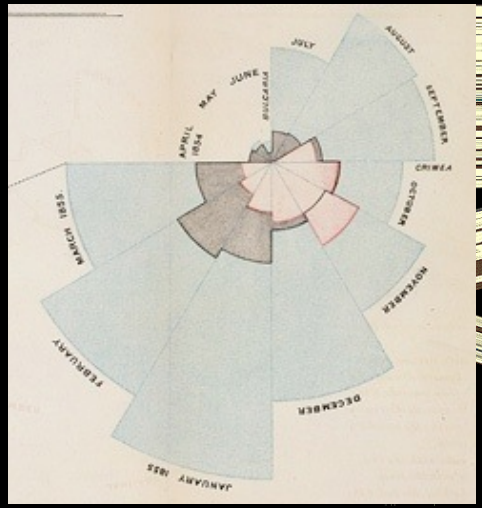
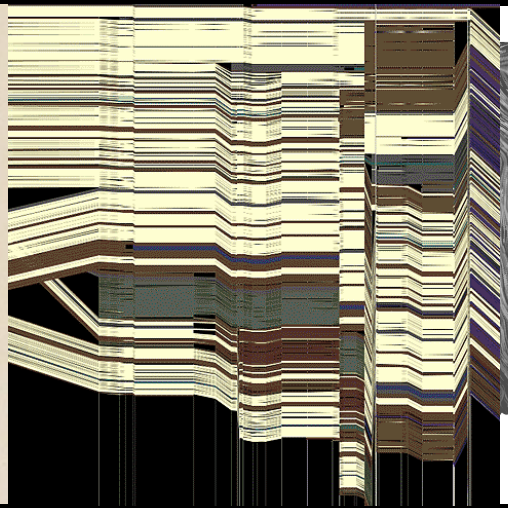
CSE 442 - Data Visualization

Web Programming + D3



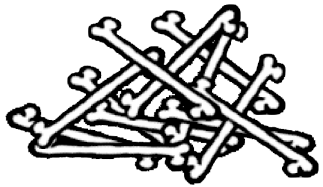




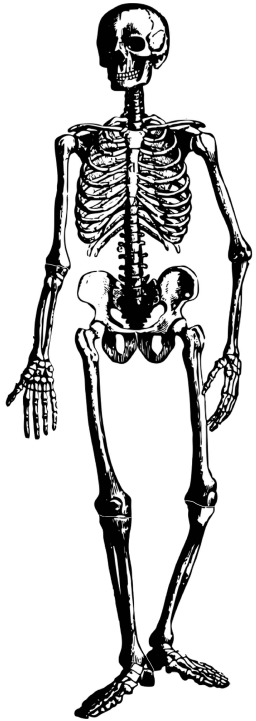
Vinay Koshy University of Washington

Web Programming

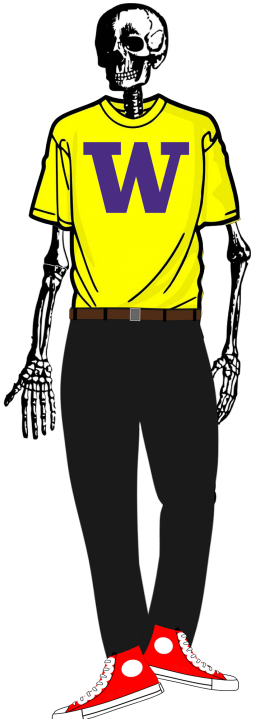
Adapted from CSE 154



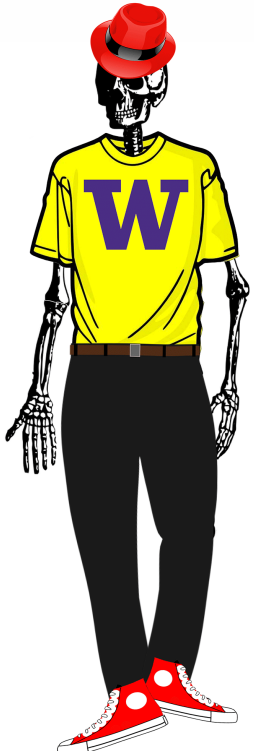
Content
Words/Images



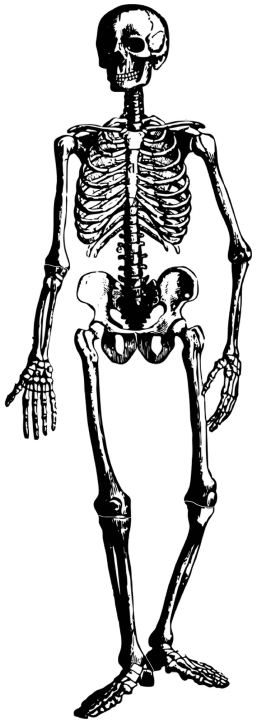
Structure
HTML



Style
CSS



Behavior
Javascript



Structure

HTML

HTML

- Describes the content and structure of information on a web page
- Surrounds text content with opening and closing tags
- Most whitespace is insignificant in HTML (ignored or collapsed to a single space)



