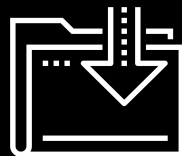




# Bar and Line Charts with

Data Boot Camp  
Lesson 16.2



# Class Objectives

---

By the end of today's class you will be able to:



Deepen their knowledge of the D3 library.



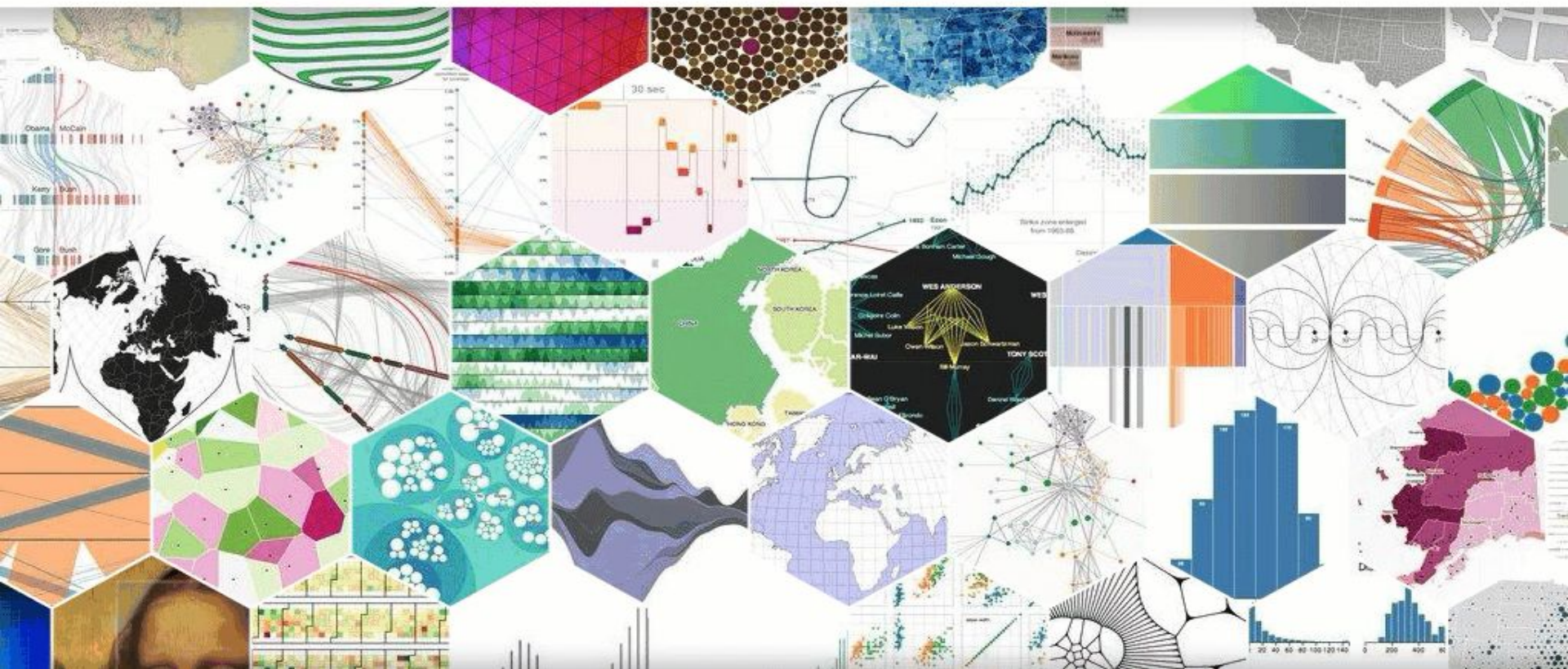
Create different types of charts and graph using D3.

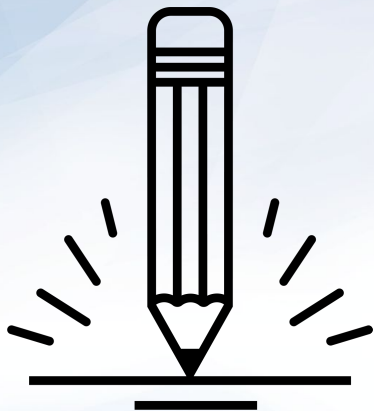


Cover creating scales and axes in D3.

# Data-Driven Documents

**WELCOME**  
Back!





## Activity: Review

In this activity, you and your partner will discuss and answer the questions sent to you.

**Suggested Time:**  
10 Minutes



# Activity: Review

---

## Instructions:

- Open [questions.js](#) (check your slack) and answer the following questions after discussing with a partner.
    1. What does **SVG** stand for? How are they used with **D3**?
    2. What's data binding?
    3. Given the following and an HTML page with no elements currently in the body, use the `enter()` pattern to render three `<li>` elements to the page with text matching the integers in the array.

```
var arr = [1, 2, 3];  
var ul = d3.select("body").append("ul");
```
    4. Imagine three `<li>` elements already exist on the page. Create code to update the text of those elements while also adding three new elements to match the array below. Leave the number 3 code uncommented as it is needed for number 4 to work properly.

```
var arr = [1, 1, 2, 3, 5, 8];  
var ul = d3.select("ul");
```
  - **Bonus**
    - Refactor your solution to number 4 above using the ES6 syntax for arrow functions. Then, modify the code to set the text of each element to “<index in the array>: <item from the array>”
    - Be sure to comment your number 4 code (not the `arr` or `ul` variables) before running the code.
-



