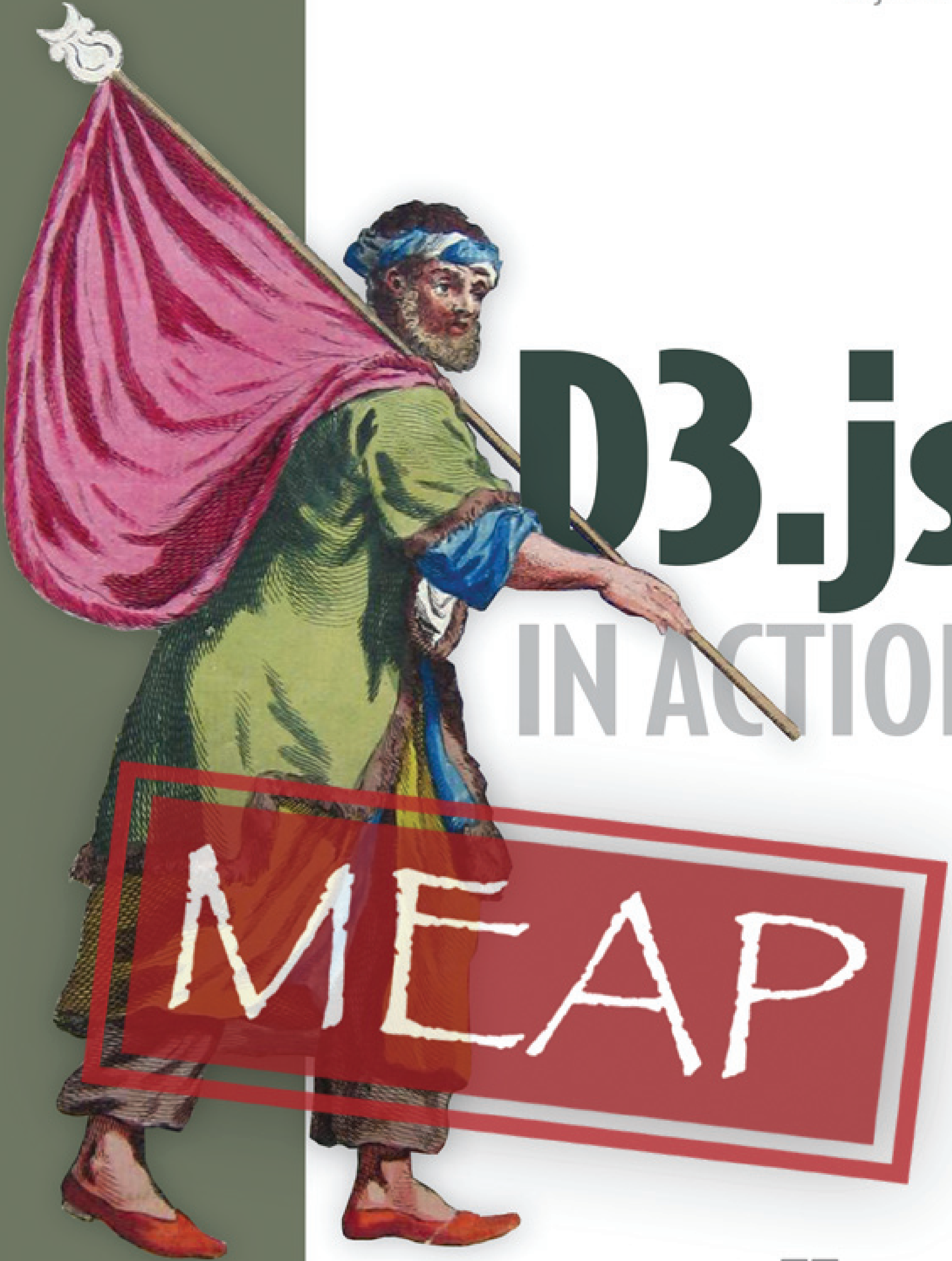
Elijah Meeks

D3.js IN ACTION



MANNING

MEAP Edition
Manning Early Access Program
D3.js in Action
Version 9

Copyright 2014 Manning Publications

For more information on this and other Manning titles go to
www.manning.com

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

Welcome

Thank you for purchasing the MEAP for *D3.js in Action*. To get the most benefit from this book, you'll want to have some established skills in programming, with experience in HTML5 and basic knowledge about CSS and DOM, or be transitioning from dealing with data in R/Python/SQL with the desire to build more sophisticated shareable applications.

When I first started experimenting with D3 two years ago, it was out of necessity. Flash was dead, and I needed an information visualization library that was feature rich and, hopefully, long lived. D3 proved to be more than that, with robust capabilities not only for data visualization, but for creating complex applications on the web.

Since that time, there have been numerous examples, some really great introductions to the library, and a few cookbooks to help new users learn the basics and get specific tasks done with D3. In contrast, *D3.js in Action* is my attempt to produce an exhaustive, highly informative, deep dive into the library, one that covers the fundamental structures of how D3 works with data to produce the stunning products we're all so impressed by. D3 does a lot, and not just with graphics, and I've tried to explain in detail the whole library.

Along with all that, I also wanted to focus on two specific spheres that D3 handles really well: networks and maps. As a result, this book spends a full chapter on each, dealing with the variety of features and functions in D3 that let you create the most amazing network visualizations and online interactive mapping applications. This is along with chapters that focus on general charts and D3-specific layouts more broadly.

Finally, throughout D3.js in Action you'll see an approach that embraces the functionality available in modern browsers.

It's a big book for a big library, and I hope you find it as useful to read as I did to write it. Please be sure to post any questions, comments, or suggestions you have about the book in the Author Online forum. Your feedback is essential in developing the best book possible.

— Elijah Meeks

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

brief contents

PART 1: AN INTRODUCTION TO D3

1 *An introduction to D3.js*

2 *Information Visualization Data Flow*

3 *Data-Driven Design and Interaction*

PART 2: THE PILLARS OF INFORMATION VISUALIZATION

4 *Chart Components*

5 *Layouts*

6 *Network Visualization*

7 *Geospatial Information Visualization*

8 *Traditional DOM Manipulation with D3*

PART 3: COMPOSING INTERACTIVE APPLICATIONS WITH D3

9 *Composing Interactive Applications*

10 *Writing Layouts and Components*

11 *Big data visualization*

Online supplement: D3 on Mobile

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

1 An introduction to D3.js

This chapter covers

- The basics of HTML, CSS, and the Document Object Model (DOM)
- The principles of Scalable Vector Graphics (SVG)
- Data-binding and selections with D3
- Different data types and their data visualization methods

"D3" stands for Data-Driven Documents. It's a brand name but also a class of applications that have been offered on the web in one form or another for years. For quite some time we've been building and working with data-driven documents, such as interactive dashboards, rich internet applications, and dynamically driven content. In one sense, the D3.js library is an iterative step in a chain of technologies used for data-driven documents, but in another sense, it's a radical step.

1.1 What is D3?

D3.js was created to fill a pressing need for web-accessible, sophisticated data visualization. Because of the library's robust design, it does more than make charts. And that's a good thing, because data visualization no longer refers to pie charts and line graphs. It now means maps and interactive diagrams and other tools and content integrated into news stories, data dashboards, reports, and everything else you see on the web.

