

# INTRODUCTION TO D3.js



# INTRODUCTIONS

**Hello**  
my name is

**JENNIFER**

**GitHub**

jmeade11

 **Skype**

jmeade11

**Linked** 

[linkedin.com/in/jenniferannmeade](https://linkedin.com/in/jenniferannmeade)

# CREATING CAREERS DAILY

learn the skills to get the job you ❤️



**CODING**

**DATA**

**PRODUCT MANAGEMENT**

**UX & DESIGN**

**DIGITAL MARKETING**

WHAT IS  
GENERAL  
ASSEMBLY



# BY THE NUMBERS



**20**

WORLDWIDE  
CAMPUSES

**60k+**

GLOBAL  
ALMUNI

**2,500+**

HIRING  
PARTNERS

**250+**

EXPERT  
INSTRUCTORS

# FOR THIS WORKSHOP

- To work through the **Code Along** and **Lab** exercises, install the Atom text editor from **atom.io**
- Then, in the Atom menubar, go to Atom > Preferences > Install and search for and install **Emmet**
- Create an account at **codepen.io** if you don't already have one!
- Wifi: **GA Guest** Password: **yellowpencil**
- Live presentation link:  
**slides.com/jennifermeade/d3js/live**

# A QUIZ ALREADY?

## **CHECKPOINT #1:** About You

### **What is your comfort level with HTML & CSS:**

- ☐ I'm a super-HTML-coding-ninja-master (expert-level)
- ☐ It ain't all that hard, ya' know? (intermediate-level)
- ☐ Wait, what's HTML & CSS? (novice -level)

### **How about Javascript:**

- ☐ Oh yeah, Javascript is my jam.
- ☐ Meh, I dabble.
- ☐ Wait, what's Javascript...am I in the wrong room?

# WORKSHOP TOPICS

- What is D3.js?
- How Web Pages Work
- HTML & SVG Basics
- Using D3.js to Create SVGs
- Building a Data-Driven SVG
- Understanding Scales
- Animating with Transitions
- Working with Events in D3.js
- Fetching Data

# WHAT IS D3?



D3 stands for **data-driven documents**.

It's a powerful  
javascript-based  
toolset for visualizing  
data.



# D3 REQUIREMENTS

HTML 5



CSS 3



D3.js



A browser that supports inline SVG,  
modern CSS and the D3.js file.

# **HOW WEB PAGES WORK**

# WEB PAGE COMPONENTS



`<h1>Title</h1>`

**HTML**

`h1 {color: red;}`

**CSS**

`insertTitle();`

**JS**

# HYPertext MARKUP LANGUAGE



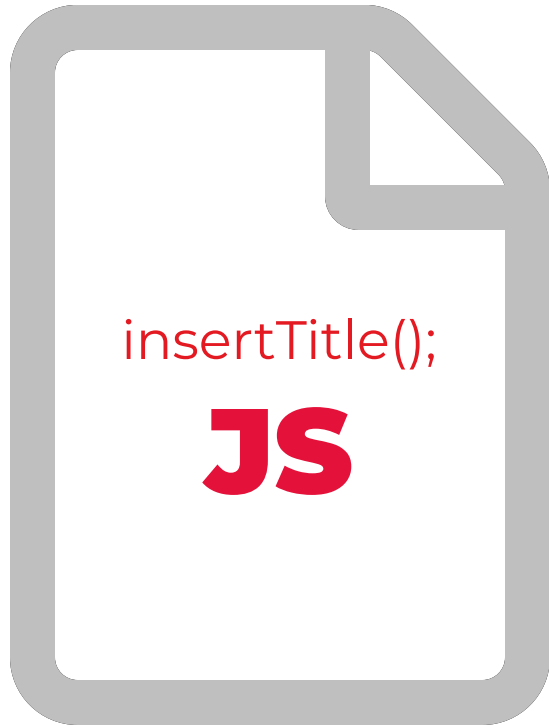
HTML provides the ***structure*** for your web pages

# CASCADING STYLE SHEETS



CSS defines how  
the elements of the  
web page ***look***

# JAVASCRIPT



Javascript allows us  
to incorporate  
***interactivity***



# SCALABLE VECTOR GRAPHICS



SVG is a ***markup language*** used to create vector graphics.

