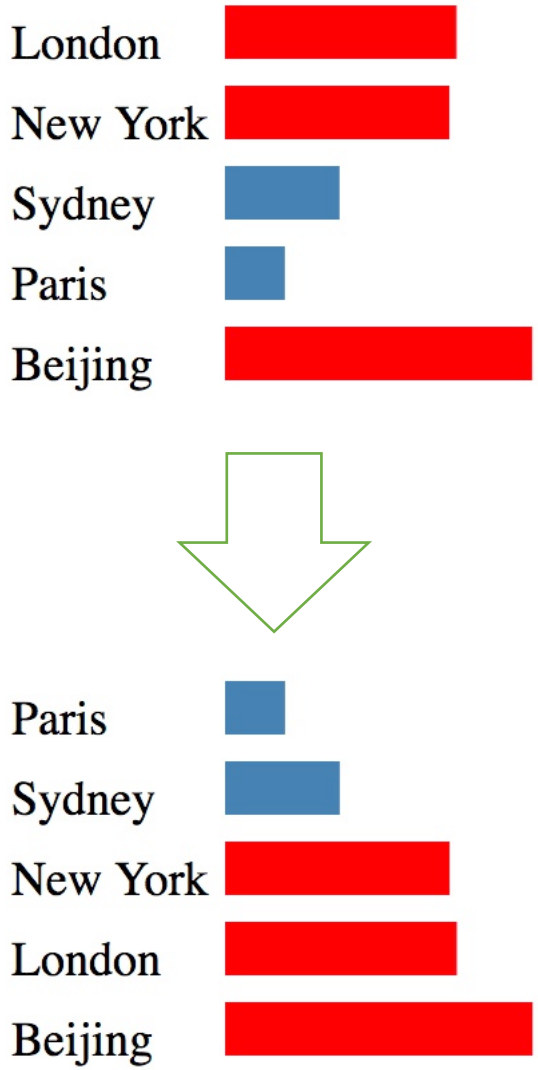
```
texts
    .sort(function(a, b) {
        return d3.ascending(a.population, b.population);
        // return d3.descending(a.population, b.population);
    })
    .attr("y", function(d, i) {
        return (i + 1) * (barHeight + padding);
    })

rects
    .sort(function(a, b) {
        return d3.ascending(a.population, b.population);
        // return d3.descending(a.population, b.population);
    })
    .attr("y", function(d, i) {
        return padding + i * (barHeight + padding);
    })
```

- Sort *texts* and *rects* in ascending/descending order

Populations of cities - Sort cities

- Re-assign *y* coordinates

```
texts
    .sort(function(a, b) {
        return d3.ascending(a.population, b.population);
        // return d3.descending(a.population, b.population);
    })
    .attr("y", function(d, i) {
        return (i + 1) * (barHeight + padding);
    })

rects
    .sort(function(a, b) {
        return d3.ascending(a.population, b.population);
        // return d3.descending(a.population, b.population);
    })
    .attr("y", function(d, i) {
        return padding + i * (barHeight + padding);
    })
```



Populations of cities – Create a table

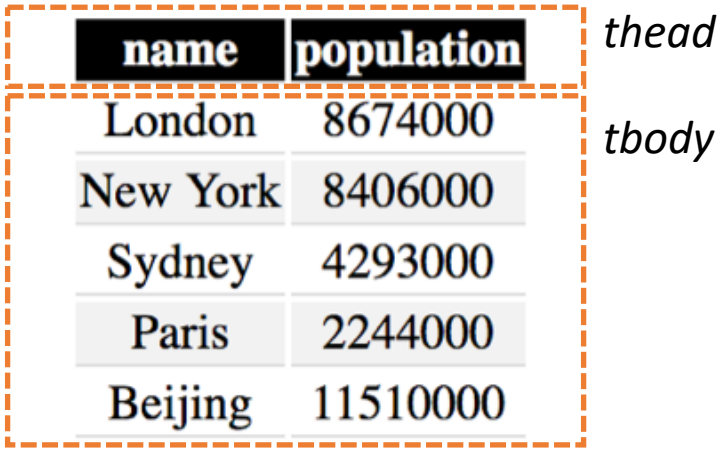
- A table can show quantities clearly
  - We can create a table with the help of d3 selections

name	population
London	8674000
New York	8406000
Sydney	4293000
Paris	2244000
Beijing	11510000

