Lab 4: D3: Data Loading and Binding

CS 7450: Information Visualization Spring 2026

Georgia Tech

Reminders

- Prototype Part 1 is due **this Friday (2/13)!!**
- Don’t forget to sign up for a team on the Prototype Team Sign Up Excel Sheet!
    - If you don’t, then you’ll be expected to do the team assignment by yourself…

Lab 4 Instructions

Goals for the labs

- ~~Learn Web Fundamentals~~
    - ~~HTML, CSS, SVG, JS, DOM, and various tooling methods~~
- D3 Fundamentals
    - **Data loading, binding**, scalings, axes, and various charts
- D3 Interactions
    - Event listeners, selection, brushing and linking, sorting, and transitions
- Advanced D3 Techniques
    - Network visualizations, scrollytelling, and geospatial projections

Georgia Tech

REVIEW: Data Binding

- In D3 we have the ability to **bind data** to DOM elements
  - This establishes a relationship between each element in the selection and each item in the data array
- Calling .data(data) binds the array of data to the selected elements
  - Now that these are attached, they can be modified directly with other D3 methods
    - Ex: .attr(), .text(), and .style()
- What happens if the number of elements and the size of the data are different?

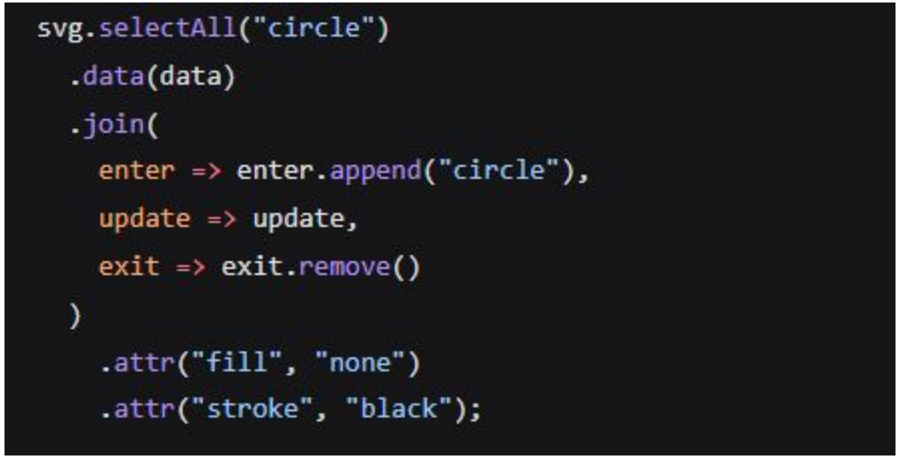
REVIEW: The Enter-Update-Exit Pattern

- Enter
  - Used when there are more data points than the elements
  - Typically used when there is new data with no elements
  - Used to create new elements with .enter().append()
- Update
  - Used when you need to update elements that already exist
  - Typically for changing element attributes like height or color
- Exit
  - Used for when there are more data elements than data points
  - Typically used for removing the extra elements that don’t have data
  - Used to remove with .exit().remove()

When in doubt, .join() !

- Important to understand the enter-update-exit states
- But when in doubt, .join() will handle all 3!
  - Ensures that the SVG always perfectly matches the data
  - Call **after** binding the data but **before** modifying the attributes

You can get pretty specific with it!



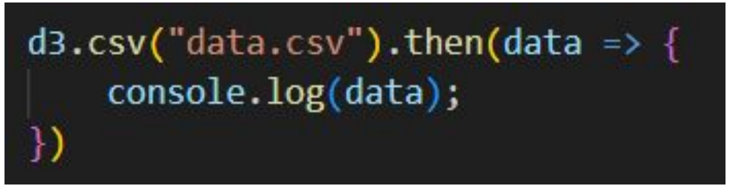
```
svg.selectAll("circle")
  .data(data)
  .join(
    enter => enter.append("circle"),
    update => update,
    exit => exit.remove()
  )
    .attr("fill", "none")
    .attr("stroke", "black");
```

REVIEW: Internal Data Loading

- Last lab we used data from an array declared inside our program
  - This is nice for quickly accessing the data
    - No waiting for loading and can pass the local variable name in directly
  - Not very practical for actual applications
- How can we use external data sources?

