

# Lab 3: Intro to D3

[Jump to bottom](#)

Guo, Grace edited this page on Feb 6 · 2 revisions

## Learning Objectives

After completing this lab you will be able to:

- Run a simple HTTP server with `python`
- Load data with D3 (includes learning about asynchronous methods)
- Basics of D3:
  - Create a D3 selection
  - Append elements with D3
  - Data-bindings with D3 to create new HTML elements based on data
  - Set HTML style and attributes with D3

## Prerequisites

- Download the corresponding lab from thecode repo (either using git, or downloading the folder) from the code of this repo (in the Code tab above)
- You have read **Chapters 5 and 6** in [D3 - Interactive Data Visualization for the Web](#) by Scott Murray

## Recommended Reading

- [Let's Make a Bar Chart](#) by Mike Bostock (creator of D3)
- [DOM Manipulation and D3 Fundamentals](#) by A. Lex of U. of Utah

## Additional Reading

- [D3.js Introduction](#)
- [Dashing D3.js](#)
- [Practical applications of a d3js selection](#)

## What to submit

1. You should have completed Activity 1, Activity 2, and Activity 3 (in each respective subfolder).
2. Rename your `lab3` folder to `LastName_FirstName_lab3`

3. Zip up `LastName_FirstName_lab3` as `LastName_FirstName_lab3.zip` and submit it to Gradescope. Note that you will not have to submit to Canvas for this lab.

## Grading

Your assignment will be graded on the following requirements:

- Functionality of Activity 1, 2, and 3 completed

## Activity 0: Run a Local Web Server

---

Before we dive into the details of D3, let's learn to run a local web server while getting a taste of the visualizations you can create using D3. Running a local web server is a crucial skill as you will use it in each of the subsequent labs.

To create data visualizations using D3, you need to store your data in a data file. Usually, the data file is in CSV format. When your visualization is being rendered by D3, your web browser fetches the CSV files from your file system. For your web browser to have access to your CSV files, you will need to run a local web server.

To run the server, open `Terminal` (for Mac) or `Command Prompt` (for Windows). Navigate to the folder `lab3/activity_0` using for example, `cd /lab3/activity_0`.

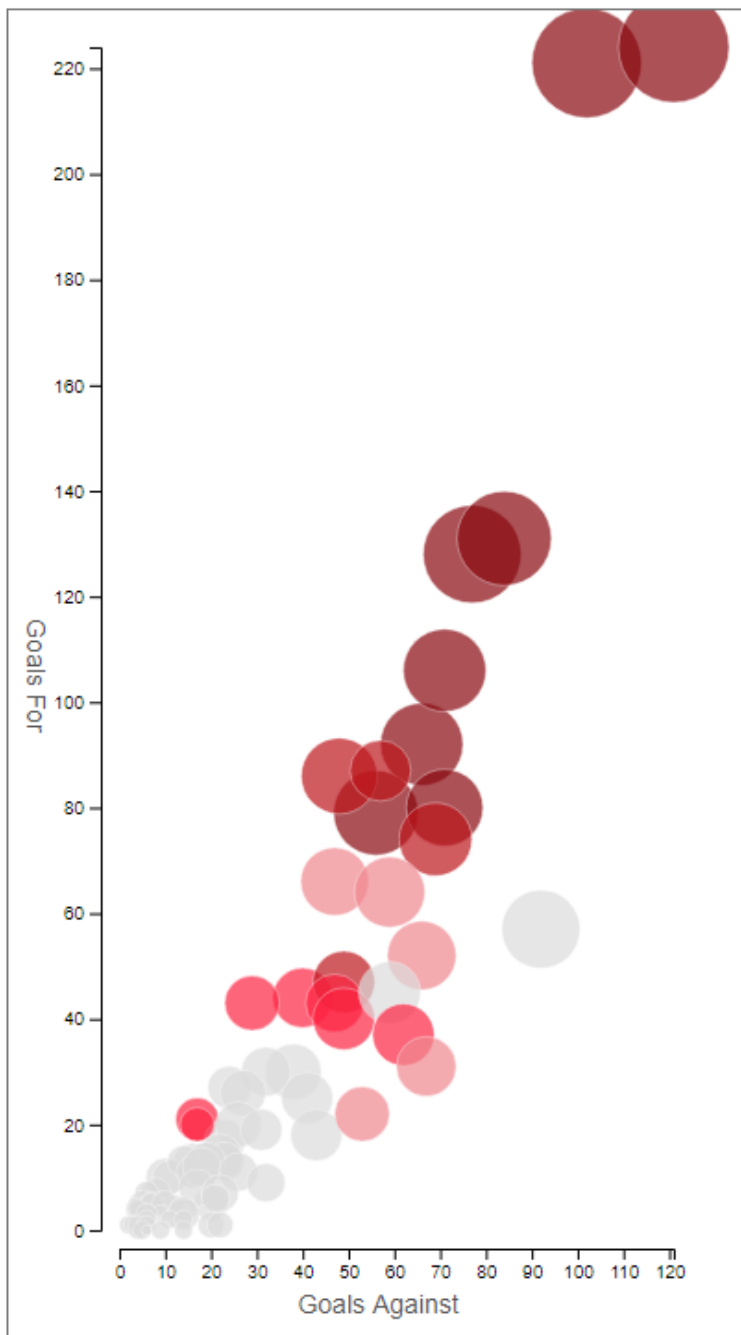
You should now be in the directory `lab3/activity_0`. Now we will run a python command which will create an HTTP web server for this current directory and all of its sub-directories. In the console execute the following command if you're running Python 2.x:

```
python -m SimpleHTTPServer 8080
```

if you're running Python 3.x or higher, use

```
python -m http.server 8080 (or python3 -m http.server 8080)
```

Now, open your browser and type `http://localhost:8080/` in the URL bar and press enter or go. You should see the following visualization:



If you go to the directory `lab3/activity_0`, you will see the file named `index.html`. When you enter `http://localhost:8080/` in your browser, your browser will automatically load the file named `index.html` in the folder.

## Tutorial 1: What is Data-Driven Documents (D3)

D3.js (Document-Driven-Data) is a powerful JavaScript library for manipulating documents based on data.

"D3 allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. For example, you can use D3 to generate an HTML table from an array of numbers. Or, use the same data to create an interactive SVG bar chart with smooth transitions and interaction." (D3.js, Mike Bostock)

A summary of D3's features and key aspects by Scott Murray:

- **Loading** data into the browser's memory
- **Binding** data to elements within the document and creating new elements as needed
- **Transforming** those elements by interpreting each element's bound datum and setting its visual properties accordingly
- **Transitioning** elements between states in response to user input

We will introduce all these concepts in the following weeks.

**We will use the version 5.x of D3 in this class.** The 2nd edition of the book (<http://alignedleft.com/work/d3-book-2e>) utilizes version 4.x. The major difference between version 4.x and version 5.x lies loading data files into the web browser (see Tutorial 3). The two versions are otherwise very similar.

## Tutorial 2: Integrating D3 into Your Project

---

Before working with D3 you have to include the D3 library first. We recommend the minified version which has a smaller file size and faster loading time. D3 hosts the latest minified version at:

<https://d3js.org/d3.v6.min.js>

You will include a `<script>` element in your html to include the D3 library:

```
<html>
  ...
    </div>
  </body>
  <script src="https://d3js.org/d3.v6.min.js"></script>
  ...
</html>
```

Including a library with a `<script>` element loads the library as a global namespace. It allows you to use `d3` in your code to reference the D3 object. With `d3`, we can access its methods and properties.

