Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 5

Introduction to D3.js

Contents

- The basics of HTML, CSS, and the Document Object Model (DOM)
- The principles of Scalable Vector Graphics (SVG)
- Node and ES2015 functionality
- Data-binding and selections with D3
- Different data types and their data visualization methods

What is D3.js?

- D3 stands for data-driven documents
- Use D3 if you want to develop web pages that needs to be:
  - Fast
  - Interactive
  - Animated
  - Shareable
- If you really want to have a simple visualization like pie chart layout, probably you can use other charting libraries
- But if you want to change the pie chart as the data changes then d3.js is useful

How D3 works

- It has several components
  - Data management
    - Data generation
    - Loading
    - Binding dataset
    - Processing
  - Charts
    - Basic charting
    - Design
    - Hierarchical layouts
    - Networks
    - Maps

How D3 works

- Interactivity
  - Mouse events
  - Brush filtering
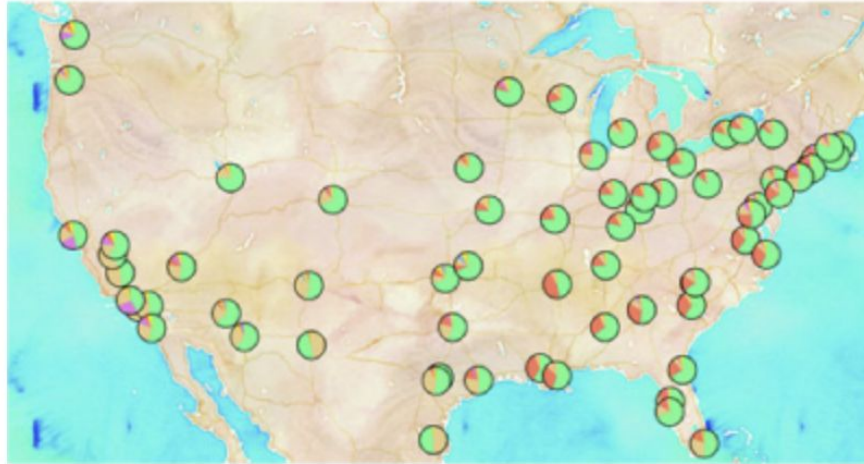 - Zoom
- MVC dashboard
- Optimization

Data visualization is more than charts

- D3 can be used for simple charts, such as this donut chart
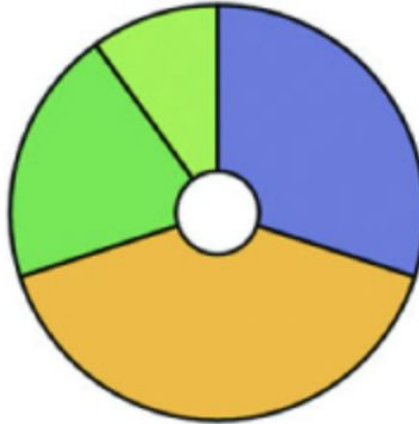
Data visualization is more than charts

- D3 can also be used to create web maps, such as this map showing the ethnic makeup of major metropolitan areas in the United States.
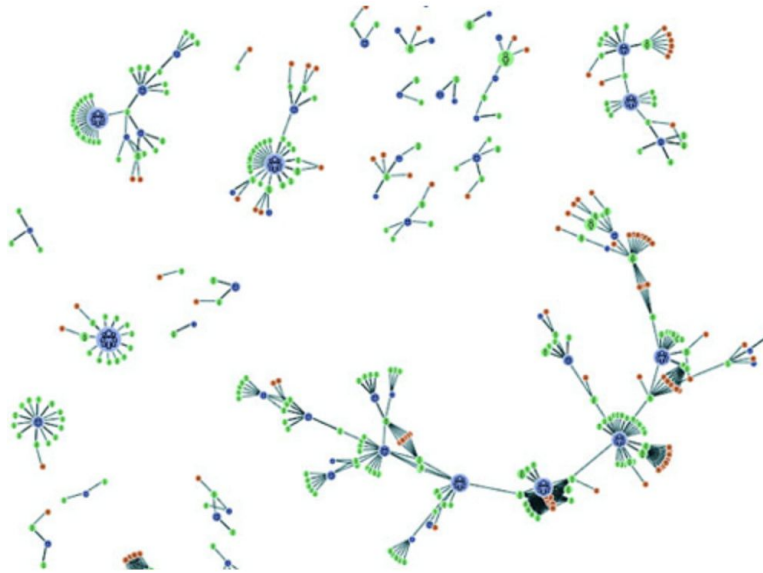
Data visualization is more than charts

- Maps in D3 aren't limited to traditional Mercator web maps. They can be interactive globes, like this map of undersea communication cables, or other, more
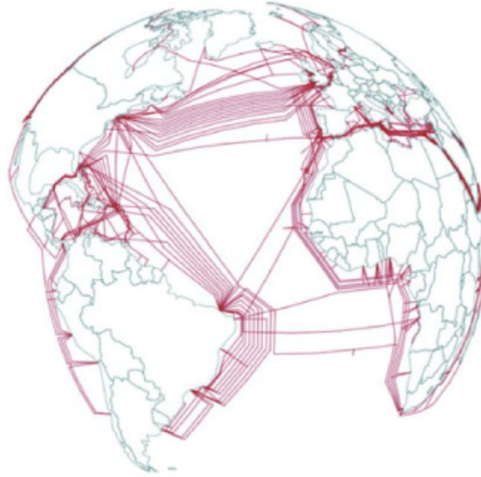
Data visualization is more than charts
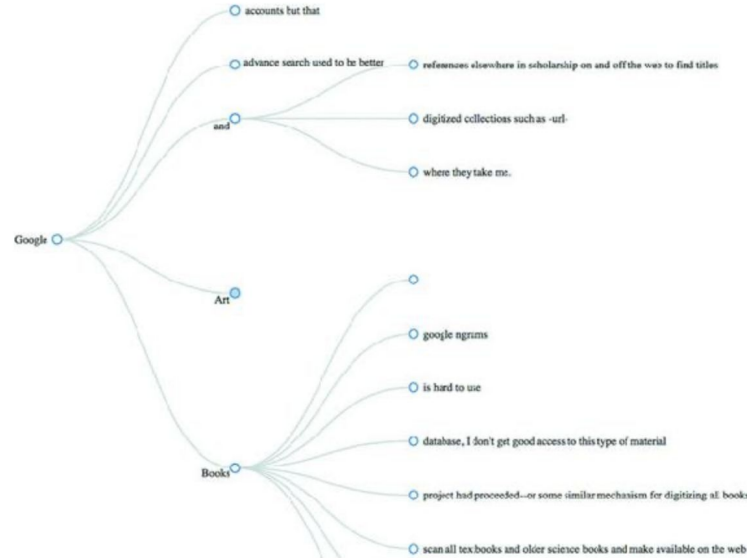
- D3 also provides robust capacities to create interactive network visualizations

