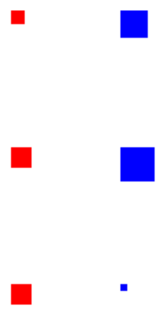
Lab 3 hints

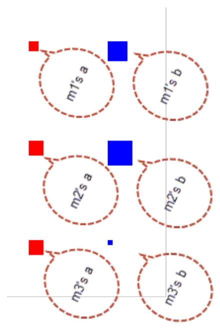
If you're still struggling with the last piece of the assignment, let me suggest you investigate the use of the SVG "g" tag. This is a grouping operator that doesn't actually do anything (much like a div… it just holds other stuff).

So let's say I have a really dumb dataset:

```
dataset = [ {"a":10,"b":20},  //m1
            {"a":15,"b":25},  //m2
            {"a":15,"b":5}];  //m3
```

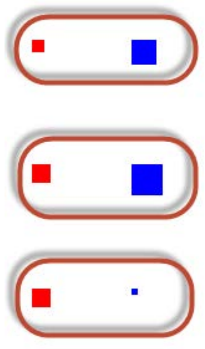
So basically 3 elements (m1, m2, m3), with two variables a and b. For whatever reason, I want a visualization that looks like this:





The first column (the red elements) are squares, one for each piece of data. Specifically, the red square encodes the 'a' values in the radius of the circle. So the width/height of the upper left circle is 10 (m1's "a"), the next down is 15 (m2's "a"), and the one after that is also 15 (m3's "a"). The second column—the blue squares—is the same for the "b" values. So what we have in the end is the columns representing first a and then b, and the rows representing each of the data elements.

Practically, one way to build this is to use the same grouping as we have in our data. Something like this:



Basically wrapping each pair of red/blue (a/b) circles into a group (denoted by the rectangle, which we won't actually draw).

In SVG it would look something like this:

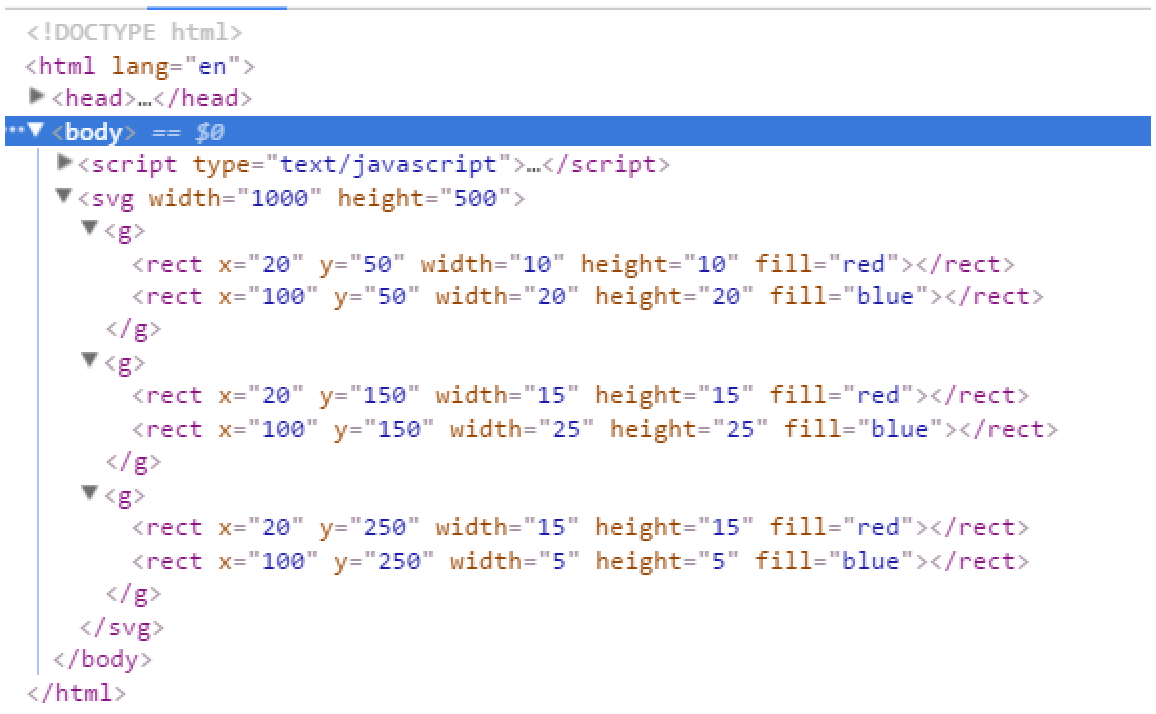
```
<svg>
<g> <rect …> <rect …>  </g>  ←group for m1 with two rectangles
<g> <rect … > <rect …> </g> ←group for m2 with two rectangles
<g> <rect …> <rect …> </g> ←group for m3 with two rectangles
</svg>
```

