D3 is *Data-Driven Documents. It* is a JavaScript library for creating data visualizations. The *data* is provided by you, and the *documents* are web-based documents, meaning anything that can be rendered by a web browser, such as HTML and SVG. D3 does the *driving*, in the sense that it connects the data to the documents.

**What it does?**

Fundamentally, D3 is an elegant piece of software that 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 or “canned” visualizations for you. This is on purpose. D3 is intended primarily for *explanatory* visualization work, as opposed to *exploratory* visualizations.
* D3 doesn’t even try to support older browsers.
* 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.
* D3’s core functionality doesn’t handle bitmap map tiles, such as those provided by Google Maps or Cloudmade.

**Chaining Methods**

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

Example code

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() method a CSS selector as input, and it will return a reference to the first element in the DOM that matches. (Use selectAll() 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() 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() 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.

**Scales**

Scales are functions that map from an input domain to an output range. The values in any dataset are unlikely to correspond exactly to pixel measurements for use in your visualization. Scales provide a convenient way to map those data values to new values useful for visualization purposes.

D3 scales are functions with parameters that you define. Once they are created, you call the scale function, pass it a data value, and it nicely returns a scaled output value. You can define and use as many scales as you like.

**Domains and Ranges**

A scale’s input domain is the range of possible input data values. A scale’s output range is the range of possible output values, commonly used as display values in pixel units. The output range is completely up to you, as the information designer.

**Creating a Scale**

D3’s scale function generators are accessed with d3.scale followed by the type of scale

you want.

var scale = d3.scale.linear();

Set the domain and range by passing the values as array to the corresponding methods.

.domain( [x, y] )

.range( [0, w] )

**Examples:**

**Bar chart**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>D3: Drawing divs, spaced out</title>

<script type="text/javascript" src="../d3.js"></script>

<style type="text/css">

div.bar {

display: inline-block;

width: 20px;

height: 75px; /\* Gets overriden by D3-assigned height below \*/

margin-right: 2px;

background-color: teal;

}

</style>

</head>

<body>

<script type="text/javascript">

var dataset = [ 5, 10, 15, 20, 25 ];

d3.select("body").selectAll("div")

.data(dataset)

.enter()

.append("div")

.attr("class", "bar")

.style("height", function(d) {

var barHeight = d \* 5;

return barHeight + "px";

});

</script>

</body>

</html>

**Scatter plot example**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>D3: A simple scatterplot with value labels</title>

<script type="text/javascript" src="../d3.js"></script>

<style type="text/css">

/\* No style rules here yet \*/

</style>

</head>

<body>

<script type="text/javascript">

//Width and height

var w = 500;

var h = 100;

var dataset = [

[5, 20], [480, 90], [250, 50], [100, 33], [330, 95],

[410, 12], [475, 44], [25, 67], [85, 21], [220, 88]

];

//Create SVG element

var svg = d3.select("body")

.append("svg")

.attr("width", w)

.attr("height", h);

svg.selectAll("circle")

.data(dataset)

.enter()

.append("circle")

.attr("cx", function(d) {

return d[0];

})

.attr("cy", function(d) {

return d[1];

})

.attr("r", function(d) {

return Math.sqrt(h - d[1]);

});

svg.selectAll("text")

.data(dataset)

.enter()

.append("text")

.text(function(d) {

return d[0] + "," + d[1];

})

.attr("x", function(d) {

return d[0];

})

.attr("y", function(d) {

return d[1];

})

.attr("font-family", "sans-serif")

.attr("font-size", "11px")

.attr("fill", "red");

</script>

</body>

</html>