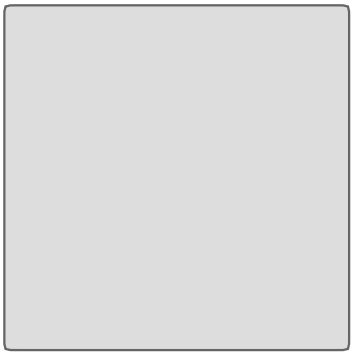
Try D3 Now



Intro

Data-Driven Documents, or D3 for short, is a new visualization library to build visualizations in SVG. But in my opinion, it's also the best javascript multipurpose SVG library when it comes to animation, interaction and above all for binding data to graphics. The community is very responsive, source code is very clean and the API is well written. There is not a lot of tutorials to learn D3 yet. But they are very good introductions and you can find plenty of examples packaged with the source code or hidden in forums. But if you know SVG, you already have a big head start. My goal was to write a simple introductory tutorial covering the main aspects of D3, with very trivial examples but with full length snippets for you to be up and running right now. So open your favorite editor, a decent browser and be ready to copy-paste.

Simple example

Let's start with a simple D3 example. It's an html file with a simple html5 doctype. The D3 script will be loaded directly from the D3 repository, but you could as well download a copy and load it locally. I've placed a `<div>` tag to be used as a container for our visualization. The D3 script lay down just before the closing `</body>` tag to be sure the page is fully loaded and ready to work with, and according to best practices. This example shows the use of D3 for DOM traversal, for adding SVG and HTML elements, for adding styles and attributes and for mouse events binding. Just copy-paste this example in a file named "index.html" opened in your favorite editor and in a recent browser. You can then replace the javascript code by any snippets found in this tutorial.

D3 simple example:

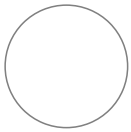
```
<!DOCTYPE html>
<html>
<head>
    <script type="text/javascript" src="http://mbostock.github.com/d3/d3.js"></script>
</head>
<body>
    <div id="viz"></div>
    <script type="text/javascript">

    var sampleSVG = d3.select("#viz")
        .append("svg")
        .attr("width", 100)
        .attr("height", 100);
```

```
    sampleSVG.append("circle")
        .style("stroke", "gray")
        .style("fill", "white")
        .attr("r", 40)
        .attr("cx", 50)
        .attr("cy", 50)
        .on("mouseover", function(){d3.select(this).style("fill", "aliceblue");})
        .on("mouseout", function(){d3.select(this).style("fill", "white");});

    </script>
</body>
</html>
```

SVG circle with hovering:



This is the SVG generated by the code from the above example:

```
<svg width="100" height="100">
<circle style="stroke: #808080; fill: #F0F8FF; " r="40" cx="50" cy="50"></circle>
</svg>
```

Why not just write this SVG fragment directly? There are several reasons to use D3 to generate SVG and HTML markup:

- Add, remove and modify multiple elements from an existing DOM
- Dynamically add attributes and styles according to a function
- Animate attributes and styles
- Bind data to automatically add elements when needed
- Benefits from a lot of helper functions

Select, append, bind event

D3, like JQuery and other javascript frameworks, simplify the selection of an element in the DOM but also in the SVG DOM. The static method `d3.select()` can take any CSS selector as an argument. The `d3.append()` method is similar, but is used to add a new element as a child of the selected element. To append an SVG element, you can prefix it by the SVG namespace like this: `d3.append("circle")`.

D3 uses a declarative style of programming. You declare what you want to select, then your modifications on the DOM, CSS styles, HTML and SVG attributes, texts, animations, and events. D3 will loop through your selection set and apply the modifications for you, hiding just enough of the DOM manipulation ugliness to leave you with the fun part and with full control on the process.

Binding events is just as simple. Select a DOM element and use the `.on()` method with the event you want as a string for the first attribute and a callback function for the second attribute, like in the example: `.on("mouseover", function(){d3.select(this).style("fill", "white");})`. You will often see this type of anonymous function as an argument in D3.

