Lab 5: D3 Enter, Update, Exit, and Filter

Jump to bottom

Chen, John T edited this page on Mar 9 · 4 revisions

Learning Objectives

After completing this lab you will be able to:

- Append new DOM elements with enter
- Update currently data-joined elements
- Remove old DOM elements with exit
- Write one function to update a chart
- Implement Filters

Prerequisites

- Download the corresponding lab from the code repo (either using git or downloading the folder from the code of this repo (in the Code tab above))
- You have **read Chapter 9** in D3 - Interactive Data Visualization for the Web by Scott Murray

Recommended Reading

- Three Little Circles by Mike Bostock
- Enter, Update, Exit by Christian Behrens

Additional Reading

- Understanding selectAll, data, enter, append sequence in D3.js
- Thinking with Joins by Mike Bostock
- Object Constancy by Mike Bostock
- General Update Pattern I by Mike Bostock
- General Update Pattern II by Mike Bostock
- Advanced D3: More on selections and data, scales, axis by A. Lex of U. of Utah

What to submit

1. You should have completed Activity 1, Activity 2, and Activity 3 (in each respective subfolder).

2. Rename your `lab5` folder to `LastName_FirstName_lab5`
3. Zip up `LastName_FirstName_lab5` as `LastName_FirstName_lab5.zip` and submit it to Canvas.

Grading

Your assignment will be graded on the following requirements:

- Correct implementation of the bar chart (visual components)
- Correct functionality of the categorical filter (dropdown); and slider

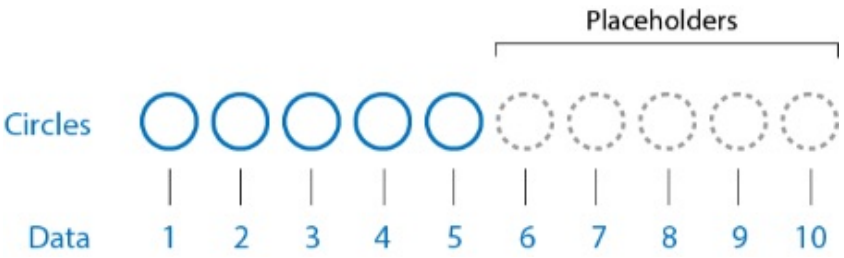
Tutorial 1: D3 Enter, Update, Exit Pattern

By now you have learned how to load external data and how to map it to visual elements like a bar chart, scatter plot, and line chart. But very often you have to deal with a continuous data stream rather than a static CSV file. Dynamic data often requires more sophisticated user interfaces that allow users to interact with the data (e.g. filter, sort, navigate).

Instead of removing and redrawing visualizations each time new data arrives, update only affected components and focus on loading times and smooth transitions. We will accomplish this by using the D3 update pattern (enter → update → exit).

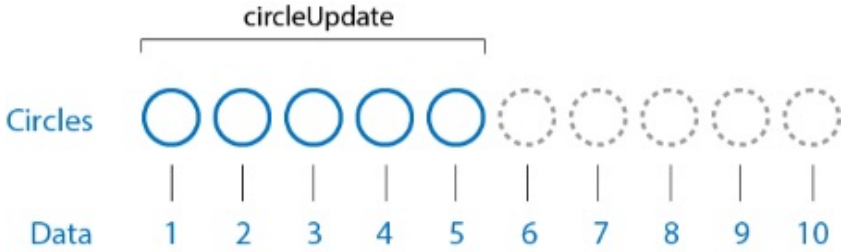
At a high level, enter, update and exit are just ways of selecting SVG elements on a SVG canvas. Let's consider an example. Let's say we already have 5 circles on the SVG canvas and we have an array with 10 elements (let's assume that the array is `[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]`). In the previous lab, we have learnt that we can bind the data to the circles using D3.

Visually, the following is what will happen after the binding. As we have more data elements than circles (we have 10 data elements and 5 circles), 5 placeholder circles are created.