# SVG IN HTML



With HTML5, we can add SVG markup directly inside our HTML page, known as ***inline SVG***.

# INLINE SVG SUPPORT TABLE



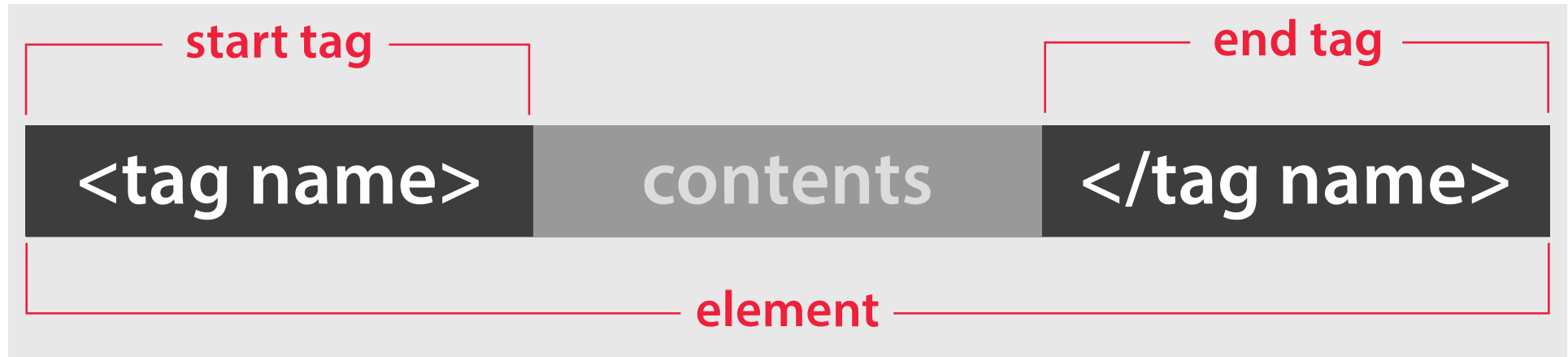
<https://caniuse.com/#search=inline%20svg>

# HTML & SVG BASICS

# OBJECTIVES

- Understand the basics of HTML and SVG markup.
- Understand the essential anatomy of a web page and create a valid HTML page.
- Create links to external files within the HTML page.
- Understand SVG primitives and their corresponding required attributes.

# **XML SYNTAX**



Some tags only have a start tag. In HTML5, we just leave off the end tag for those, **but for SVG** we end the start tag with `/>`.



# ADDING ATTRIBUTES



# GETTIN' ALL MARKUPY

```
<!-- XML Tags -->
<customer-invoice>
  <invoice-date>Wed Sep 12 2018 18:57:00</invoice-date>
  <customer-account>003492748</customer-account>
</customer-invoice>

<!-- HTML Tags -->
<p>This is a paragraph.  The p tag requires no attributes.</p>

<ul>
  <li>This is an unordered list item.</li>
</ul>

<a href="https://developer.mozilla.org">Anchor tags require an href attribute</a>

<input type="text" placeholder="input tags are forbidden to have an end tag">

<!-- SVG Tags -->
<svg viewBox="0 0 400 200">
  <circle cx="50" cy="50" r="10" />
  <rect width="20" height="20" x="100" y="100" />
</svg>
```

# WHAT'S UP DOC?

- The doctype is the first thing that goes on an html page
- The **HTML5** doctype below tells the browser to behave strictly according to the standards
- Case-insensitive but conventionally written as:

```
<!DOCTYPE html>
```



