CS 171

Lab 4

Learning Objectives

- Be able to differentiate between *scales* and *axes*
- Understand the concept of *input domains* and *output ranges*
- Know how to create scale functions in D3
- Know how to group and transform SVG elements
- Know how to use scales to create axes in D3

Prerequisites

- You have read and **programmed** along with chapter 7 and 8 (until page 130) in *D3 - Interactive Data Visualization for the Web*.
- You have successfully filled in the pre-quiz for the fourth lab.

Last week you created your first D3 visualizations. You added content to the DOM, mapped datasets to visual elements on the webpage and defined dynamic, data-dependent properties. This week we will focus on Scales and Axes to create visualizations that adapt dynamically to input.

Scales

Until now, when creating a D3 visualization, we used only *x* and *y* values that corresponded directly to pixel measurements on the screen, within a pre-defined SVG drawing area. That is not very flexible and only feasible for static data. What if our data attributes are suddenly doubled? We can not increase the size of the chart every time a value increases. At some point, the user might have to scroll through a simple bar chart to get all the information.

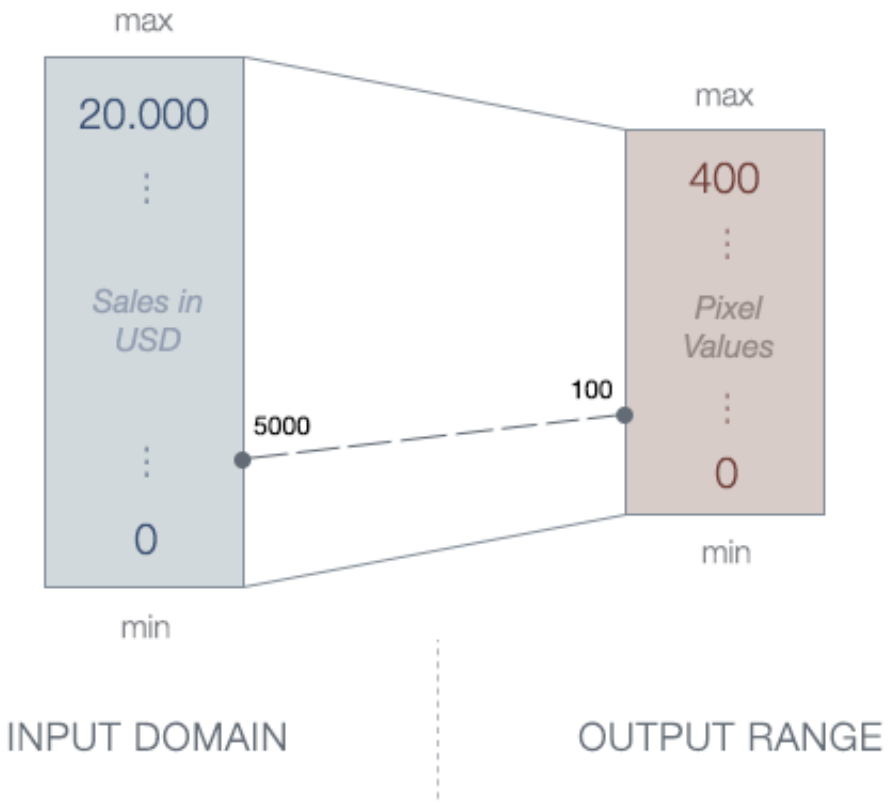
→ This is where ***scales*** come in. We specify a fixed SVG drawing space in our webpage and scale the data to fit in the dedicated area.

> D3 provides built-in methods for many different scales: linear, ordinal, logarithmic, square root etc. Most of

> the time you will use *linear scale functions* , so we will focus on learning this type of scale.
>
> You can read more about D3 scales here: https://github.com/mbostock/d3/wiki/Scales

"Scales are functions that map from an input domain to an output range." (Mike Bostock)

Example: We want to visualize the monthly sales of an ice cream store. The input data are numbers between 0 and 20,000 USD and the maximum height of the chart is 400px. We take an input inverval (called ***Domain***) and transform it into a new output interval (called ***Range***).