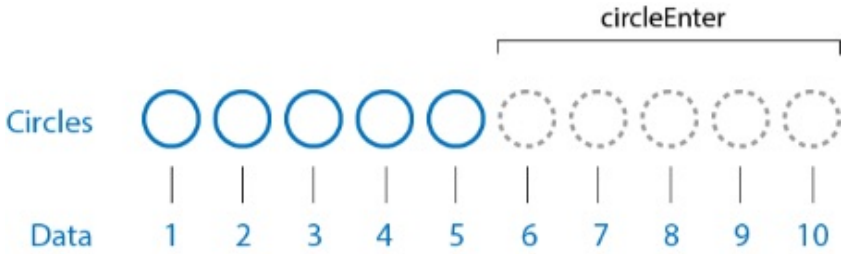
To select the 5 circles that have corresponding data points, you use the update selection. `circleUpdate` below stores the 5 circles that are bound with data points.

```
var circleUpdate = d3.select('svg').selectAll('circle')
        .data([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]);
```

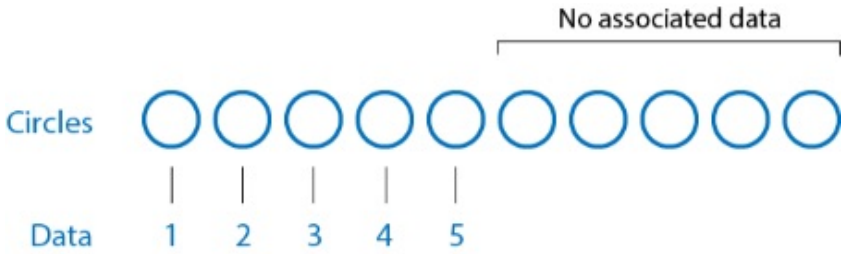


To select the 5 placeholder circles, you use the enter selection. `circleEnter` below stores the 5 placeholder circles. (You might have noticed that we've actually already used this pattern multiple times in previous labs)

```
var circleEnter = d3.select('svg').selectAll('circle')
        .data([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ])
        .enter();
```

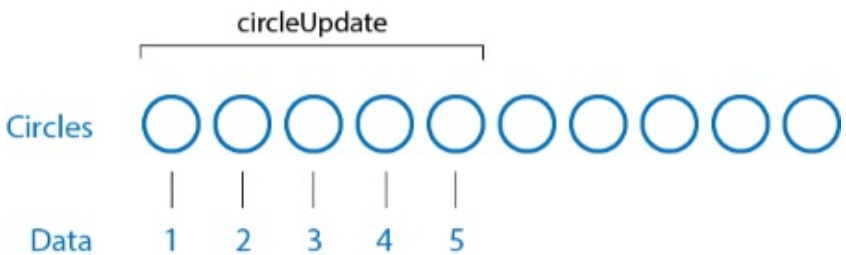


Now, let's consider another scenario. What if we have 10 circles on the SVG canvas and we have an array with 5 elements? (let's assume that the array is `[ 1, 2, 3, 4, 5 ]`) After data binding, the following will happen. 5 circles are bound with data values but the other 5 are not associated with any data.



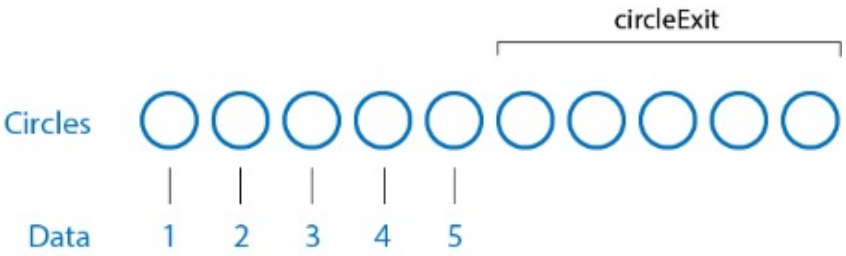
To select the 5 circles that have corresponding data points, you again use the update selection. `circleUpdate` below stores the 5 circles that are bound with data points.

```
var circleUpdate = d3.select('svg').selectAll('circle')
        .data([ 1, 2, 3, 4, 5 ]);
```



To select the 5 circles that are not associated with data points, you use the exit selection. `circleExit` below stores the 5 circles that are not associated with data points.

```
var circleExit = d3.select('svg').selectAll('circle')
        .data([ 1, 2, 3, 4, 5 ])
        .exit();
```



To summarize:

- **Enter** selects placeholder elements
- **Update** selects elements that are bounded with data
- **Exit** selects elements that are not bounded with data

Let's take a look at another example.

Tutorial 2: Enter, Update, and Exit for Letters

Let's work with some letters. We'll be learning the ABCs of D3's update pattern with... wouldn't you know it the ABCs. We are going to create grouped circles with text for each letter. You can follow the instructions to copy and paste the code to `main.js` in the `/lab5/tutorial2` folder.

First, we will bind the data to .letter `<g>` elements:

```
var letter = d3.select('svg').selectAll('.letter')
        .data(['A', 'B', 'C']);
```

The length of the dataset is 3 and we select all SVG elements with a classname of `letter` in the SVG. Remember from the last lab that `d3.select('svg')` is needed to specify the `parent` for this selection. That means, if there are 3 or more existing groups, the **enter selection** is empty, otherwise it contains placeholders for the missing elements.