EVERY HTML PAGE STARTS WITH A DOCTYPE

# ANATOMY OF A WEBPAGE

Every HTML page has the same foundational structure

```
<!DOCTYPE html>
<html>
  <head>

    <meta charset="UTF-8">
    <title>Page Title</title>

  </head>
  <body>

    <!-- Where your content goes -->

  </body>
</html>
```



**HTML BOILERPLATE**

# LINKING FILES TO HTML

To keep our HTML manageable and maintainable, we link separate files for our CSS and Javascript.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Page Title</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>

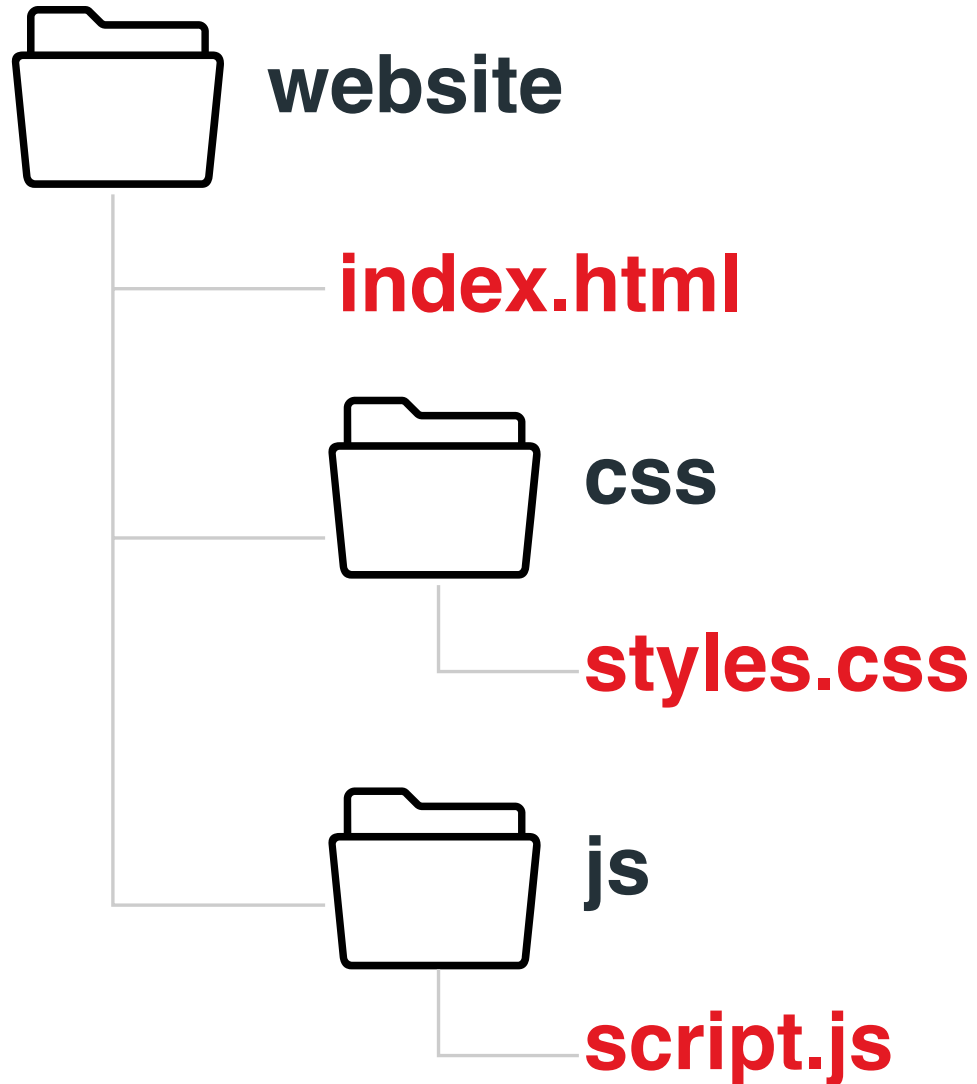
    <!-- Where your content AND Javascript goes -->
    <script src="scripts.js">
  </body>
</html>
```



Notice how the link tag goes in the head but the script tag is in the body.



# LINKING FILES RELATIVELY



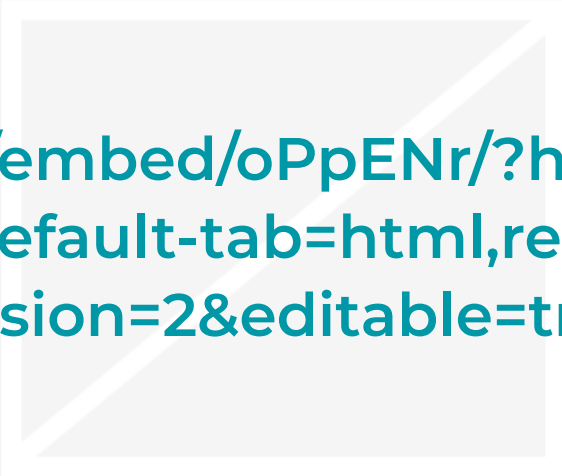
- index.html → styles.css:

