D3 Reference Guide

Background

What is D3?

D3 stands for Data-Driven Documents; it is a JavaScript library allowing developers to bind arbitrary data to a **D**ocument **O**bject **M**odel. Powerful, data-driven visualizations are created using HTML, CSS, and SVG.

Selecting Elements

Selecting different DOM elements requires the use of `.select` and `.selectAll`.

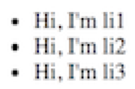
```
d3.select("ul").selectAll("li")
```

In the above code, we are using D3 to select all `li` elements inside of a `ul` tag in an HTML document. After execution, the code returns a selection object containing the `li` elements from the DOM.

`.each()`

Once elements have been selected, different methods can be chained. The `.each()` method allows us to call a function on each element within the object. This is similar to using `.map()` or `.forEach()` to iterate through an array.

Take this basic webpage, for instance:



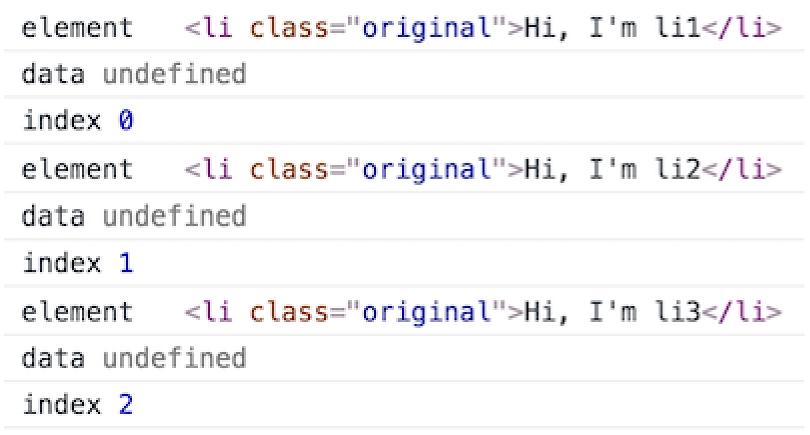
- Hi, I'm li1
- Hi, I'm li2
- Hi, I'm li3

We would use the following code to select the list items and iterate through them:

```
d3.select("ul").selectAll("li")
  .each(function (d, i) {
    console.log("element", this);
    console.log("data", d);
    console.log("index", i);
  });
```

Now let's break down the code.

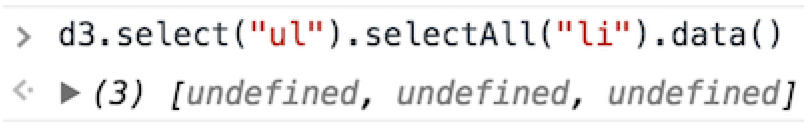
- The `.each()` method includes the `(d, i)` parameters; `d` is the data property of the element and `i` is the index in the selector object.
- The keyword `this` is used to log the element.
- Each `console.log` returns the element, data (undefined because we have not bound any to the element yet), and the index position of the element.



```
element   <li class="original">Hi, I'm li1</li>
data undefined
index 0
element   <li class="original">Hi, I'm li2</li>
data undefined
index 1
element   <li class="original">Hi, I'm li3</li>
data undefined
index 2
```

`.data()`

Using the same simple webpage as the example above, we can chain `.data()` to the selector.



```
> d3.select("ul").selectAll("li").data()
<· ▶ (3) [undefined, undefined, undefined]
```

In the above image, the current data property is undefined because there is none. Let's change that by adding an array to the mix.

```
var arr = [50, 55];
d3.select("ul").selectAll("li").data(arr)
d3.select("ul").selectAll("li").data()
> (3) [50, 55, undefined]
```

Now we have an array, but its length is smaller than the number of elements, resulting in an undefined data property.

```
var arr = [50, 55, 53, 56, 68];
d3.select("ul").selectAll("li").data(arr)
d3.select("ul").selectAll("li").data()
> (3) [50, 55, 53]
```

In the above example, the length of the array is more than the number elements in the selector. In this case, the extra items are ignored.