We could transform the numbers from one domain into the other manually but what if the sales rise above 20.000 and the interval changes? That means a lot of manual work. Thankfully, we can use D3's built-in scaling methods to do this automatically.

Scaling in D3

D3 provides scale functions to convert the *input domain* to an *output range*. We can specify the domain and range by using *method chaining syntax*:

```
// Creating a scale function
var iceCreamScale = d3.scale.linear()
    .domain([0, 20000])
    .range([0, 400]);

// Call the function and pass an input value
iceCreamScale(5000);    // Returns: 100
```

This was pretty easy, because we already knew the max value of the data. What if we load data from an external source and don't know the data range the data is going to be in? Instead of specifying fixed values for the domain, we can use the convenient array functions `d3.min()`, `d3.max()` or `d3.extent()`.

```
var quarterlyReport = [
    { month: "May", sales: 6900 },
    { month: "June", sales: 14240 },
    { month: "July", sales: 25000 },
    { month: "August", sales: 17500 }
];

// Returns the maximum value in a given array (= 25000)
var max = d3.max(quarterlyReport, function(d) {
    return d.sales;
});

// Returns the minimum value in a given array (= 6900)
var min = d3.min(quarterlyReport, function(d) {
    return d.sales;
});

// Returns the min. and max. value in a given array (= [6900,25000])
var extent = d3.extent(quarterlyReport, function(d) {
    return d.sales;
});
```