**Time's Up!** Let's Review.



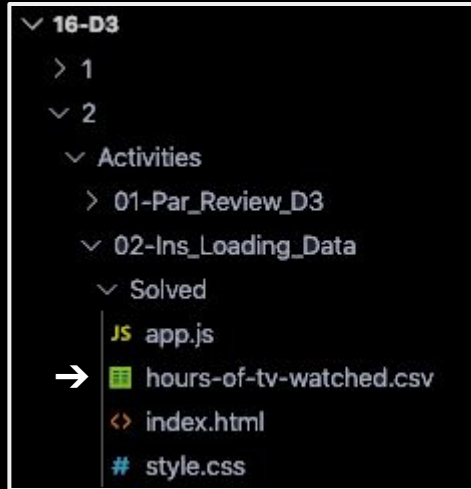
# Instructor Demonstration

## Loading Data



# Loading Data

- File Structure:



→ We need to parse the CSV file using `d3.csv()`

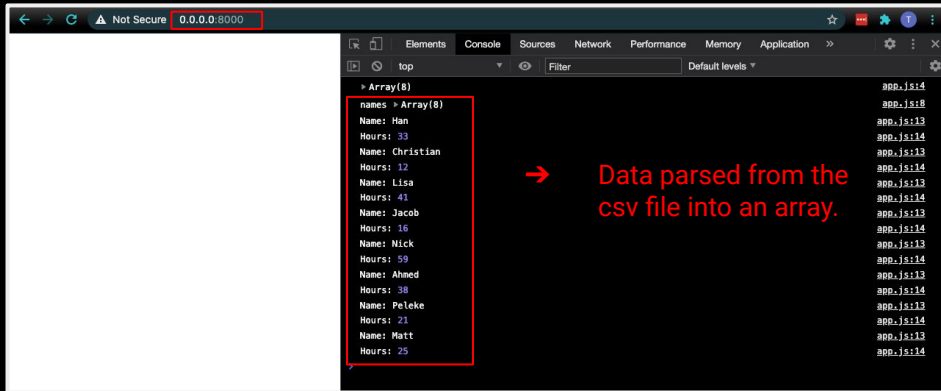
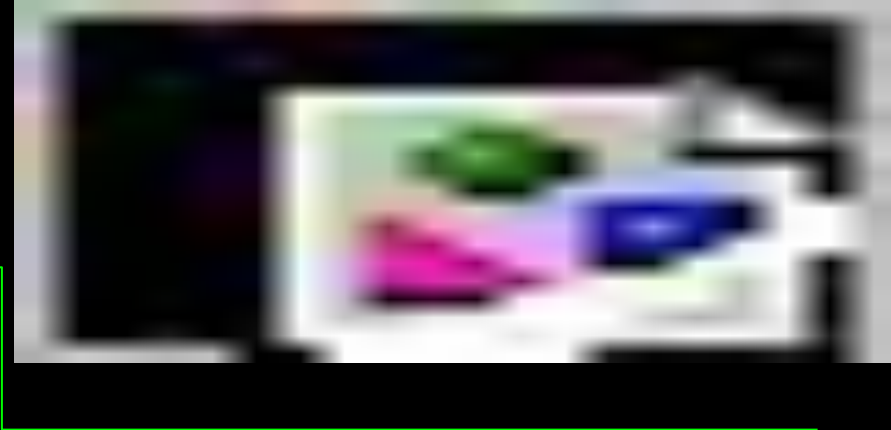




# Loading Data

- Open Terminal:
  - Activate the appropriate virtual environment.
  - Go to the appropriate directory.
  - Run: `python -m http.server`.

- Open browser



→ Visit `http://0.0.0.0:8000/`



## Activity: Bar Chart from CSV

In this activity, you and your partner will be challenged to interpret and complete the starter code with the goal of creating a bar chart to reflect CSV data.

**Suggested Time:**  
15 Minutes



# Activity: Bar Chart from CSV

---

## Instructions:

- First, take a few minutes to review the code and discuss what you need to do with a partner.
- Organize the provided files in a folder, open the folder, and start the server. All code which you will write will be inside of the callback method passed into `d3.csv`.
- Using `chartGroup`, select all of the elements inside with a class of `bar` and bind the `tvData` data to the selection.
- Still chaining to the code written in the previous step, run the `.enter()` method and append a `rect` with a class of `bar` for each element in the array.
- Set the `width` property of each rectangle to `barWidth`.
- Set the `height` property of each rectangle using a callback function, which is passed the data bound to the rectangle. Scale the height of the rectangle by the value of `yscale`.
- Set the `x` attribute of each rectangle using a callback function which is passed the data bound to the rectangle. Space the rectangles by the value of `'barSpacing'`.
- Set the `y` attribute of each rectangle using a callback function which is passed the data bound to the rectangle. Remember to invert your `y` values using the `chartHeight` so that the bar chart is right-side-up.

- **Hints:**



- For assistance understanding data joins with D3, see [the article](#) slacked to you written by Mike Bostock, the creator of **D3**.
- For more information on transformations with SVG files, see the [Tutorial on Basic Transformations](#) slacked to you.