D3's creator, Mike Bostock, helped develop an earlier data visualization library, Protovis, and also developed Polymaps, a JavaScript library that provided vector and tile mapping capability in a lightweight form. These earlier endeavors would inform the creation of D3.js, which focuses on modern standards and modern browsers. As Bostock describes it, "This avoids proprietary representation and affords extraordinary flexibility, exposing the full capabilities of web standards such as CSS3, HTML5 and SVG." This is the radical nature of

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

D3.js. Although it won't run on Internet Explorer 6, the widespread adoption of standards on modern browsers has finally let web developers deliver dynamic and interactive content seamlessly in the browser.

Until recently, you couldn't build the kind of high-performance rich internet applications in the browser unless you built them in Flash or as a Java applet. Flash and Java are still around on the internet, and especially for internal web apps, for this reason. D3.js provides the same performance, but integrated into web standards and the Document Object Model at the core of HTML. D3 provides developers with the ability to create rich interactive and animated content based on data and tie that content to existing web page elements. It gives you the tools to create high-performance data dashboards and sophisticated data visualization, and to dynamically update traditional web content.

But D3 isn't easy for people to pick up, because they often expect it to be a simple charting library. A good example is the pie chart layout, which you'll see in chapter 5. D3 doesn't have one single function to create a pie chart. Rather, it has a function that processes your dataset with the necessary angles so that, if you pass the dataset to D3's `arc` function, you get the drawing code necessary to represent those angles. And you need to use yet another function to create the paths necessary for that code. It's a much longer process than using dedicated charting libraries, but the D3 process is also its strength. Although other charting libraries conveniently allow you to make line graphs and pie charts, they quickly break down when you want to make something more than that. Not D3, which allows you to build whatever data-driven graphics and interactivity that you can imagine, and that's why D3 is behind much of the most innovative and exciting information visualization on the web today.

1.2 How D3 works

Let's take a look at the principles of data visualization, as well as how D3 works in general. In figure 1.1 you see a rough map of how you might start with data and use D3 to process and represent that data, as well as add interactivity and optimize the data visualization you've created. In this chapter, we'll start by establishing the principles of how D3 selections and data-binding work and learning how it interacts with SVG and the DOM. Then we'll look at data types that you'll commonly encounter. Finally, you'll use D3 to create simple DOM and SVG elements.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

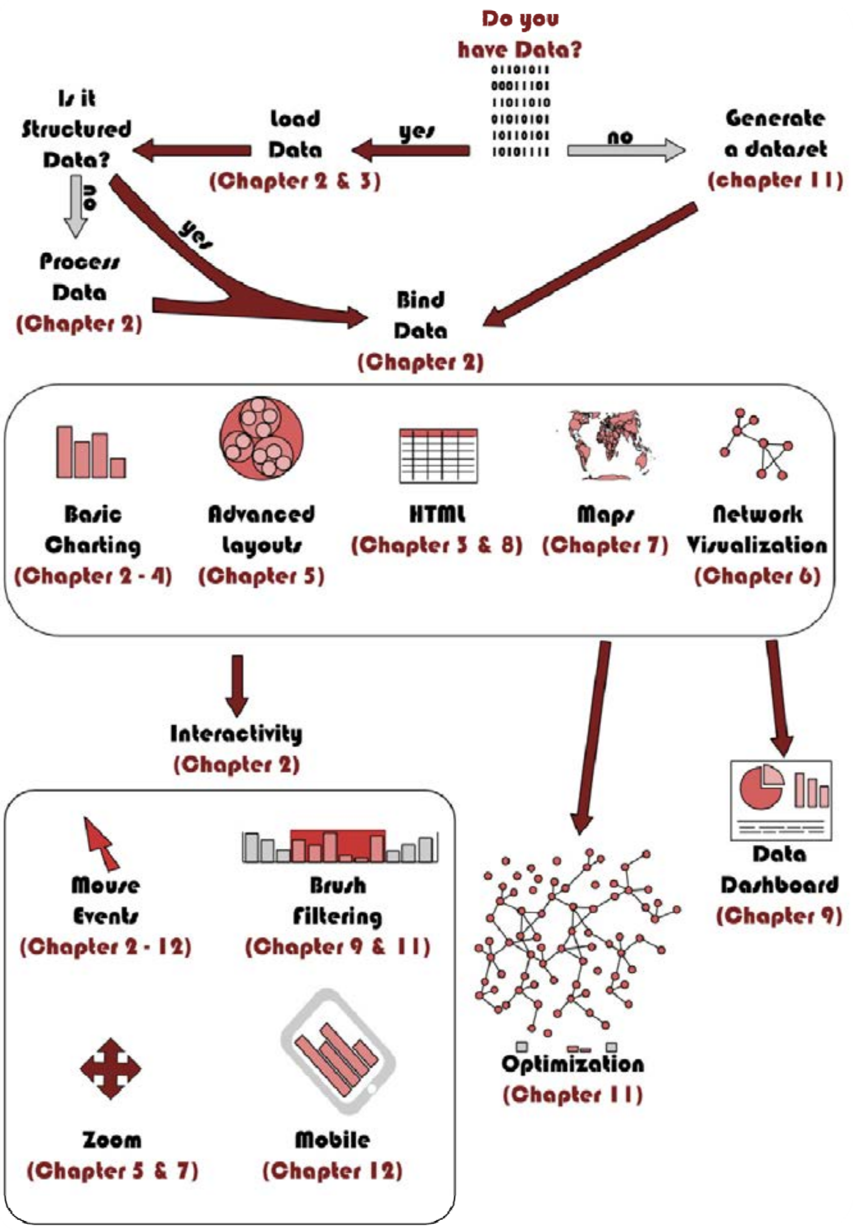

Do you have Data?
01101011
00011101
11011010
01010101
10110101
10101111
yes
no
Load Data (Chapter 2 & 3)
Is it Structured Data?
Generate a dataset (chapter 11)
no
yes
Process Data (Chapter 2)
Bind Data (Chapter 2)
Basic Charting (Chapter 2 - 4)
Advanced layouts (Chapter 5)
HTML (Chapter 3 & 8)
Maps (Chapter 7)
Network Visualization (Chapter 6)
Interactivity (Chapter 2)
Mouse Events (Chapter 2 - 12)
Brush Filtering (Chapter 9 & 11)
Zoom (Chapter 5 & 7)
Mobile (Chapter 12)
Optimization (Chapter 11)
Data Dashboard (Chapter 9)

Figure 1.1 A map of how to approach data visualization with d3.js that highlights the approach in this book. Start at the top with data, and then follow the path depending on the type of data and the needs you're addressing.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

1.2.1 Data visualization is more than data visualization

You may think of data visualization as limited to pie charts, line charts, and the variety of charting methods popularized by Tufte and deployed in research. It’s much more than that. One of the core strengths of d3.js is that it allows for the creation of vector graphics for traditional charting, but also the creation of geospatial and network visualizations as well as traditional HTML elements like tables, lists, and paragraphs. This broad-based approach to data visualization, where a map or a network graph or a table is just another kind of representation of data, is the core of the D3.js library’s appeal for application development.

Figures 1.2 through 1.8 show data visualization pieces that I’ve created with D3. They include maps and networks, along with more traditional pie charts and completely custom data visualization layouts based on the specific needs of my clients.

