# CS 571 - Data Visualization & Exploration

## DOM and D3 Basics

Instructor: Hamza Elhamdadi

UMassAmherst

# Upcoming Dates

**Feb 28 (Tomorrow): Project Proposal Due**

**Homework 2 will be released Feb 28 (due Mar 14)**

From the HTML and CSS Lecture...

# The Document Object Model (DOM)

You might have noticed that an HTML document looks a lot like a tree.

The root of the tree is the <html> tag, and each node may have children that may contain nodes themselves.

The DOM is a programming interface for HTML documents

We will use “DOM” to mean the tree that is created by web browsers to represent the HTML document, as well as the API browsers provide to access this tree

# DOM Manipulation

# DOM Manipulation

We can use Javascript to manipulate the structure of our HTML code

To do this, we can use the global Javascript object: **document**

# Manipulating HTML using the DOM

Some useful functions:

- document.createElement()
- document.createTextNode()
- .appendChild()
- document.getElementById()

# Setting Attributes using the DOM

Two ways to do this:

- .setAttribute(“<attribute-name>”, “<attribute-value>”);
- .<attribute-name> = <attribute-value>;

# Javascript Events

Example events:

- onclick
- onload
- onmouseover
- onmouseout
- onchange
- onresize

Attribute can be set in HTML or in Javascript:

- 
- document.getElementById(“mainDiv”).onclick = func;

# D3 (Data-Driven Documents)

# D3 (Data-Driven Documents)

D3 is a Javascript library for manipulating the DOM based on data

It can be used to manipulate pure HTML, but most commonly we use it to manipulate SVG (to create visualizations!)

Link to D3 in your HTML header using the following script tag:

- <script src="https://d3js.org/d3.v7.js"></script>

Or, you can download the library from the D3 Website and include a local version

# D3 Selections

Like we did with vanilla DOM manipulation, we can select elements using **d3.select()**

For example:

- d3.select(“p”) selects the first p element
- d3.select(“#main”) selects the first element with id = “main”
- d3.select(“.hat”) selects the first element with class = “hat”

# D3 Selections (Multiple Elements)

To select all of the elements that match your selector, use **d3.selectAll()**

For example:

- d3.selectAll(“p”) selects all p elements
- d3.select(“.hat”) selects all elements with class = “hat”

Note: D3 is declarative (no loops needed)

# D3 and Data

The most important feature in D3 is the ability to map DOM elements with data.

To do this, we call the **data()** function on a selection

# enter()

If you have more data than DOM elements in your selection, you can use **.enter()** to create new DOM elements for the remaining data

For example (if there were only 3 rectangles in the DOM):

- d3.selectAll("rect").data([126, 61, 256, 71]).enter().append("rect")

would create a new rectangle for the fourth element (70) in our data.

# merge()

To avoid writing duplicate code for the existing elements and the elements we created using enter(), we can merge these two selections using the **merge()** function.

To do this, we need to create a variable that contains our selection mapped to data:

- let selection = d3.selectAll("rect").data([126, 61, 256, 71]);

Then we can use combine enter() and merge():

- selection = selection.enter().append("rect").merge(selection);