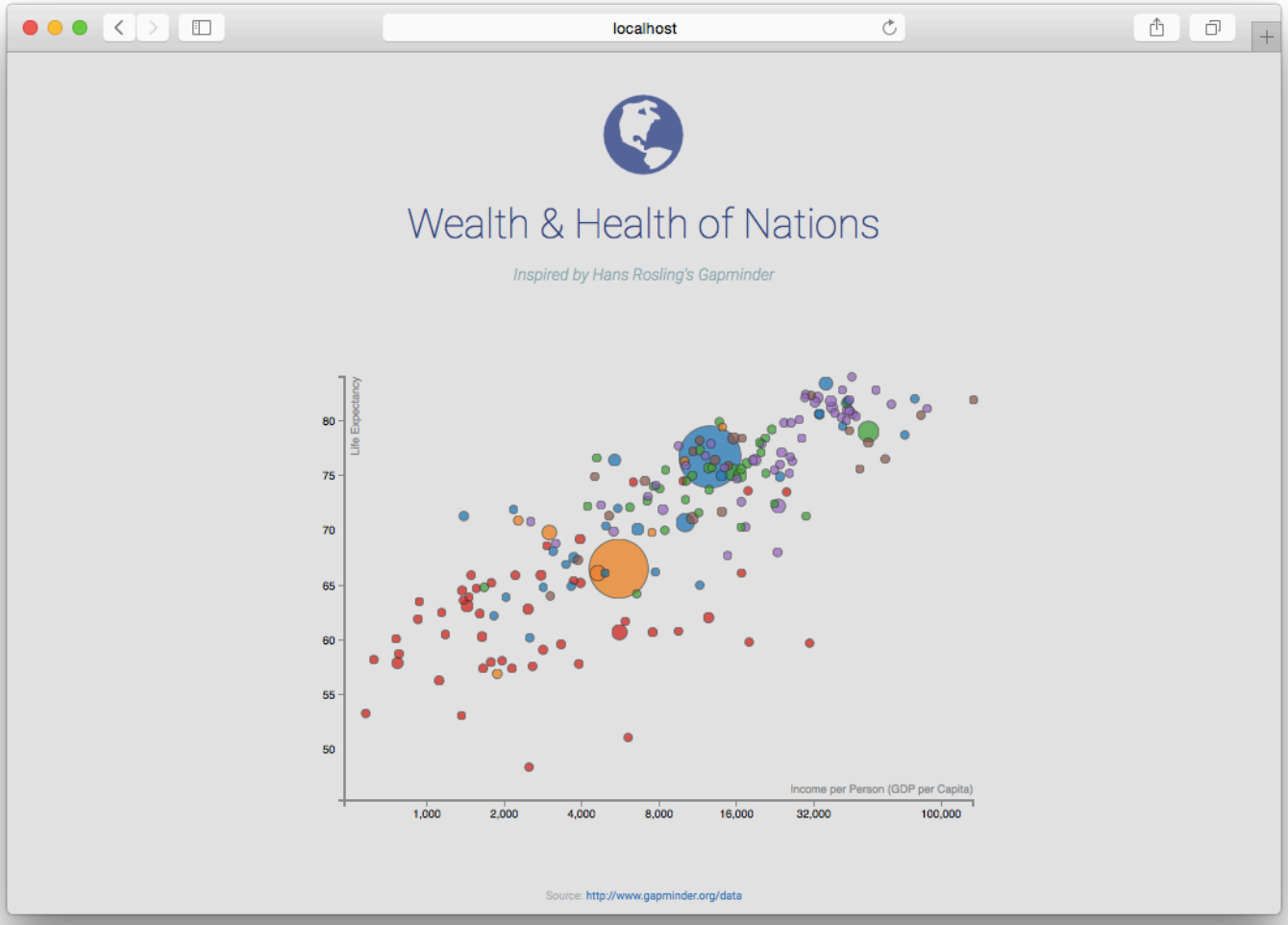
Activity I

Health & Wealth of Nations

In this lab you will work on a scatterplot with flexible scales and axes. You will compare the lifetime expectancy

(health) against the GDP per capita (wealth) of 188 countries. All activities in this lab are building on each other, so it is very important that you complete all tasks in order.

Result:



Data: `Country` | `Income` | `LifeExpectancy` | `Population` | `Region`

1. **Download the framework**

   http://www.cs171.org/2016/assets/scripts/lab4/template.zip

   We have included *Bootstrap*, *D3* and the dataset *wealth-health-2014.csv*. The HTML document contains an empty *div* container for the scatterplot and the CSV import is implemented in the JS file `main.js`.

2. **Analyze and prepare the data**

   - Use the web console to get a better idea of the dataset (e.g. quantitative or ordinal data?)

- Convert numeric values to numbers. All the imported values are *Strings* and calculations with *Strings* will lead to errors.

3. **Append a new SVG area with D3**

   - The ID of the target div container in the html file is `#chart-area`
   - Use the variables `height` and `width` for the SVG element
   - Save the new D3 selection in a variable (`var svg = d3.select("#chart-area")...`)

4. **Create linear scales by using the D3 scale functions**

   - You will need an *income* scale (x-axis) and a scale function for the *life expectancy* (y-axis). Call them `incomeScale` and `lifeExpectancyScale` .
   - Use *d3.min()* and *d3.max()* for the *input domain*
   - Use the variables `height` and `width` for the *output range*

5. **Try the scale functions**

   You can call the functions with example values and print the result to the web console.

```
// Examples:
incomeScale(5000)       // Returns: 23.2763
lifeExpectancyScale(68) // Returns: 224.7191
```

6. **Map the countries to SVG circles**

   - Use D3 to bind the data to visual elements, as you have done before (using D3's `select()` , `data()` , `enter()` , `append()` , etc.). Use svg circles as marks.
   - Instead of setting the x- and y-values directly, you have to use your scale functions to convert the data values to pixel measures

```
// Ice Cream Example
.attr("cx", function(d){ return iceCreamScale(d.sales); })
```

   - Specify the circle attributes: `r` , `stroke` and `fill`

7. **Refine the range of the scales**

   You have used the min and max values of the dataset, which means that some circles are positioned exactly on the border of the chart and are getting cut off.