External Data Loading

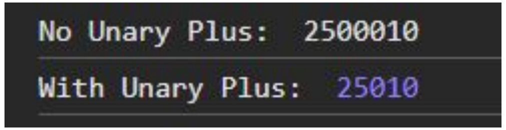
- D3 comes with some built-in methods to help with this!
  - This is for D3 v5, older versions of D3 may work differently!
  - .csv(), .json(), and .tsv()
  - Can be used for external files or URLS
- External files take time to download but JavaScript will try to run your script anyways! We need a way to make it wait until you have your data
  - Use .then() to make sure the rest of the method can't proceed before the data is done loading



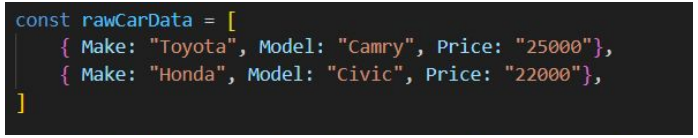
```
d3.csv("data.csv").then(data => {
    console.log(data);
})
```

Data Cleaning

- Values in CSV are imported as **strings**, D3 has no idea if a column should be a number, date, etc.
- Option 1: Unary Plus
  - Forces the string into a number before the mathematical operation

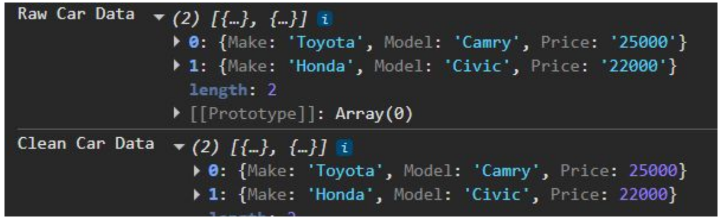


```
No Unary Plus:   2500010
With Unary Plus:  25010
```

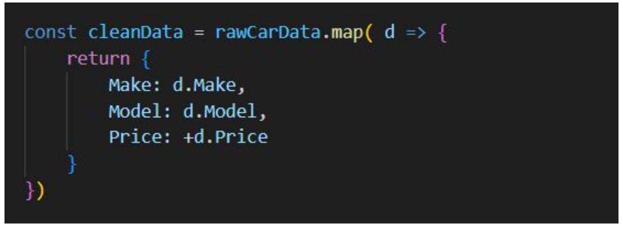


```
const rawCarData = [
    { Make: "Toyota", Model: "Camry", Price: "25000"},
    { Make: "Honda", Model: "Civic", Price: "22000"},
]
```

- Option 2: Row-Level Cleaning
  - Maps the data to the “cleaned” version



```
Raw Car Data   ▾ (2) [{…}, {…}] i
                 ▸ 0: {Make: 'Toyota', Model: 'Camry', Price: '25000'}
                 ▸ 1: {Make: 'Honda', Model: 'Civic', Price: '22000'}
                   length: 2
                 ▸ [[Prototype]]: Array(0)
Clean Car Data   ▾ (2) [{…}, {…}] i
                   ▸ 0: {Make: 'Toyota', Model: 'Camry', Price: 25000}
                   ▸ 1: {Make: 'Honda', Model: 'Civic', Price: 22000}
```



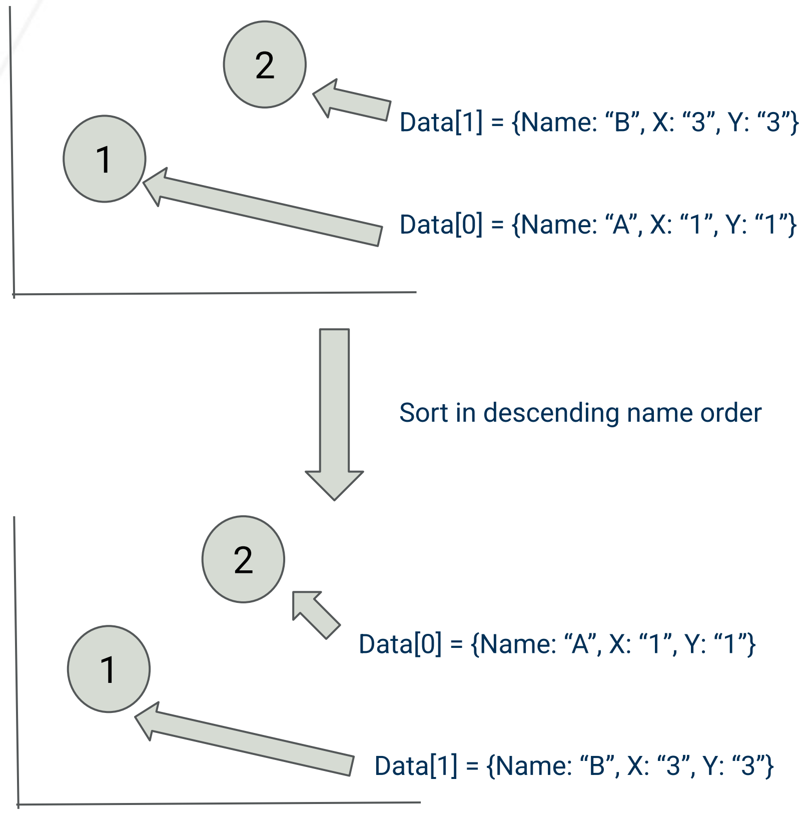
```
const cleanData = rawCarData.map( d => {
    return {
        Make: d.Make,
        Model: d.Model,
        Price: +d.Price
    }
})
```