Enter

The page is empty because we have not appended any groups yet. We can access the enter selection and append a new group for each placeholder with the following statement:

```
var letterEnter = letter.enter()
    .append('g')
```

```
        .attr('class', 'letter')
        .attr('transform', function(d,i) {
            return 'translate('+[i * 30 + 50, 50]+')';
        });
```

This will create spaced out `<g>` elements with the classname `letter`, but now we need to add the circles and text. This can be achieved by appending to `letterEnter`.

```
    letterEnter.append('circle')
        .attr('r', 10);

    letterEnter.append('text')
        .attr('y', 30)
        .text(function(d) {
            return d;
        });
```

Update

Now with data-bound elements in the SVG canvas we can use the `update` selection to change them. In our drawing function we call:

```
        var letterCircle = d3.select('svg').selectAll('.letter circle');
```

Which returns a selection of 3 circles - `.letter circle` is a css selector for all elements with a classname of `letter` and then selects any and all `circle` elements inside of the `.letter` element.

We can use this selection to update the already drawn circles:

```
        letterCircle.attr('r', 20);
```

Exit

Often you want to remove elements from the SVG. If someone filters the dataset you may want to remove existing elements. In this case, you have to use the exit selection. `exit` contains the leftover elements for which there is no corresponding data anymore.

We call the drawing function again with new data:

```
var letter = d3.select('svg').selectAll('.letter')
        .data(['A', 'B']);
```

All that's left to do, then, is to remove the exiting elements:

```
letter.exit().remove();
```

And now we have two letters:

Now if we want to add more data, we can use all three selections `enter`, `update`, `exit` in the same drawing function. This will update our circles based on any new data. Note that the `letterEnter.merge(letter)` creates the **update + enter selection** This is useful for changing styles and attributes on *new and old elements* at the same time.

```
function updateCircles(letters) {
    var letter = d3.select('svg').selectAll('.letter')
        .data(letters);

    var letterEnter = letter.enter()
        .append('g')
        .attr('class', 'letter');

    letterEnter.merge(letter)
        .attr('transform', function(d,i) {
            return 'translate('+[i * 60 + 50, 50]+')';
        });

    letterEnter.append('circle')
        .attr('r', 20);

    letterEnter.append('text')
        .attr('y', 30)
        .text(function(d) {
            return d;
        });
```

```
        letter.exit().remove();
    }

    updateCircles(['A', 'B', 'C']);
    updateCircles(['A', 'B']);
    updateCircles(['A', 'B', 'C', 'D', 'E', 'F']);
```

By default, the data join happens **by index**: the first element is bound to the first datum, and so on. Thus, either the enter or exit selection will be empty, or both. If there are more data than elements, the extra data are in the enter selection. And if there are fewer data than elements, the extra elements are in the exit selection.

Activity 1: Drawing a Bar Chart

> Reminder: Start an http server for this lab's directory. From command line call `python -m SimpleHTTPServer 8080` (for Python 2) or `python -m http.server 8080` (for Python 3).

For activities 1 - 3, you will be working with the same HTML/CSS/JavaScript code. All of it can be found in `/lab5/activities`. At the end of this activity, rename the `activities` folder as `activity_1`. Duplicate this folder and name it as `activity_2` and start there for the next activity.

During this lab, you will be working with the `letters_freq.csv` dataset. The dataset includes 26 rows. Each row corresponds to a letter in the English alphabet and the frequency that it is used. Here is a snippet of the data table:

letter	frequency
A	0.08167
B	0.01492
C	0.02782
D	0.04253

At the end of Activity 1 and 2, your vis should look and function like this:

As last time, the following steps should help scaffold your progress. However, you do not necessarily have to follow them.

1. How to structure your code for interaction

Similar to the previous week, we have already added structure to your `activities\main.js` code. There are a number of additions to the code:

- `onCategoryChanged` method - used to handle `change` events from the `select` input widget.
- Layout parameters for configuring the spacing of your bar chart. `barBand` can be used to space out your bars evenly. `chartG` is a group that has been positioned based on the `padding`. Add your bars to this group with `chartG.selectAll('.bar').data()`
- `d3.csv('letter_freq.csv', dataPreprocessor).then(...)` is included
- `updateChart(filterKey)` the method to be called for new data

Take some time to look through the template and read the comments.