Data visualization is more than charts

- D3 includes a library of common data visualization layouts, such as the dendrogram

Data visualization is more than charts

- D3 has SVG and canvas drawing functions so you can create your own custom visualizations, such as this representation of musical scores.
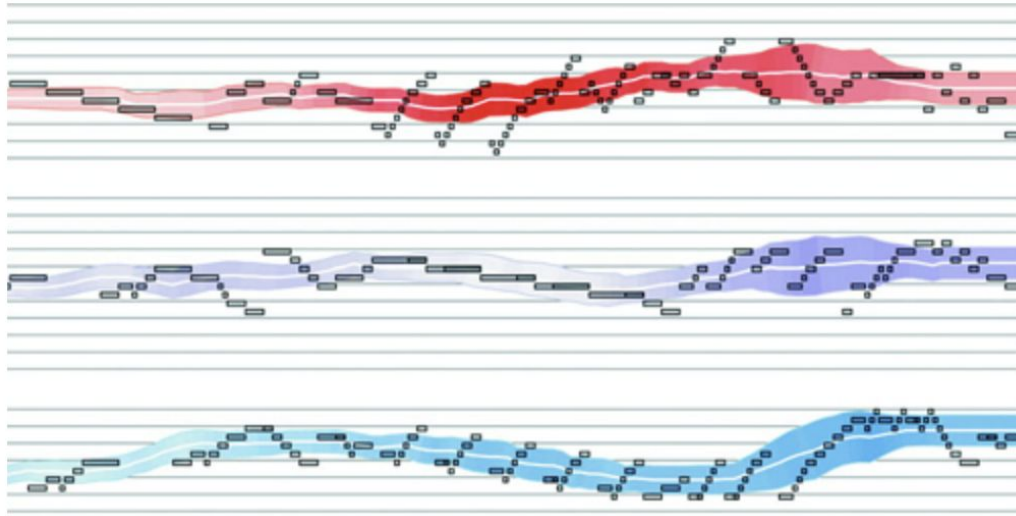
Data visualization is more than charts

You can combine these layouts and functions to create a data dashboard. You can also use the drawing functions to make your bar charts look distinctive, such as this "sketchy" style.
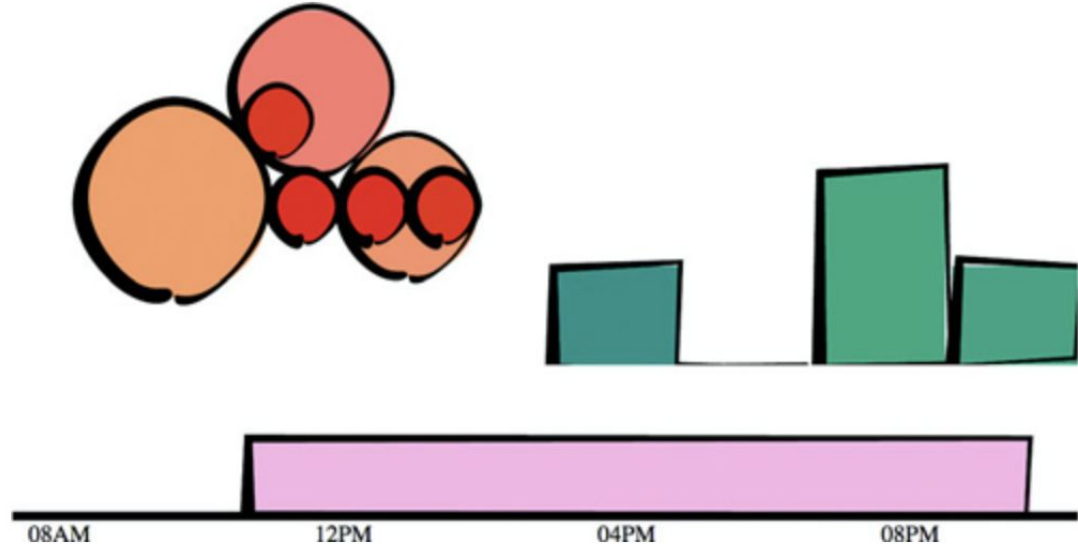
D3.js

- The fundamental principles for all these visualizations are same!
- Not only the method is same, the data could be also same
- The only difference is the representation
- We will now go through some of the fundamental concepts

D3 is about selecting and binding

- Concentrate on two things
  - The selection of elements in your web page
  - The data that you want to bind with the elements
- Here's a selection without any data:

```
d3.selectAll("circle.a").style("fill", "red").attr("cx", 100);
```

- Likewise, this code turns every div on our web page red and changes its class to b:

```
d3.selectAll("div").style("background", "red").attr("class", "b");
```

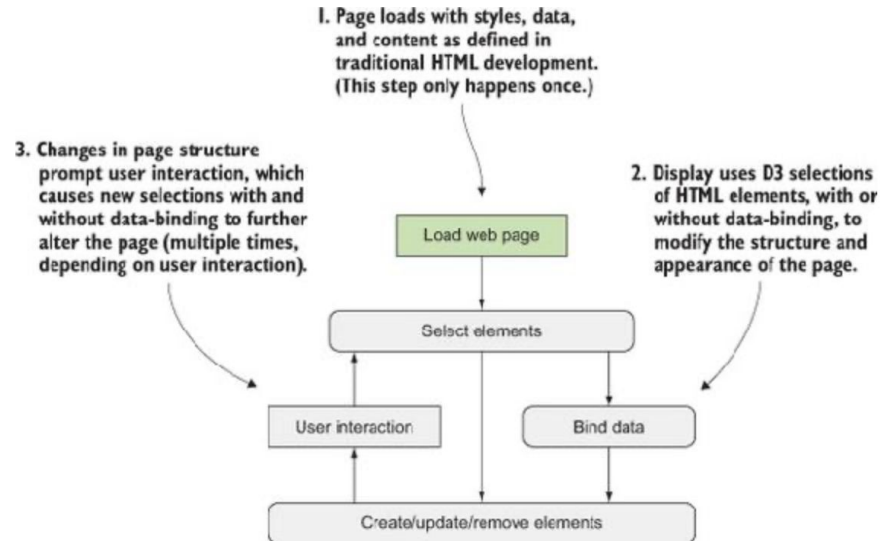# D3 is about selecting and binding

- So far we have seen selecting element, let's now bind some data:

  ```
  d3.selectAll("div.market").data([1,5,11,3])
  ```

- What if the number of data is more or less than the number of elements?
- There are nice ways to tackle these situations
- We will see in future lectures

D3 is about selecting and binding

An application created with D3 can use selections and data binding over and over again, together and separately, to update the content of the data visualization based on interaction.

1. Page loads with styles, data, and content as defined in traditional HTML development. (This step only happens once.)

3. Changes in page structure prompt user interaction, which causes new selections with and without data-binding to further alter the page (multiple times, depending on user interaction).

2. Display uses D3 selections of HTML elements, with or without data-binding, to modify the structure and appearance of the page.