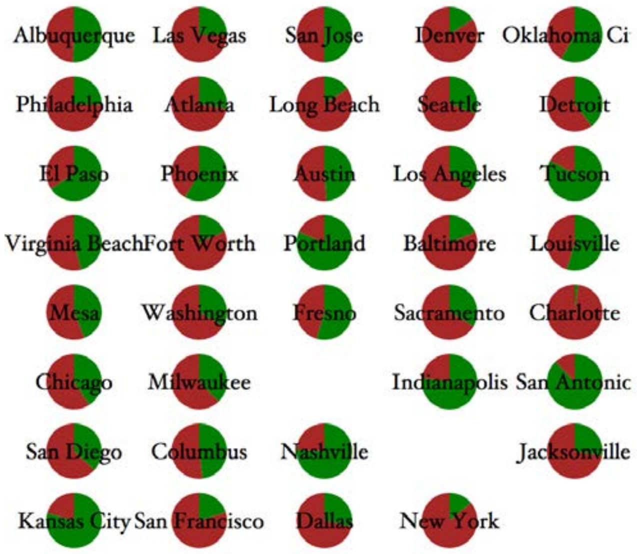


Figure 1.2 D3 can be used for simple charts, such as this set of multiple pie charts (explained in chapter 5) used to represent the differences in the use of language about nature in major US city planning (from the City Nature project at citynature.stanford.edu). Each pie shows the ratio of language referring to parks and open space (green) versus habitat (red) in city plans.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

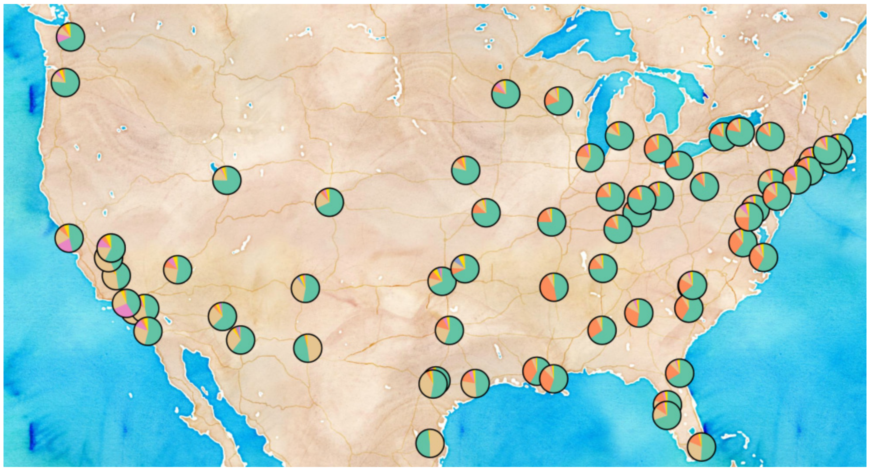


Figure 1.3 D3 can also be used to create web maps (see chapter 7), such as this map showing the ethnic makeup of major metropolitan areas in the United States.

Although the ability to create rich and varied graphics is one of D3's strong points, more important for modern web development is the ability to embed the high level of interactivity that users expect. With D3, every element of every chart, from a spinning globe to a single, thin slice of a pie chart, is made interactive in the same way. And because D3 was written by someone well versed in data visualization practice, it includes a number of interactive components and behaviors that are standard in data visualization and web development.

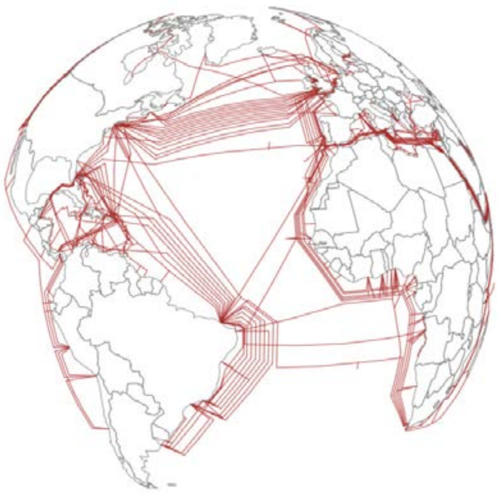


Figure 1.4 Maps in D3 aren't limited to traditional Mercator web maps, and can be interactive globes, like this map of undersea communication cables, or other more unorthodox maps (see chapter 7).

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

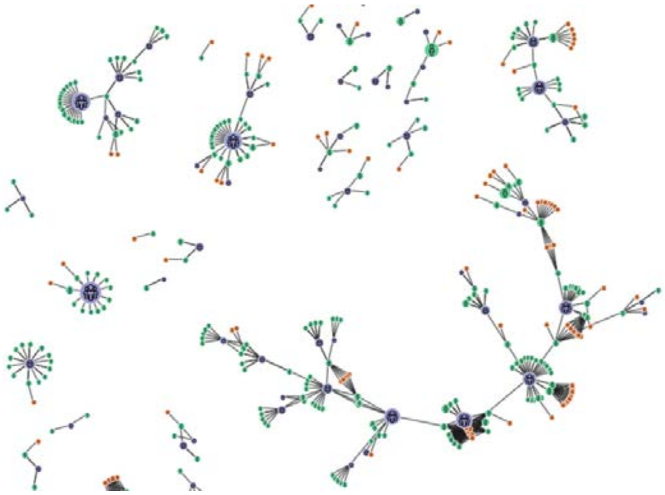
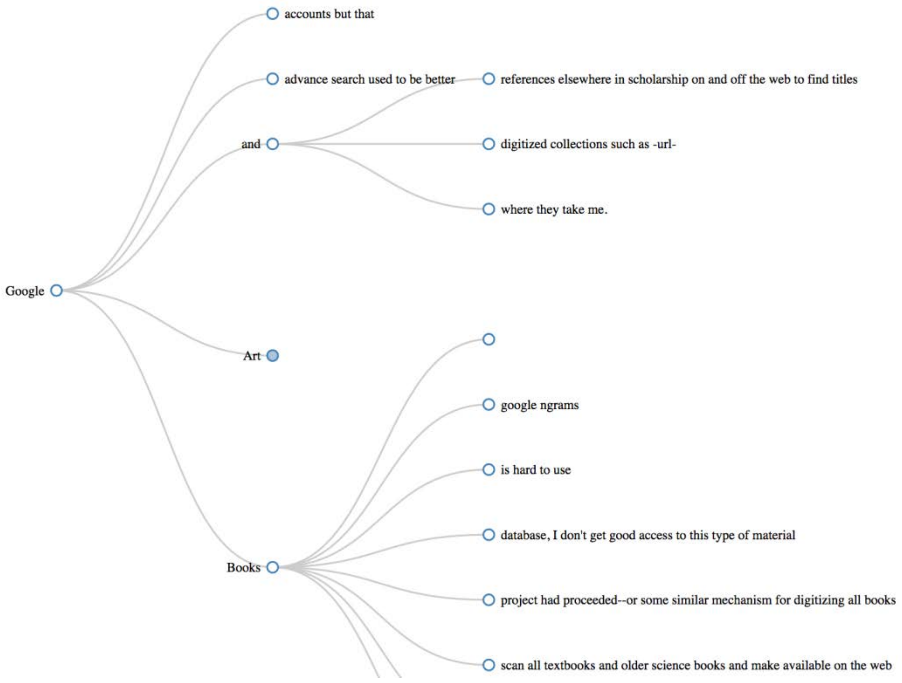


Figure 1.5 D3 also provides robust capacities to create interactive network visualizations (see chapter 6). Here you see the social and coauthorship network of archaeologists working at the same dig for nearly 25 years.


accounts but that
advance search used to be better
references elsewhere in scholarship on and off the web to find titles
and
digitized collections such as -url-
where they take me.
Google
Art
google ngrams
is hard to use
database, I don't get good access to this type of material
Books
project had proceeded--or some similar mechanism for digitizing all books
scan all textbooks and older science books and make available on the web
Figure 1.6 D3 includes a library of common data visualization layouts, such as the dendrogram (explained in chapter 5), that let you represent data such as this word tree.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

You don't invest your time learning D3 so that you can deploy Excel-style charts on the web. For that, there are easier, more convenient libraries. You learn D3 because it gives you the ability to implement almost every major data visualization technique. It also gives you the power to create your own data visualization techniques, something a more general library can't do.