**Time's Up!** Let's Review.



## Everyone Do: Scales

In this activity, we all will work on **D3** Scales.

**Suggested Time:**  
15 Minutes



# Everyone Do: Scales

- Finding the min and max in a JavaScript Array using **D3**.

- `min()` Returns the lowest element value in an array.
- `max()` Returns the highest element value in an array.
- `extent()` Returns both, the highest and the lowest elements in an array.

```
> var dataArr = [10, 20, 2000];
```

```
→ console.log("min value ", d3.min(dataArr));
```

```
→ console.log("max value ", d3.max(dataArr));
```

```
→ console.log("min and max values ", d3.extent(dataArr));
```

```
min value 10 ←
```

VM98:3

```
max value 2000 ←
```

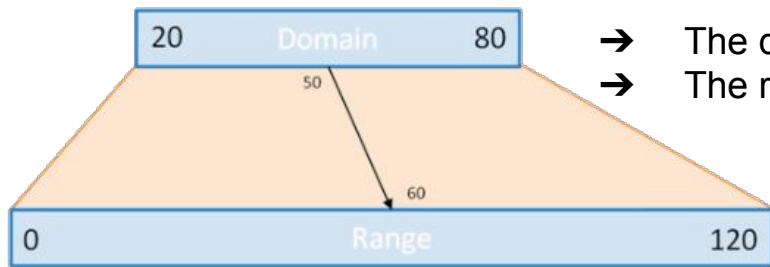
VM98:4

```
min and max values ▶ (2) [10, 2000] ←
```

VM98:5

# Everyone Do: Scales

- `scaleLinear()`
  - In order to give an accurate visual representation, the data represented in a graph is scaled.
  - D3 provides an easy way to convert those values to suit the visualization.



- The domain is the bounds of the data.
- The range is the bounds of the chart.

```
> var yScale = d3.scaleLinear()
  .domain([0, 100])
  .range([0, 1000]);

console.log(`50 returns ${yScale(50)}`);
console.log(`75 returns ${yScale(75)}`);
console.log(`100 returns ${yScale(100)}`);

50 returns 500
75 returns 750
100 returns 1000

< undefined
```

VM13:5

VM13:6

VM13:7



# Everyone Do: Scales

- `d3.max()`
  - The parameters passed in this part of the code is slight different.
  - We want the highest element of testScores for the max of the domain.
  - We use a variable for the upper boundary in range.

```
> var testScores = [50, 90, 95, 75, 85];
```

```
var svgHeight = 1000;
```

```
var yScale = d3.scaleLinear()  
  .domain([0, d3.max(testScores)])  
  .range([0, svgHeight]);
```

```
console.log(`50 returns ${yScale(50)}`);  
console.log(`75 returns ${yScale(75)}`);  
console.log(`95 returns ${yScale(95)}`);
```

```
50 returns 526.3157894736842
```

VM40:10

```
75 returns 789.4736842105264
```

VM40:11

```
95 returns 1000
```

VM40:12

```
← undefined
```

# Everyone Do: Scales

---

- `d3.extent()`
  - In this instance we are using a function to pass the highest and the lowest element of the array.

```
> var testScores = [50, 90, 95, 75, 85];
```

```
var svgHeight = 1000;
```

```
var yScale = d3.scaleLinear()  
  .domain(d3.extent(testScores))  
  .range([0, svgHeight]);
```

```
console.log(`50 returns ${yScale(50)}`);
```

```
console.log(`75 returns ${yScale(75)}`);
```

```
console.log(`95 returns ${yScale(95)}`);
```

```
50 returns 0
```