Load web page

Select elements

User interaction

Bind data

Create/update/remove elements

Web page elements can now be divs, countries, and flowcharts

- The fundamental principle is to selecting and binding data
- You may need to do it continuously because of user interaction or due to other events
- The elements will change, which will create different types of visualizations

The power of HTML5

- 20-30 years ago the only type of animation in web was GIF
- GIF is one type of animation but user do not have any control
- Modern HTML has different feature to control the contents:
  - The DOM
  - SVG
  - Canvas
  - CSS
  - Javascript
  - ES2015 and Node

The DOM

```
<!doctype html>
<html>
<head>
  <script src="d3.v4.min.js"></script>
</head>
<body>
  <div id="someDiv" style="width:200px;height:100px;border:black
<input id="someCheckbox" type="checkbox" />
  </div>
</body>
</html>
```

The DOM

- From the HTML code the browser creates a tree
- For example the <html> is the root of the tree
- The <head> and <body> are children of the root
- ….
- Three categories of information about each element determine its behavior and appearance: styles, attributes, and properties.

```
d3.select("#someDiv").style("border", "5px darkgray dashed");
d3.select("#someDiv").attr("id", "newID");
d3.select("#someCheckbox").property("checked", true);
```

The DOM

- One technical point: from the CSS lecture, we remember that there is z-index that determines the order of overlapping element
- In SVG version 1 this feature is not available
- So we want to append the elements carefully

Examining the DOM in the console

The developer tools in Chrome place the JavaScript console on the rightmost tab, labeled Console, with the element inspector available using the arrow in a rectangle (circled above) on the top left or by browsing the DOM in the Elements tab.

Examining the DOM in the console

You can run JavaScript code in the console and call global variables or declare new ones as necessary. Any code you write in the console and changes made to the web page are lost as soon as you reload the page.
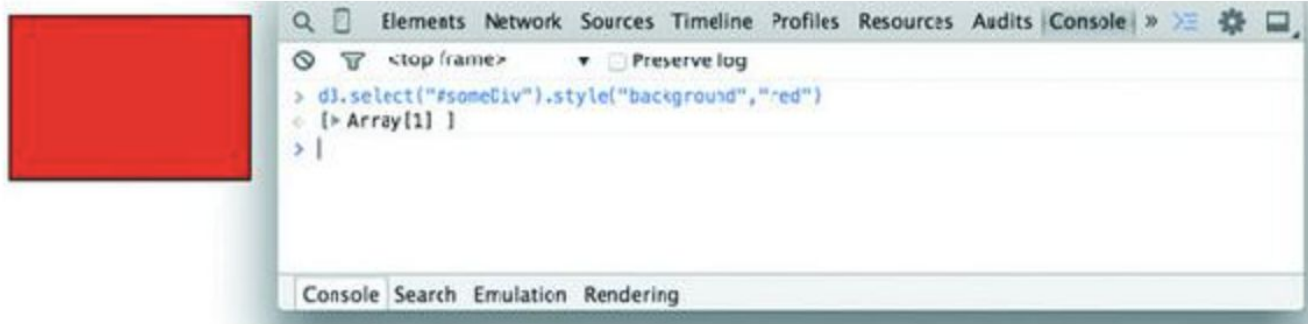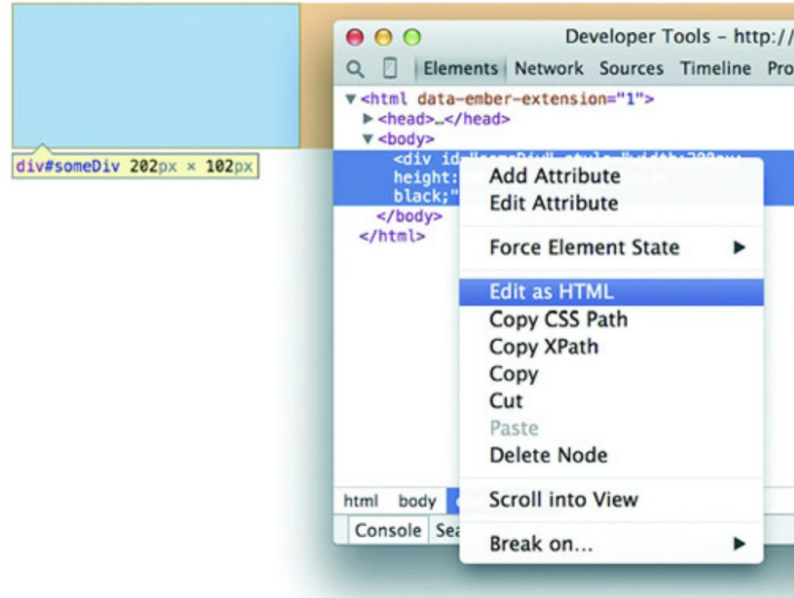
Examining the DOM in the console

- You can put breakpoints, which really helps for debugging
- You can modify elements in console
- You can delete elements
- But the elements can be reloaded
- You can manually change different elements

Examining the DOM in the console

Rather than adding or modifying individual styles and attributes, you can rewrite the HTML code as you would in a text editor. As with any changes, these only last until you reload the page.

# Examining the DOM in the console

Changing the content of a DOM element is as simple as adding text between the opening and ending brackets of the element.

```
▶ <head>…</head>
▼ <body>
    <div id="someDiv"
    style="width:200px;height:100px;border
    :1px solid black;">Here's some text to
    put into my div</div>
  </body>
</html>
```
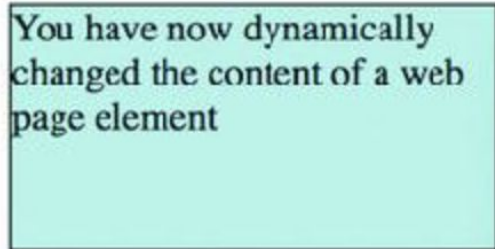
Examining the DOM in the console

The page is updated as soon as you finish making your changes. Writing HTML manually in this way is only useful for planning how you might want to dynamically update the content.
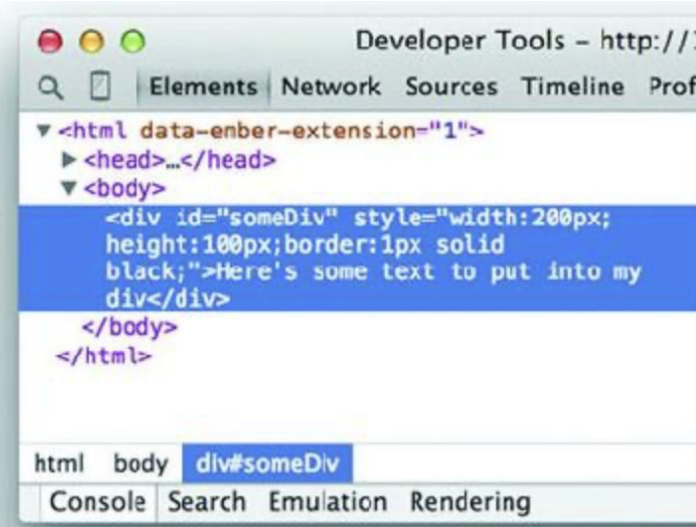
Coding in the console

d3.select("div").style("background", "lightblue").style("border", "solid black 1px").html("You have now dynamically changed the content of a web page element");