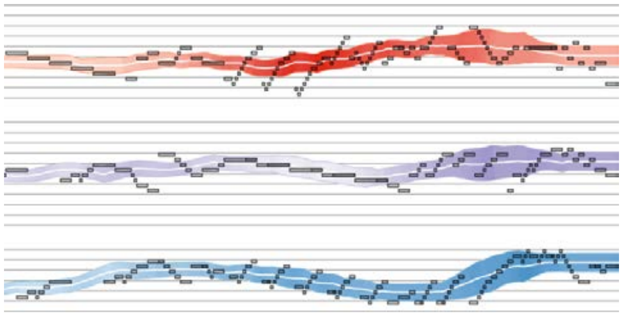


Figure 1.7 D3 has numerous SVG drawing functions (see chapter 4) so you can create your own custom visualizations, such as this representation of musical scores.

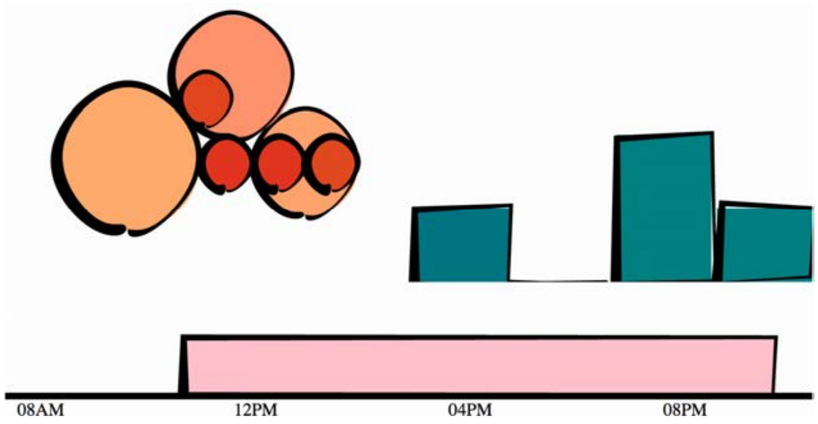


Figure 1.8 You can combine these layouts and functions to create a data dashboard like you'll do in chapter 9. You can also use the drawing functions to make your bar charts look distinctive, such as this "sketchy" style.

For more examples of the variety of different data visualization techniques realized with D3, take a look at Christophe Viau's gallery of over 2000 D3 examples here: http://christopheviau.com/d3list/gallery.html

By requiring a break with the practice of supporting long-obsolete browsers, D3.js affords developers the capacity to make not only richly interactive applications but also applications that are styled and served like traditional web content. This makes them more portable, more amenable to the growing linked data web, and more easily maintained by large teams.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

The decision on Bostock’s part to deal broadly with data, and create a library capable of presenting maps as easily as charts as easily as networks as easily as ordered lists, also means that a developer doesn’t need to try to understand the abstractions and syntax of one library for maps, and another for dynamic text content, and another for data visualization. Instead, the code for running an interactive force-directed network layout is very close to pure JavaScript and also similar to the code representing dynamic points of interest on a D3.js map. Not only are the methods the same, but the very data could be the same, formulated in one way for lists and paragraphs and spans while formulated in another way for geospatial representation. The class of data-driven document is already broad and becomes even more all-encompassing when you also treats images and text as data.

1.2.2 D3 is about selecting and binding

Throughout this chapter, you’ll see code snippets that you can run in your browser to make changes to the graphical appearance of elements on your web site. At the end of the chapter is an application written in D3 that explains the basics of the code you’re running in JavaScript. But before that you’ll explore the principles of web development using D3, and you’ll see this pattern of code over and over again: selecting.

Imagine you have a set of data, such as the price and size of a few houses, and a set of web page elements, whether graphics or traditional <div> elements, that you want to represent that data, whether with text or through size and color. A *selection* is the group of all of them together, and you perform actions on the elements in the group, such as moving them, changing their color, or updating the values in the data. You work with the data and the web page elements separately, but the real power of D3 comes from using selections to combine data and web page elements.

Here’s a selection without any data:

```
d3.selectAll("circle.a").style("fill", "red").attr("cx", 100);
```

This takes every circle on your page with the class of "a" and turns it red and moves it so that its center is 100 pixels to the right of the left side of your <svg> canvas. Likewise, this code turns every div on your web page red and changes its class to "b".

```
d3.selectAll("div").style("background", "red").attr("class", "b");
```

But before you can change your circles and divs, you’ll need to create them, and before you do that, it’s best to understand what’s happening in this pattern.

The first part of that line of code, `d3.selectAll()`, is part of the core functionality necessary for understanding D3: selections. Selections can be made with `d3.select()`, which selects the first single element found, but more often you’ll use `d3.selectAll()`, which can be used to select multiple elements. Selections are a group of one or more web page elements that may be associated with a set of data, like the following code, which binds the elements in the array [1,5,11,3] to <div> elements with the class of "`market`".

```
d3.selectAll("div.market").data([1,5,11,3])
```

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

This association is known in D3 as *binding data,* and you can think of a selection as a set of web page elements and a corresponding, associated set of data. Sometimes there are more data elements than DOM elements, or vice versa, in which case D3 has functions designed to create or remove elements that you can use to generate content. We'll cover selections and data-binding in detail in chapter 2. Selections might not include any data-binding, and won't for most of the examples in this chapter, but the inclusion allows the powerful information visualization techniques of D3. You can make a selection on any elements in a web page, including items in a list, circles, or even regions on a map of Africa. Just as the elements can take a number of shapes, the data associated with those elements (where applicable) can take many forms.

1.2.3 D3 is about deriving the appearance of web page elements from bound data

After you have a selection, you can then use D3 to modify the appearance of web page elements to reflect differences in the data. You may want to make the length of a line equal to the value of the data, or change the color to a particular color that corresponds to a class of data. You may want to hide or show elements as they correspond to a user's navigation of a dataset. As you can see in Figure 1.9, after the page has loaded, you use D3 to select elements and bind data for the purpose of creating, removing or changing DOM elements. You continue to use this process in response to user interaction.