VM65:10

```
75 returns 555.5555555555555
```

VM65:11

```
95 returns 1000
```

VM65:12

```
< undefined
```

# Everyone Do: Scales

- `scaleBand()`
  - This function is used to build a new band scale with the domain as an array of categorical data values and the range as the min and max extents of it. It will split the range into  $x$  bands where  $x$  is the number of values in the domain array.

```
> svgHeight = 600;
   svgWidth = 1000;

testScores = [90, 85, 75, 90];
var students = ["Han", "Sarah", "Matt", "Ruchi"];

var xScale = d3.scaleBand()
  .domain(students)
  .range([0, svgWidth]);

→ console.log(`Han's x-coordinate: ${xScale("Han")}`);
   console.log(`Sarah's x-coordinate: ${xScale(students[1])}`);
   console.log(`Matt's x-coordinate: ${xScale("Matt")}`);
   console.log(`Ruchi's x-coordinate: ${xScale(students[3])}`);

→ console.log(`Each band is ${xScale.bandwidth()} pixels wide.`);

// The y values are scaled separately.

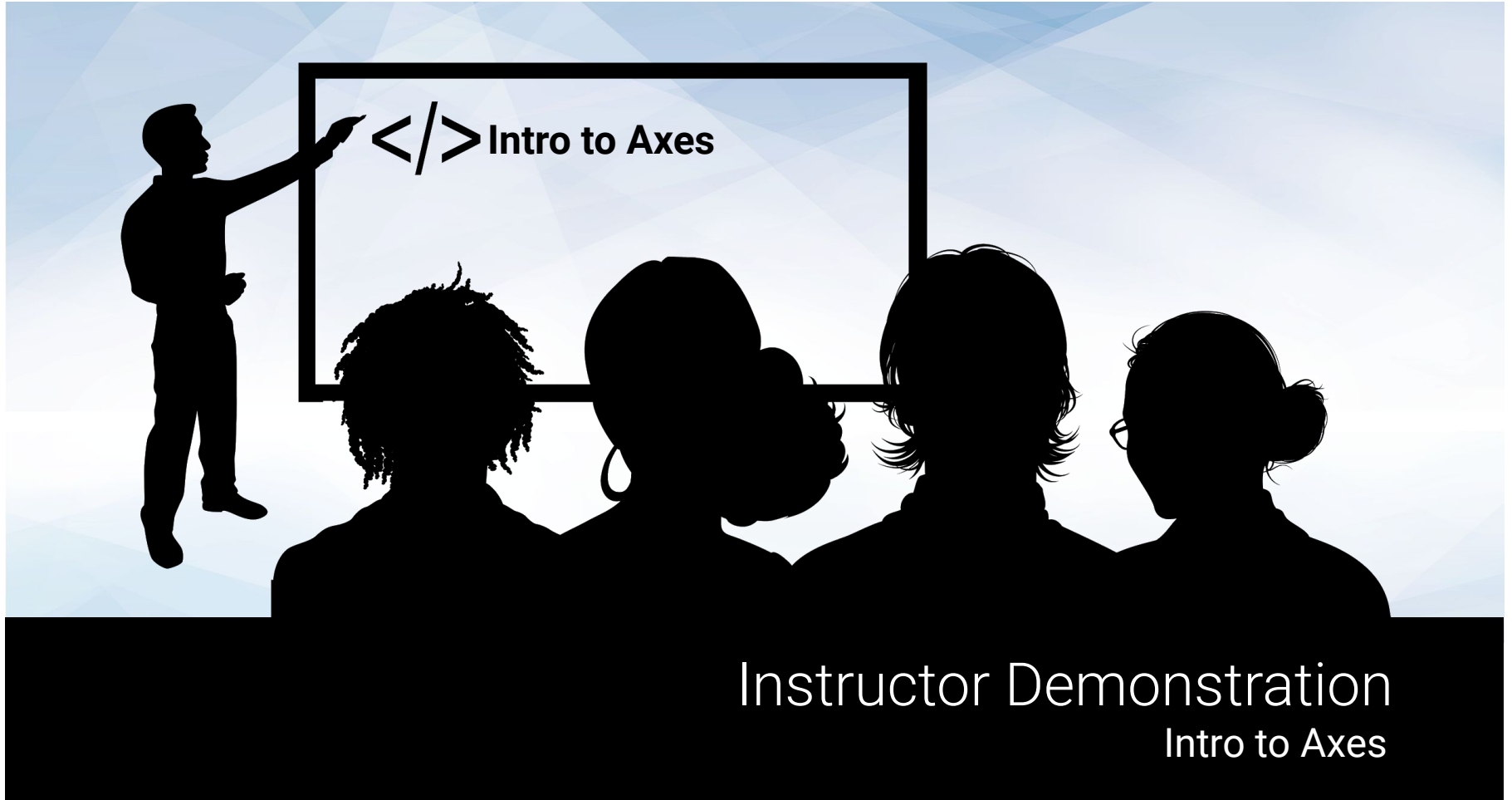
var yScale = d3.scaleLinear()
  .domain([0, 100])
  .range([0, svgHeight]);

console.log(`The height of Han's bar: ${yScale(testScores[0])}`); ←
```

Han's x-coordinate: 0  
Sarah's x-coordinate: 250  
Matt's x-coordinate: 500  
Ruchi's x-coordinate: 750

→ Each band is 250 pixels wide.