SVG

The commands to draw an SVG path (right) and the resulting graphic (left)

m 25.447534,104.50879 c 35.89856,7.45849 74.730166,45.01366
108.571426,11.42857 28.09984,-27.887135 47.81916,-92.985425
111.42857,-74.285715 31.98127,9.40176 26.13586,71.972865
62.85714,71.428565 55.39866,-0.82115 101.36764,-112.52070547
160,-45.714275 44.69105,50.921525 55.38932,45.730385
91.42358,37.142855
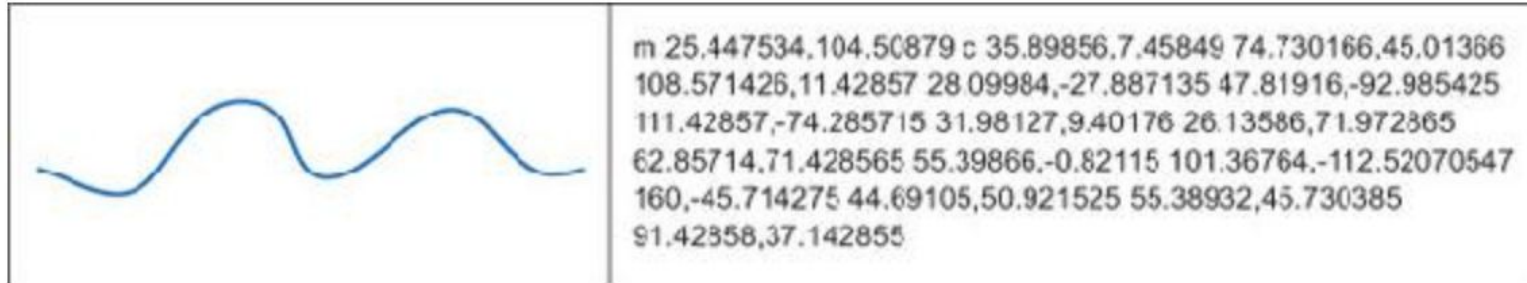
SVG

A sample web page with SVG elements

```html
<!doctype html>
<html>
    <script src="d3.v4.min.js">
</script>
<body>
  <div id="infovizDiv">
  <svg style="width:500px;height:500px;border:1px lightgray s
    <path d="M 10,60 40,30 50,50 60,30 70,80"
        style="fill:black;stroke:gray;stroke-width:4px;" />
    <polygon style="fill:gray;"
        points="80,400 120,400 160,440 120,480 60,460" />
  <g>
  <line x1="200" y1="100" x2="450" y2="225"
  style="stroke:black;stroke-width:2px;"/>
  <circle cy="100" cx="200" r="30"/>
  <rect x="410" y="200" width="100" height="50"
        style="fill:pink;stroke:black;stroke-width:1px;" />
  </g>
  </svg>
  </div>
</body>
</html>
```
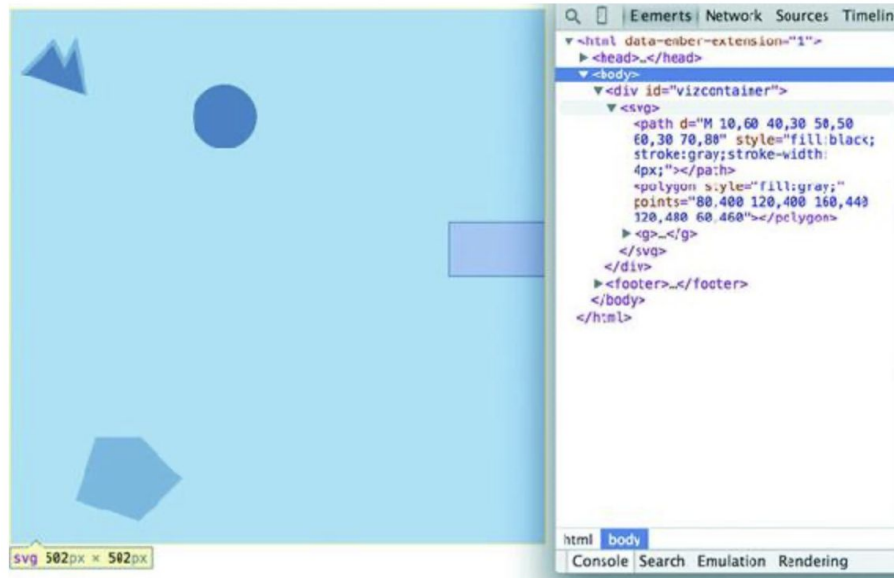
SVG

```
d3.select("circle").remove()                    1
d3.select("rect").style("fill", "purple")       2
```

- *1* Deletes the circle
- *2* Changes the rectangle color to purple

SVG

Inspecting the DOM of a web page with an SVG canvas reveals the nested graphical elements as well as the style and attributes that determine their position. Notice that the circle and rectangle exist as child elements of a group.

SVG

```
d3.select("svg").style("background", "darkgray");          1
```

- *1* Infoviz is always cooler on a dark background

Canvas

- Creating static images
- Large amount of data
- WebGL for 3d

Geometric primitives

- Circle
- Rect
- Line
- Ellipse
- Polygon

Geometric primitives

Modifying the height and width attributes of a <rect> element changes the appearance of that element. Inspecting the element also shows how the stroke adds to the computed size of the element.

Text in SVG

- doesn't have the formatting support found in HTML elements
- primarily used for labels
- If you do want to do basic formatting, you can nest <tspan> elements in <text>

Grouping element

- Use the <g> tag for grouping, like a node and a label
- Does not has any graphical representation
- It's a logical grouping
- Is used extensively
- You can move a group together

```
<g>
<circle r="2"/>
<text>This circle's Label</text>
</g>
```

Grouping

All SVG elements can be affected by the transform attribute, but this is particularly salient when working with <g> elements, which require this approach to adjust their position. The child elements are drawn by using the position of their parent <g> as their relative 0,0 position. The scale() setting in the transform attribute then affects the scale of any of the size and position attributes of the child elements.

No transform:
Like all SVG elements, the <g> is by default drawn at the 0,0 position. The result in this case is that the graphics are drawn such that you only see a tiny piece of the bottom right corner of the circle, and none of the label text.

Translate(100,50):
The entire <g> element is moved 100 pixels to the right (along the x-axis) and 50 pixels down (along the y-axis). The child elements are drawn from this position.

