Javascript + D3 II

More D3 + Reusable Charts Pattern
Scales

- Functions that map from an input domain to an output range
- Specify dataspace values instead of pixel space (i.e. converts data vals to pixel vals)

```javascript
.attr("y", function(d) {
  return h - (d * 4) + 14;
});
```
Scales

```javascript
var dataset = [ 5, 10, 13, 19, 21, 25, 22, 18, 15, 13, 11, 12, 15, 20, 18, 17, 16, 18, 23, 25 ];

• Want to create bars for this data on an svg canvas that is 50px tall
  • Domain: 0..25
  • Range: 0..50

varyscale = d3.scale.linear().domain([0,25]).range([0,h]);
```
var yscale = d3.scale.linear()
  .domain([0, d3.max(dataset, function(d) {
    return d[1];
}))
  .range([h, 0]);

• What if data was an array of points...how would you set the range?

dataset = [ [23, 12], [14, 35], [2, 55]]; 
d3.max(dataset, function(3)  
  { return d[1]; })}
Scale Variations

- Quantitative: linear, log, pow, etc..
- Ordinal
  
  ```javascript
  data = [ "A", "B", "C" ];

  var x = d3.scale.ordinal()
    .rangeRoundBands([0, width], .1);

  x.domain(data);

  ///Use it to create an axis
  var xAxis = d3.svg.axis()
    .scale(x).orient("bottom");

  svg.append("g")
    .attr("class", "x axis")
    .call(xAxis);
  ```
A visual representation of a scale

```javascript
var yAxis = d3.svg.axis()
  .scale(yScale)
  .orient("left");
```

Now, attach it to the chart

```javascript
svg.append("g").call(yAxis)
```
Interaction

• `selection.on(event type, listener)`
  – event type
    • “mouseover”
    • “mouseout”
    • etc.
    • For a list:
  – listener
    • user defined function
    • has access to d, i
Reusable Charts

```javascript
function chart(width, height) {
    // generate chart here,
    // using `width` and `height`
}
```

• User must store and manage width, height, etc. separate from the chart!
• Will have to pass them in whenever an update is needed
function chart(config) {
    // generate chart here,
    // using `config.width` and `config.height`
}

• Better...but must still manage config object separate from the chart
• Really want to ‘bind’ the config object to the chart so don’t have to manage it!
• A **closure** is a function that has access to the **context** (or outer scope) in which it was created

• The pattern is as follows:
  – Create the context you want to attach to the function
  – Call a ‘constructor’ that returns the function
  – You now have a handle to the function and it’s context!
Closure attempt #1

function chart(config) {
    return function() {
        // generate chart here, using
        // `config.width` and `config.height`
    };
}

var myChart = chart({width: 720, height: 80});

• Not bad, however the ‘config’ is trapped in the closure and I cannot edit it!
Closure Attempt #2

• Use the closure to bind the configuration parameters
• Use getters/setters to update them
• How do we allow the getters/setters to access the closure context?
  – Make sure they are created when it exists...or that they can access a function that can access them!
See closureDemo3.js