The height of Han's bar: 540 ←

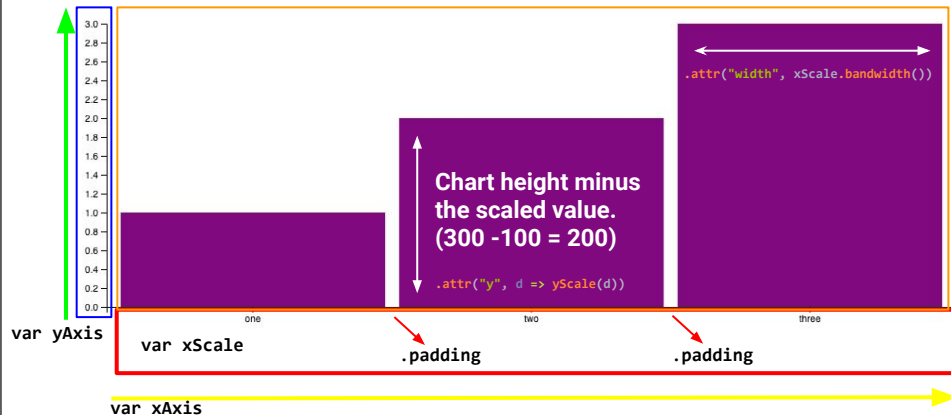


# Instructor Demonstration

## Intro to Axes

# Intro to Axes

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <div id="svg-area">
      <svg height="400" width="1000">
        <g transform="translate(50, 50)">
          <g transform="translate(0, 300)" fill="none" font-size="10" font-family="sans-serif" text-anchor="middle">
            <path class="domain" stroke="currentColor" d="M0,5,6V0.5H900.5V6"></path>
            <g class="tick" opacity="1" transform="translate(154.9180327868852,0)">
              <line stroke="currentColor" y="6"></line>
              <text fill="currentColor" y="9" dy="0.71em">one</text>
            </g>
            <g class="tick" opacity="1" transform="translate(449.99999999999994,0)"></g> → One "g" tag per axis
            <g class="tick" opacity="1" transform="translate(745.0819672131147,0)"></g>
          </g>
          <g fill="none" font-size="10" font-family="sans-serif" text-anchor="end"></g>
          <rect class="bar" x="14.75480936065559" y="100.00000000000003" width="280.327868852459" height="99.99999999999997"></rect>
          <rect class="bar" x="309.83606557377844" y="100.00000000000001" width="280.327868852459" height="200"></rect>
          <rect class="bar" x="604.9180327868852" y="0" width="280.327868852459" height="300"></rect>
        </g>
      </svg>
    </div>
    <script src="https://d3js.org/d3.v5.min.js"></script>
    <script type="text/javascript" src="app.js"></script>
  </body>
</html>
```



One rect per element in the array. Attributes calculated with the scale functions

```
var dataArray = [1, 2, 3];
var dataCategories = ["one", "two", "three"];

var svgHeight = 400;
var svgWidth = 1000;

var margin = {
  top: 50,
  right: 50,
  bottom: 50,
  left: 50
};

var chartHeight = svgHeight - margin.top - margin.bottom;
var chartWidth = svgWidth - margin.left - margin.right;

var svg = d3.select("#svg-area").append("svg")
  .attr("height", svgHeight)
  .attr("width", svgWidth);

var chartGroup = svg.append("g")
  .attr("transform", `translate(${margin.left}, ${margin.top})`);

var yScale = d3.scaleLinear()
  .domain([0, d3.max(dataArray)])
  .range([chartHeight, 0]);

var xScale = d3.scaleBand()
  .domain(dataCategories)
  .range([0, chartWidth])
  .padding(0.05);

var yAxis = d3.axisLeft(yScale);
var xAxis = d3.axisBottom(xScale);

chartGroup.append("g")
  .attr("transform", `translate(0, ${chartHeight})`)
  .call(xAxis);

chartGroup.append("g")
  .call(yAxis);