2. Create a global data variable

Because you will need to access the loaded dataset in the `updateChart` method, you will need to create a global variable of the loaded dataset. **Hint:** declare the global variable `letters` as the loaded dataset within the `function(dataset) {...}` data callback method.

3. Create a width scale

Create a linear-scale for the `frequency` data attribute. You will want the output range of this scale to be `[0, chartWidth]`. **Again you might want this to be a global variable.**

4. Create the bar chart

Inside the `updateChart` method, use the `filteredLetters` array to create the `<rect>` elements and `<text>` elements that make up the bar chart.

5. Create the axes

Create the pair of axes as shown, using `.tickFormat()` to display ticks as percentages. **Note:** this should not be in the updateChart method or since the axes are static.

Your web page should look this now:

At this point, rename the `activities` folder as `activity_1`. Duplicate this folder and name it as `activity_2`. You will start there for the next activity.

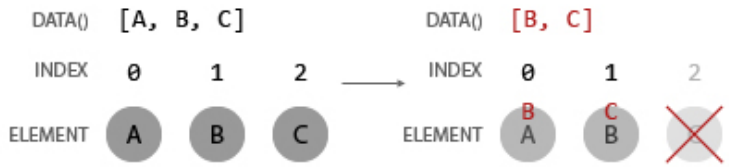
Tutorial 3: Key Function

Now that you understand update, enter, and exit selections, it's time to dig deeper into data joins (or data binding).

The default join is by index order, meaning the first data value is bound to the first DOM element in the selection, the second value is bound to the second element, and so on.

You can control precisely which datum is bound to which element by specifying a key function in the `selection.data()` method. For example, by using the identity function `function(d){ return d; }`, you can rebind the circles to new data while ensuring that existing circles are rebound to the same value in the new data, if any.

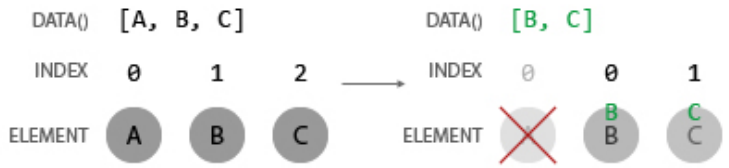
Take our letter circles example. Say we already have 3 circles on the canvas for `['A', 'B', 'C']`. And we want to update the diagram with the letters `['B', 'C']`. If we use the default index order then the following items will be removed with the `exit` selection:



The index will be used again as the default key to match the new data to the actual circles. There are three circles on the webpage and two items in the new dataset. Therefore, the last circle will be removed and the other two circles will be bound to the new data.

This is the simplest method of joining data and often sufficient. However, when the data and the elements are not in the same order, joining by index is insufficient. In this case, you can specify a key function as the second argument (callback function).

This key function takes a data point as input and returns a corresponding key: a string, such as a name, that uniquely identifies the data point. The objects stay constantly bound to their original data because of this key function:



Here's an example of a key function for our letters:

```
var letter = svg.selectAll('.letter')
    .data(['A', 'B'], function(d){
        return d;
    });
```

They function is the second input for the `.data()` method. And as you might have guessed it is optional. So when should you use they key function?

Key functions are needed when using transitions. They can also be useful for improving performance independent of transitions.

Use a key function whenever you want to follow graphical elements through animation and interaction: **filtering (adding or removing elements)**, reordering (sorting), switching dimensions within multivariate data, etc. If you forget to specify a key function, the default join-by-index can be misleading! Assist your viewers by maintaining object constancy.

In the following activity you will need to use the key function to retain object constancy in the bar chart.

Tutorial 4: Interaction via Event Listeners

We are using a `change` event from the `select` element to update our chart. An event acts as a "trigger," something that happens after page load to apply updates to our chart.

In JavaScript, events are happening all the time. Not exciting events, like huge parties, but really insignificant events like mouseover and click. Most of the time, these insignificant events go ignored. But if someone is listening, then the event will be heard, and can go down in posterity, or at least trigger some sort of DOM interaction.

An event listener is an anonymous function that listens for a specific event on a specific element or elements. In today's Activity, the `select` element has an attribute for an `onchange` listener. The listener listens for a change event. When that happens, the listener function is executed. You can put whatever code you want in between the brackets of the anonymous function:

```
    <select class="custom-select" id="categorySelect" onchange="onCategoryChanged()">
      <option selected value="all-letters">All Letters</option>
      <option value="only-consonants">Only Consonants</option>
      <option value="only-vowels">Only Vowels</option>
</select>
```

When a user selects a new value, `onCategoryChanged` is executed in `main.js`. From there we can access the newly set value with:

```
// Global function called when select element is changed
function onCategoryChanged() {
    var select = d3.select('#categorySelect').node();
    // Get current value of select element
    var category = select.options[select.selectedIndex].value;
    // Update chart with the selected category of letters
    updateChart(category);
}
```

We'll cover more on interactivity in the following Labs.