```
href="css/styles.css"
```

- What would the javascript link look like?

```
src="js/scripts.js"
```

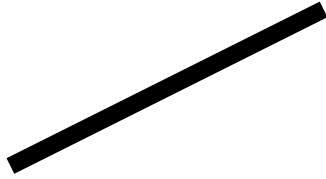
# SVG BASICS



[//codepen.io/jme11/embed/oPpENr/?height=265&theme-id=default&default-tab=html,result&embed-version=2&editable=true](https://codepen.io/jme11/embed/oPpENr/?height=265&theme-id=default&default-tab=html,result&embed-version=2&editable=true)

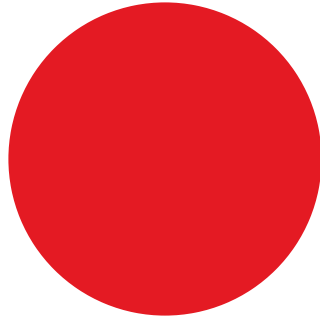
<https://codepen.io/jme11/pen/oPpENr/>

# REQUIRED ATTRIBUTES



line

- x1
- y1
- x2
- y2



circle

- cx
- cy
- r



path

- d



rect

- width
- height
- x
- y
- rx
- ry

# **USING D3 TO GENERATE SVGS**

# SELECTIONS

- Selections target the page elements we want to operate on
- D3.js uses selection methods similar to jQuery and the `.querySelector()` and `.querySelectorAll()` methods in Javascript
- Any valid CSS selector except pseudo elements can be used in the selection method

# SELECTION METHODS

```
// Select only the element with an id of chart  
d3.select('#chart');  
  
// Select only the first element with a class of triangle  
d3.select('.triangle');  
  
// Select ALL of the elements with a tag of div  
d3.selectAll('div');
```



# APPENDING ELEMENTS

- The `.append()` method is used to append elements to the selection.

```
// Select the element with an id of chart
// Append an SVG element

d3.select( '#chart' )
    .append( 'svg' );
```



It's conventional to break chained methods on to separate lines and indent them to make them more readable.

# STORING THE SELECTION

- Storing the selection in a variable makes it easier and more efficient to access this element later in our script

```
// Select the element with an id of chart  
  
let chart = d3.select('#chart')  
                .append('svg');
```

# ADDING ATTRIBUTES

- The `.attr()` method allows us to add attributes.
- Takes two parameters: the attribute name and the value to be applied.

```
let svg = d3.select('#chart')  
    .append('svg');  
  
// Add a viewBox attribute to the svg  
svg.attr('viewBox', '0 0 100 100');
```

# ADDING CLASSES

- The `.classed()` method adds or removes classes.
- Takes two parameters: the class name and `true` to add or `false` to remove.