## Tutorial 3: Loading Data Files Using D3

---

Instead of typing the data in a local variable in a `.js` file (like we did in the previous lab), we can load data asynchronously from external files. The D3 built-in methods make it easy to load JSON, CSV and other data file types. The use of the right file format depends on the data - JSON should be used for hierarchical data and CSV is usually a proper way to store tabular data.

By calling D3 methods like `d3.csv()`, `d3.json()`, `d3.tsv()` etc. we can load external data resources in the browser:

```
d3.csv('baseball_hr_leaders_2017.csv').then(function(dataset) {  
    // the variable `dataset` is an array of data elements  
});
```

The above function takes two arguments: a string representing the path of the file, and an anonymous function. Usually, we call the anonymous function a **callback function**.

You may wonder how the callback function works. In our data loading problem, loading a file from the disk or an external server takes a while. The callback function is executed only until receiving a notification that the data loading process is complete.

The implication is that **code that depends on the dataset should exist only in the callback function**. This makes sure that your data-dependent code will have access to the data. To clarify this, let's look at the following example:

```
d3.csv('baseball_hr_leaders_2017.csv').then(function(dataset) {  
    console.log("Line 1: inside the callback function");  
});  
console.log("Line 2: outside the callback function");
```

In this example, `console.log("Line 2: outside the callback function");` is executed first. The CSV file is then loaded into the browser. Finally, `console.log("Line 1: inside the callback function");` is executed. Hence, if you put your data-dependent code outside the callback function, it might be executed before your data file is loaded and may not have access to the data.

A NOTE ON FILE NAMING CONVENTION: Always use something like

`d3.csv("baseball_hr_leaders_2017.csv")` (without `./`) and not `d3.csv("./baseball_hr_leaders_2017.csv")` (with `./`). The later one works in Mac on local server but not on Windows, while the first one works in both Mac and Windows.

## Tutorial 4: Method Chaining and D3 Selections

---

### Method Chaining

Method or function chaining is a common technique in JavaScript, especially when working with D3. It can be used to simplify code in scenarios that involve calling multiple methods on the same object consecutively. The functions are "chained" together with periods.

```
d3.selectAll('circle')  
    .attr('r', 10)  
    .style('fill', '#777');
```

Here is the execution sequence of the above example. `d3.selectAll('circle')` returns all the circles in a page. `.attr('r', 10)` changes the radii of the returned circles to 10. `.attr('r', 10)` again returns all the circles in a page. `.style('fill', '#777');` changes the color of the circles to '#777'.

Here is the alternative code without method chaining:

```
var circles = d3.selectAll('circle');
circles.attr('r', 10);
circles.style('fill', '#777');
```

The main drawback here is that you have to instantiate a new variable. The example code from here on out will use method chaining extensively.

## D3 Selections

D3 selection allows you to select [DOM elements](#) for manipulation. DOM elements consists of any HTML elements and SVG elements in a page. D3 selection is your entry point for finding, creating, and removing DOM elements. You also use selections to apply styles and attributes that dictate how those elements appear on the page and respond to events.

`d3.select()` selects the first element that matches a selector. Selectors can specify tags ( `p` in our example below), classes, and IDs:

```
d3.select('p')
  .style('font-size', '0.9em')
  .text('First Paragraph');
```

Notice, however, that as mentioned previously, only the first element that matches is selected. Of course, it is more practical to select all elements of a certain type, which we can achieve with `d3.selectAll()`. The code below selects all the paragraph `p` and changes the font size.

```
d3.selectAll('p')
  .style('font-size', '0.9em');
```

This example illustrates **the declarative approach of D3**: we don't have to iterate over a list of elements and apply the style. Instead we select a set of elements through rules and declare properties.

Once you have a selection, you can bulk-modify it's content, not only in terms of style, but we can modify arbitrary attributes using `selection.attr(name[, value])`, the textual content of the elements with `selection.text([value])`, etc. We can also append elements to a selection.

## Append Elements

With a D3 selection, we can now `append()` elements to the selection. For example, calling `d3.select('svg').append('g')` selects the `svg` canvas, creates a new `<g>` element and adds it as a child of the `svg` canvas. Note that `append()` adds child elements at the end of the child list, while `insert()` adds them to the beginning of the child list of an element. Here is another example:

```
var group = d3.select('svg').append('g');
group.append('circle');
group.append('text');
```

In the above example, `d3.select('svg').append('g')` appends a `<g>` element to the `svg` canvas and returns the appended `<g>` element. Calling `group.append('circle');` and `group.append('text');` will add a `<circle>` element and a `<text>` element to the `<g>` element.

```
<svg>
  <g>
    <circle/>
    <text></text>
  </g>
</svg>
```

## Binding Data

Data visualization is a process of mapping data to visuals. Data in, visual properties out. Maybe bigger numbers make taller bars, or special categories trigger brighter colors. The mapping rules are up to you. With D3, we bind our data input values to elements in the DOM. Binding is like “attaching” or associating data to specific elements, so that later you can reference those values to apply mapping rules.

We use D3's `selection.data()` method to bind data to DOM elements. Binding data requires two things:

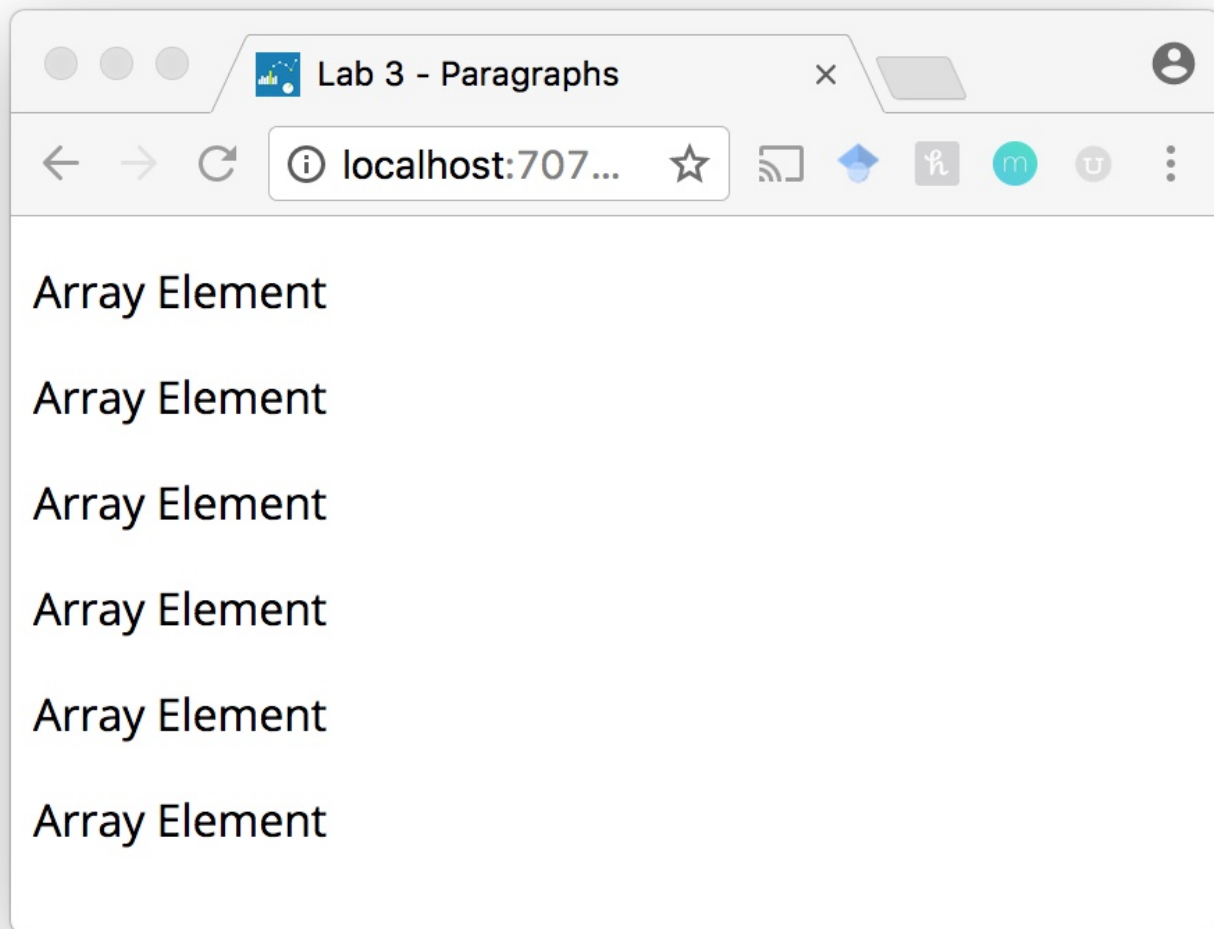
- The data (an array of objects, strings, numbers, etc.)
- A selection of DOM elements – the elements your data will be associated with