chartGroup.selectAll(".bar")
  .data(dataArray)
  .enter()
  .append("rect")
  .classed("bar", true)
  .attr("x", (d, i) => xScale(dataCategories[i]))
  .attr("y", d => yScale(d))
  .attr("width", xScale.bandwidth())
  .attr("height", d => chartHeight - yScale(d));
```





Countdown timer

**15:00**

(with alarm)



## Activity: Complete Bar Chart

In this activity, you will create a bar chart from CSV data complete with axes. It will require a combination of skills from today and mostly build on the solution from the previous activity.

**Suggested Time:**  
15 Minutes





# Activity: Complete Bar Chart

---

## Instructions:

- From the command line, start a Python server to serve the csv and web files: `python -m http.server`.
- Load the data from the `hours-of-tv-watched.csv` using `d3.csv`.

Within the `d3.csv` method callback, do the following:

- Cast the hours as numbers.
- Create scale functions for your x and y values.
- Create functions to generate your x and y axes.
- Render your axes to the page.
- Render rectangles to the page to create your bar chart and give them hover effects.

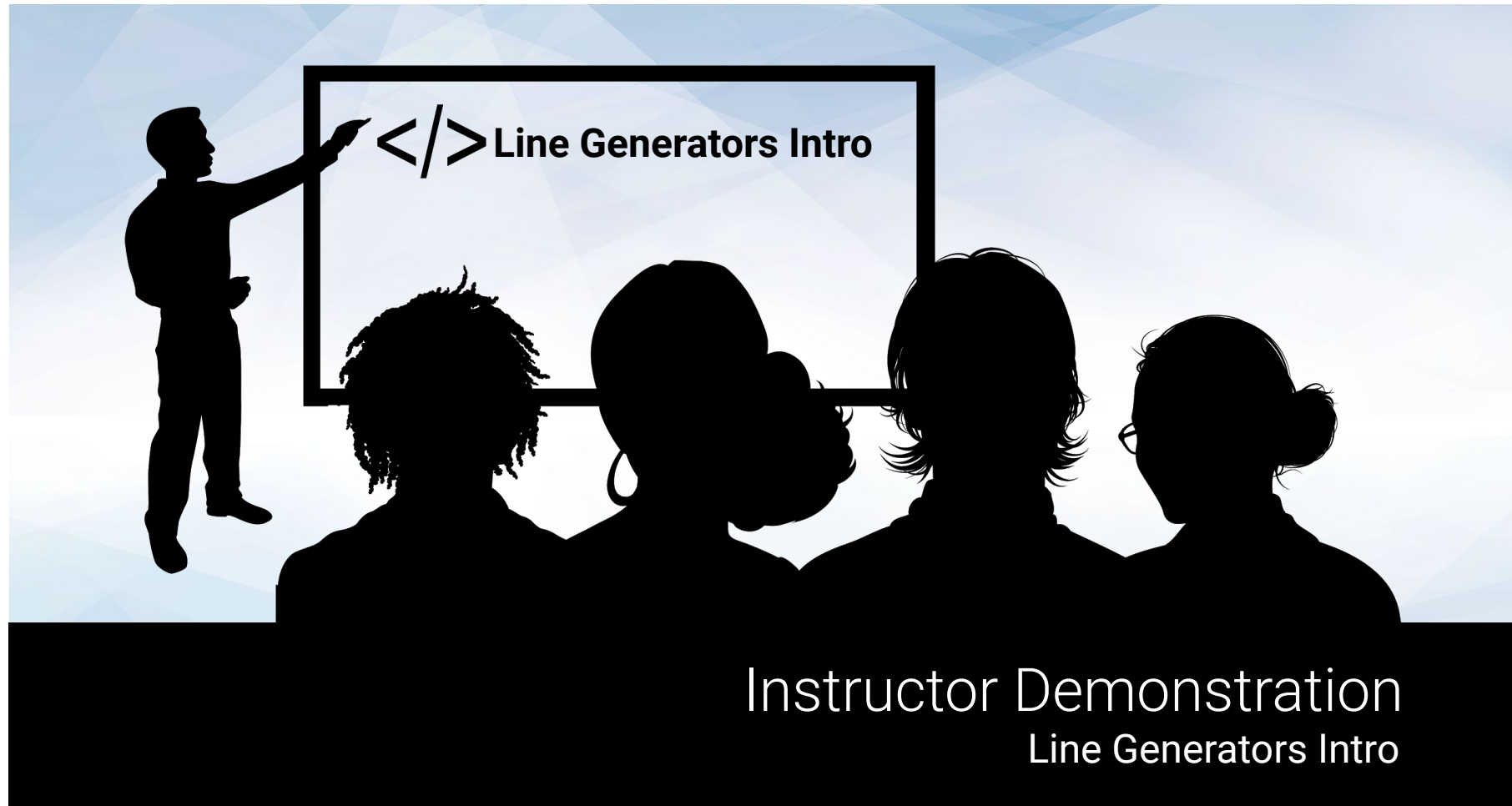
- **Hints:**



- Use prior activities for reference.
  - Checkout your slack for this reference material on [D3 Scales](#).
  - See this example of create [D3 Band Scales & Bottom Axes](#). Remember that the domain and range must both be arrays.
  - For assistance with axis creation with D3, see the [d3-axis documentation](#).
-



**Time's Up!** Let's Review.



# Instructor Demonstration

## Line Generators Intro

# <Time to Code>





## Activity: Generating Lines

In this activity, you will generate a line chart using life expectancy data from CSV data.

**Suggested Time:**  
10 Minutes



# Activity: Generating Lines

---

## Instructions:

- First, take a few minutes to review the code.
- Note that we cast `year` as a number here. This is for demonstration purposes and will normally be cast as a date object. You will see this in the next activity.
- Create scales for x and y so that your line fits the SVG properly.
- Create and store a line generator function using the scale functions for x and y.
- Append a path element to the SVG using the line generator and add styling attributes.
- Test your code by running a server and viewing in the browser. Refine as needed.
- Hints:



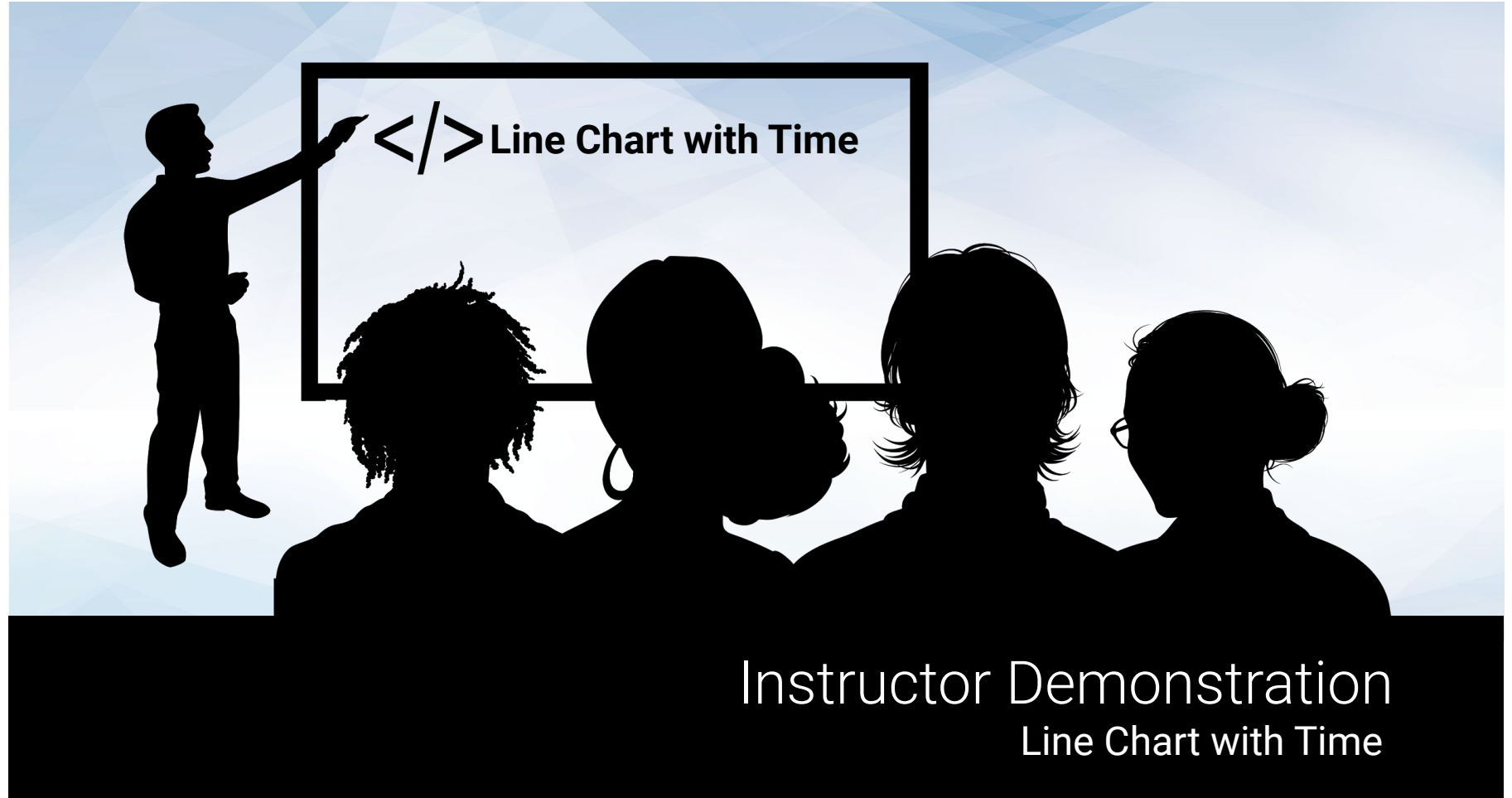
○ Check out [D3 in Depth - Line Generator](#).

---



**Time's Up!** Let's Review.



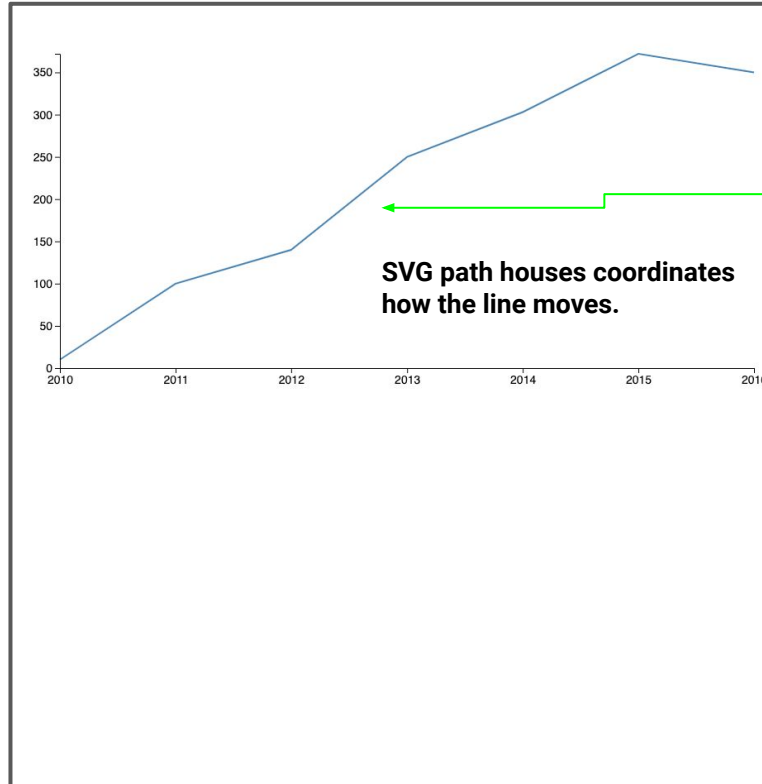


# Instructor Demonstration

## Line Chart with Time

# Line Chart with Time

- `<path>`