Tutorial 5: A Brief Introduction to Bootstrap

Before we move on to the final activity, a quick note on the Bootstrap framework that we are using to style the `select` input element.

Rather than coding from scratch, frameworks enable you to utilize ready made blocks of code to help you get started. They give you a solid foundation for what a typical web project requires and usually they are also flexible enough for customization.

We have chosen Bootstrap as an example open source HTML, JS and CSS framework. It is one of the most widely used frameworks, it is easy to understand and it provides a great documentation with many examples. The question whether a framework can be useful depends on the individual project and on the developer. Therefore, it is up to you to decide if you want to use it in your programming assignments.

Here is a summary of the main aspects of Bootstrap:

- Open source HTML, CSS, and JS framework
- Provides a base styling for common used HTML elements
- The grid system helps you to create multi-column and nested layouts, especially if your website should
- works on different devices
- Extensive list of pre-styled components (navigation, dropdown-menu, forms, tables, icons ...)
- Customizable: All CSS rules can be overridden by your own rules
- Compatible with the latest versions of all major browsers

In general, frameworks are very helpful when you need standard widgets or web page elements.

Activity 2: Updating with New Data

In this second activity you will make your bar chart interactive! You will need to use the **Enter, Update, Exit pattern** to achieve this. You will need to reformat your code in `updateChart`.

Again, the following steps should help, but are not required.

1. Create an update selection

You will first want to create an update selection of all the bars. Remember to use the key function.

2. Enter and append all new elements

Use your code from the previous activity to create the bars for each letter. You may need to reformat it as needed.

3. Exit and remove filtered bars

The final step is to use the `exit` selection to remove the bars that have been filtered out by the data.

At this point, your vis should look and function like this:

At the end of Activity 2, duplicate this folder and name it as `activity_3` . You will start there for the next activity.

Activity 3: Add a Cutoff Filter

For this activity, you need to add a cutoff filter to your bar chart above. Changing the cutoff (to a higher or lower value) and clicking “Filter Data” should add or remove bars to show all the bars that that have frequency greater than or equal to the cutoff value. Place the cutoff input form and filter button below your dropdown. Note that the dropdown needs to also still function properly.

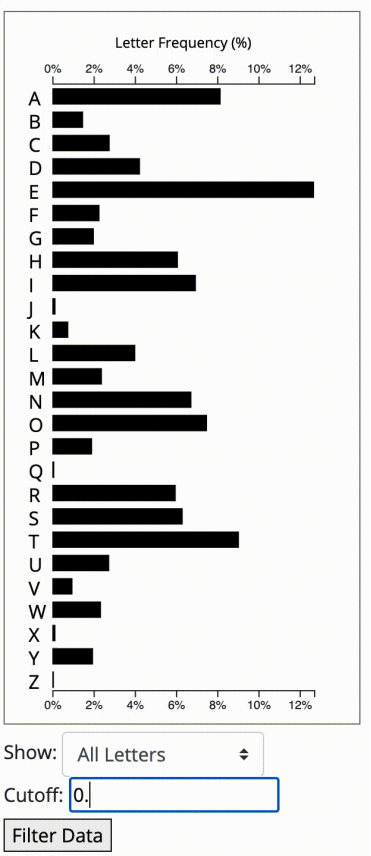
You can copy this code to add an input form to your html:

```
<form name="cutoff">
      <label for="cutoffFilter" class="show">Cutoff:</label>
      <input type="text" id="cutoff">
</form>
```

This is an example of how to append a button:

```
var main = document.getElementById('main');
d3.select(main)
        .append('p')
        .append('button')
        .style("border", "1px solid black")
        .text('Filter Data')
        .on('click', function() {
            // Add code here
        });
```

This is what it should look like:



At this point, you should have three subfolders, `activity_1`, `activity_2`, and `activity_3`, in your `lab5` folder.

What to Turn In

- Complete Activity 1, 2, and 3 and submit your code for the `/lab5/activities` folder
- Please name the file as "lastname_firstname_lab5.zip"

This lab was based on the following material:

- Hanspeter Pfister's CS171 Lab Material (Harvard)

- D3 - Interactive Data Visualization for the Web by Scott Murray
- Enter, Update, Exit by Christian Behrens
- Three Little Circles by Mike Bostock
- Object Constancy by Mike Bostock

Clone this wiki locally

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