Let's take a look at this example:

```
var states = ["Connecticut", "Maine", "Massachusetts", "New Hampshire", "Rhode Island",
"Vermont"];

var p = d3.select("body").selectAll("p")
  .data(states)
  .enter()
  .append("p")
  .text("Array Element");
```

You will see the following:



Here's what's happening:

```
d3.select("body")
```

Finds the body in the DOM and hands off a reference to the next step in the chain.

```
.selectAll("p")
```

Selects all paragraphs in the DOM. Because none exist yet, this returns an empty selection. Think of this empty selection as representing the paragraphs that will soon exist.

```
.data(states)
```

There are six values in our array called `states`. This step binds each value to a `<p>` element that does not yet exist. In total, there are six `<p>` elements that do not yet exist because there are six values in the `states` array. You can consider these `<p>` elements that do not yet exist placeholders.

```
.enter()
```



Select the six placeholder elements.

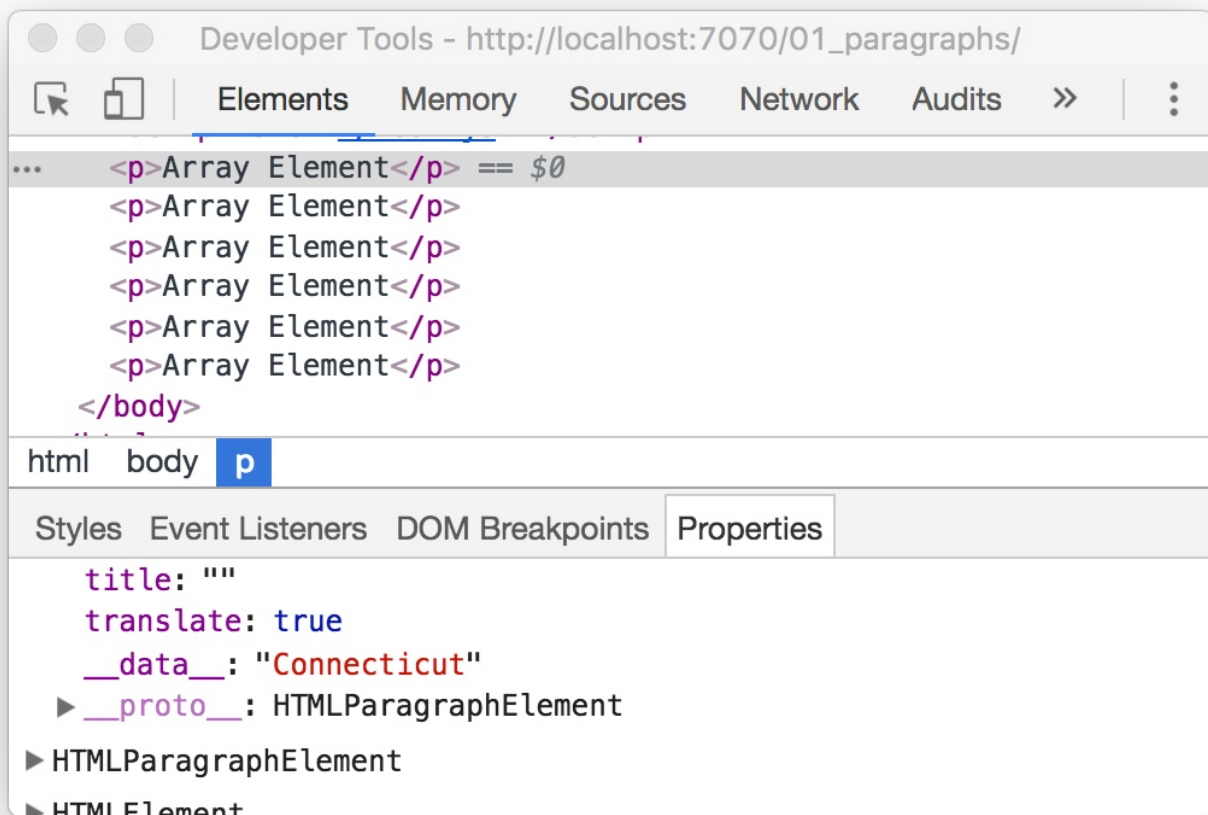
```
.append("p")
```

Create the six placeholder elements, which now becomes six `<p>` elements. Each `<p>` element is associated with a state value.

```
.text("Array Element")
```

Changes the text of each of the six `<p>` elements to "Array Element".

If we inspect the `p` elements that we added to the `body` with the Web Inspector, we will see that in the "Properties" tab for each of these `p` elements that there is a `__data__` property associated with each.



When D3 binds data to an element, that data doesn't exist in the DOM, but it does exist in memory as a `__data__` attribute of that element. And the console is where you can go to confirm whether or not your data was bound as expected.