A Label

Translate(100,400)Scale(3.5):
The <g> element is moved to XY position 100,400, lining it up along the y-axis with the above <g>, and the child elements are drawn from there at 3.5 times their size.
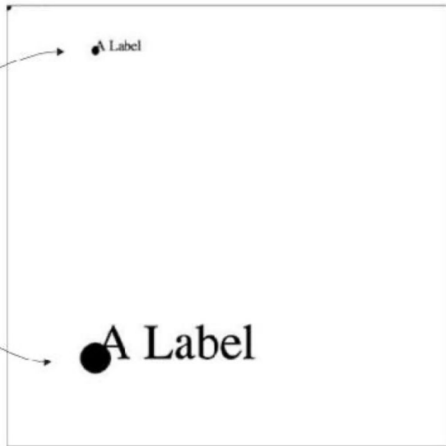
A Label

```
<g>
    <circle r="2"/>
    <text>This circle's Label</text>
</g>
<g transform="translate(100,50)">
    <circle r="2" />
    <text>This circle's Label</text>
</g>
<g transform="translate(100,400) scale(2.5)">
    <circle r="2"/>
    <text>This circle's Label</text>
</g>
```

Path

Open – unfilled:
Path elements are by default filled with no stroke. You need to set the fill style to "none" and stroke and stroke-width style if you want to draw it as a line.

Open – filled:
An open path can be filled just like a closed path, with the fill area defined by the same area that would be bounded if the path were closed.

Closed – unfilled:
A path will always close by drawing a line from the end point to the start point.

Closed – filled:
Notice the stroke overlaps the fill area slightly.

```
<path style="fill:none;stroke:gray;stroke-width:4px;"
    d="M 10,60 40,30 50,50 60,30 70,80"
transform="translate(0,0)" />
<path style="fill:black;stroke:gray;stroke-width:4px;"
    d="M 10,60 40,30 50,50 60,30 70,80"
transform="translate(0,100)" />
<path style="fill:none;stroke:gray;stroke-width:4px;"
    d="M 10,60 40,30 50,50 60,30 70,80Z"
transform="translate(0,200)" />
<path style="fill:black;stroke:gray;stroke-width:4px;"
    d="M 10,60 40,30 50,50 60,30 70,80Z"
transform="translate(0,300)" />
```

CSS

- Use to style the elements in the DOM
- Can exist in a separate CSS file
- Can be embedded in the HTML
- Refers to
  - ID
  - Class
  - Type of element
- Inline style (for a single element) to set the opacity:

```
d3.select(#someElement).style(opacity, .5)
```

CSS

Examining an SVG rectangle in the console shows that it inherits its fill style from the CSS style applied to <rect> types and its stroke style from the .active class.
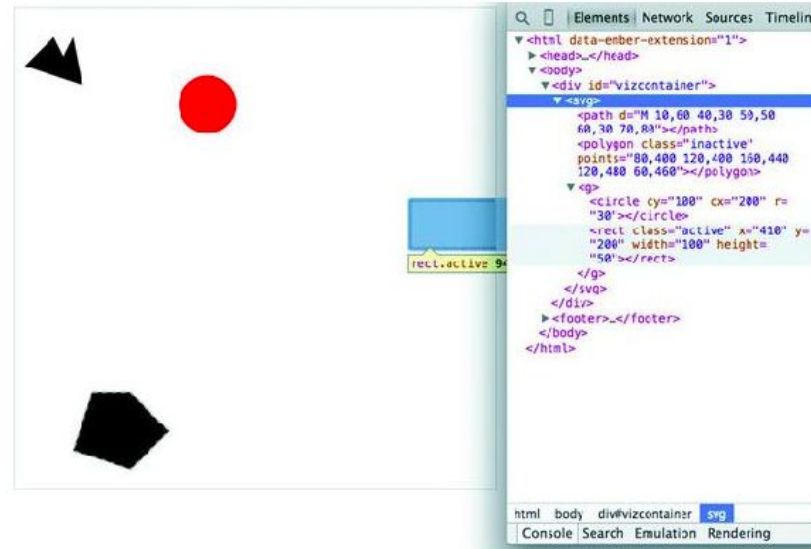
CSS

- A simple approach:
  - Create different style for different class
  - Change the class of the element

```
d3.select("circle").attr("class", "tentative");
```

- This overwrites the entire class attribute
- One can also add or remove classes from the set of classes:

```
d3.select("circle").classed("active", true);
```

CSS

```
<style>
.inactive, .tentative {
  stroke: darkgray;
  stroke-width: 2px;
  stroke-dasharray: 5 5;
}
.tentative {
  opacity: .5;
}
.active {
  stroke: black;
  stroke-width: 4px;
  stroke-dasharray: 1;
}
circle {
  fill: red;
}
rect {
  fill: darkgray;
}
</style>
```

An SVG circle with fill style determined by its type and its opacity and stroke settings determined by its membership in the tentative class

Javascript

- Can create and modify web elements
- Makes drawing and updating SVG reusable and maintainable
- Two fundamental subjects:
  - Method chaining
  - Array

```
d3.selectAll("div").data(someData).enter().append("div").html("Wow").append("span").html("Even
More Wow").style("font-weight", "900");
```

Javascript

```
d3.selectAll("div")                          1
  .data(someData)                            2
  .enter()                                   3
  .append("div")                             4
  .html("Wow")                               5
  .append("span")                            6
  .html("Even More Wow")                     7
  .style("font-weight", "900");              8
```

1 **Returns a selection, an object with various functions available**

Javascript

```
d3.selectAll("div")                           1
  .data(someData)                             2
  .enter()                                    3
  .append("div")                              4
  .html("Wow")                                5
  .append("span")                             6
  .html("Even More Wow")                      7
  .style("font-weight", "900");               8
```

2 **Sets the data on the selection and returns the selection**

Javascript

```
d3.selectAll("div")                          1
  .data(someData)                            2
  .enter()                                   3
  .append("div")                             4
  .html("Wow")                               5
  .append("span")                            6
  .html("Even More Wow")                     7
  .style("font-weight", "900");              8
```

3 **Returns the selection.enter() object**

Javascript

```
d3.selectAll("div")                              1
  .data(someData)                                2
  .enter()                                       3
  .append("div")                                 4
  .html("Wow")                                   5
  .append("span")                                6
  .html("Even More Wow")                         7
  .style("font-weight", "900");                  8
```

4 Sets .append() behavior on the selection.enter object and returns the selection.enter object

Javascript

```
d3.selectAll("div")                              1
  .data(someData)                                2
  .enter()                                       3
  .append("div")                                 4
  .html("Wow")                                   5
  .append("span")                                6
  .html("Even More Wow")                         7
  .style("font-weight", "900");                  8
```

5 Sets .html() for the selection.enter object and returns the selection.enter object

Javascript

```
d3.selectAll("div")                              1
   .data(someData)                               2
   .enter()                                      3
   .append("div")                                4
   .html("Wow")                                  5
   .append("span")                               6
   .html("Even More Wow")                        7
   .style("font-weight", "900");                 8
```