You can use a *padding* variable when setting the *range* of the scales. This is one option to push the elements away from the edges of your SVG drawing area:

```
var padding = 20;

// Modify range for x-axis
var xScale = d3.scale.linear()
    .domain(...)
    .range([padding, width - padding]);

// Modify range for y-axis
var xScale = d3.scale.linear()
    .domain(...)
    .range([height - padding, padding]);
```

SVG Groups

In the last lab you have learned how to create basic SVG shapes, like rectangles or circles. But there is another SVG element that is very useful for programming in D3: the group element (`<g></g>`). In contrast to graphical elements the group element does not have a visual presence, but it helps you to organize other elements and to apply *transformations*. In this way, you can create hierarchical structures.

```
// Create group element
var group = svg.append("g");

// Append circle to the group
var circle = group.append("circle")
    .attr("r", 4)
    .attr("fill", "blue");
```

Group elements are invisible but you can apply transformations, for example *translate()* or *rotate()*, to the group and it will affect the rendering of **all child elements**!

```
// Group element with 'transform' attribute
// x = 70, y = 50 (...moves the whole group 70px to the right and 50px down)
var group = svg.append("g")
    .attr("transform", "translate(70, 50)");
```

Axes

The current visualization does not really look like a scatterplot yet. It is just a bunch of circles that are nicely arranged. We need x- and y-axes to allow the user to actually extract meaningful insights from the visualization.

→ An ***axis*** is the visual representation of a scale.

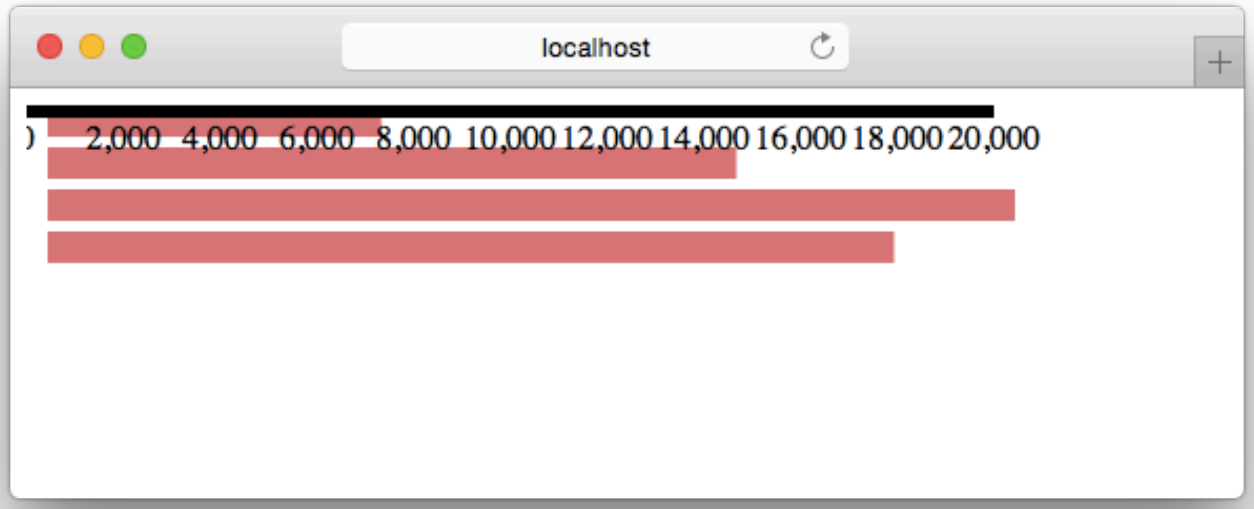
D3 provides methods to create axes, or more precisely, it can display reference lines for D3 scales automatically. These axis components contain lines, labels and ticks.

```
// Create a generic axis function
var xAxis = d3.svg.axis();

// Pass in the scale function
xAxis.scale(xScale);

// Specifiy orientation (top, bottom, left, right)
xAxis.orient("bottom");
```

Finally, to add the axis to the SVG graph, we need to specify the position in the DOM tree and then we have to *call* the axis function.