Key Functions

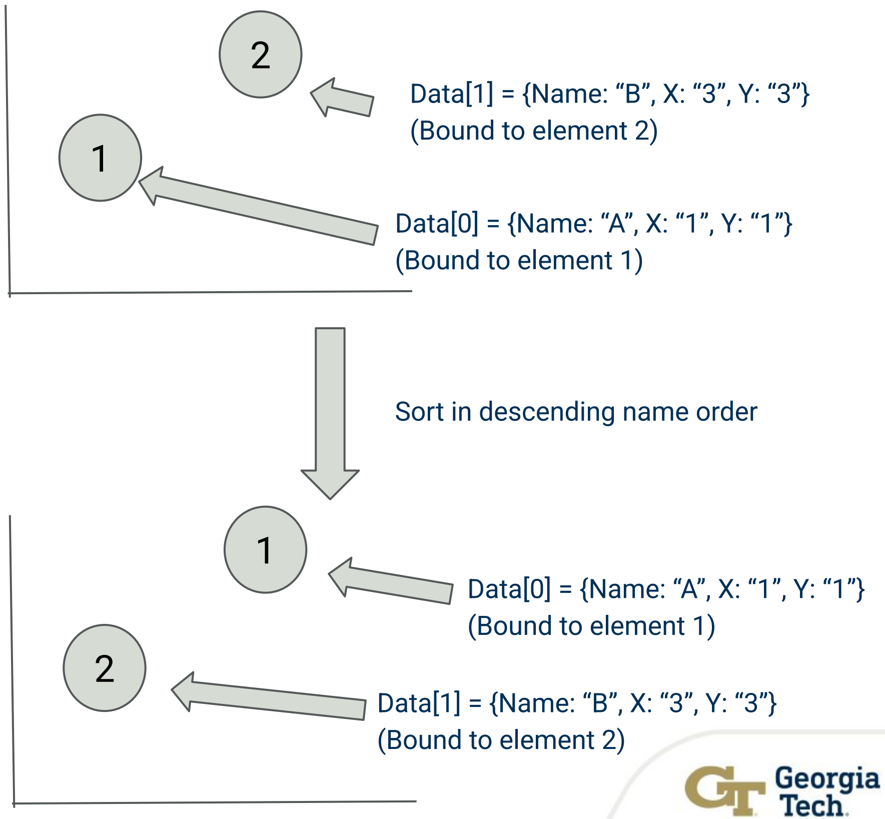
- By default, D3 pairs the 1st data point with the 1st element, and so on
- If you were to sort the data, D3 will just reassign the 1st element to the new 1st data point instead of move the original element to the new spot
- **The key function** gives an identifier to the data
    - This modifies D3's logic to match the identifiers to the elements
    - When the data index changes, it will tell that the data is already "bound" to an element, and instead move that originally bound element

Key Functions Cont.

Without Key Functions


2
1
Data[1] = {Name: “B”, X: “3”, Y: “3”}
Data[0] = {Name: “A”, X: “1”, Y: “1”}
Sort in descending name order
2
1
Data[0] = {Name: “A”, X: “1”, Y: “1”}
Data[1] = {Name: “B”, X: “3”, Y: “3”}

With Key Functions


2
1
Data[1] = {Name: “B”, X: “3”, Y: “3”}
(Bound to element 2)
Data[0] = {Name: “A”, X: “1”, Y: “1”}
(Bound to element 1)
Sort in descending name order
1
2
Data[0] = {Name: “A”, X: “1”, Y: “1”}
(Bound to element 1)
Data[1] = {Name: “B”, X: “3”, Y: “3”}
(Bound to element 2)