SYNTAX: <tag>content</tag>

EXAMPLE: This is a paragraph

HTML

- Browser will interpret specific tags/structure as having certain basic visual attributes
- Tags can be nested to create a "hierarchy" of content

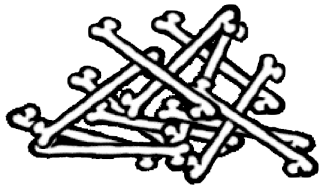
```
<body>
 <p>
  This is a <em>really, <strong>REALLY</strong></em> awesome paragraph.
  And here's a neat list:
 </p>
 <ol>
  <li>with one list item</li>
  <li>with another list item</li>
 </ol>
</body>
```

Code

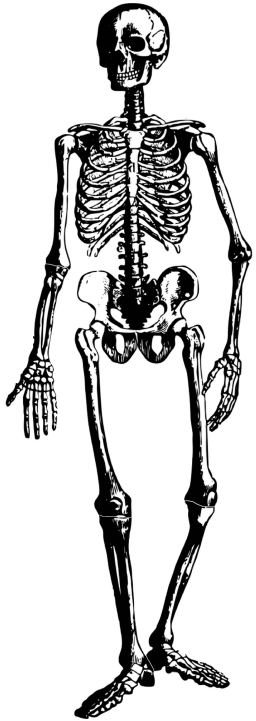
This is a *really, **REALLY*** awesome paragraph. And here's a neat list...

- with one list item
- and another list item!

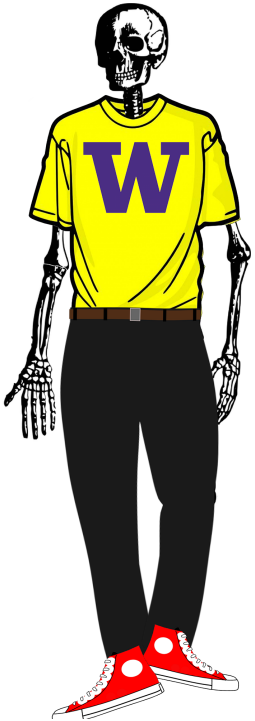
Output



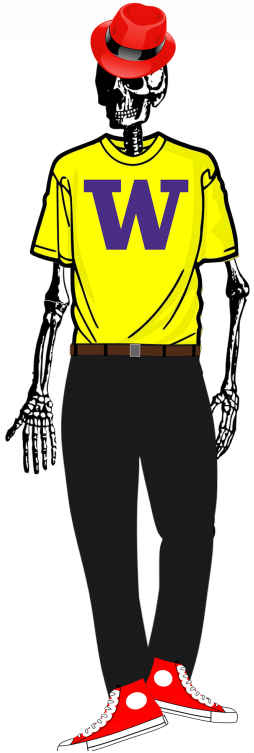
Content
Words/Images



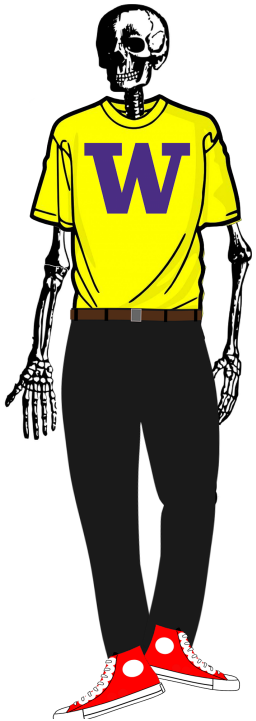
Structure
HTML



Style
CSS



Behavior
Javascript



Style

CSS

CSS

```
p {
  color: red;
  text-decoration: underline;
  font-family: sans-serif;
}
```

- A CSS file consists of one or more rulesets containing one or more rules
- Selectors designate exactly which element(s) to apply styles to
- Properties determine the styles that will be applied to the selected element(s)
- Together, a selector and the list of properties that follows is a rule.

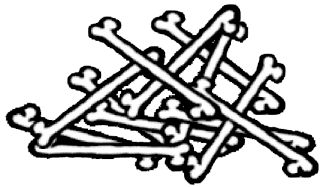
CSS

We can use select on...

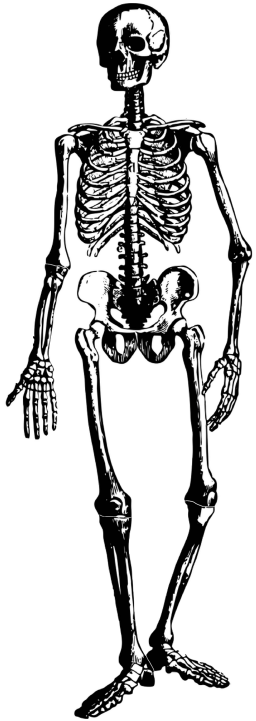
Type
- Tag type (p, div, ol, etc)