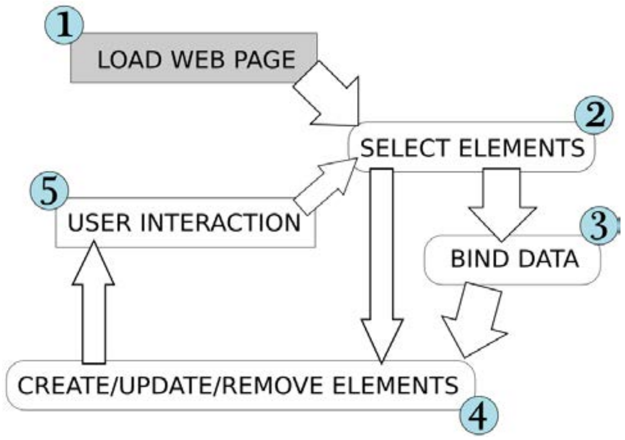


Figure 1.9 A page utilizing D3 is typically built in such a way that the page loads with styles, data, and content as defined in traditional HTML development (1) with its initial display using D3 selections of HTML elements (2) either with data-binding (3) or without it to modify the structure and appearance of the page (4) . The changes in structure prompt user interaction (5), which causes new selections with and without data-binding to further alter the page. Step 1 is shown differently because it only happens once (when you load the page), whereas every other step may happen multiple times, depending on user interaction.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

You modify the appearance of elements by using selections to reference the data bound to an element in a selection. D3 iterates through the elements in your selection and performs the same action using the bound data, which results in different graphical effects. Although the action you perform is the same, the effect is different because it's based on the variation in the data. You'll first see data-binding at the end of the chapter, and in much more detail throughout this book.

1.2.4 Web page elements can now be divs, countries, and flowcharts

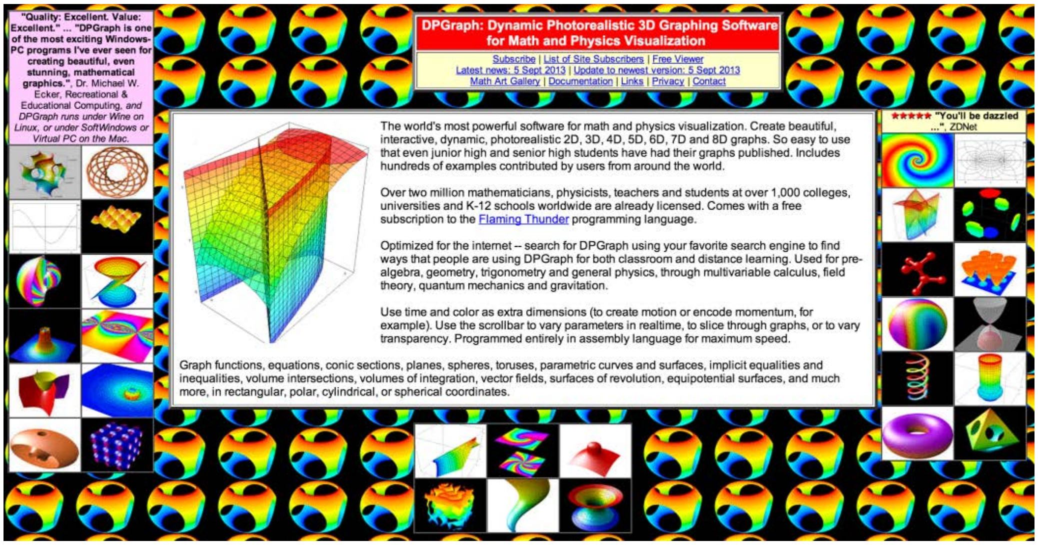
We've grown accustomed to thinking of web pages as consisting of text elements with containers for pictures, videos, or embedded applications. But as you grow more familiar with D3, you'll begin to recognize that every element on the page can be treated with the same high-level abstractions. The most basic element on a web page, a `<div>` that represents a rectangle into which you can drop paragraphs, lists, and tables, can be selected and modified in the same way you can select and modify a country on a web map, or individual circles and lines that make up a complex data visualization.

To be able to select items on a web page, you have to ensure that they're built in a manner that makes them a part of the traditional structure of a web page. You can't select items in a Java applet, or in a Flash runtime, nor could you select the labels on an embedded Google map, but if you create these elements so that they exist as elements in your web page, then you give yourself tremendous flexibility. To get a taste of this, look at chapter 7, where you'll build robust mapping applications in D3, and you'll use the `d3.select()` syntax to update the appearance of a mapping application in the same manner as it's being used here and elsewhere to create and move circles or `<div>` elements.

1.3 Using HTML5

We've come a long way from the days when animated gifs and frames were the pinnacle of dynamic content on the web. In figure 1.10, you can see why gifs never caught on for robust data visualization on the web. GIFs, like the infoviz libraries designed to use VML, are still necessary for earlier browsers, but D3 is designed for modern browsers that don't need the helper libraries necessary for backward compatibility. D3 development isn't for everyone, but if your audience can be assumed to have access to a modern web browser, D3 also brings a significant reduction in the cost necessary not only to code for older browsers but also to learn and keep updated on the various libraries that support backward compatibility with those older browsers.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905



"Quality: Excellent. Value: Excellent." ... "DPGraph is one of the most exciting Windows-PC programs I've ever seen for creating beautiful, even stunning, mathematical graphics.", Dr. Michael W. Ecker, Recreational & Educational Computing, *and DPGraph runs under Wine on Linux, or under SoftWindows or Virtual PC on the Mac.*

DPGraph: Dynamic Photorealistic 3D Graphing Software for Math and Physics Visualization

Subscribe | List of Site Subscribers | Free Viewer
Latest news: 5 Sept 2013 | Update to newest version: 5 Sept 2013
Math Art Gallery | Documentation | Links | Privacy | Contact

★★★★★ "You'll be dazzled ...", ZDNet

The world's most powerful software for math and physics visualization. Create beautiful, interactive, dynamic, photorealistic 2D, 3D, 4D, 5D, 6D, 7D and 8D graphs. So easy to use that even junior high and senior high students have had their graphs published. Includes hundreds of examples contributed by users from around the world.

Over two million mathematicians, physicists, teachers and students at over 1,000 colleges, universities and K-12 schools worldwide are already licensed. Comes with a free subscription to the Flaming Thunder programming language.

Optimized for the internet -- search for DPGraph using your favorite search engine to find ways that people are using DPGraph for both classroom and distance learning. Used for pre-algebra, geometry, trigonometry and general physics, through multivariable calculus, field theory, quantum mechanics and gravitation.

Use time and color as extra dimensions (to create motion or encode momentum, for example). Use the scrollbar to vary parameters in realtime, to slice through graphs, or to vary transparency. Programmed entirely in assembly language for maximum speed.

Graph functions, equations, conic sections, planes, spheres, toruses, parametric curves and surfaces, implicit equalities and inequalities, volume intersections, volumes of integration, vector fields, surfaces of revolution, equipotential surfaces, and much more, in rectangular, polar, cylindrical, or spherical coordinates.

Figure 1.10 Before GIFs were weaponized to share cute animal behavior, they were your only hope for animated data visualization on the web. Few examples from the 1990s like dpgraph.com exist, but this page has more than enough GIFs to remind us of their dangers.

A modern browser typically can not only display SVG graphics and obey CSS3 rules, but also has great performance. Along with Cascading Style Sheets (CSS) and Scalable Vector Graphics (SVG), we can break down HTML5 into the Document Object Model (DOM) and JavaScript. The following sections treat each of them and include code you can run to see how D3 uses their functionality to create interactive and dynamic web content.