6 Sets the append() behavior on the selection.enter object and returns the enter.append object

Javascript

```
d3.selectAll("div")                          1
   .data(someData)                           2
   .enter()                                  3
   .append("div")                            4
   .html("Wow")                              5
   .append("span")                           6
   .html("Even More Wow")                    7
   .style("font-weight", "900");             8
```

7 Sets the html() for the enter.append object and returns the enter.append object

Javascript

```
d3.selectAll("div")                           1
  .data(someData)                             2
  .enter()                                    3
  .append("div")                              4
  .html("Wow")                                5
  .append("span")                             6
  .html("Even More Wow")                      7
  .style("font-weight", "900");               8
```

8 Sets the font-weight style on the enter.append object and returns the enter.append object

Javascript

```
var function1 = d3.selectAll("div");
var function1withData = function1.data(someData);
var function2 = function1withData.enter();
var function3 = function2.append("div");
function3.html("Wow");
var function4 = function3.append("span");
function4.html("Even More Wow");
function4.style("font-weight", "900");
```

Javascript

```
<!doctype html>
<html>
<script src="d3.v4.min.js"></script>
<style>
   #borderdiv {
   width: 200px;
   height: 50px;
border: 1px solid gray;
}
</style>
<body>
   <div id="borderdiv"></div>
   </body>
</html>
```

Javascript

```
var someData = ["filler", "filler", "filler", "filler"];
d3.select("body").selectAll("div")
  .data(someData)
  .enter()
  .append("div")
  .html("Wow")
  .append("span")
  .html("Even More Wow")
  .style("font-weight", "900");
```

Javascript

By binding an array of four values to a selection of <div> elements on the page, the .enter() function created three new <div> elements to reflect the size mismatch between the data array and the selection.
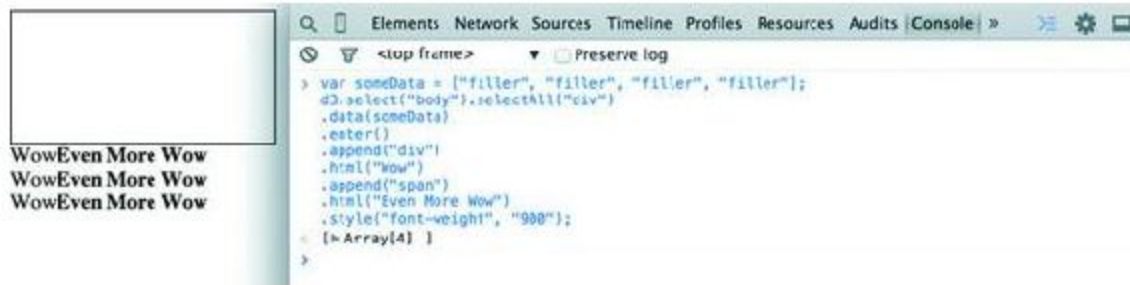
Javascript

Inspecting the DOM shows that the new <div> elements have been created with unformatted content followed by the child <span> element with style and content set by your code.
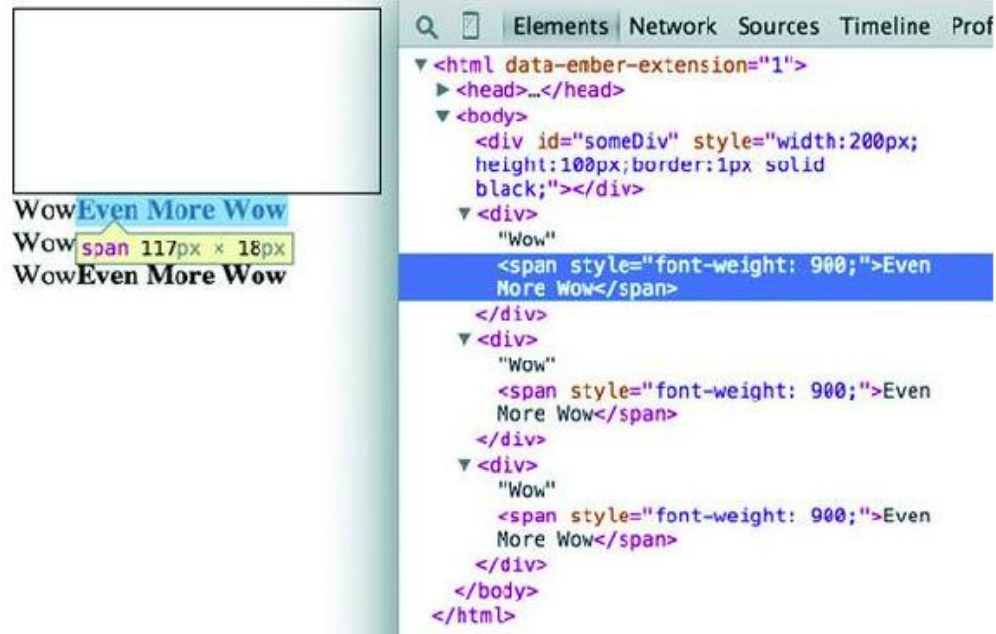
Arrays and array function

- Arrays can contain numbers, string etc.

  ```
  someNumbers = [17, 82, 9, 500, 40];
  someColors = ["blue", "red", "chartreuse", "orange"];
  ```

- Or it can contain objects.

  ```
  somePeople = [{name: "Peter", age: 27}, {name: "Sulayman", age: 24},
  {name: "K.C.", age: 49}];
  ```

Arrays and array function
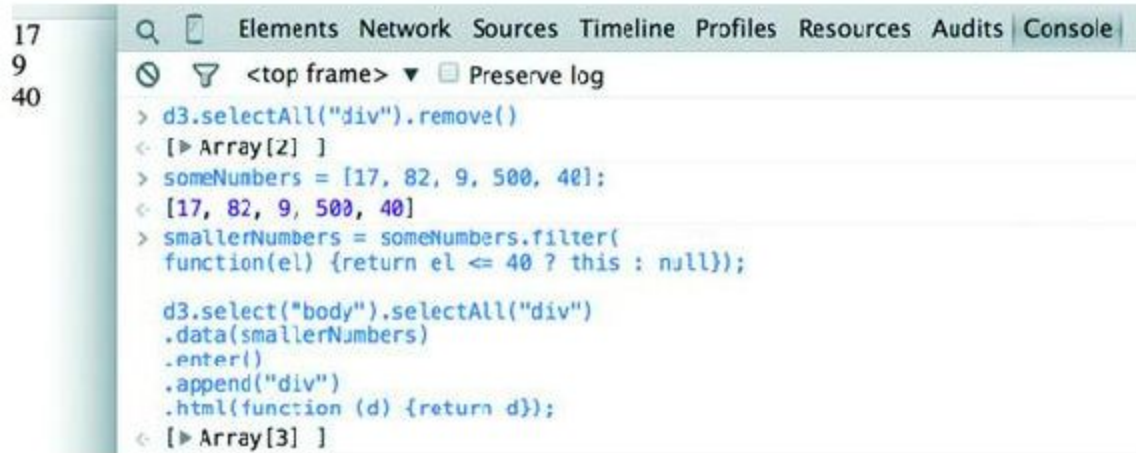
- You can filter based on a condition

```
someNumbers.filter(function(d) {return d >= 40});
```

```
someColors.filter(function(d) {return d.length < 5});
```