ID
- Unique identifier for an element
- Each unique id can only appear once per page
- Each element can only have one id

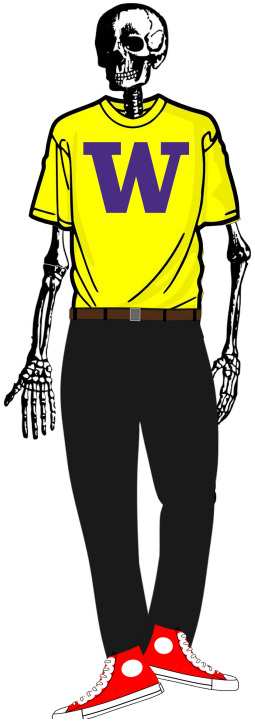
Class
- Non-unique grouping attribute to share with many elements
- Many elements (even of different types) can share the same class
- Each element can have many different classes



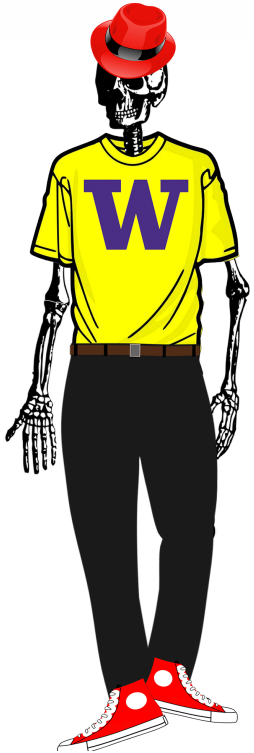
Content
Words/Images



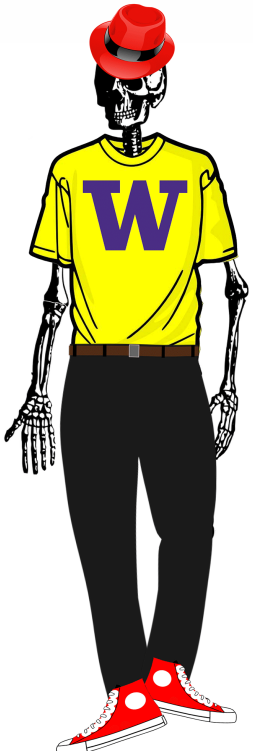
Structure
HTML



Style
CSS



Behavior
Javascript



Behavior

Javascript

JS: Adding Behavior to HTML/CSS

We can use write JavaScript functions to...

- Insert dynamic text into HTML (e.g., based on dataset)
- React to events (e.g., page load, user's mouse click)
- Get information about a user's computer
  (e.g., what browser they are using)
- Request additional data needed for the page
  (e.g. load data from URL)

JS: Defining Functions

```
// Standalone function definition
function myFunction(x) {
  return x+2;
}

// In-line arrow function definition
const myFunction = x => x + 2;

//Multi-line arrow function definition
const myFunction = (x) => {
  return x + 2;
};

// Arrow functions often passed as args
const scores = [0.9, 0.7, 0.25];
const widths = scores.map(d => d * 200);   // [180, 140, 50]
```

JS: Selectors

- Selectors let us pick out specific HTML elements to manipulate dynamically
- Essentially same syntax as CSS selectors

```
let x = 0;
const p = document.querySelector("p"); // find first p element
p.style.color = "blue";
```

JS: Append

- We can create and insert new elements HTML elements as well

```
const bar = document.createElement("div");
bar.className = "bar";
document.body.appendChild(bar);
```

JS: Event Handlers

- JS Code runs once top-to-bottom
- But often we want code to run in response to user action
- Event handles tells browser what to run on user action
    - Often bound to specific HTML elements

```
let x = 0;
const p = document.querySelector("p"); // find first p element
p.addEventListener("click", () => {x = x + 1}); //increment x every time user clicks p element
```

Raw HTML/CSS/JS sufficient, but not ideal for making interactive visualizations

SVG Preferable to HTML Elements

- Complicated logic for arranging HTML elements spatially
- Annoying to make HTML elements into arbitrary shapes
- SVG elements are another kind of element your browser can render
- Can be interspersed with HTML and selected/manipulated by CSS+JS

SVG Preferable to HTML Elements

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
  <title>SVG Example</title>
 </head>
 <body>
  <p>A bar chart drawn with SVG:</p>
  <svg width="220" height="140">
   <rect x="0" y="0"   width="180" height="25" fill="teal" />
   <rect x="0" y="30"  width="140" height="25" fill="teal" />
   <rect x="0" y="60"  width="50"  height="25" fill="teal" />
   <rect x="0" y="90"  width="70"  height="25" fill="teal" />
   <circle cx="200" cy="12" r="8" fill="orange" />
   <text x="5" y="130">SVG text</text>
  </svg>
 </body>
</html>
```

Manually specify position

Visual features as attributes

Different shapes

d3.js Data-Driven Documents



Mike Bostock, Vadim Ogievetsky, Jeffrey Heer [TVCG 2011]

Jason Davies (d3-geo, 2011–13), Philippe Rivière (2016–)

What is D3?

1. A collection of reusable visualization utilities
2. A tool for updating the browser's Document Object Model (DOM) in response to input data

Select: query DOM content

Join: bind input data to DOM elements

Update: set DOM element properties

Why D3?

Enable highly custom visualization design

Support animation and dynamic displays

Support rich and varied interactions

Interoperate via web standards (HTML, SVG, CSS)

Avoid artificial limits. If a browser can do it, D3 should be able to take advantage of it.

D3 Selections

The core abstraction in D3 is a ***selection***.

D3 Selections

The core abstraction in D3 is a ***selection***.