- We need two parameters
  - columnNames: names of columns (i.e., an array [”name”, “population”])
  - data: names of cities and populations of cites

Populations of cities – Create a table

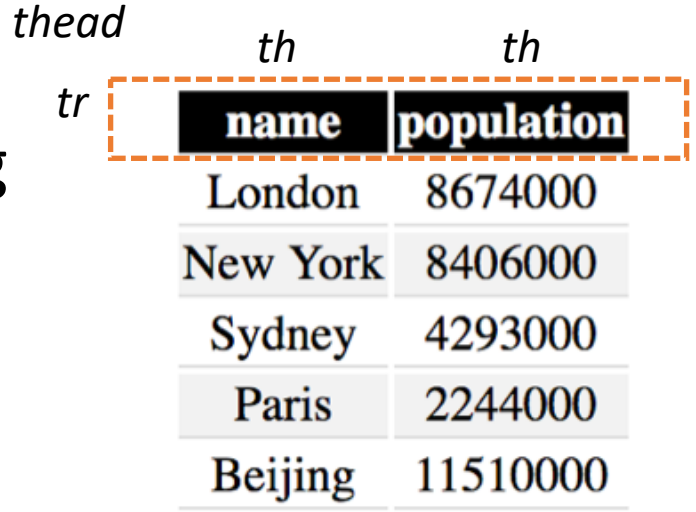
- Create *table* tag
  - In the *table* tag, we create *thead* tag and *tbody* tag
  - *thead* tag shows names of columns
  - *tbody* tag shows data




```
var table = d3.select('body').append('table');
var thead = table.append('thead');
var tbody = table.append('tbody');
```

Populations of cities – Create a table

- We add names of columns into the *thead* tag
  - Create a row by *tr* tag
  - Add two header cells by *th* tag in this row
    - The column names are put in the cells




```
thead.append('tr')
  .selectAll('th')
  .data(columnNames)
  .enter()
  .append('th')
  .text(function (d) {
    return d;
  });
```

```
▼<thead>
  ▼<tr>
      <th>name</th>
      <th>population</th>
    </tr>
  </thead>
```

Populations of cities – Create a table

- We add data (i.e., names and populations of cities) into the *tbody* tag
  - Create rows by *tr* tag
  - Add two standard cells by *td* tag in each row
    - The data are put in the cells

tbody

name	population
London	8674000
New York	8406000
Sydney	4293000
Paris	2244000
Beijing	11510000

tr

```
var rows = tbody.selectAll('tr')
  .data(data)
  .enter()
  .append('tr');
```

```
var cells = rows.selectAll('td')
  .data(function (row) {
    return columnNames.map(function (columnName) {
      return {
        key: columnName,
        value: row[columnName]
      };
    });
  })
  .enter()
  .append('td')
  .text(function (d) {
    return d.value;
  });
```

```
▼<tbody>
 ▼<tr>
    <td>London</td>
    <td>8674000</td>
  </tr>
 ▼<tr>
    <td>New York</td>
    <td>8406000</td>
  </tr>
 ▼<tr>
    <td>Sydney</td>
    <td>4293000</td>
  </tr>
 ▼<tr>
    <td>Paris</td>
    <td>2244000</td>
  </tr>
 ▼<tr>
    <td>Beijing</td>
    <td>11510000</td>
  </tr>
</tbody>
```

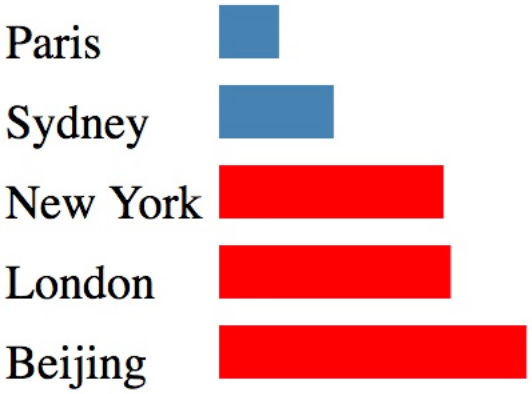
Data Loading

- Loading data from external files
  - d3.json, d3.csv, d3.html, d3.txt, d3.tsv, .d3xml
- d3.json(*input*, *callback*)
  - *callback* function will be invoked after data is loaded