If we provide a new array that is again less than the number of elements, only the new elements are updated.

```
var arr = [1, 2];
d3.select("ul").selectAll("li").data(arr)
d3.select("ul").selectAll("li").data()
> (3) [1, 2, 53]
```

`.text()`

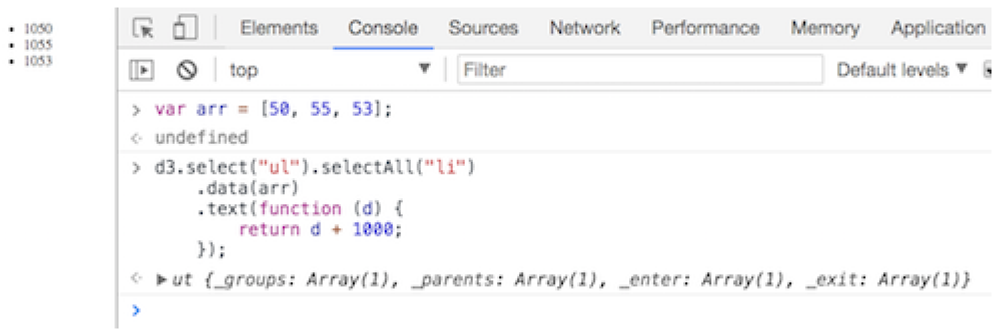
After data has been bound to an element, there are a variety of functions available to manipulate the elements.

```
var arr = [50, 55, 53];

d3.select("ul").selectAll("li")
  .data(arr)
  .text(function (d) {
    return d;
  });

> ut {_groups: Array(1), _parents: Array(1), _enter: Array(1), _exit:
Array(1)}
```

Above, we have chained `.text` with a callback function. This callback function is called with each element in the selection. This also changes the element text on our webpage:

- 50
- 55
- 53

The element text can be further modified before assigning new text:

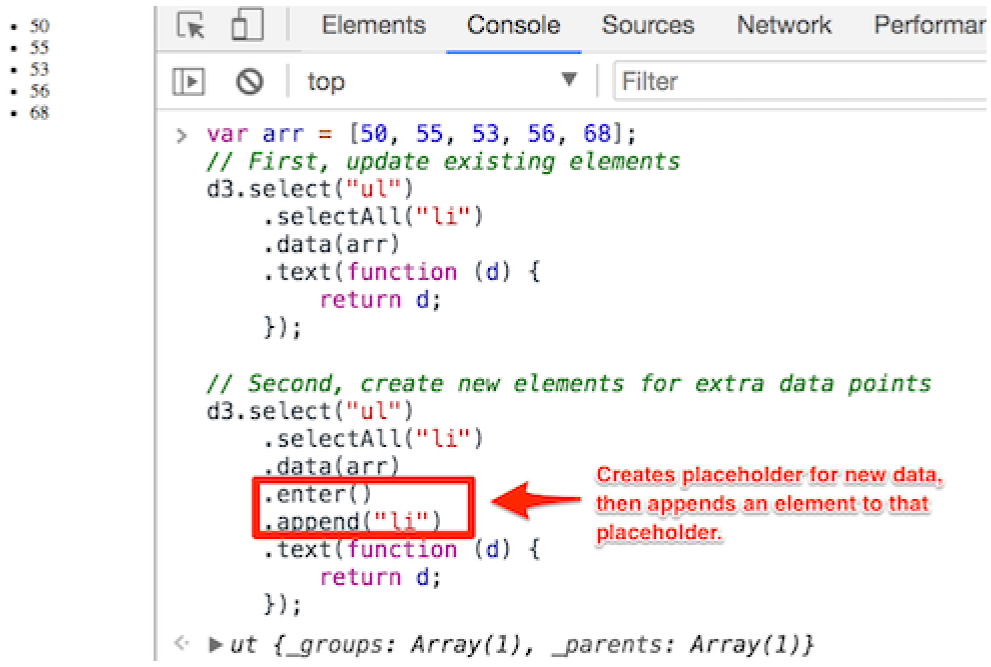


```
> var arr = [50, 55, 53];
< undefined
> d3.select("ul").selectAll("li")
      .data(arr)
      .text(function (d) {
          return d + 1000;
      });
< ▶ut {_groups: Array(1), _parents: Array(1), _enter: Array(1), _exit: Array(1)}
```

• 1050
• 1055
• 1053
Elements Console Sources Network Performance Memory Application
top
Filter
Default levels

`.enter()` & `.append()`

When it comes to handling additional data such as the extra array values above, using `.enter()` will create a sub-selection for data that hasn't been mapped to an element yet. `.append()` will then pair an element to the sub-selection.

