D3 – Data-Driven Documents

**Chapter 2 Notes**

* D3 – also known as D3 or d3.js
* D3 is a JavaScript library for creating data visualizations.
* The data is provided by you
* The documents are web-based documents – this means anything that can be rendered by a web browser e.g. HTML.
* D3 connects the data to the documents.

What does D3 do?

It facilitates generation and manipulation of web documents with data. It does this by:

* Loading data into the browser’s memory
* Binding data to elements within the document, creating new elements as needed.
* Transforming those elements by interpreting each element’s bound datum and setting its visual properties accordingly.
* Transitioning elements between states in response to user input.

What it doesn’t do

* D3 doesn’t generate predefined visualisations for you.
* It does not support old browsers.
* D3’s core functionality doesn’t handle bitmap map tiles, such as those provided by Google Maps or Cloudmade.
* D3 doesn’t hide your original data. Because D3 code is executed on the client side (meaning, in the user’s web browser, as opposed to on the web server), the data you want visualized must be sent to the client. If your data can’t be shared, then don’t use D3.

A benefit of D3 is that, because it operates directly on the web document itself, it allows for easier debugging, easier experimentation and more visual possibilities. The only downside to this is a potentially steeper learning curve.

**Chapter 3 Notes (SVG)**

D3 is most useful when used to generate and manipulate visuals as Scalable Vector Graphics (SVG).

The SVG Element

* SVG is a text-based image format.
* Each SVG image is defined using markup code similar to HTML.
* SVG code can be included directly within any HTML document, or inserted dynamically into the DOM.
* Every browser supports SVG except IE 8 and older.
* SVG is XML-based thus elements don’t have a closing tag (as they are self closing).

Before you can draw anything, you must create an SVG element.

At a minimum, it’s good to specify width and height values e.g.

**<svg** width="500" height="50"**>** **</svg>**

Simple Shapes

There are a few visual elements that you can include between those svg tags, including:

* **rect** (defined using width and height).
* **circle** (defined using cx and cy to specify co-oridnates of the centre and r to specify radius).
* **ellipse** (same as circle except needs rx and ry for radius).
* **line** (use x1 and y1 to specify co-ordinates of one end of the line and x2 and y2 for the other end. A stroke colour must be defined for the line to be visible).
* **text** (text renders text. Use x to specify the position of the left edge, and y to specify the vertical position of the type’s *baseline*. (Baseline is a typographical term for the invisible line on which the letters appear to rest.)
* **path** (for drawing anything more complex).

Styling SVG Elements

Common SVG properties are as follows:

* fill - A colour value. Just as with CSS, colours can be specified as named colours, hex values, or RGB or RGBA values.
* Stroke - A colour value.
* stroke-width - A numeric measurement (typically in pixels).
* opacity - A numeric value between 0.0 (completely transparent) and 1.0 (completely opaque).

With text, you can also use these properties, which work just like in CSS:

* font-family
* font-size

In another parallel to CSS, there are two ways to apply styles to an SVG element: either directly (inline) as an attribute of the element, or with a CSS style rule.

**Chapter 5 – Data**

In the context of programming for visualization, data is stored in a digital file, typically in either text or binary form.

Within the scope of D3 and browser-based visualization, however, we will limit ourselves to *text-based data*. That is, anything that can be represented as numbers and strings of alpha characters. If you can get your data into a *.txt* plain text file, a *.csv* comma-separated value file, or a *.json* JSON document, then you can use it with D3.

Whatever your data, it can’t be made useful and visual until it is *attached* to something. In D3 lingo, the data must be *bound* to elements within the page.

Chain Syntax:

D3 smartly employs a technique called *chain syntax*, which you might recognize from jQuery. By “chaining” methods together with periods, you can perform several actions in a single line of code.

By the way, *functions* and *methods* are just two different words for the same concept: a chunk of code that accepts an argument as input, performs some action, and returns some other information as output.

Here’s an example:

d3.select("body").append("p").text("New paragraph!");

* **d3** - References the D3 object, so we can access its methods. Our D3 adventure begins here.
* .**select("body")** - Give the [select()](http://bit.ly/12h4SF1" \t "_top) method a CSS selector as input, and it will return a reference to the first element in the DOM that matches. (Use [selectAll()](http://bit.ly/WwINKs" \t "_top) when you need more than one element.) In this case, we just want the body of the document, so a reference to body is handed off to the next method in our chain.
* **.append("p")** - [append()](https://github.com/mbostock/d3/wiki/Selections" \l "wiki-append" \t "_top) creates whatever new DOM element you specify and appends it to the end (but *just inside*) of whatever selection it’s acting on. In our case, we want to create a new p within the body. We specified "p" as the input argument, but this method also sees the reference to body that was passed down the chain from the select()method. So an empty p paragraph is *appended* to the body. Finally, append() hands off a reference to the new element it just created.
* **.text("New paragraph!")** - [text()](https://github.com/mbostock/d3/wiki/Selections" \l "wiki-text" \t "_top) takes a string and inserts it between the opening and closing tags of the current selection. Because the previous method passed down a reference to our new p, this code just inserts the new text between <p> and </p>. (In cases where there is existing content, it will be overwritten.)
* **;** - The all-important semicolon indicates the end of this line of code. Chain over.

Binding Data

Data visualization is a process of *mapping* data to visuals. Data in, visual properties out. It might be that bigger numbers make taller bars, or special categories trigger brighter colours. The mapping rules are up to you.

With D3, we *bind* our data input values to elements in the DOM.

Binding is like “attaching” (or associating) data to specific elements, so that later you can reference those values to apply mapping rules.

Without the binding step, we have a bunch of data-less, unmappable DOM elements.

We use D3’s [selection.data()](https://github.com/mbostock/d3/wiki/Selections" \l "wiki-data" \t "_top) method to bind data to DOM elements. But there are two things we need in place first, before we can bind data:

* The data
* A selection of DOM elements

**Data**

D3 is smart about handling different kinds of data.

It will accept practically any array of numbers, strings, or objects (themselves containing other arrays or key/value pairs).

It can handle JSON (and GeoJSON), and even has a built-in method to help you load in CSV files.

CSV stands for comma-separated values. A CSV data file might look something like this:

Food, Deliciousness

Apples, 9

Green Beans, 5

Egg Salad Sandwich, 4

Cookies, 10

Vegemite, 0.2

Burrito, 7