Arrays and array function

Running JavaScript in the console allows you to test your code. Here you've created a new array called smallerNumbers that consists of only three values, which you can then use as your data in a selection to update and create new elements.

```
17
9
40
```

```
Q  🔲   Elements  Network  Sources  Timeline  Profiles  Resources  Audits  Console
🚫  🔽  <top frame> ▼  ☐ Preserve log
> d3.selectAll("div").remove()
⟵ [▶ Array[2] ]
> someNumbers = [17, 82, 9, 500, 40];
⟵ [17, 82, 9, 500, 40]
> smallerNumbers = someNumbers.filter(
  function(el) {return el <= 40 ? this : null});

  d3.select("body").selectAll("div")
  .data(smallerNumbers)
  .enter()
  .append("div")
  .html(function (d) {return d});
⟵ [▶ Array[3] ]
```

Arrays and array function

- Anonymous function are use to access the bounded data

```
.style("background", function(d) {return d})
.attr("cx", function(d,i) {return i})
.html(function(d) {return d})
```

ES2015 and Node

- EcmaScript is a standard for scripting languages
- Javascript follows EcmaScript 6 (ES2015)
- Widely used in browser
- But can be used in backend using Node.js
  - Uses Node package Manager (NPM)
  - Do not need to use script tags
  - Can import already existing modules
  - Uses array functions (don't access "this" keyword)

```
someArray.forEach(d => {
    console.log(d)
})
```

```
someArray.forEach(function (d) {
    console.log(d)
})
```

Data standards

- tabular data
- nested data
- network data
- geographic data
- raw data
- objects

Tabular data

Delimited data can be expressed in different forms

| Name,Age,Salary	Name Age Salary	Name|Age|Salary
1.1.1 d3.csv	1.1.2 d3.tsv	1.1.3 d3.dsv
1.1.4 Sal,34,50000	1.1.5 Sal 34 50000	1.1.6 Sal|34|50000
1.1.7 Nan,22,75000	1.1.8 Nan 22 75000	1.1.9 Nan|22|75000

Tabular data

- d3.csv()
- d3.tsv()
- d3.dsv()

Nested data

Nested data represents parent/child relationships of objects, typically with each object having an array of child objects, and is represented in a number of forms, such as this dendrogram. Notice that each object can have only one parent.
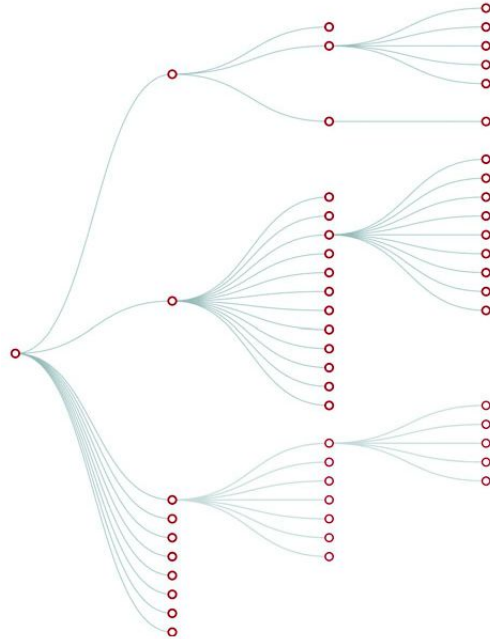
Network data

Network data consists of objects and the connections between them. The objects are typically referred to as nodes or vertices, and the connections are referred to as edges or links. Networks are often represented using force-directed algorithms, such as the example here, that arrange the network in such a way as to pull connected nodes toward each other.
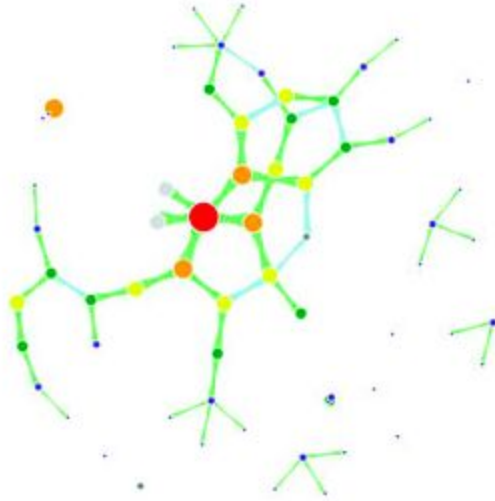
Geographic data

Geographic data stores the spatial geometry of objects, such as states. Each of the states in this image is represented as a separate feature with an array of values indicating its shape. Geographic data can also consist of points, such as for cities, or lines, such as for roads.
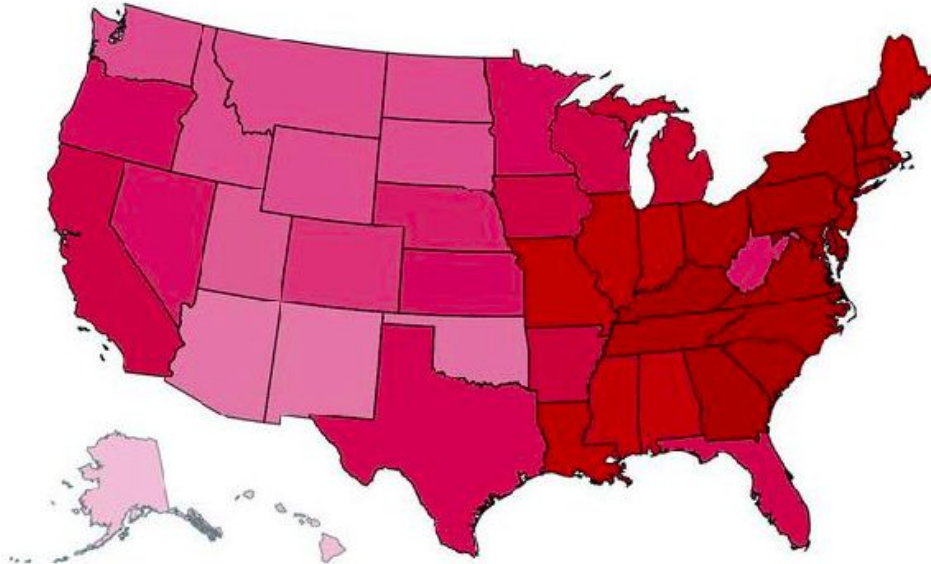
Objects

- JSON (JavaScript Object Notation)

var person = {name: "Charlie", age: 55, employed: true, childrenNames: ["Ruth", "Charlie Jr."]}

```
person.name // Charlie
person["name"] // Charlie
person.name = "Charles" // Sets name to Charles
person["name"] = "Charles" // Sets name to Charles
person.age < 65 // true
person.childrenNames // ["Ruth", "Charlie Jr."]
person.childrenNames[0] // "Ruth"
```