1.3.1 The DOM

A web page is structured according to the DOM. You need a passing familiarity with the DOM to do web development, so we'll take a quick look at DOM elements and structure in a simple web page in your browser and touch on the basics of the DOM. To get started, you'll need a web server that you can access from the computer that you're using to code. With that in place, you can download the D3 library from d3js.org (d3.js or d3.min.js for the minified version) and place that in the directory where you'll make your web page. You'll create a page called d3ia.html in the text editor with the following contents:

```
<!doctype html>
<html>
<head>
  <script src=”d3.v3.min.js” type=”text/JavaScript”></script>#a
</head>
<body> #b
<div id="someDiv" style="width:200px;height:100px;border:black 1px solid;"> #c
<input id="someCheckbox" type="checkbox" /> #d
</div>
```

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

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

#a a child element of <html>
#b a child element of <html>
#c a child element of <body>
#d a child element of <div>

Basic HTML like the kind here follows the DOM. It defines a set of nested elements, starting with an `<html>` element with all its child elements and their child elements and so on. In this example, the `<script>` and `<body>` elements are children of the `<html>` element and the `<div>` element is a child of the `<body>` element. The `<script>` element loads the D3 library here, or it can have inline JavaScript code, whereas any content in the `<body>` element shows up onscreen when you navigate to this page.

UTF-8 and D3.js

D3 utilizes UTF-8 characters in its code, which means that you can do one of three things to make sure you don't have any errors:

You can set your document to UTF-8:

```
<!DOCTYPE html><meta charset="utf-8">
```

Or you can set the charset of the script to UTF-8:

```
<script charset="utf-8" src="d3.js"></script>
```

Or you can use the minified script, which shouldn't have any UTF-8 characters in it:

```
<script src="d3.min.js"></script>
```

Three categories of information about each element determine its behavior and appearance: styles, attributes, and properties. *Styles* can determine transparency, color, size, borders, and so on. *Attributes* typically refer to classes, IDs, and interactive behavior, though some attributes can also determine appearance, depending on which type of element you're dealing with. *Properties* typically refer to states, such as the "checked" property of a check box, which is true if the box is checked and false if the box is unchecked. D3 has three corresponding functions to modify these values. If you wanted to modify the HTML elements in the previous example, you could use D3 functions that abstract this process:

```
d3.select("#someDiv").style("border", "5px darkgray dashed");
d3.select("#someDiv").attr("id", "newID");
d3.select("#someCheckbox").property("checked", true);
```

Like many D3 functions of this kind, if you don't signify a new value, then the function returns the existing value. You'll see this in action throughout this book, and later in the

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

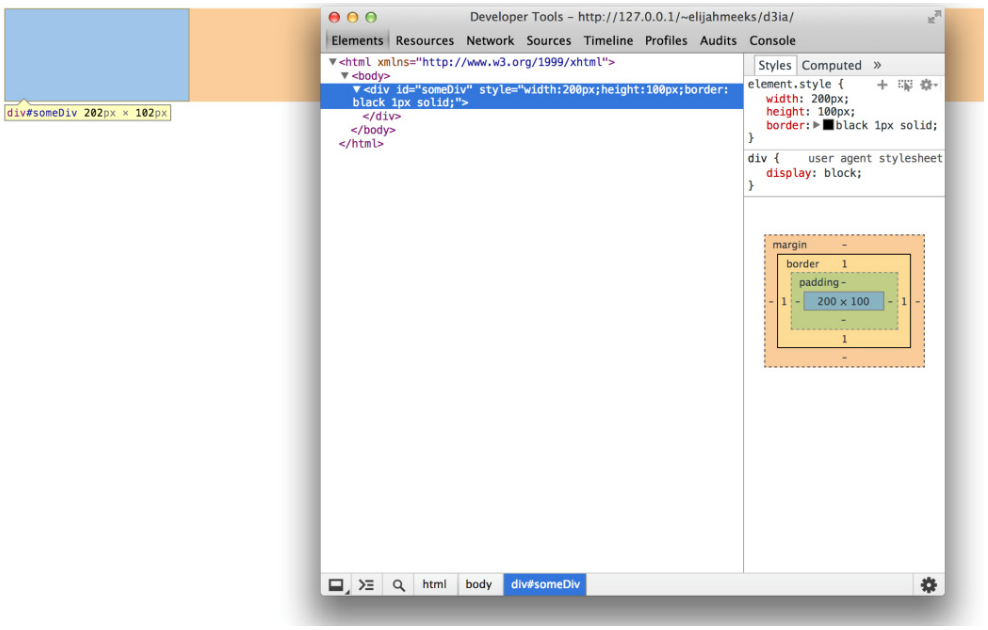
chapter as you write more code, but for now remember that these three functions allow you to change how an element appears and interacts.

The DOM also determines the drawing order of elements onscreen, with child elements drawn after and inside parent elements. Although you have some control over drawing elements above or below each other with traditional HTML using `z-index`, this isn't available for SVG elements (though it might be implemented at some point using the `render-order` attribute).

EXAMINING THE DOM IN THE CONSOLE

Navigate to d3ia.html, and you can get exposure to how D3 works. The page isn't very impressive, with just a single black-outlined rectangle. You could modify the look and feel of this web page by updating d3ia.html, but you'll find that it's easy to modify the page by using your web browser's developer console. This is useful for testing changes to classes or elements before implementing them in your code. Open up the developer console, and you'll have two useful screens, shown in figures 1.11 and 1.12, which we'll go back to over and over.

NOTE: You'll see the console in this first chapter, but in chapter 2, once you're familiar with it, I'll show only the output.



```
▼<html xmlns="http://www.w3.org/1999/xhtml">
  ▼<body>
    ▼<div id="someDiv" style="width:200px;height:100px;border:
    black 1px solid;">
      </div>
    </body>
  </html>
```

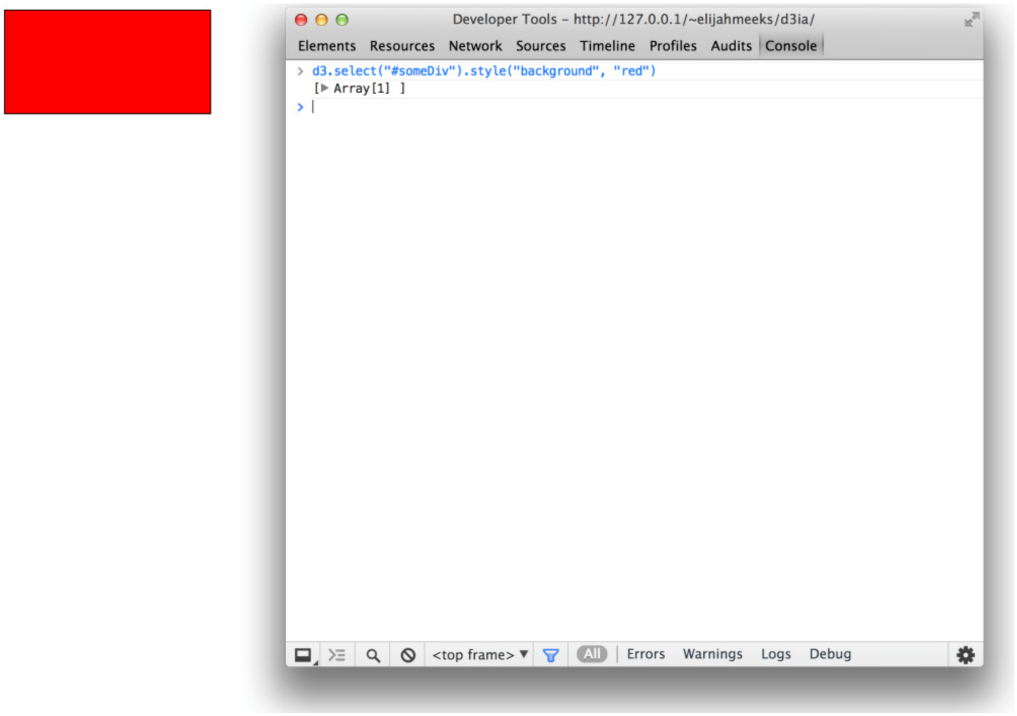
```
element.style {
   width: 200px;
   height: 100px;
   border: ▶■black 1px solid;
}
div {      user agent stylesheet
   display: block;
}
```