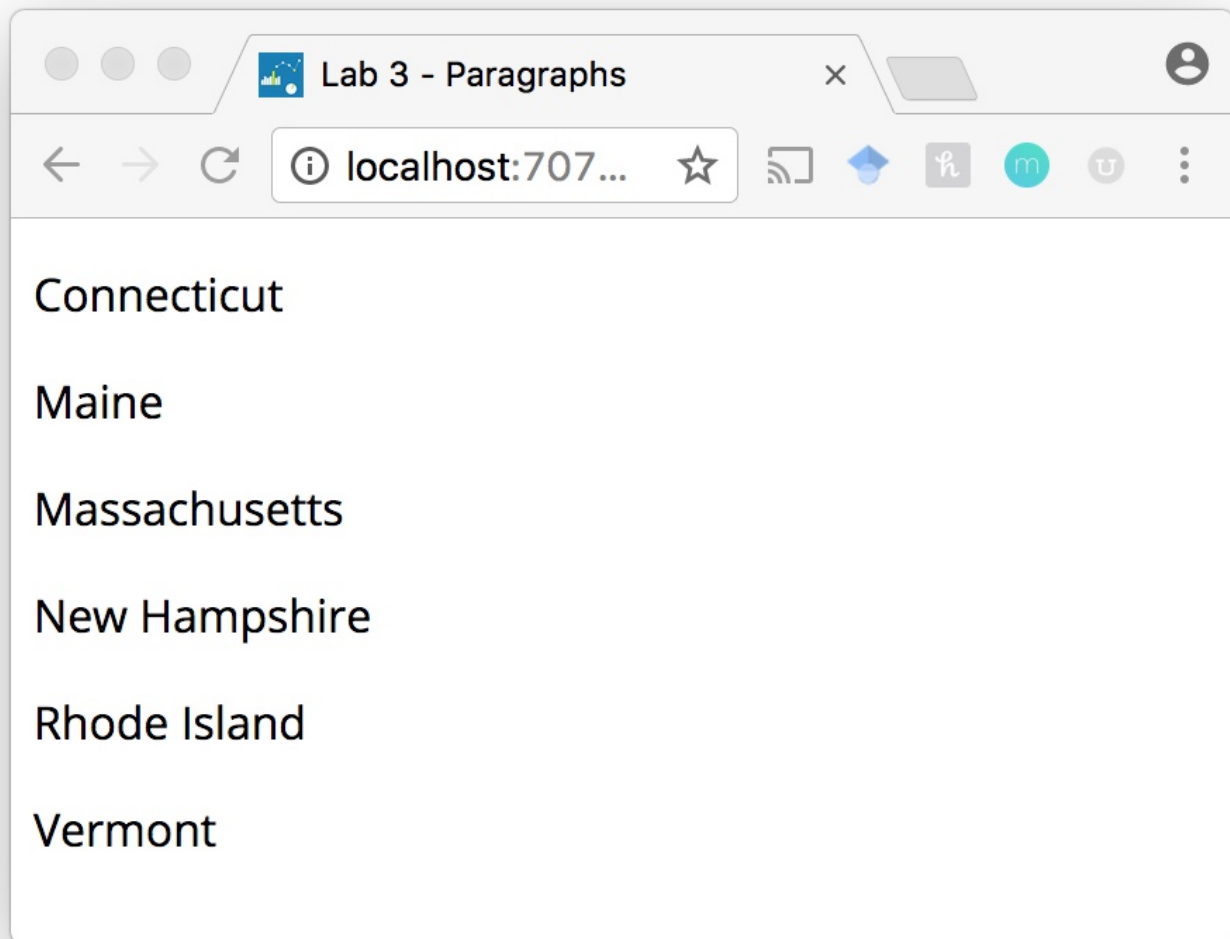
We can see that the data has been loaded into the page and is bound to our newly created elements in the DOM. Now we are going to use each string bound to the `p` elements by setting the text contents of each `p` to the state name:

```
var p = d3.select("body").selectAll("p")  
  .data(states)  
  .enter()  
  .append("p")  
  .text("Array Element");
```

Let's change the last line to:

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

Now we will see the following on our page:



Here, we used an anonymous function that is called by D3 for each element in the selection. The anonymous function `function(d, i)` takes two inputs.

- `d` = the data element bound (in this case each state, e.g. "Connecticut", "Maine", etc.)
- `i` = the index for that bound array element (e.g. 0, 1, etc.)

Generally in D3 documentation and tutorials, you'll see the parameter `d` used for the current data element and `i` (or `index`) used for the index of the current data element. The index is passed in as the second element to the function calls and is optional.

The function `function(d, i)` loops through all the six

elements to change the text values. The return statement `return d;` specifies that the text value of a

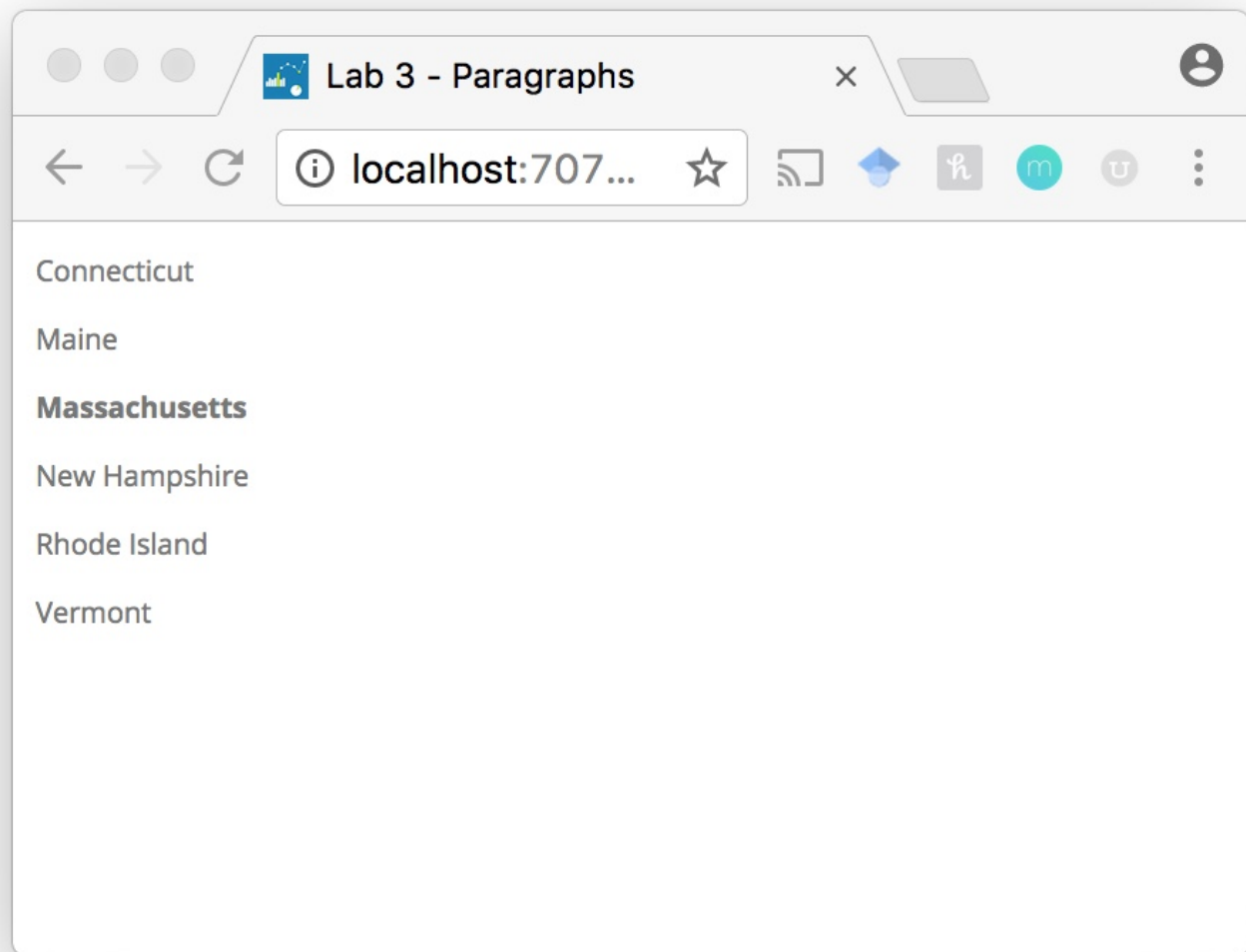
elements should be equal to `'d'`, which is a state name.