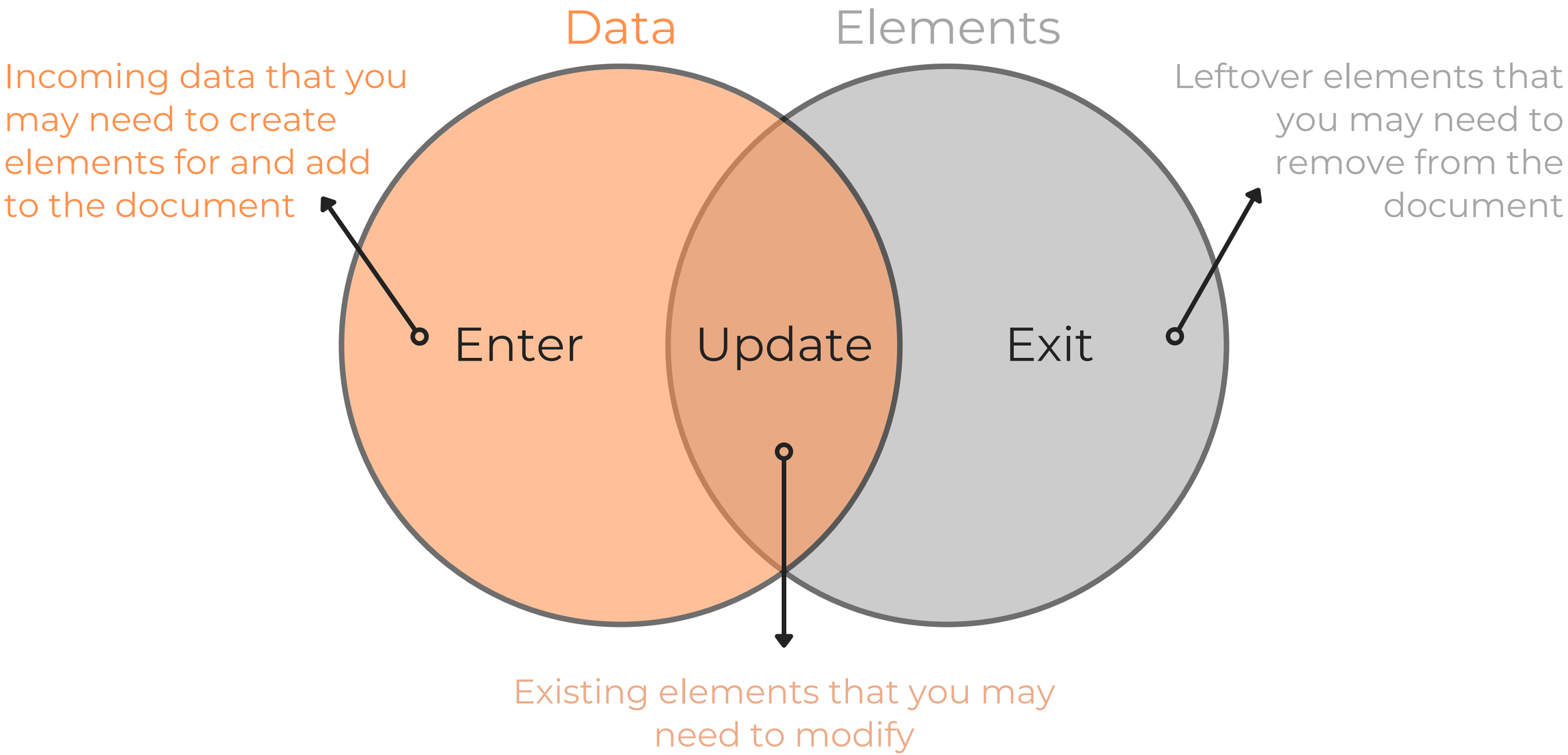
Then, apply attributes and styles to selection as normal

# exit()

If you have fewer data than DOM elements, you can use **.exit()** to define what happens to the remaining DOM elements.

For example, we could do the following to remove the remaining DOM elements:

- selection.exit().remove();


Data
Elements
Incoming data that you may need to create elements for and add to the document
Leftover elements that you may need to remove from the document
Enter
Update
Exit
Existing elements that you may need to modify

# join()

In D3 v5, the developers added a new way to handle selections and data: **selection.join()**

Two ways to use selection.join():

- selection.join("rect");
  - assumes you want to append entering elements and remove exiting elements
- selection.join( enter,  update,  exit );
  - allows you to specify what happens to entering elements, existing elements (update), and exiting elements

# D3 Transitions

Transitions are baked into D3! To specify them, use:

- **selection.transition()**

We can specify:

- .**duration(ms)** - how long to transition for
- .**delay(ms)** - how long to wait before transitioning

# D3 Paths

If you want to draw complex lines in d3, you can **append a “path”**

For example:

- d3.select("svg").append("path")
  .attr("d", "M 10 10 L 200 200 L 200 400 L 300 100 L 400 150");

# d3.line()

Instead of writing the path code manually, D3 provides us with **d3.line()**

First, let’s declare the points along the path as a object:

- let points = [{ x: 10, y: 10 }, { x: 200, y: 200 }, { x: 200, y: 400 }, { x: 300, y: 400 }, { x: 400, y: 140 }];

Then, we can declare our line function:

- let lineFn = d3.line().x( d => d.x ).y( d => d.y );

Finally, we specify points as our data and pass lineFn to the “d” attribute:

- svg.append("path").datum(points).attr("d", lineFn) ...

# Exercise

Instead of writing the path code manually, D3 provides us with **d3.line()**