Figure 1.11 The developer tools in Chrome place the JavaScript console on the rightmost tab labeled "Console," with the element inspector available using the hourglass on the bottom left or by browsing the DOM in the Elements tab.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

The element inspector allows you to look at the elements that make up your web page by navigating through the DOM (represented as nested text, where each child element is shown indented). You can also select an element onscreen graphically, typically represented as a magnifying glass or cursor icon.

The other screen you'll want to use quite often is the console (figure 1.12), which allows you to write and run JavaScript code right on your web page:



```
> d3.select("#someDiv").style("background", "red")
  [▶ Array[1] ]
> |
```

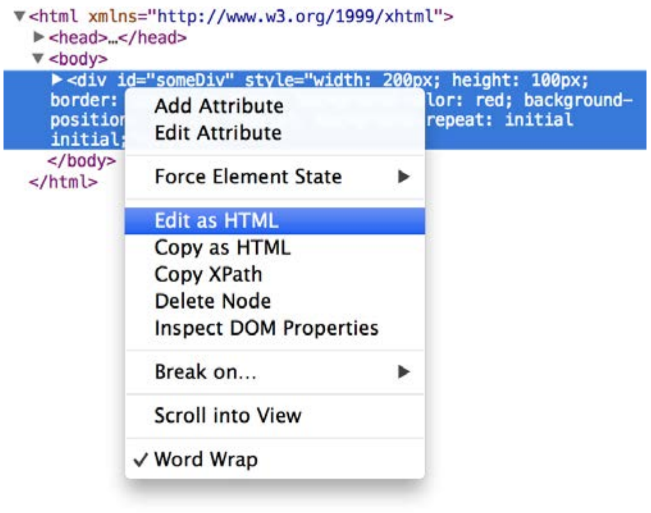
Figure 1.12 You can run JavaScript code in the console and also call global variables or declare new ones as necessary. Any code you write in the console and changes made to the web page are lost as soon as you reload the page.

The examples in this book use Google Chrome and its developer console, but you could use Safari's developer tools or Firebug in Firefox, or whatever developer console you're most comfortable with. You can see and manipulate DOM elements such as <div> or <body> by clicking on the element inspector or looking at the DOM as represented in HTML. You can click one of these elements and change its appearance by modifying it in the console.

You can even delete elements in the console. Give it a try: select the div either in the DOM or visually, and press Delete. Now your web page is very lonely. Press refresh so that your page reloads the HTML and your div comes back. You can adjust the size and color of your div by adding new styles or changing the existing one, so you can increase the width of the border

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

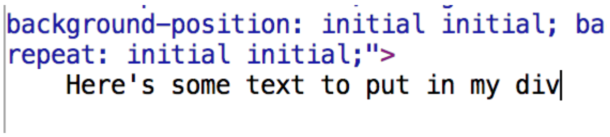
and make it dashed by changing the border style to “black 5px dashed.” You can add content to the div in the form of other elements, or you can add text by right-clicking on the element and selecting “edit as HTML,” as shown in figures 1.13 and 1.14.



```
▼<html xmlns="http://www.w3.org/1999/xhtml">
  ▶<head>…</head>
  ▼<body>
    ▶<div id="someDiv" style="width: 200px; height: 100px;
    border:                            lor: red; background-
    positio                            repeat: initial
    initial;
    </body>
  </html>
```

Figure 1.13 Rather than adding or modifying individual styles and attributes, you have the ability to rewrite the HTML code as you would in a text editor. As with any changes, these only last until you reload the page.

You can then write whatever you’d like in between the opening and closing HTML:

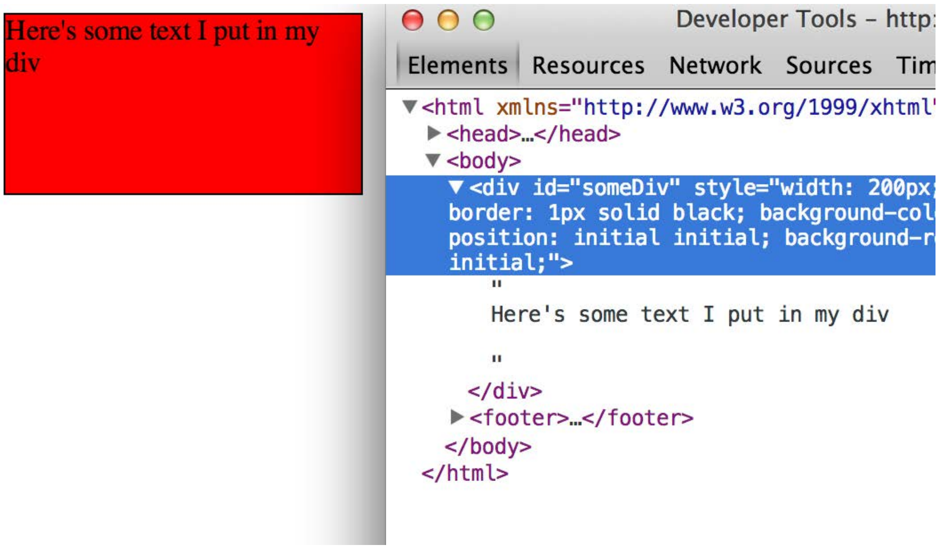


```
background-position: initial initial; ba
repeat: initial initial;">
    Here's some text to put in my div
```

Figure 1.14 Changing the content of a DOM element is as simple as adding text between the opening and ending brackets of the element.

Any changes you make, regardless of whether they’re well structured or not, will be reflected on the web page. In figure 1.15 you see the results of modifying the HTML, which is rendered immediately on your page.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905



Here's some text I put in my div

```
▼<html xmlns="http://www.w3.org/1999/xhtml'
  ▶<head>…</head>
  ▼<body>
    ▼<div id="someDiv" style="width: 200px;
    border: 1px solid black; background-col
    position: initial initial; background-r
    initial;">
        "
        Here's some text I put in my div

        "
      </div>
    ▶<footer>…</footer>
   </body>
</html>
```

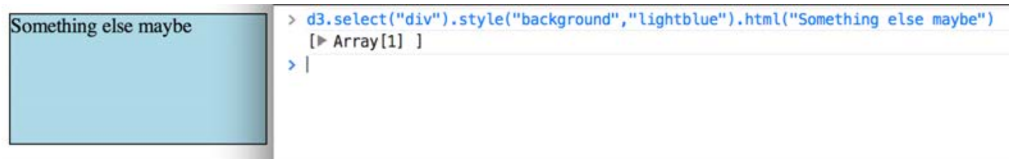
Figure 1.15 The page is updated as soon as you finish making your changes. Writing HTML manually in this way is only useful for planning how you might want to dynamically update the content.