```
<!DOCTYPE html>
<html lang="en">
<head>_</head>
<body> == $0
<script src="https://d3js.org/d3.v5.min.js"></script>
<script src="app.js"></script>
<svg width="960" height="500">
  <g transform="translate(60, 60)">
    <path d=
      "M0,369.78494623655916L139.9361022364217,277.8494623655914L279.8722044728434,236.989247
      31182797L420.19169329873486,124.62365591397851L560.1277955271565,70.48387096774192L700.
      0638977635783,0L840,22.473118279569874" class="line"></path>
    <g class="axis" fill="none" font-size="10" font-family="sans-serif" text-anchor="end">
      <path class="domain" stroke="currentColor" d="M-6,380.5H0.5V0.5H-6"></path>
      <g class="tick" opacity="1" transform="translate(0,380.5)"></g>
      <g class="tick" opacity="1" transform="translate(0,329.4247311827957)"></g>
      <g class="tick" opacity="1" transform="translate(0,278.3494623655914)"></g>
      <g class="tick" opacity="1" transform="translate(0,227.2741935483871)"></g>
      <g class="tick" opacity="1" transform="translate(0,176.1989247311828)"></g>
      <g class="tick" opacity="1" transform="translate(0,125.12365591397851)"></g>
      <g class="tick" opacity="1" transform="translate(0,74.0483870967742)"></g>
      <g class="tick" opacity="1" transform="translate(0,22.973118279569874)"></g>
    </g>
    <g class="axis" transform="translate(0, 380)" fill="none" font-size="10" font-family=
      "sans-serif" text-anchor="middle"></g>
  </g>
</svg>
</body>
</html>
```

html body

# Line Chart with Time

- `app.js`

```
var svgWidth = 960;  
var svgHeight = 500;
```

→ Sets the height and the width of the SVG container.