```
// Add and configure an SVG element (<svg width="500" height="300">)
svg = d3.append("svg")          // add new SVG to page body
    .attr("width", 500)         // set SVG width to 500px
    .attr("height", 300);       // set SVG height to 300px

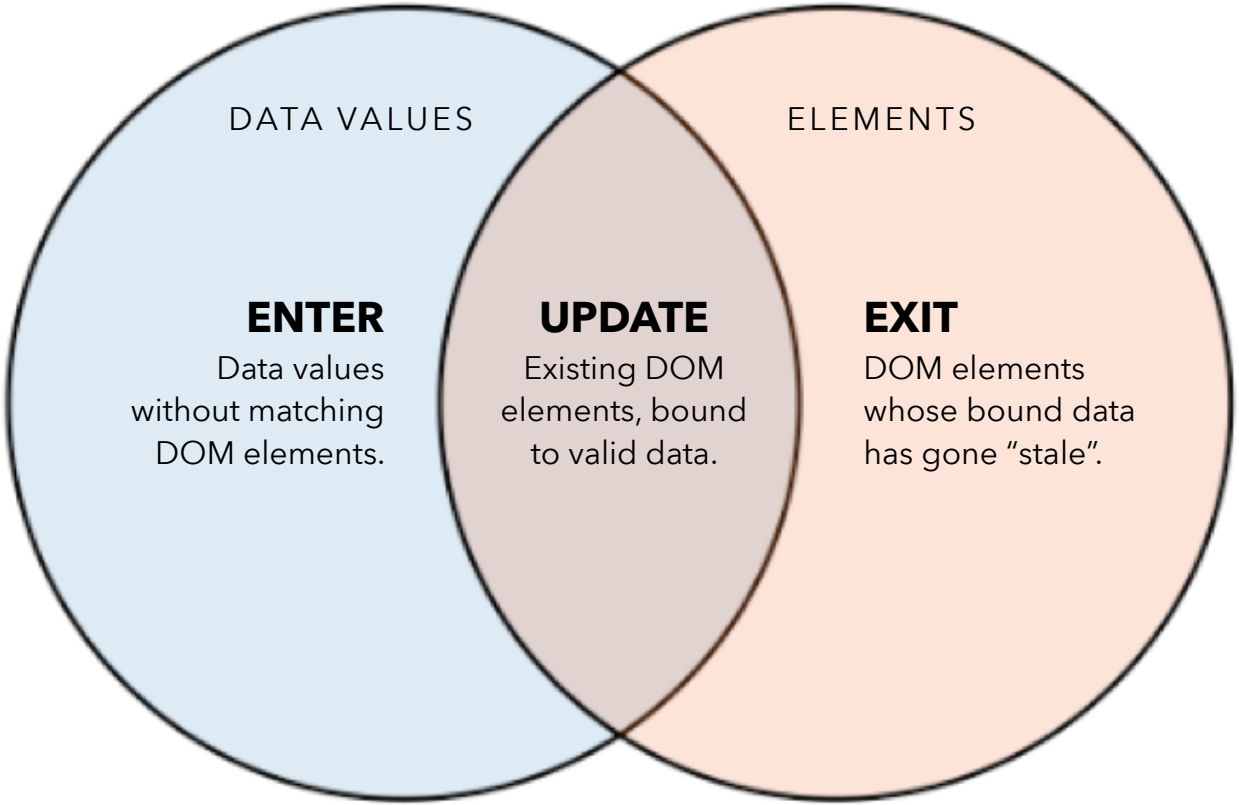
// Select & update existing rectangles contained in the SVG element
svg.selectAll("rect")           // select all SVG rectangles
    .attr("width", 100)         // set rect widths to 100px
    .style("fill", "steelblue"); // set rect fill colors
```

Data Binding

Selections can ***bind*** **data and DOM elements**.

values = [{…}, {…}, {…}, …]; // input data as JS objects