The nice thing about this structure is that we can break apart the d3 process into: (1) create a new g element for every piece of data, (2) to each of the g's add a new square for the "a" values, and (3) add a new square for each of the "b" values. The appeal is that we only need to call the whole "enter()" stuff once. We are creating groups and then we "append" the squares to those groups. The order that we do this is important. Notice in the code that we are first creating all the groups and then drawing all the 'a' squares (the red ones) and only then drawing all the blue ones (the 'b' squares). Our code would look something like this:

```
       // make the svg
        var svg = d3.select("body").append("svg")
                                  .attr("width",1000).attr("height",500);

        // dataset
        dataset = [ {"a":10,"b":20}, //m1
                     {"a":15,"b":25}, //m2
                     {"a":15,"b":5}]; //m3

        // create a grouping operator, one for each
        var groups = svg.selectAll("g")// grab all g's (we have none)
                           .data(dataset) // loop over dataset
                           .enter() // this will make us as many "g's" as we have data
                           .append("g");

        // CREATE THE RED SQUARE
        // into every group, we're going to first draw
        // a square for the "a" dimension. Notice that we don't
        // have to "enter" anymore. There are 3 pieces of data
        // and we previously created 3 g's.
        groups
                     .append("rect")
                     ~~.data(dataset) // we don't actually need this line~~
                     .attr("x",20) // always in the same column, but move the y:
                     .attr("y", function(d,i) {return ((i*100) + 50); })
                     .attr("width",function (d) {return d['a'];})
                     .attr("height",function (d) {return d['a'];})
                     .attr("fill","red");

        // CREATE THE BLUE SQUARE
        groups
                     .append("rect")
                     ~~.data(dataset) // we don't actually need this line~~
                     .attr("x", 100) // always in the same column, but move the y:
                     .attr("y", function(d,i) {return ((i*100) + 50); })
                     .attr("width",function (d) {return d['b'];})
                     .attr("height",function (d) {return d['b'];})
                     .attr("fill","blue");
```

The other cool thing about this approach is that D3 will propagate your data to nested objects. Because we've bound "dataset" to our "g" objects, the data will get pushed to the circles that are contained inside those "g" objects. So strictly speaking, we can get rid of the lines in red and it would still work.

The SVG for this looks like:



```
<!DOCTYPE html>
<html lang="en">
 ▶<head>…</head>
…▼<body> == $0
   ▶<script type="text/javascript">…</script>
   ▼<svg width="1000" height="500">
     ▼<g>
         <rect x="20" y="50" width="10" height="10" fill="red"></rect>
         <rect x="100" y="50" width="20" height="20" fill="blue"></rect>
       </g>
     ▼<g>
         <rect x="20" y="150" width="15" height="15" fill="red"></rect>
         <rect x="100" y="150" width="25" height="25" fill="blue"></rect>
       </g>
     ▼<g>
         <rect x="20" y="250" width="15" height="15" fill="red"></rect>
         <rect x="100" y="250" width="5" height="5" fill="blue"></rect>
       </g>
     </svg>
   </body>
</html>
```