Object constancy -- switch to barChart_exit

Each bar in a bar chart represents a piece of data. What happens to a bar if it's data disappears (a top ten chart, where one country falls off the list?)

```
d3 .csv (' . / countryData_topten .csv' ,  function ( dataIn ) {…
```

Object constancy is a principle of visualization that states that each object (bar in a bar chart) should be related to one unique piece of data.

Ensuring object constancy requires use of a **key function**. It pairs each bound DOM element to the key property specified in the function:

```
 svg.selectAll('rect' )
       .data ( pointData ,  function ( d ) { return d. countryCode } )
```

Change in both the original data bind and the update function.

The “Exit” in enter, exit, update:

When your data is bound to a DOM element using a key function, you can’t just rebind a new array to the same element if the list of keys isn’t the same; you’d have an orphaned bar left over.

Instead, you need to add an .exit() function to the .enter() and update functions that we’ve been using.

First, move the data bind into the drawPoints( ) function. Separate the selection and data bind from the rest, and store it in a variable:

```
var rects = svg.selectAll('.bars')
     .data(pointData, function(d){return d.countryCode;});
```

Next, get rid of any bars that are no longer needed:

```
rects .exit ( )
     .remove ( );
```

Enter, exit, update:

Update the remaining bars, just like before:

```
rects
    .attr ( 'x‘ , function ( d ) { …
```

And then add in any new bars needed:

```
rects
    .enter ( )
    .append ( 'rect‘ )
    .attr ( 'class‘ ,‘bars‘ )
    .attr ( 'fill‘ , "slategray“ )
    .attr ( 'x‘ , function(d){ …
```

Update scales to match new data:

Because the contents of your data array are changing, you also need to reset the x axis domain each time the data updates (otherwise your new bars don’t have a position on the x axis)

Add a class to x axis in d3.csv function

Move scaleX domain def’n to drawPoints( )

Select the axis group by its class, and call the axis function

```
d3 .selectAll ( ' .xaxis ‘ )
      .call ( d3 .axisBottom ( scaleX ) );
```

Add transitions!

Add a transition to the update pattern to make the object constancy more noticeable:

```
rects
	.transition ( )
	.duration ( 200 )
	.attr ( 'x' , function ( d ) {….
```

Linked charts

Use bootstrap to structure page

www.layoutit.com

Make a container with two columns in it, download the HTML, plug into index.html code.

Move svg inside of bootstrap page

Give the col-md-6 columns a class svgBox, and style it in the CSS:

```
.svgBox {

    height : 400px;
    width : 100%;

}
```

Update the svg attributes to 100% for both width and height

Fix chart drawing to fit

Console.log width

Add an id “svg1” to the svg.

Switch over to getElement to read in properties:

```
var width = document .getElementById ( 'svg1‘ ) .clientWidth;
var height = document .getElementById ( 'svg1‘ ) .clientHeight;
```

Adjust scale values to calculate based on svg size:

```
var scaleX = d3.scaleBand()
                    .rangeRound([0, width-2*marginLeft]).padding(0.1);
var scaleY = d3.scaleLinear().range([height-2*marginTop, 0]);
```

Replace “400” values in bar chart drawing and update, and x axis translate functions as well

Add a second SVG

Duplicate the SVG code in the second bootstrap column (remember to update id!)

In this case, I expect SVG1 and SVG2 to have the same width and height, and I want the padding to be equal for both. So, leave them alone for now.

Grab the second svg, and save it to a variable in JS. Also update the selection tag to an id for both svg1 and svg2:

```
var svg2 = d3 .select (‘#svg2‘ )
    .append ( 'g‘ )
    .attr ( 'transform‘ ,  'translate (' + marginLeft + ',' + marginTop + ')‘ );
```

Draw a bar chart on the second SVG

Copy and paste the code for selection and drawing inside of drawPoints(). Update the selection to grab svg2 instead of svg1.

Plotting different things: set up the Y axis

Now, we want the second chart to share and x axis but have a different y.
Declare a new Y scale:, and update axis drawing calls for svg2:

```
var scaleY2 = d3 .scaleLinear ( ) .range ( [ height-2*marginTop, 0 ] );


 svg2 .append ( "g“ )
       .attr ( 'class‘ ,  'yaxis2‘ )
       .call ( d3.axisLeft ( scaleY2 ) );

 scaleY2. domain( [ 0 , d3.max(
                pointData.map(function (d) { return +d.totalPop}))
]);

d3 .selectAll ( '.yaxis2‘ )
       .call ( d3 .axisLeft ( scaleY2 ) );
```

Update rects2 drawings to use scaleY2

Plotting different things: change Y axis variable

```
scaleY2 .domain ( [ 0 , d3.max ( pointData.map ( function ( d ) {
                return +d.caloriesPerCap }
) ) ] );
```

Update drawing code to match, anywhere that you use scaleY2:

```
rects2
      .transition ( )
      .duration ( 200 )
      .attr ( 'x‘ , function ( d ) {
         return scaleX ( d.countryCode ) ;
      })
      .attr ( 'y‘ , function ( d ) {
         return scaleY2 ( d.caloriesPerCap );
```

Linking selection: create IDs

We now have two charts with shared x axes and different ys. What if we want to highlight a bar for the same country in both charts?

Set a new id attribute in the enter( ) section of the drawing, for both SVGs:

```
 rects
       .enter ( )
       .append ( 'rect‘ )
       .attr ( 'class', 'bars‘ )
       . attr ( 'id‘,  function ( d ) { return d .countryCode; } )
```

Linking selection: add mouse behavior

In the .enter( ) section of the code, add mouse highlighting behavior, like we did before:

```
.on( 'mouseover‘ ,  function ( d ) {
          d3.select ( this ) .attr ( 'fill‘ , ‘purple‘ )
})
.on ( 'mouseout‘ , function ( d ) {
          d3 .select ( this ) .attr ( 'fill‘ , 'slategray‘ )
})
```

Do this for both SVG1 and SVG2

Linking selection: one chart affects the other

Really, when we highlight one bar, we'd like it to update both charts. To do that, we use the ids:

First, save the id for my current bar selection:

```
currentID = d3.select ( this ) .attr ( 'id' );
```

Then, use that to select a bar in SVG2 and change its properties:

```
svg2 .selectAll ( '#' + currentID ) .attr ( 'fill' , 'green' )
```

Update the mouseout function to do the same, and apply to both charts.