The Data Join



Data Binding

Selections can ***bind*** **data and DOM elements**.

values = [{…}, {…}, {…}, …]; // input data as JS objects

// Select SVG rectangles and bind them to data values.

bars = svg.selectAll(“rect.bars”).data(values);

// What if the DOM elements don’t exist yet? The **enter** set represents data
// values that do not yet have matching DOM elements.

bars.enter().append(“rect”).attr(“class”, “bars”);

Data

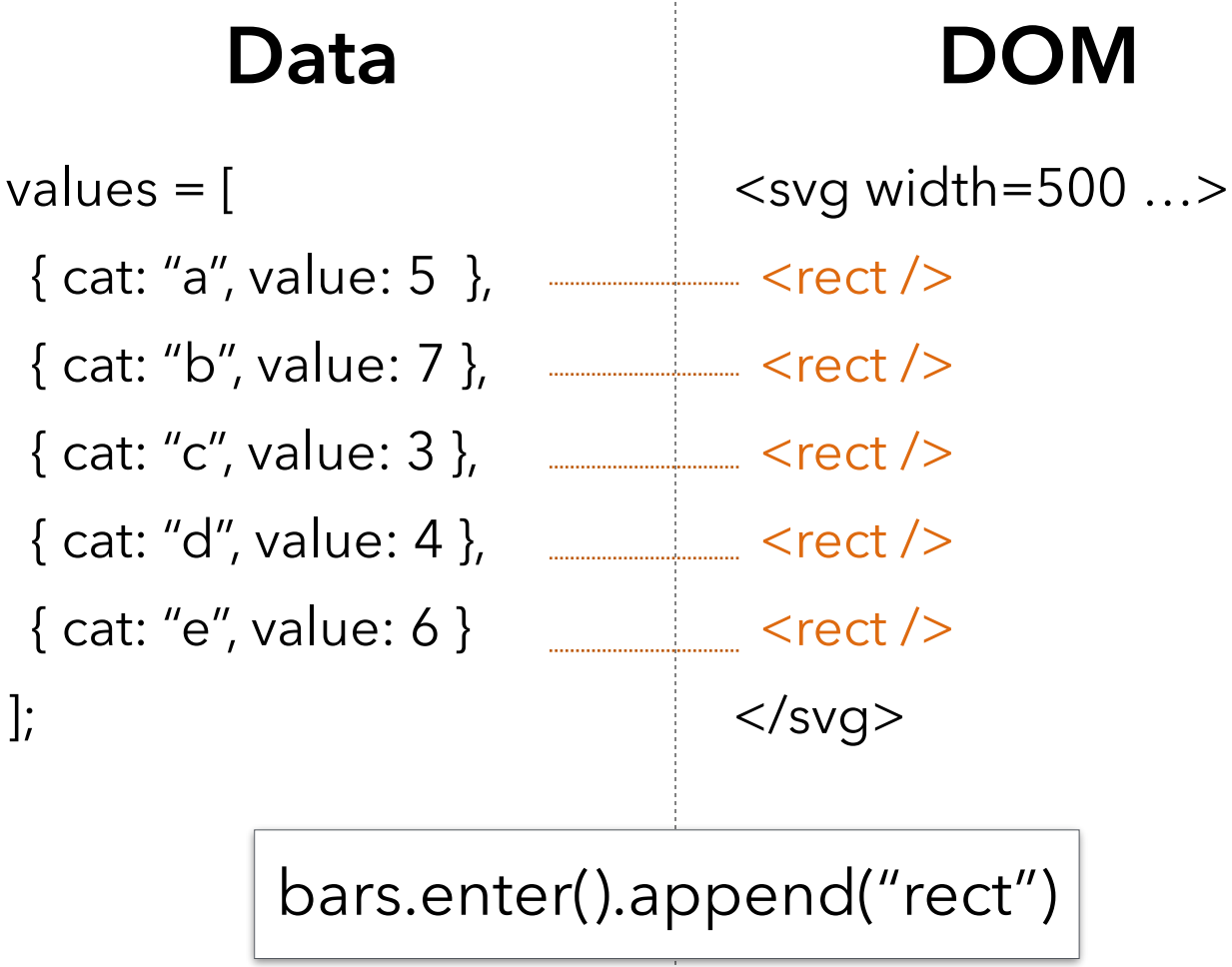
```
values = [
  { cat: “a”, value: 5  },
  { cat: “b”, value: 7 },
  { cat: “c”, value: 3 },
  { cat: “d”, value: 4 },
  { cat: “e”, value: 6 }
];
```

?
?
?
?
?

DOM