```
// Add a highlight class to all divs
// Remove the active class from any divs

d3.selectAll('div')
  .classed('highlight', true)
  .classed('active', false);
```

# ADDING INLINE STYLES

- The .style() method adds an inline style.
- Takes two parameters: the style property and its corresponding value.

```
// Set the opacity of all divs to .9  
  
d3.selectAll('div')  
  .style('opacity', .9);
```



**MAKING SVGS WITH D3**

**BINDING DATA**

# BINDING DATA

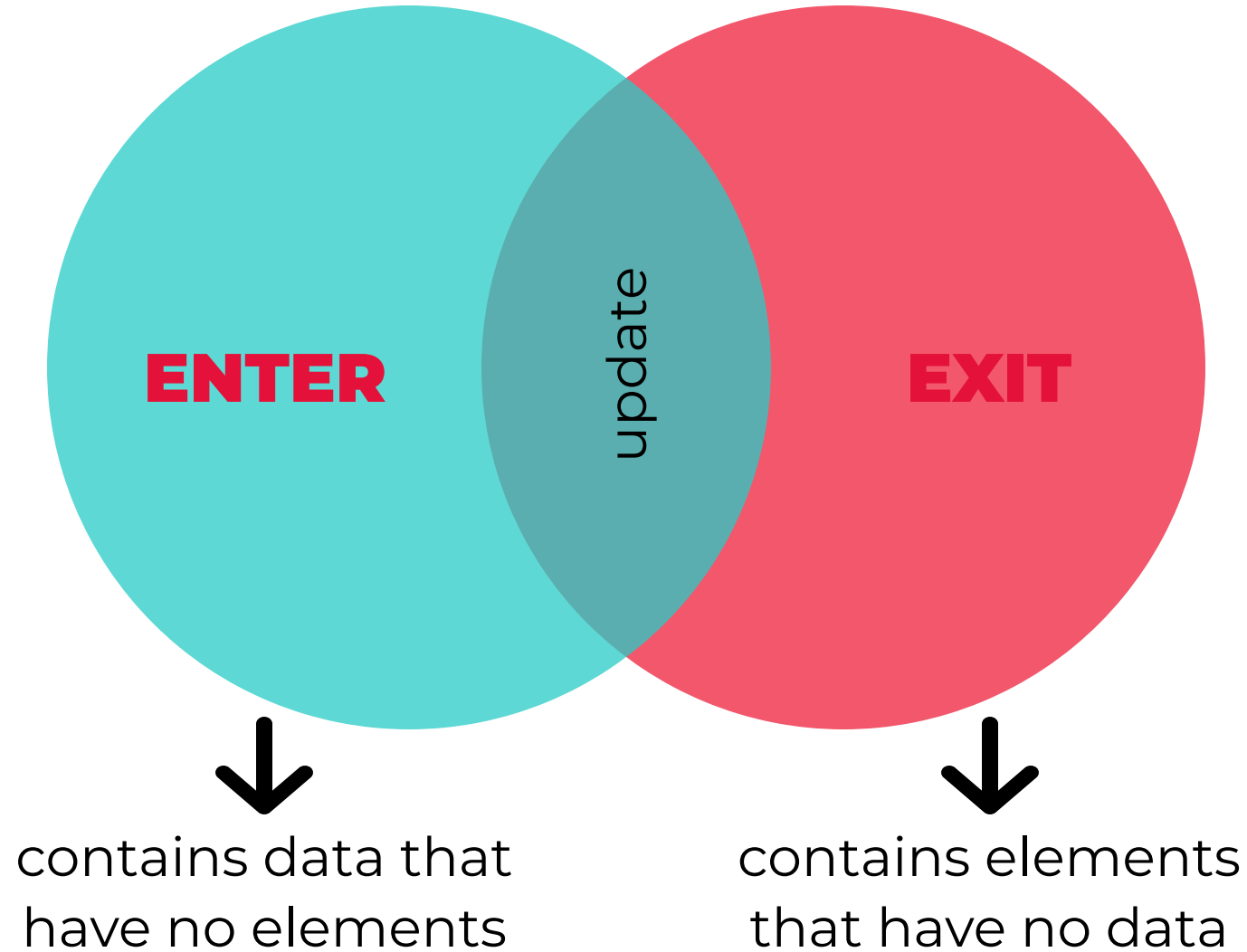
- The `.data()` method is how we bind our data to a selection.