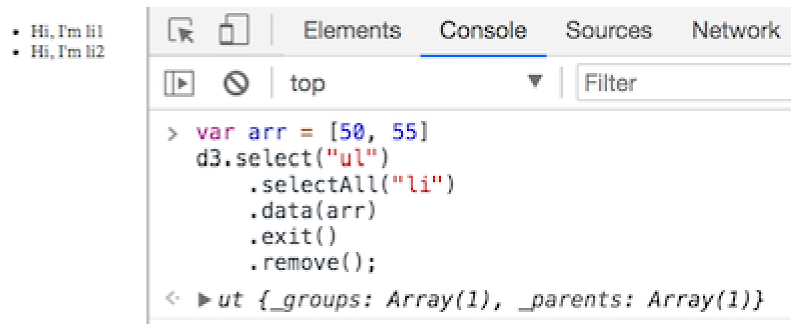


```
> var arr = [50, 55, 53, 56, 68];
  // First, update existing elements
  d3.select("ul")
      .selectAll("li")
      .data(arr)
      .text(function (d) {
          return d;
      });

  // Second, create new elements for extra data points
  d3.select("ul")
      .selectAll("li")
      .data(arr)
      .enter()
      .append("li")
      .text(function (d) {
          return d;
      });
< ▶ut {_groups: Array(1), _parents: Array(1)}
```

• 50
• 55
• 53
• 56
• 68
Elements Console Sources Network Performar
top
Filter
Creates placeholder for new data, then appends an element to that placeholder.

`.exit()` & `.remove()`

When removing an element based on the number of data properties, `.exit()` will create a selection of the surplus and `.remove()` will remove them from the DOM.

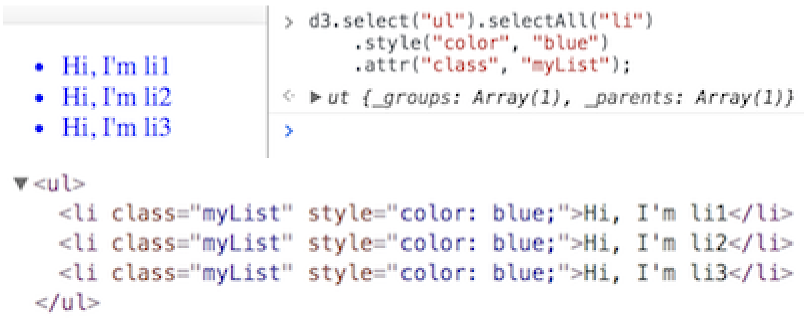


- Hi, I'm li1
- Hi, I'm li2

```
> var arr = [50, 55]
  d3.select("ul")
      .selectAll("li")
      .data(arr)
      .exit()
      .remove();
< ▶ut {_groups: Array(1), _parents: Array(1)}
```

`.attr()` & `.style()`

Attributes and styling can be applied with `.attr()` and `.style()`.



- Hi, I'm li1
- Hi, I'm li2
- Hi, I'm li3

```
> d3.select("ul").selectAll("li")
      .style("color", "blue")
      .attr("class", "myList");
< ▶ut {_groups: Array(1), _parents: Array(1)}
>
```

```
▼<ul>
   <li class="myList" style="color: blue;">Hi, I'm li1</li>
   <li class="myList" style="color: blue;">Hi, I'm li2</li>
   <li class="myList" style="color: blue;">Hi, I'm li3</li>
 </ul>
```

In the above code, we selected the `li` elements and applied a blue color to the text. Additionally, with `.attr()`, we added a class to each `li` element.