```
<svg width=500 …>
</svg>
```

```
bars = svg.selectAll(“rect”) .data(values)
```


Data
DOM
values = [
{ cat: “a”, value: 5 },
{ cat: “b”, value: 7 },
{ cat: “c”, value: 3 },
{ cat: “d”, value: 4 },
{ cat: “e”, value: 6 }
];
<svg width=500 …>
<rect />
<rect />
<rect />
<rect />
<rect />
</svg>
bars.enter().append(“rect”)

Data

```
values = [
  { cat: “a”, value: 5  },
  { cat: “b”, value: 7 },
  { cat: “c”, value: 3 },
  { cat: “d”, value: 4 },
  { cat: “e”, value: 6 }
];
```

DOM

```
<svg width=500 …>
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
</svg>
```

```
bars.enter().append(“rect”).attr(“class”, “bars”)
```

Data

```
values = [
  { cat: “a”, value: 5  },
  { cat: “b”, value: 7 },
  { cat: “c”, value: 3 },
  { cat: “d”, value: 4 },
  { cat: “e”, value: 6 }
];
```

DOM

```
<svg width=500 …>
  <rect x=“…” />
  <rect x=“…” />
  <rect x=“…” />
  <rect x=“…” />
  <rect x=“…” />
</svg>
```

```
bars.enter().append(“rect”)
    .attr(“x”, d => d.cat)
```

Data

```
values = [
  { cat: “a”, value: 5  },
  { cat: “b”, value: 7 },
  { cat: “c”, value: 3 },
  { cat: “d”, value: 4 },
  { cat: “e”, value: 6 }
];
```

DOM

```
<svg width=500 …>
  <rect height=“…” />
  <rect height=“…” />
  <rect height=“…” />
  <rect height=“…” />
  <rect height=“…” />
</svg>
```

```
bars.enter().append(“rect”)
.attr(“height”, d => d.value)
```

Data Binding

Selections can ***bind*** **data and DOM elements**.

values = [{…}, {…}, {…}, …]; // input data as JS objects

// Select SVG rectangles and bind them to data values.

bars = svg.selectAll(“rect.bars”).data(values);

// What if the DOM elements don’t exist yet? The **enter** set represents data
// values that do not yet have matching DOM elements.

bars.enter().append(“rect”).attr(“class”, “bars”);

// What if data values are removed? The **exit** set is a selection of existing
// DOM elements who no longer have matching data values.

bars.exit().remove();

Data

```
values = [
  { cat: “a”, value: 5  },
  { cat: “b”, value: 7 },
  { cat: “c”, value: 3 },
  { cat: “d”, value: 4 },
  { cat: “e”, value: 6 }
];
```

DOM

```
<svg width=500 …>
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
</svg>
```

Data

```
values = [
  { cat: “a”, value: 5  },
  { cat: “b”, value: 7 },
  { cat: “c”, value: 3 },
  { cat: “d”, value: 4 },
  { cat: “e”, value: 6 }
];
```

DOM

```
<svg width=500 …>
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
</svg>
```

```
values.filter(d => ![‘b’, ‘d’].includes(d.cat))
```

Data

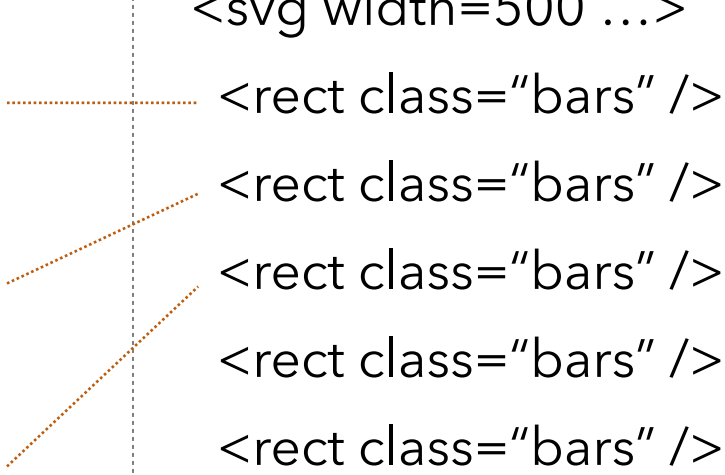
```
values = [
  { cat: “a”, value: 5  },