## HTML Attributes and CSS Properties

Styles, attributes, and other properties can also be specified based on the data.

For example, in the previous example we can emphasize "Massachusetts" with a bold font-weight. We can also style all of the `p` elements:

```
var p = d3.select('body').selectAll('.state-name')
    .data(states)
    .enter()
    .append('p')
    .attr('class', 'state-name')
    .text(function(d, i) { return d; })
    .style('color', '#777')
    .style('font-size', '10px')
    .style('font-weight', function(d) {
        return d == 'Massachusetts' ? 'bold' : 'normal';
    });
```



- We use D3 to set the paragraph content, the HTML class, the `color` and as the last property, the `font-weight` which depends on the individual array value.
- If you want to assign specific styles to the whole selection (e.g. `font-color: blue`), we recommend you define an HTML class ("state-name" in our example) and add these rules in an external stylesheet. That will make your code concise and reusable.

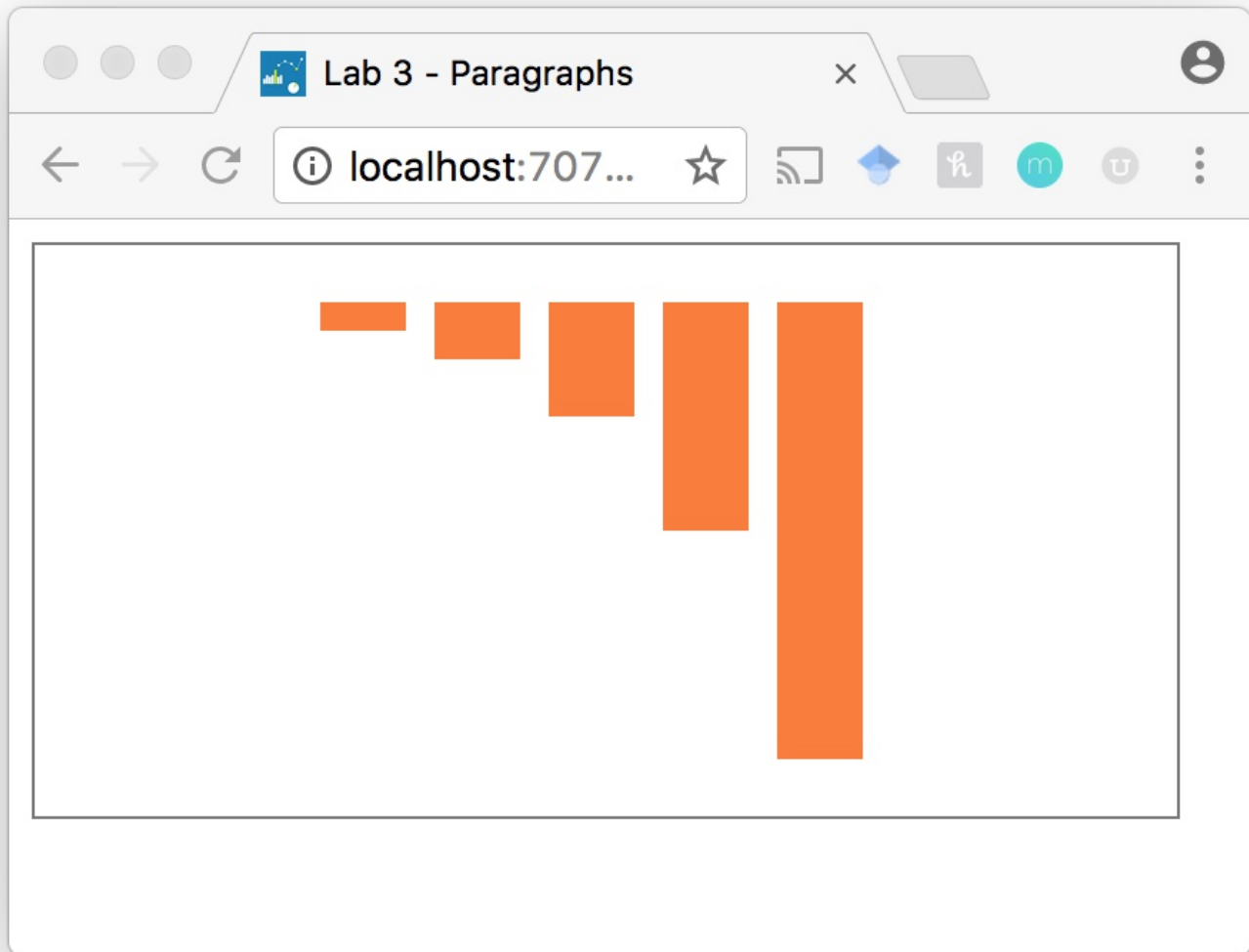
In the following example we use numeric data to create five rectangles. We then style them by data:

```
var numericData = [1, 2, 4, 8, 16];

var svg = d3.select('svg');

// Add rectangles
svg.selectAll('rect')
  .data(numericData)
  .enter()
  .append('rect')
  .attr('fill', '#f77e46')
  .attr('width', 30)
```

```
.attr('height', function(d){  
  return 160 * d / 16;  
})  
.attr('y', 20)  
.attr('x', function(d, i) {  
  return (i * 40) + 100;  
});
```



- We have appended SVG elements to the DOM tree in our second example.
- It is crucial to set the SVG coordinates. If we don't set the x and y values, all the rectangles will be drawn on the same position at (0, 0). By using the index - of the current element in the selection - we can create a dynamic x property and shift every newly created rectangle 40px to the right (plus an initial 100px to center them).
- We also set the height attribute of each rectangle based on the bound data. Remember from the SVG lab that the y-coordinate system in SVG starts at the top (0px) and goes down. This is why our bar chart is aligned at the top. We would need to set the y-position of the rectangles based on data to align them at the bottom (hint for upcoming homework).