We create an SVG group element as a selection and use the *call()* function to hand it off to the *xAxis* function. All the axis elements are getting generated within that group.

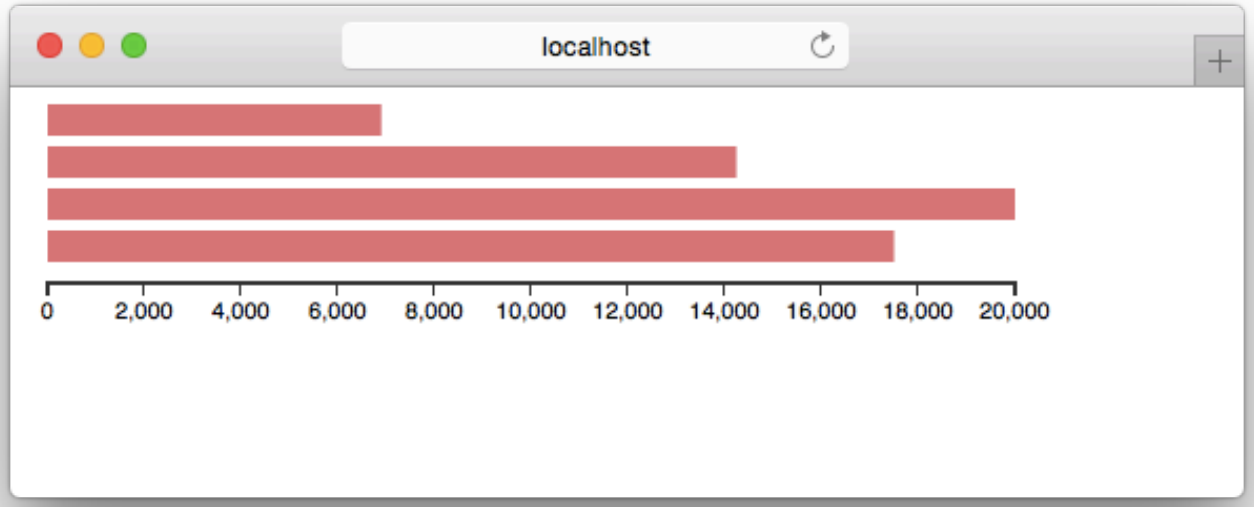
```
// Draw the axis
svg.append("g")
    .attr("class", "axis x-axis")
    .call(xAxis);
```



Recall that we can use the *transform* attribute to change the position and move the axis to the bottom. Additionally, you can use the HTML *class* property as a selector and modify the style with CSS:

```
.axis path,
.axis line {
    fill: none;
    stroke: #333;
    shape-rendering: crispEdges;
}

.axis text {
    font-family: sans-serif;
    font-size: 11px;
}
```

`shape-rendering` *is an SVG property which specifies how the SVG elements are getting rendered. We have used it in this example to make sure that we don't get blurry axes.*



Refine the axis

D3 axis functions automatically adjust the spacing and labels for a given scale and range. Depending on the data and the type of the visualization you may want to modify these settings.

```
var xAxis = d3.svg.axis()
    .scale(xScale)
    .orient("bottom")
    . ... // Add options here
```

There are many different options to customize axes:

- Number of ticks: `.ticks(5)`
- Tick format, e.g. as percentage: `.tickFormat(d3.format(".0%"))`
- Tick values: `.tickValues([0, 10, 20, 30, 40])`

You can read more about D3 axis, ticks and tick formatting in the *D3 API reference*:
https://github.com/mbostock/d3/wiki/SVG-Axes