  { cat: “c”, value: 3 },

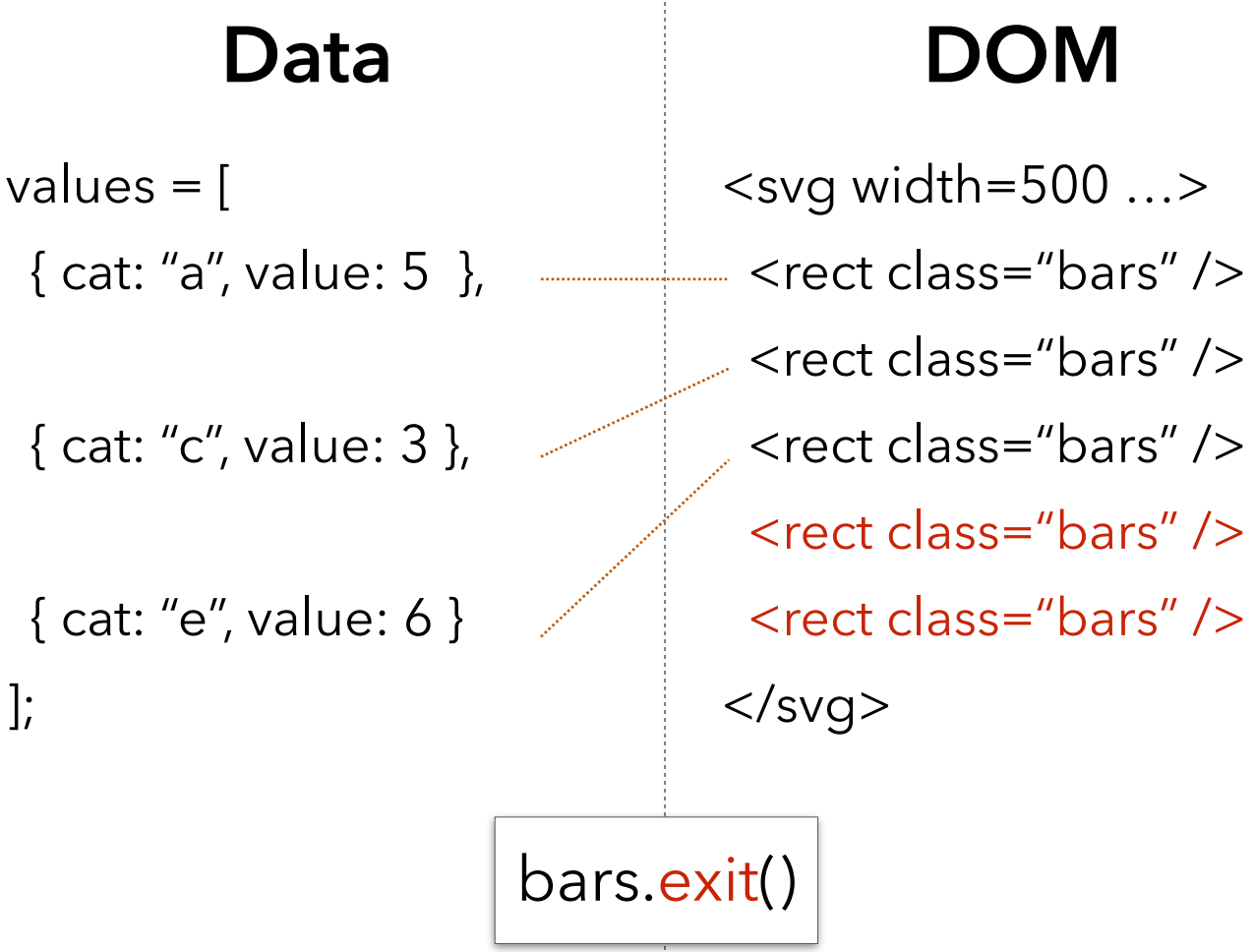
  { cat: “e”, value: 6 }
];
```

DOM

```
<svg width=500 …>
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
</svg>
```



```
bars = svg.selectAll(“rect.bars”).data(values)
```


Data
DOM
values = [
{ cat: “a”, value: 5 },
{ cat: “c”, value: 3 },
{ cat: “e”, value: 6 }
];
<svg width=500 …>
<rect class=“bars” />
<rect class=“bars” />
<rect class=“bars” />
<rect class=“bars” />
<rect class=“bars” />
</svg>
bars.exit()

Data

```
values = [
 { cat: “a”, value: 5  },

 { cat: “c”, value: 3 },

 { cat: “e”, value: 6 }
];
```

DOM

```
<svg width=500 …>
 <rect class=“bars” />
 <rect class=“bars” />
 <rect class=“bars” />
 <rect class=“bars” />
 <rect class=“bars” />
</svg>
```

bars.exit().remove()

Data

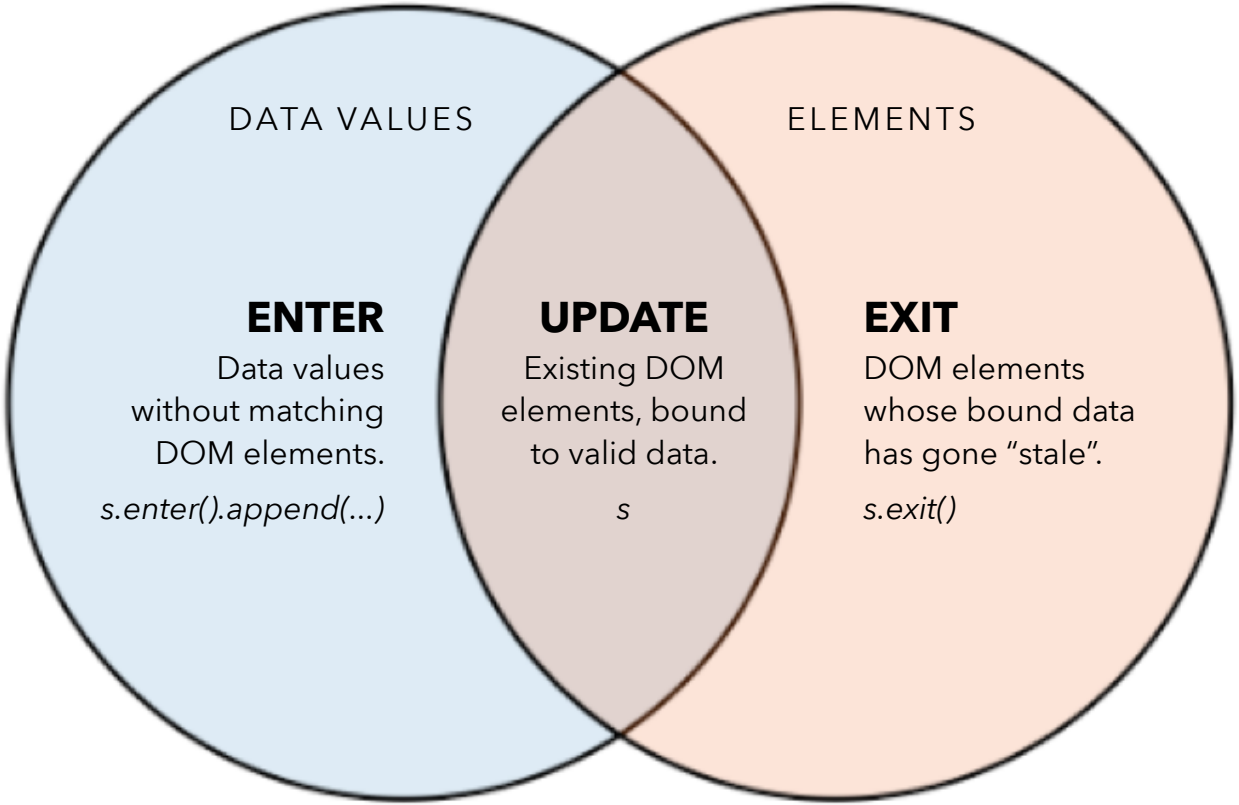
```
values = [
  { cat: “a”, value: 5  },
  { cat: “c”, value: 3 },
  { cat: “e”, value: 6 }
];
```

DOM