## Activity 1: Creating a List of Data

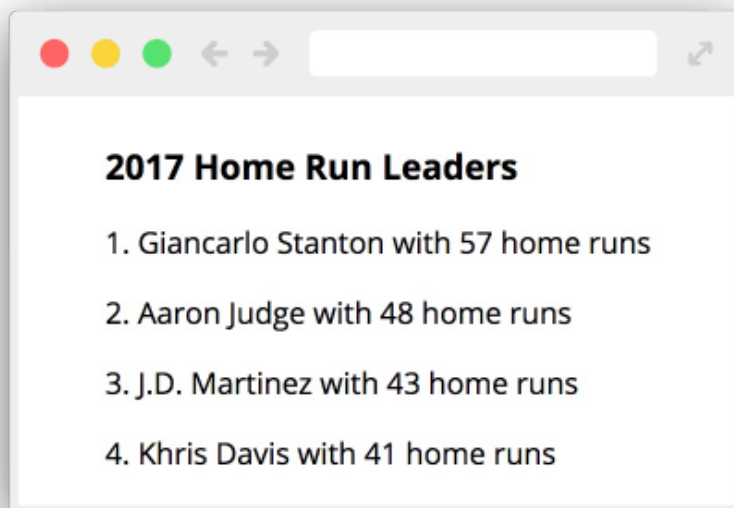
Start an http server for this lab's directory. You can accomplish this by opening a command line window, navigating to this lab's directory (e.g. `cd ~/Fall-2019-Labs/lab3/` ). From there you will start the http server by executing: `python -m SimpleHTTPServer 8080` (for Python 2) or `python -m http.server 8080` . You will need to start an http server for every lab from here on out. Servers are required to serve local files and run JavaScript. If you are using a different version of Python, you may need to use a different command to launch your local webserver.

Similar to the examples above, you will be creating a list of paragraph elements in your web page. As always, please start by opening your code editor to `lab/activity_1/` . In that directory you will see a familiar project structure with `index.html` , `style.css` , and `paragraphs.js` files. `paragraphs.js` is where you will write JavaScript code for this activity. Also notice that there is a CSV data file in the directory: `baseball_hr_leaders_2017.csv` .

The CSV dataset is 2017's top 10 Home Run (HR) leaders from the Major League Baseball. (For those of you who aren't baseball fans, a home run is when a player hits the ball over the fence in the outfield which will score one or more runs, so a very good thing.) See below for a snippet of the dataset:

| name              | rank | year | homeruns |
|-------------------|------|------|----------|
| Giancarlo Stanton | 1    | 2017 | 57       |
| Aaron Judge       | 2    | 2017 | 48       |
| J.D. Martinez     | 3    | 2017 | 43       |
| Khris Davis       | 4    | 2017 | 41       |

For this activity you will use d3 data binding to create a paragraph ( `<p>` ) for each player in the list. You will also add text content to each paragraph to describe the player's Home Runs for this season and their rank. In the end your web page should look like this:



You will edit the `activity_1/paragraphs.js` file for this activity.

### 1. Load the CSV file

Using `d3.csv()`, load the dataset at `./baseball_hr_leaders_2017.csv`. Make sure to write a callback function to access the loaded data.

We recommend using `console.log()` on the loaded data to double check everything went according to plan.

### 2. Create your paragraphs

Append a `p` element to the `#homerun-leaders` div for each player in the dataset. (Recall that '#' indicates id.) You will need to follow the above examples to achieve this. Remember that we are now reading from a data file rather than declaring some data within the javascript file.

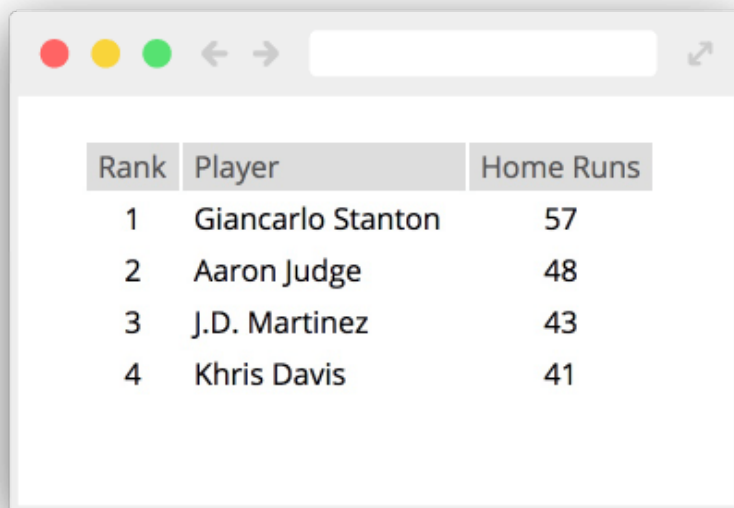
### 3. Add text content

Add text content using `text()` for your D3 data selection. The text content should include the player's HR rank, their name, and the number of Home Runs they hit this season. Again, you can follow the examples above to add text content. Hint: you need to create only one text object with all the right words in it.

### 4. Highlight the Home Run King

Add a bold font styling to the player that has the most Home Runs this season, Giancarlo Stanton. There are a number of different ways to select and style this `p` element based on the rank value, index of the player JS object, or the player's name.

Create a table of the dataset like in the below snippet and add the table to the `#main` div after the `#homerun-leaders` div:



| Rank | Player            | Home Runs |
|------|-------------------|-----------|
| 1    | Giancarlo Stanton | 57        |
| 2    | Aaron Judge       | 48        |
| 3    | J.D. Martinez     | 43        |
| 4    | Khris Davis       | 41        |

We have provided the following table HTML to get you started:

```
<table id="homerun-table">
  <thead>
    <tr>
      <td>Rank</td>
      <td>Player</td>
      <td>Home Runs</td>
    </tr>
  </thead>
  <tbody>
    </tbody>
</table>
```

You will need to append a table row `<tr>` for each player to the `<tbody>`. You will then need to add a `<td>` cell element for each column. Think about what code should go in your html file and what should go in the javascript file. This can be pretty challenging if you haven't worked much with D3 before.

Style your table so that readers can differentiate the column headers from the column data content.

## Activity 2: Drawing a Pixel Scatterplot

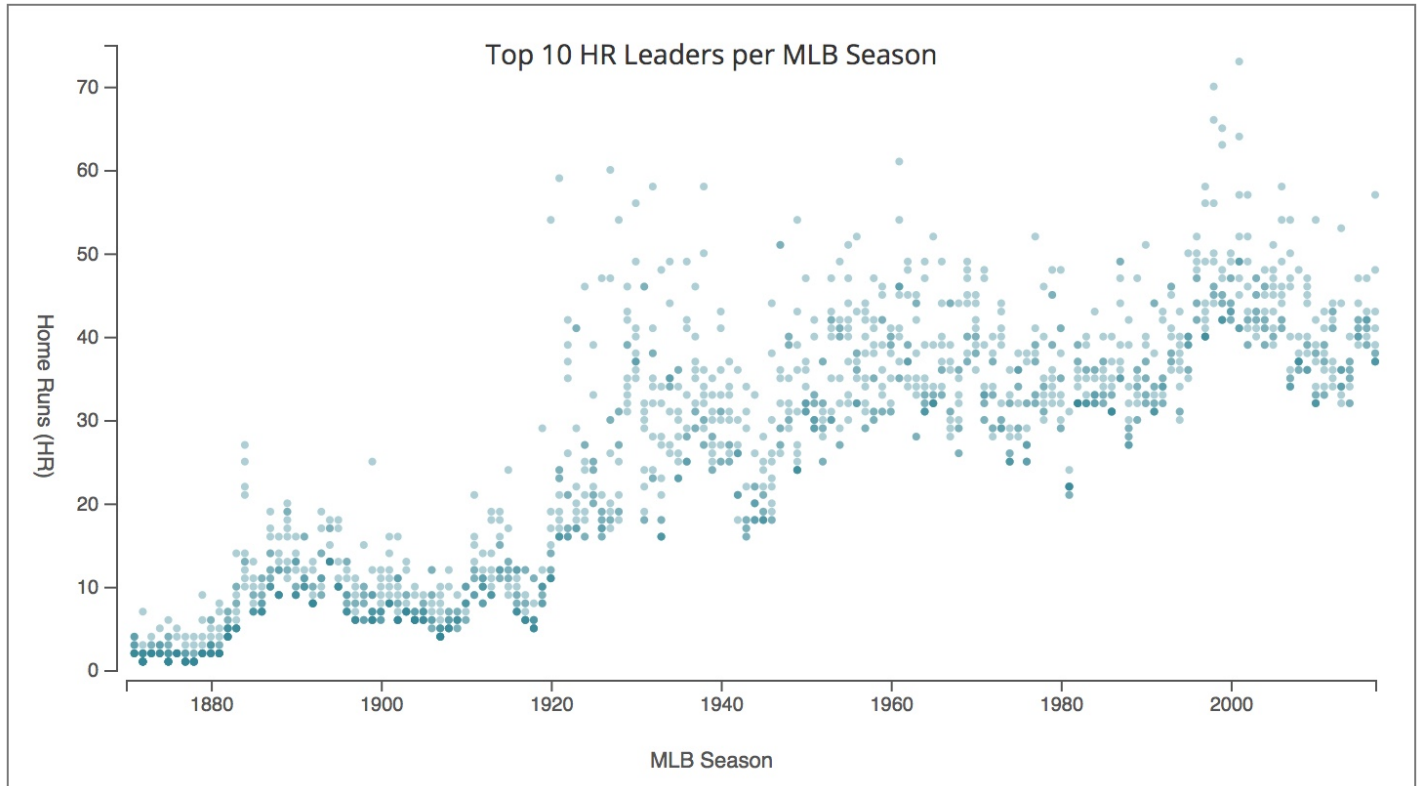
The starter code for this activity can be found at `lab3/activity_2/`.