```
// Select all of the rect elements and  
// bind the data array to them
```

```
d3.selectAll('rect')  
  .data([30,20,10,15]);
```



# ENTER, UPDATE, EXIT



# WHHHHAAAAT?

When you attach data with the `.data()` method in D3, each data element in your data array is attached to an element within the **current selection**.

Three things can happen...

# MORE DATA THAN ELEMENTS

The first thing that can happen is that there are more data elements than elements in the selection.

## *THE RESULT:*

The extra data elements go into the **enter selection**, which can be accessed with the `.enter()` method.

# MORE ELEMENTS THAN DATA

The second thing that can happen is that there are more elements in the selection than data elements in your data array.

## *THE RESULT:*

The extra elements from the selection that have no data attached go into the **exit selection**, which can be accessed with the `.exit()` method.

# AT LEAST AS MANY ELEMENTS AS DATA POINTS

The **intersection** of the data and the selection elements is called the update selection (i.e, every data element is associated with a selection element).

## **THE RESULT:**

The data is attached to the elements and any updates are applied. If there are leftover elements in the selection they go into the exit selection.

# GENERAL UPDATE PATTERN

```
// Run this function whenever the data changes.

function update(data) {

  // Join new data with old elements, if any.
  const selection = svg.selectAll(/* your selector here */)
    .data(data);

  // Update old elements as needed.
  selection
    .text('Updated!') // run whatever updates methods

  // Create new elements as needed in the enter selection.
  selection
    .enter().append(/* your selector here */)
    .text('Added!') // run whatever updates methods

  // Merge the entered elements with the update selection.
  // Apply operations to both.
  .merge(selection) // merge takes the selection as its argument
    .text('Merged and added!') // run whatever updates methods

  // Update the elements in the exit selection
  // Most of the time we'll run the remove method here
  div.exit()
    .text('Ready for removal!') // run whatever updates methods
    .remove();
}
```



**ENTER-UPDATE-EXIT**

# THE CALLBACK

```
// d is the data on the currently selected element
// i is the index of the currently selected element