```
<svg width=500 …>
  <rect class=“bars” />
  <rect class=“bars” />
  <rect class=“bars” />
</svg>
```

The Data Join

var s = d3.selectAll(...).data(...)



Administrivia

A2: Peer Reviews

Due Thursday EoD

You will be assigned two peer W2 submissions to review on Canvas -- submit on Canvas and Gradescope

For each:

- Try to determine which is earnest and which is deceptive
- Share a rationale for how you made this determination
- Share feedback using the “I Like / I Wish / What If” rubric

A3: Team Registration

Submit team registration by *EOD* **Thu Oct 29**.

Submit even if working **alone**

Fill out all parts of the form

Dedicated Ed thread for finding teammates

Quiz Grades

Hopefully tonight

D3 in Observable

- More built-in functions/libraries you can call to make life easier
- Individual cell renders its "return value"
- Reactivity: Cells automatically re-run when a dependent variable is changed by another cell
- Mostly do everything in JS

Handy D3 Feature #1: Scale Functions

.style('width', function(d) {return 12*d.value + "px";}) // bad, have to hard-code scaling

Handy D3 Feature #1: Scale Functions

```
.style('width', function(d) {return 12*d.value + "px";}) // bad, have to hard-code scaling
```

```
let width = 600px;
xscale = d3.scaleLinear()
  .domain([0, d3.max(barData, d => d.value)])
  .range([0, width])
[...]
 .style('width', d => `${xscale(d.value)}px`) // good, no magic number
```

Handy D3 Feature #1: Scale Functions

```
.style('width', function(d) {return 12*d.value + "px";}) // bad, have to hard-code scaling
```

```
xscale = d3.scaleLinear()
  .domain([0, d3.max(barData, d => d.value)])
  .range([0, width]) //Observable has pre-defined width values!
[...]

.style('width', d => `${xscale(d.value)}px`) // good, no magic number
```

Scale Functions for Color

```
color = d3.scaleOrdinal(d3.schemeTableau10).domain(barData.map(d => d.key));

[...]

.style('background', d => color(d.key)) // <-- Use color scale to specify the bar color
```

Scale Functions for Bands

```
yscale = d3.scaleBand()
  .domain(barData.map(d => d.key))
  .range([0, height]) // Band scales useful for placing bars
```

Scale Functions for Bands

```
yscale = d3.scaleBand()
  .domain(barData.map(d => d.key))
  .range([height, 0]) // (0,0) top left, so often inverted!
```

Baking Margins into Scale Functions

```
margin = ({top: 10, right: 10, bottom: 20, left: 20})

xscale = d3.scaleLinear()
  .domain(0, d3.max(barData, d => d.value)])
  .range([margin.left, width - margin.right])


yscale = d3.scaleBand()
  .domain(barData.map(d => d.key))
  .range([height - margin.bottom, margin.top])
```

Handy D3 Feature #2: Axes

```
const axis = d3.axisLeft(yScale);  // axisLeft takes in a scale function

// axisLeft returns a function, which can
// can then be used to create an axis element inside another element
const g = container.append('g')
  .attr('transform', `translate(${margin.left}, 0)`);

axis(g); // creates the axis inside g
```

Handy D3 Feature #2: Axes

```
container.append('g')
  .attr('transform', `translate(${margin.left}, 0)`)
  .call(d3.axisLeft(yScale)); // alt syntax
```