We will learn from our lectures that a pixel chart is a great way to visualize data when there is a lot of data. Pixel charts are also effective when the pixel represents one data case.



In this exercise you will create a pixel scatterplot for the expanded Home Run leaders dataset. The expanded dataset includes not just the top 10 HR leaders from 2017, but all of the top 10 Home Run leaders from 1871 to the current MLB season. In total there are 1,679 data cases in the `baseball_hr_leaders.csv` dataset.

You will create the following scatterplot during this activity:



You will edit the `activity_2/scatterplot.js` file for this activity. We have already added code to create the axes, scales and labels for the scatterplot. You will be calling the following two functions that we have written for you: `scaleYear(year)` and `scaleHomeruns(homeruns)`. Each function returns a scaled pixel value for the input numeric data value.

### 1. Load the CSV file

Load the dataset at `./baseball_hr_leaders.csv` using `d3.csv()` as you did in the previous activity. Again be sure to include all code that requires the loaded dataset within the callback function.

### 2. Create and center your circles

Create a `circle` element for each data case by: Select all `circle` elements in the `svg`, then create a data-binding with the dataset, and finally enter/append `circle` elements.

Set the center point of each circle by data.

- The `cx` position should be set to the `year` data attribute for each circle (use the provided `scaleYear(year)` method).

- The `cy` position should be set to the `homeruns` data attribute (use the provided `scaleHomeruns(homeruns)` method).

Set the radius of each circle to all be `2px`.

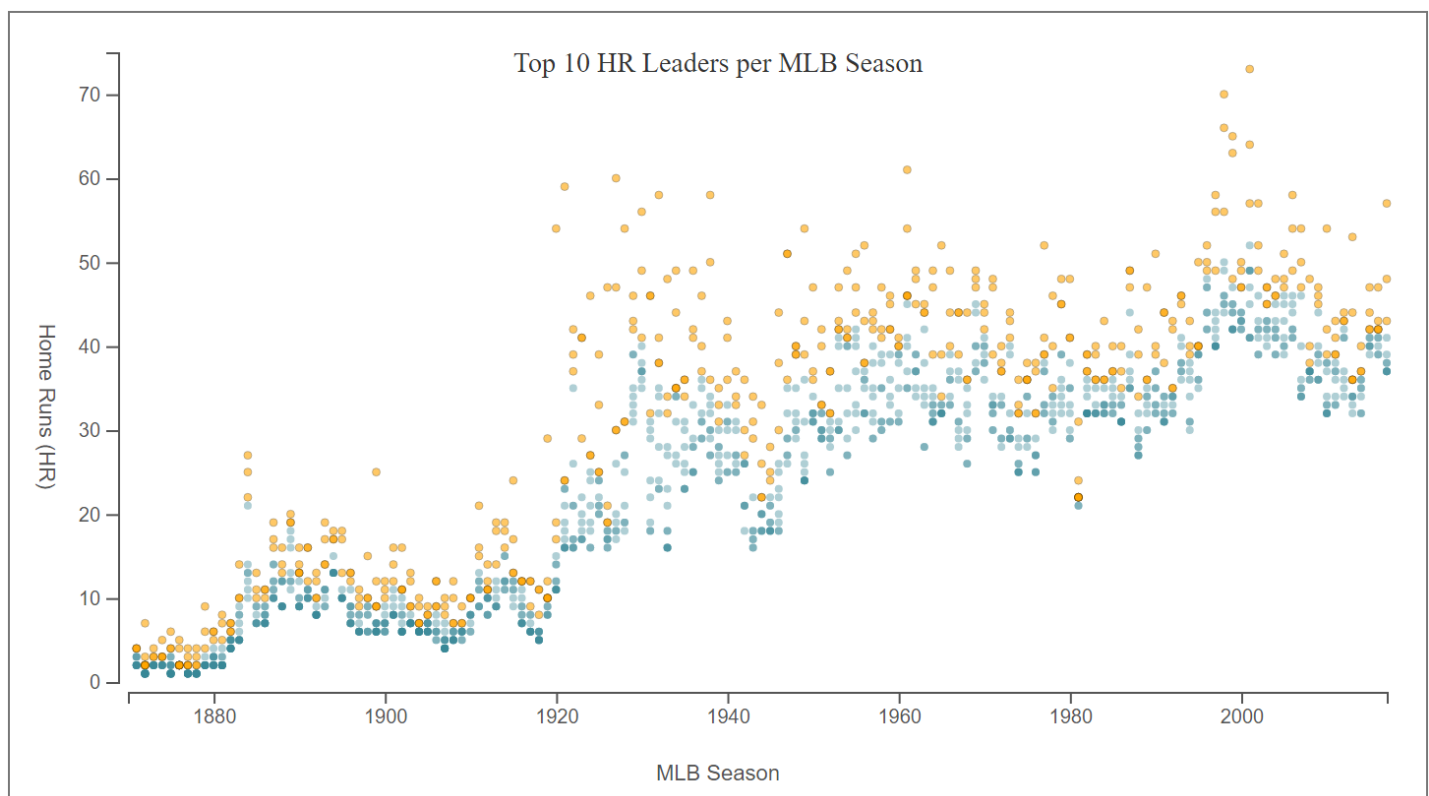
### 3. Style your circles

You should now see all of the black circles placed in the chart. You cannot see it because of the black-opaque fill of the circles, but a lot of home run leaders are on top of each other - this is due to players being tied for Home Runs in the same year. To fix this we ask that you style the `fill` and `opacity` of each circle, so that we can see where circles occlude each other.