```
var margin = {  
  top: 60,  
  right: 60,  
  bottom: 60,  
  left: 60  
};
```

→ Sets margins for the chart.

```
var chartWidth = svgWidth - margin.left - margin.right;  
var chartHeight = svgHeight - margin.top - margin.bottom;
```

→ Sets the height and the width for the chart area.

```
var svg = d3.select("body")  
  .append("svg")  
  .attr("width", svgWidth)  
  .attr("height", svgHeight);
```

→ Appends a SVG container to the body with the svgWidth and the svgHeight.

```
var chartGroup = svg.append("g")  
  .attr("transform", `translate(${margin.left}, ${margin.top})`);
```

→ Appends a group area and set its margins.

```
var parseTime = d3.timeParse("%Y");
```

→ Configures the function to return a new date object in a string.

```
d3.csv("forcepoint.csv").then(function(forceData) {  
  console.log(forceData);  
});
```

→ Loads data from the file.

# Line Chart with Time

```
forceData.forEach(function(data) {  
  data.date = parseTime(data.date);  
  data.force = +data.force;  
});
```

→ Formats the date and converts the force value to an integer.

```
var xTimeScale = d3.scaleTime()  
  .domain(d3.extent(forceData, data => data.date))  
  .range([0, chartHeight]);
```

→ Configures the time scale.

```
var yLinearScale = d3.scaleLinear()  
  .domain([0, d3.max(forceData, data => data.force)])  
  .range([chartHeight, 0]);
```

→ Configures the linear scale .

```
var bottomAxis = d3.axisBottom(xTimeScale);  
var leftAxis = d3.axisLeft(yLinearScale);
```

→ Configures two functions to create the chart axes passing the scales as arguments.

```
var drawLine = d3.line()  
  .x(data => xTimeScale(data.date))  
  .y(data => yLinearScale(data.force));
```

→ Configures a line function to plot the x and y coordinates using the scales.

```
chartGroup.append("path")  
  .attr("d", drawLine(forceData)) → Function returns the instructions to  
  .classed("line", true);          create the line for the forceData.
```

→ Configures the time scale.

```
chartGroup.append("g")  
  .classed("axis", true) → Appends an SVG group element.  
  .call(leftAxis);       Creates the left axis inside.
```

```
chartGroup.append("g")  
  .classed("axis", true)  
  .attr("transform",  
    `translate(0, ${chartHeight})`)  
  .call(bottomAxis);  
}).catch(function(error) {  
  console.log(error);  
});
```

→ Appends an SVG Group. Creates inner bottom axis translating it to the bottom of the page.



## Activity: Line Chart

In this activity, you will create a line chart using a new data set. In the line chart demonstrate the number of miles per month the user of a fitness application has walked since they started using the app.

**Suggested Time:**  
10 Minutes



# Activity: Line Chart

---

## Instructions:

- Take a moment to study the new data set `miles-walked-this-month.csv`.
  - Define the dimensions for `chartWidth` and `chartHeight`.
  - Append an SVG to the body, and then append a group element to it and translate it to adhere to the margins.
  - Load the data from the CSV. Remember that you will need to run a server to check your page.
  - Use the `d3.timeParse` method in order to create a new function to parse the month from the CSV data and save the function to a variable. You will need to pass the `%B` token as an argument into the `d3.timeParse` method in order to properly configure the new function to create a `Date` object from a string month.
  - Run a `forEach` loop on the `milesData`. Cast the `miles` property of each element in the `milesData` array to a number. Use the time parser function created in the last step to convert the `month` for each element into a `Date` object.
  - Configure your x and y scales as `xTimeScale` and `yLinearScale`.
  - Create your axes generator functions.
  - Run the `d3.line` method to create and save a new line generator function. Configure this function to plot the x-axis using `xTimeScale` function, passing in the `date` value for each element in the data set. Then, configure this function to plot the y-axis using the `yLinearScale` function, passing in the `miles` property for each element in the data set.
  - Append an SVG `path` to the SVG group element. Set the `d` attribute of the new SVG `path` using the line generator function created in the last step. Pass `milesData` into the line generator as an argument. Give this element a class of "line".
  - Append two `<g>` elements and use the axes generator functions you created in step 7 to append an axis to each. Make sure to place these elements to correctly display your axes.
-

# Activity: Line Chart

---

## Instructions:

- **Hints:**



- See D3 documentation for [local.parse](#) to better understand the d3.timeParse method.
- See D3 documentation for [local.format](#) to better understand the tokens used with the d3.timeParse method.
- See D3 documentation for [d3.line](#) to better understand the steps for creating a line generator function.
- Check out a [basic line chart example](#) made by D3 creator, Mike Bostock.
- Check the browser often, print any values you're unsure about to the console.





**Time's Up!** Let's Review.

*The  
End*