Animating

All attributes and styles can be animated using `.transition()`, `.delay()` and `.duration()`. D3 will recognize the type of the value and will interpolate numbers as well as colors, matrices, paths and percentages.

Simple animation:

```
var sampleSVG = d3.select("#viz")
    .append("svg")
    .attr("width", 100)
    .attr("height", 100);

sampleSVG.append("circle")
    .style("stroke", "gray")
    .style("fill", "white")
    .attr("r", 40)
    .attr("cx", 50)
    .attr("cy", 50)
  .transition()
    .delay(100)
```

```
    .duration(1000)
    .attr("r", 10)
    .attr("cx", 30)
    .style("fill", "black");
```

Animation chaining

One way to chain animations is by using `.delay()` to start one animation after the other. This example shows one way of using this technique on a mouse press. The `mousedown` event pass the current context, in this case the element where the event occurred, as an argument for the callback function that can use it under the name `this`. Notice how the duration of the first transition is inherited in the second. The main idea here is to delay the second transition for the duration of the first one so it is only triggered after.

Animation chaining trigged by the mouse:

```
var sampleSVG = d3.select("#viz")
    .append("svg")
    .attr("width", 100)
    .attr("height", 100);

sampleSVG.append("circle")
    .style("stroke", "gray")
    .style("fill", "white")
    .attr("r", 40)
    .attr("cx", 50)
    .attr("cy", 50)
    .on("mouseover", function(){d3.select(this).style("fill", "aliceblue");})
    .on("mouseout", function(){d3.select(this).style("fill", "white");})
    .on("mousedown", animate);

function animate() {
    d3.select(this).transition()
        .duration(1000)
        .attr("r", 10)
      .transition()
        .delay(1000)
        .attr("r", 40);
};
```

A better way of chaining animation is to listen to the `end` event with the `.each();` method to detect the end of each animation. See this example for a way to pass arguments to the function and to call it recursively.

Animation chaining using the `end` event:

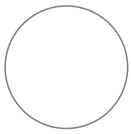
```
var sampleSVG = d3.select("#viz")
    .append("svg")
    .attr("width", 100)
    .attr("height", 100);

sampleSVG.append("circle")
    .style("stroke", "gray")
    .style("fill", "white")
    .attr("r", 40)
    .attr("cx", 50)
    .attr("cy", 50)
    .on("mouseover", function(){d3.select(this).style("fill", "aliceblue");})
    .on("mouseout", function(){d3.select(this).style("fill", "white");})
    .on("mousedown", animateFirstStep);

function animateFirstStep(){
    d3.select(this)
      .transition()
        .delay(0)
        .duration(1000)
        .attr("r", 10)
        .each("end", animateSecondStep);
};

function animateSecondStep(){
    d3.select(this)
      .transition()
        .duration(1000)
        .attr("r", 40);
};
```

Animation chaining trigged by a mouse press:



Binding data

D3 means Data-Driven Document because of the way it can bind data to graphics. Here is a simple example adding as much elements as there is members in the dataset. Once again, anonymous functions are used to dynamically calculate values for styles and attributes. Those functions provide two arguments: the data bound to the element and the index of the element in the selection set. Follow this tutorial directly from the master to learn how to add, update and remove elements on the fly.

Adding element from data:

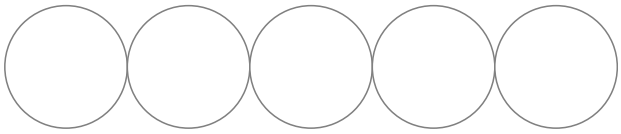
```
var dataset = [],
    i = 0;

for(i=0; i<5; i++){
    dataset.push(Math.round(Math.random()*100));
}

var sampleSVG = d3.select("#viz")
    .append("svg")
    .attr("width", 400)
    .attr("height", 75);

sampleSVG.selectAll("circle")
    .data(dataset)
    .enter().append("circle")
    .style("stroke", "gray")
    .style("fill", "white")
    .attr("height", 40)
    .attr("width", 75)
    .attr("x", function(d, i){return i*80})
    .attr("y", 20);
```

One SVG circle generated for each data member:



Binding two-dimensional data

To bind data made from an array of arrays, often called two-dimensional array, we must use a little trick. We will build a simple html table. First we add a `<table>` tag, then `<tr>` tags to add rows. For this, we bind the dataset to an empty `<tr>` selection and we then append as many `<tr>` tags as there is members in the first dimension of the dataset by chaining the two methods `.enter().append()`. But when it's time to bind data to the `<td>` tags to add columns, you have to bind the sub-array with this trick: `.data(function(d){return d;})`.