Hint, we recommend using the `style.css` file to style all circles. In-line styling is better reserved for data-bound styling.

### 4. Make top-ranked players stand out.

Once your scatterplot looks like the one shown in the image above, you will need to update the style of circles so that top-ranked players (players with rank [1-3]) stand out. It should look like this:



**Hint:** You can accomplish these changes using CSS classes or using d3's `.style()`. For the first approach, look at `style.css` and think about how you could style specific circles differently. For the second approach, read more about using `.style()` [here](#) or look at [this example](#).

At this point, save your work. For Activity 3 below, you will want to duplicate your Activity 2 folder and files as some changes will be conflicting. Name this new folder `activity_3`.

## Activity 3: Drawing a Pixel Scatterplot

---

Starting with the files in your new folder, let's add some more features.

### 1. Create a text label for each player.

For this part you will need to append nested SVG elements. You will want to append a `<g>` element for each data case. You will want to translate that group based on the `year` and `homers` data attributes as you did earlier for the circles.

Then, you will want to append a `<text>` and `<circle>` element to each `<g>`. **Note, you do not need to set the x or y positions of the text or circle because of the transform of the enclosing `<g>` element.** (Remember the previous lab on SVG transform attributes). However, try to style the text elements and make them showing right above each circle element as shown in the following example gif.

When you achieve this you will see a ton of text elements!

### 2. On hover each group to show the text label.

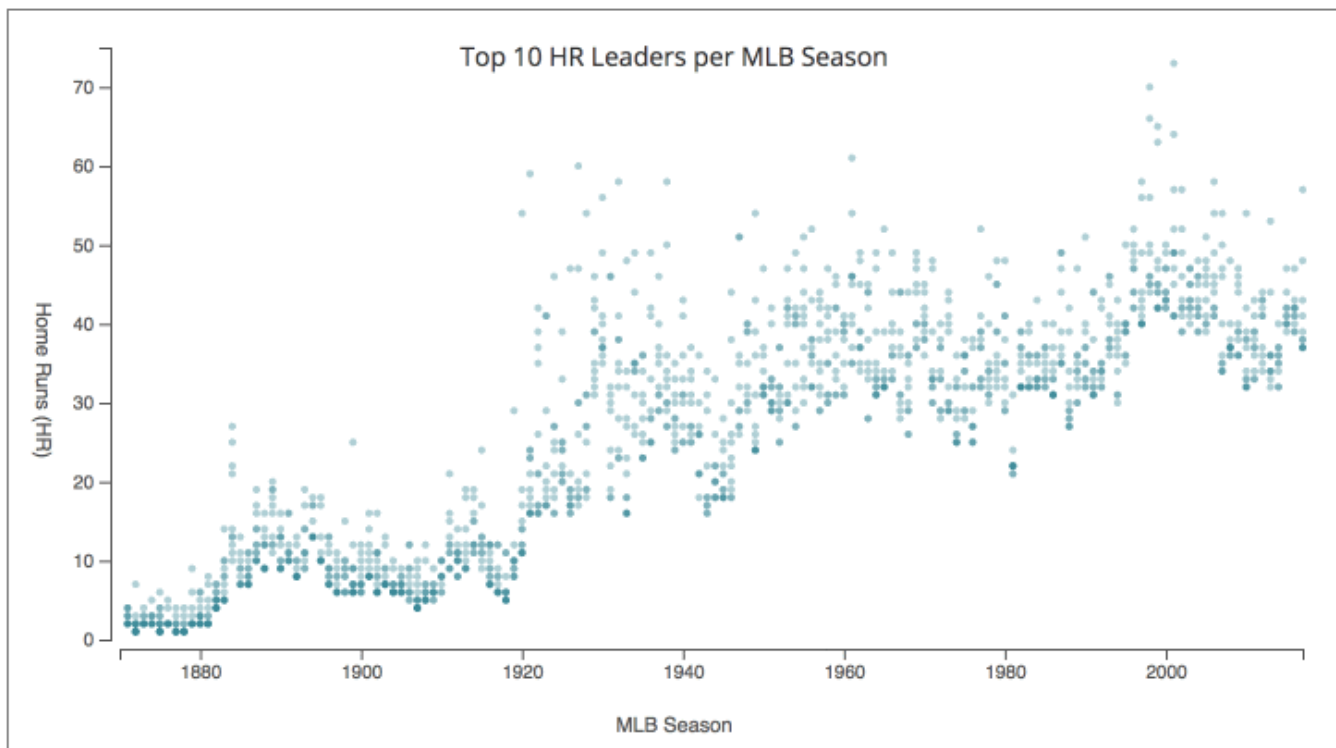
To create this effect, you will want to set all text labels to have an 'opacity' of 0 to start. We recommend using the `style.css` file to make a CSS rule.

Next you will create another CSS rule for the text labels, however you will add the `:hover` CSS selector to the group element (e.g. `g:hover text`).

What this does is styles the text elements when the parent `g` element is mouse-hovered. As you may have guessed, within this rule you want to set the text `opacity` to be 1. This makes the text visible on mouse-hover.

In addition, create a similar CSS rule for the circles to set the circle `opacity` to be 1 when its parent `g` element is mouse-hovered.

See below for how the hover interaction looks:



To further clarify, what we are looking for here is that the hover interaction is **only** triggered when you **hover over the point**.

This lab is based on the following material:

- Hanspeter Pfister's CS171 Lab Material (Harvard)
- [DOM Manipulation and D3 Fundamentals](#) by Alex Lex of U. of Utah
- [Practical applications of a d3js selection](#)
- [D3 - Interactive Data Visualization for the Web](#) by Scott Murray

▼ Pages 11

► [Home](#)

► [Lab 1: Intro to HTML, CSS, and SVG](#)

► [Lab 2: Javascript 101](#)

▼ [Lab 3: Intro to D3](#)

Learning Objectives

Prerequisites

Recommended Reading

Additional Reading

What to submit

## Grading

Activity 0: Run a Local Web Server

Tutorial 1: What is Data-Driven Documents (D3)

Tutorial 2: Integrating D3 into Your Project

Tutorial 3: Loading Data Files Using D3

Tutorial 4: Method Chaining and D3 Selections

Method Chaining

D3 Selections

Append Elements

Binding Data

HTML Attributes and CSS Properties

Activity 1: Creating a List of Data

1. Load the CSV file
2. Create your paragraphs
3. Add text content
4. Highlight the Home Run King

Activity 2: Drawing a Pixel Scatterplot

1. Load the CSV file
2. Create and center your circles
3. Style your circles
4. Make top-ranked players stand out.

Activity 3: Drawing a Pixel Scatterplot

1. Create a text label for each player.
2. On hover each group to show the text label.

▶ [Lab 4: D3 Selections and Grouping](#)

▶ [Lab 5: D3 Enter, Update, Exit, and Filter](#)

▶ [Lab 6: Brushing and Linking](#)

▶ [Lab 7 A: Force Directed Graph](#)

▶ [Lab 7 B: Brushing and Linking](#)

▶ [Lab 7 C: Scrollytelling](#)

▶ [Lab 7 D: Interactive Visual Comparison](#)

## Clone this wiki locally

<https://github.gatech.edu/CS4460/Spring23-Labs-PUBLIC.wiki.git>