d3.method(function(d, i){
  return d;
});
```

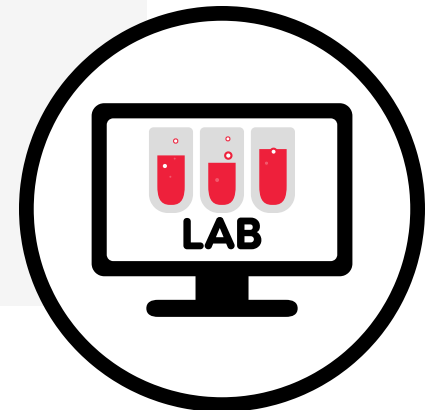
- This function is called an anonymous function in Javascript.
- In D3, it always gets 2 values: the **data** in the selection and the **index**.
- The value in the return statement is the value that is used in the method for the current element.



# DATA-DRIVEN ELEMENTS

[//codepen.io/jme11/embed/wRMqPm/?height=265&theme-id=default&default-tab=html,result&embed-version=2&editable=true](https://codepen.io/jme11/embed/wRMqPm/?height=265&theme-id=default&default-tab=html,result&embed-version=2&editable=true)

<https://codepen.io/jme11/pen/LBpQdG>



## Creating Shapes in D3:

# LAB TASKS

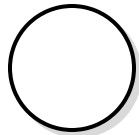
1. Add rect elements to the svg for each data point.
2. Remember to use the enter() selection followed by append() since there are no rect elements in the svg
3. Also, make sure that you add the following required attributes for rect elements: x, y, height, width
4. **Bonus:** Can you think of how to evenly space them apart?  
Hint: the iterator is a good multiplier 😊

# LET'S SEE HOW YOU'RE DOING

**CHECKPOINT #2:** Just Checking In

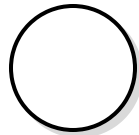
**On a scale of 1 to 4 — where 1 is completely lost, and 4 is feeling pretty comfortable — how are you feeling about your grasp of the content so far?**

**1**



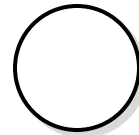
Wait, what just happened?

**2**



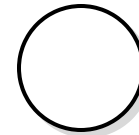
Dazed and confused

**3**



Patiently waiting for Nirvana

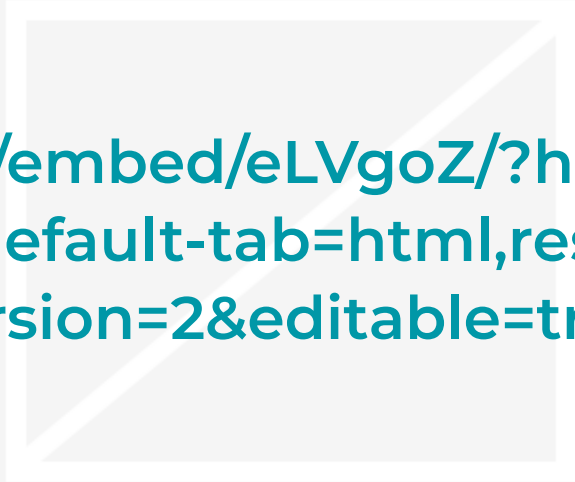
**4**



So far, so good!

# **BUILDING A BARCHART**

# SOLUTION



[//codepen.io/jme11/embed/eLVgoZ/?height=265&theme-id=default&default-tab=html,result&embed-version=2&editable=true](https://codepen.io/jme11/embed/eLVgoZ/?height=265&theme-id=default&default-tab=html,result&embed-version=2&editable=true)

<https://codepen.io/jme11/pen/eLVgoZ>

# FLIPPING THE BARS

In SVG, **y values increase from top to bottom.**  
This is inconsistent with how we tend to visualize data in a Cartesian coordinate system.

```
/* To flip our bars we will subtract their y value  
   from the height of the SVG  
*/  
    .attr('y', function(d,i){  
        return svgHeight - y;  
    })
```

# LINEAR SCALES

The linear scales in D3 **map data to a specific range of values**. Linear scales are also called continuous scales because it maps a serial set of input values to output values.

- `d3.scaleLinear()`
- `domain([min, max])` **input**
- `range([newmin, newmax])` **output**

# MAKING THE BARS FIT VERTICALLY

Linear scale maps the data to the height of the SVG.

```
const yScale = d3.scaleLinear()  
    .domain([0, d3.max(data)])  
    .range([0, height]);  
  
// Use this for our height attribute  
// and y attribute  
  
... .attr('height', function(d) {  
    return yScale(d)  
})  
    .attr('y', function(d){  
        return height - yScale(d);  
    });
```



# ORDINAL SCALES

Sometimes the **order** of the data matters most, such as when we're plotting data that corresponds to dates. In these cases, we'll use an ordinal scale.

- `d3.scaleOrdinal()`
- `scaleBand()` **helpful for barcharts**
- `padding()`, `paddingInner()`, `paddingOuter()`

# FIT THE DATA HORIZONTALLY

```
const xScale = d3.scaleBand()  
    .domain(bardata)  
    .paddingInner(.3) // also padding()  
    .range([0, width]);  
  
// Use this for our width attribute  
// and x attribute  
  
... .attr('width', function(d,i){  
    return xScale.bandwidth(); //bandwidth built in method  
})  
    .attr('x', function(d,i){  
    return xScale(d);  
})
```

# ORDINAL SCALE FOR COLORS