city_population.json

```
{
    "cities":
        [
            {
                "name": "London",
                "population": 8674000
            },
            {
                "name": "New York",
                "population": 8406000
            },
            {
                "name": "Sydney",
                "population": 4293000
            },
            {
                "name": "Paris",
                "population": 2244000
            },
            {
                "name": "Beijing",
                "population": 11510000
            }
        ]
}
```

```
d3.json("city_population.json", drawBars);

function drawBars(error, data) {
    var cities = data["cities"];

......

var texts = svg.selectAll('text')
    .data(cities)

......

var rects =  svg.selectAll("rect")
    .data(cities)
......
```



Data Loading - CSV

- d3.csv(*input*, *callback*)
- For example, draw ten points

points.csv

```
ID,x,y,r
0,3,6,8
1,9,10,1
2,9,5,5
3,10,2,9
4,3,5,5
5,2,7,9
6,6,7,10
7,8,8,4
8,5,2,2
9,9,1,1
```

```
d3.csv("points.csv", drawPoints);

function drawPoints(error, points) {

▼Array(10) i
  ▶0: {ID: "0", x: "3", y: "6", r: "8"}
  ▶1: {ID: "1", x: "9", y: "10", r: "1"}
  ▶2: {ID: "2", x: "9", y: "5", r: "5"}
  ▶3: {ID: "3", x: "10", y: "2", r: "9"}
  ▶4: {ID: "4", x: "3", y: "5", r: "5"}
  ▶5: {ID: "5", x: "2", y: "7", r: "9"}
  ▶6: {ID: "6", x: "6", y: "7", r: "10"}
  ▶7: {ID: "7", x: "8", y: "8", r: "4"}
  ▶8: {ID: "8", x: "5", y: "2", r: "2"}
  ▶9: {ID: "9", x: "9", y: "1", r: "1"}
  ▶columns: (4) ["ID", "x", "y", "r"]
   length: 10
```

- The variable *points* stores the received csv data
- The loaded data is in the form of key: value

Data Loading - CSV

- d3.csv(*input*, *callback*)
- For example, draw ten points

points.csv

```
ID,x,y,r
0,3,6,8
1,9,10,1
2,9,5,5
3,10,2,9
4,3,5,5
5,2,7,9
6,6,7,10
7,8,8,4
8,5,2,2
9,9,1,1
```

```
d3.csv("points.csv", drawPoints);

function drawPoints(error, points) {

▼ Array(10)
  ▶ 0: {ID: "0", x: "3", y: "6", r: "8"}
  ▶ 1: {ID: "1", x: "9", y: "10", r: "1"}
  ▶ 2: {ID: "2", x: "9", y: "5", r: "5"}
  ▶ 3: {ID: "3", x: "10", y: "2", r: "9"}
  ▶ 4: {ID: "4", x: "3", y: "5", r: "5"}
  ▶ 5: {ID: "5", x: "2", y: "7", r: "9"}
  ▶ 6: {ID: "6", x: "6", y: "7", r: "10"}
  ▶ 7: {ID: "7", x: "8", y: "8", r: "4"}
  ▶ 8: {ID: "8", x: "5", y: "2", r: "2"}
  ▶ 9: {ID: "9", x: "9", y: "1", r: "1"}
  ▶ columns: (4) ["ID", "x", "y", "r"]
    length: 10
```

- Column names become the keys

- The *points.columns* is an array of column names

Data Loading - CSV

- We then draw circles and a table

```
function drawPoints(error, points) {
    var svg = d3.select('svg');
    svg.selectAll('circle')
        .data(points).enter()
        .append('circle')
        .attr('cx', function(d) {
            return xScale(d.x);
        })
        .attr('cy', function(d) {
            return yScale(d.y);
        })
        .attr('r', function(d) {
            return d.r;
        })
        .attr('fill', 'steelblue');

    var columnNames = points.columns;
    // Create a table
    tabulate(columnNames, points);
```

ID	x	y	r
0	3	6	8
1	9	10	1
2	9	5	5
3	10	2	9
4	3	5	5
5	2	7	9
6	6	7	10
7	8	8	4
8	5	2	2
9	9	1	1