Table generated by data binding:

```
var dataset = [],
tmpDataset = [],
i, j;

for (i = 0; i < 5; i++) {
    for (j = 0, tmpDataset = []; j < 3; j++) {
        tmpDataset.push("Row:"+i+",Col:"+j);
    }
    dataset.push(tmpDataset);
}

d3.select("#viz")
    .append("table")
    .style("border-collapse", "collapse")
    .style("border", "2px black solid")

    .selectAll("tr")
    .data(dataset)
```

```
    .enter().append("tr")

    .selectAll("td")
    .data(function(d){return d;})
    .enter().append("td")
    .style("border", "1px black solid")
    .style("padding", "10px")
    .on("mouseover", function(){d3.select(this).style("background-color", "aliceblue")})
    .on("mouseout", function(){d3.select(this).style("background-color", "white")})
    .text(function(d){return d;})
    .style("font-size", "12px");
```

HTML table generated from a 2D array:

Row:0,Col:0	Row:0,Col:1	Row:0,Col:2
Row:1,Col:0	Row:1,Col:1	Row:1,Col:2
Row:2,Col:0	Row:2,Col:1	Row:2,Col:2
Row:3,Col:0	Row:3,Col:1	Row:3,Col:2
Row:4,Col:0	Row:4,Col:1	Row:4,Col:2

Loading data from a file

For the last examples, we generated arbitrary data. But D3 can also help you load a data file, for example a Comma Separated Value (CSV) file using Ajax. This example will work as is on a web server, local or remote. But you could also bypass the security restrictions of your browser that prevent you from loading data from a local file, for example using the command switch `--allow-file-access-from-files` in Google Chrome. If you understand what this is, you probably also know the risks. If not, use a web server. Anyway to load a CSV file with D3, you just need to use the `d3.csv()` method. But here, I will use an equivalent method, `d3.text()`, to load it as plain text and use a d3 helper to parse the CSV.

Table from external CSV file:

```
d3.text("auto_mpg_tmp.csv", function(datasetText) {

var parsedCSV = d3.csv.parseRows(datasetText);

var sampleHTML = d3.select("#viz")
    .append("table")
    .style("border-collapse", "collapse")
    .style("border", "2px black solid")

    .selectAll("tr")
    .data(parsedCSV)
    .enter().append("tr")

    .selectAll("td")
    .data(function(d){return d;})
    .enter().append("td")
    .style("border", "1px black solid")
    .style("padding", "5px")
    .on("mouseover", function(){d3.select(this).style("background-color", "aliceblue")})
    .on("mouseout", function(){d3.select(this).style("background-color", "white")})
    .text(function(d){return d;})
    .style("font-size", "12px");
});
```

Content of the external CSV file:

```
car name,miles/gallon,cylinders,displacement,horsepower,weight,acceleration,model year,origin
"chevrolet chevelle malibu",18,8,307,130,3504,12,70,1
"buick skylark 320",15,8,350,165,3693,11.5,70,1
"plymouth satellite",18,8,318,150,3436,11,70,1
```

Table generated by the data from a CSV file:

car name	miles/gallon	cylinders	displacement	horsepower	weight	acceleration	model year	origin
chevrolet chevelle malibu	18	8	307	130	3504	12	70	1
buick skylark 320	15	8	350	165	3693	11.5	70	1
plymouth satellite	18	8	318	150	3436	11	70	1

That's about it. I hope this tutorial was useful for you. If you want to learn more, continue to other great

tutorials!

Follow @d3visualization

Christophe Viau

In partnership with