```
const colors = d3.scaleOrdinal()  
    .range([  
        'yellowgreen',  
        'darkorange',  
        'deepskyblue',  
        'darkviolet',  
        'deeppink'  
    ]);  
  
// or use built in ones with  
  
// const colors = d3.scaleOrdinal(d3.schemeAccent);  
  
// then use them to style our bars  
  
... .style('fill', function(d,i){  
    return colors(i);  
});
```

# GROUPING & TRANSFORMS

The `<g>` element allows us to **group** multiple elements together so that we can apply a single transform to them.

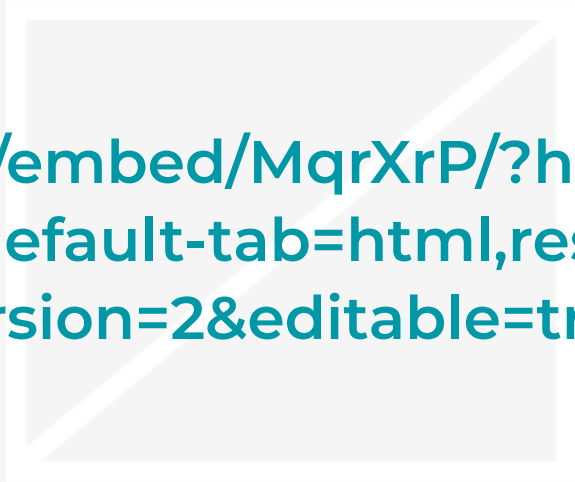
```
/* Our g elements can wrap other elements in SVG */  
  
<g transform="translate(x, y)">  
  <circle ... >  
  <circle ... >  
  <circle ... >  
</g>
```

# CREATING MARGINS THE D3 WAY

The `<g>` element allows us to **group** multiple elements together so that we can apply a single transform to them.

```
const margin = {top: 15, right: 15, bottom: 15, left: 30},  
  w = 400,  
  h = 300,  
  height = h - margin.top - margin.bottom,  
  width = w - margin.left - margin.right;
```

# UNDERSTANDING VIEWBOX



[//codepen.io/jme11/embed/MqrXrP/?height=265&theme-id=default&default-tab=html,result&embed-version=2&editable=true](https://codepen.io/jme11/embed/MqrXrP/?height=265&theme-id=default&default-tab=html,result&embed-version=2&editable=true)

<https://codepen.io/jme11/pen/MqrXrP>

# ADDING A Y-AXIS

```
// This scale is identical to our yScale that fits
// the data to the height of the svg EXCEPT that the
// range is reversed so the 0 is at the bottom ;-)
```

```
yAxisValues = d3.scaleLinear()
                  .domain([0, d3.max(bardata)])
                  .range([height,0]),
```

```
// The ticks method value is just a suggestion. D3
// figures out what makes sense based on the range
// of values supplied.
```

```
yAxisTicks = d3.axisLeft(yAxisValues)
                .ticks(3);
```

```
// Append a group so the axis can be transformed into
// position easily and call the axis method stored in
// our yAxisTicks variable.
```

```
svg.append('g')
    .attr('transform', 'translate('+margin.left+', '+margin.top+')')
    .call(yAxisTicks);
```

**TRANSITIONS**



# TRANSITIONS IN D3

- `transition()` **set up**
- `duration(ms)` **controls speed**
- `delay(ms)` **when to start**
- `ease(predefined)`



Using ease requires an additional Javascript file (see: <https://github.com/d3/d3-ease>)

# **WORKING WITH EVENTS**

# EVENTS IN D3

- `on(event, function(d){ return ;})`



Standard Javascript events (e.g., click or mouseover) and some additional D3 specific events can be used in the on method.

**FETCHING DATA**

# FETCHING DATA

```
d3.json(url)
  .then(function(json) {
    /* Do Something with the Data */
  });
```



This uses the Javascript Fetch API. That means, we can't run this as a file locally (it requires an actual server).

**YOU DID IT!** 🙌

**GO BUILD  
AWESOME THINGS!**