Objects

```
for (x in person) {console.log(x); console.log(person[x]);}
```

- One can also use Object.keys(person)
- Once we have the keys, we can iterate through them
- d3.csv() also returns a json object

A simple D3 app

```
<!doctype html>
<html>
<head>
    <script src="d3.v4.min.js"></script>
</head>
<body>
</body>
</html>
```

A simple D3 app

Using d3.select to set style and HTML content

```
d3.select("body").append("div")
  .style("border", "1px black solid")
  .html("hello world");
```

A simple D3 app

Using d3.select to set attributes and event listeners

```
d3.select("div")
  .style("background-color", "pink")
  .style("font-size", "24px")
  .attr("id", "newDiv")
  .attr("class", "d3div")
  .on("click", () => {console.log("You clicked a div")});
```

A simple D3 app

Using console.log(), you can test to see if an event is properly firing. Here you create a <div> and assign an onclick event handler using the .on() syntax. When you click that element and fire the event, the action is noted in the console.

Another simple app

A simple web page with an SVG canvas

```
<html>
<head>
  <script src="d3.v4.min.js"></script>
</head>
<body>
  <div id="vizcontainer">
  <svg style="width:500px;height:500px;border:1px lightgray solid;" />
  </div>
</body>
</html>
```
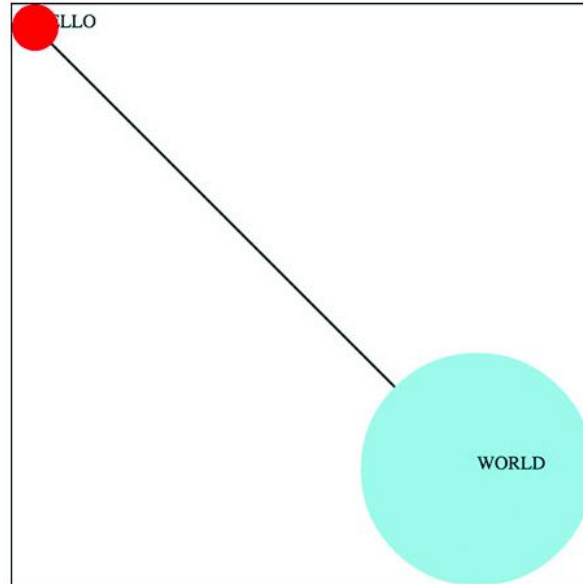
Another simple app

Creating lines and circles with select and append

```
d3.select("svg")
  .append("line")
  .attr("x1", 20)
  .attr("y1", 20)
  .attr("x2",400)
  .attr("y2",400)
  .style("stroke", "black")
  .style("stroke-width","2px");
d3.select("svg")
  .append("text")
  .attr("x",20)
  .attr("y",20)
  .text("HELLO");
d3.select("svg")
  .append("circle")
  .attr("r", 20)
  .attr("cx",20)
  .attr("cy",20)
  .style("fill","red");
```

```
d3.select("svg")
  .append("circle")
  .attr("r", 100)
  .attr("cx",400)
  .attr("cy",400)
  .style("fill","lightblue");
d3.select("svg")
  .append("text")
  .attr("x",400)
  .attr("y",400)
  .text("WORLD");
```

Another simple app

The result of running listing 1.10 in the console is the creation of two circles, a line, and two text elements. The order in which these elements are drawn results in the first label covered by the circle drawn later.
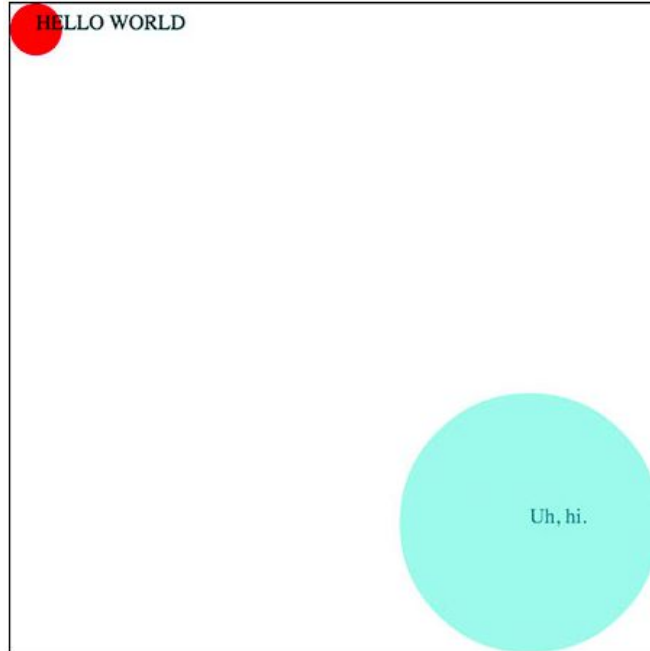
Another simple app

SVG elements with IDs and transparency

```
d3.select("svg")
  .append("circle")
  .attr("r", 20)
  .attr("cx",20)
  .attr("cy",20)
  .style("fill","red");
d3.select("svg")
  .append("text")
  .attr("id", "a")
  .attr("x",20)
  .attr("y",20)
  .style("opacity", 0)
  .text("HELLO WORLD");
```

```
d3.select("svg")
  .append("circle")
  .attr("r", 100)
  .attr("cx",400)
  .attr("cy",400)
  .style("fill","lightblue");
d3.select("svg")
  .append("text")
  .attr("id", "b")
  .attr("x",400)
  .attr("y",400)
  .style("opacity", 0)
  .text("Uh, hi.");
```
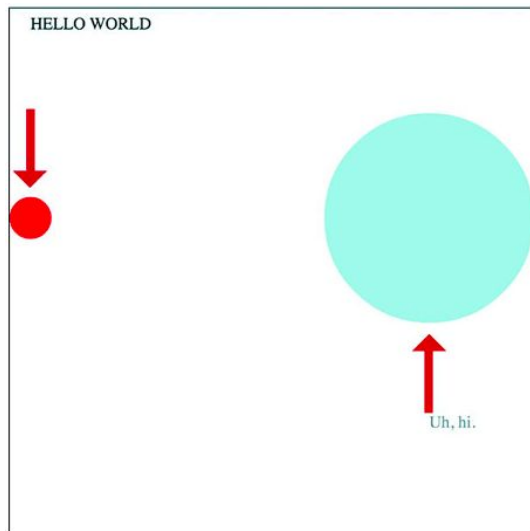
Another simple app

Transition behavior when associated with a delay results in a pause before the application of the attribute or style.

Another simple app

```
d3.select("#a").transition().delay(1000).style("opacity", 1);
d3.select("#b").transition().delay(3000).style("opacity", .75);
```



```
d3.selectAll("circle").transition().duration(2000).attr("cy", 200);
```