In this way, you could slowly and painstakingly create a web page in the console. You're not going to do that. Instead, you're going to use D3 to create elements on the fly with size, position, shape, and content based on your data.

1.3.2 Coding in the console

You'll do a lot of your coding in the IDE of your choice, but one of the great things about web development is that you can test JavaScript code changes by using your console. Later you'll focus on writing JavaScript, but for now, to demonstrate how the console works, copy the following code into your console and press enter. You should see the effect shown in figure 1.16.

```
d3.select("div").style("background","lightblue").style("border", "solid black
      1px").html("Something else maybe");
```



Something else maybe

```
> d3.select("div").style("background","lightblue").html("Something else maybe")
  [▶ Array[1] ]
> |
```

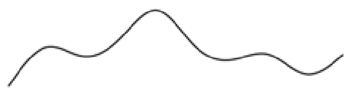
Figure 1.16 The D3 `select` syntax modifies style using the `.style()` function and traditional HTML content using the `.html()` function.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

We'll see a few more uses of traditional HTML elements in this chapter, and then again in chapter 3, but then we won't see traditional DOM elements again until chapter 8, where you'll use D3 to create complex, data-driven spreadsheets and galleries using `<div>`, `<table>`, and `<select>` elements. If all D3 could do was select HTML elements and change their style and content like this, then it wouldn't be that useful for data visualization. To do more, we have to move away from traditional HTML and focus on a special type of element in the DOM: SVG.

1.3.3 SVG

A major value of HTML5 is the integrated support for SVG. These allow for simple mathematical representation of images that scale and are amenable to animation and interaction. Part of the attractiveness of D3 is that it provides an abstraction layer for drawing SVG. This is because SVG drawing can be a little confusing. SVG drawing instructions for complex shapes, known as `<path>` elements, are written a bit like the old LOGO programming language. You start at a point on a canvas and draw a line from that point to another. If you want it to curve, you can give the SVG drawing code coordinates on which take make that curve. So if you want to draw the line on the right, you'd create a `<path>` element in an `<svg>` canvas element in your web page, and you'd set the d attribute of that `<path>` element equal to the text on the left:



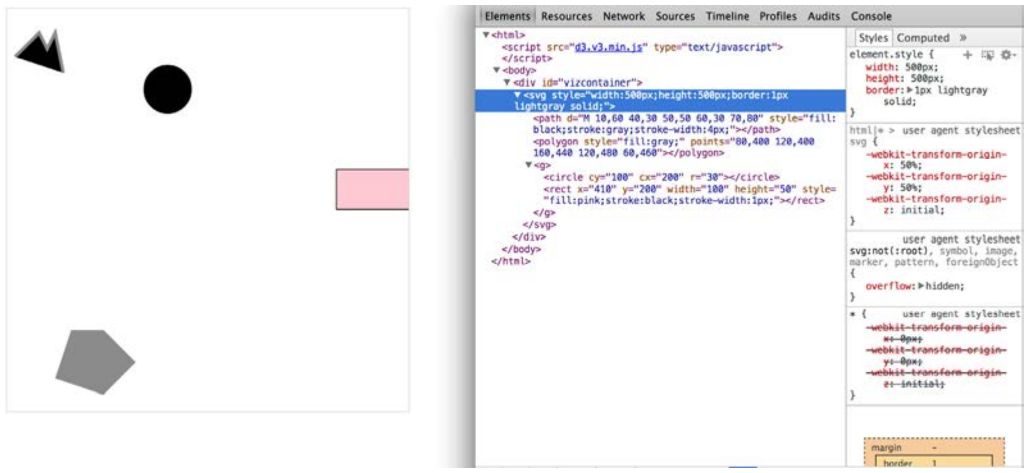
```
M50,485.714857L58.3332,473.809238C66.666
66,461.90487,83.33331,438.09381,99.9999,
433.33333C116.66,428.5714285714285 6,133.
34,442.8571428571428 3,150,445.22C166.669
,447.6190476190476,183.3331,438.09523809
52381,199.9997,421.4285714285714C216.663
,404.7619047619047,233.331,380.952380952
3809,250,383.33C266.667,385.714285714285
67,283.337,414.2857142857143,300,430.952
38095238096C316.6674,447.6190476190476,3
33.337,452.3809523809524,350,450C366.667
,447.6190476190476,383.33,438.0952380952
381,400,442.857142857142 83C416.663,447.6
190476190476,433.33,466.6663,449.994,469
.049C466.66663,471.4285714285714 4,483.33
33333,457.1428571,491.663,450L500,442.85
71429
```

But you'd almost never want to create SVG by manually writing drawing instructions like this. Instead, you'll want to use D3 to do the drawing with a variety of helper functions, or rely on other SVG elements that represent simple shapes (known as geometric or graphical primitives) using more readable attributes. You'll start doing that in chapter 4, where you'll use `d3.svg.line` and `d3.svg.area` to create line and area charts. For now, you'll update d3ia.html to look like the following listing to include the necessary elements to display SVG, as well as examples of the various shapes you might use.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

Listing 1.1 A sample web page with SVG elements

```
<!doctype html>
<html>
   <script src="d3.v3.min.js" type="text/JavaScript">
</script>
<body>
<div id="infovizDiv">
<svg style="width:500px;height:500px;border:1px lightgray solid;">
  <path d="M 10,60 40,30 50,50 60,30 70,80"
     style="fill:black;stroke:gray;stroke-width:4px;" />
  <polygon style="fill:gray;"
     points="80,400 120,400 160,440 120,480 60,460" />
<g>
<line x1="200" y1="100" x2="450" y2="225"
style="stroke:black;stroke-width:2px;"/>
<circle cy="100" cx="200" r="30"/>
<rect x="410" y="200" width="100" height="50"
     style="fill:pink;stroke:black;stroke-width:1px;" />
</g>
</svg>
</div>
</body>
</html>
```



```
▼<html>
   <script src="d3.v3.min.js" type="text/javascript">
   </script>
 ▼<body>
  ▼<div id="vizcontainer">
   ▼<svg style="width:500px;height:500px;border:1px
    lightgray solid;">
      <path d="M 10,60 40,30 50,50 60,30 70,80" style="fill:
      black;stroke:gray;stroke-width:4px;"></path>
      <polygon style="fill:gray;" points="80,400 120,400
      160,440 120,480 60,460"></polygon>
     ▼<g>
        <circle cy="100" cx="200" r="30"></circle>
        <rect x="410" y="200" width="100" height="50" style=
        "fill:pink;stroke:black;stroke-width:1px;"></rect>
      </g>
    </svg>
   </div>
  </body>
</html>
```

```
element.style {
   width: 500px;
   height: 500px;
   border: ▶1px lightgray
      solid;
}
html|* > user agent stylesheet
svg {
   -webkit-transform-origin-
      x: 50%;
   -webkit-transform-origin-
      y: 50%;
   -webkit-transform-origin-
      z: initial;
}
            user agent stylesheet
svg:not(:root), symbol, image,
marker, pattern, foreignObject
{
   overflow: ▶hidden;
}
* {         user agent stylesheet
   ~~-webkit-transform-origin-~~
      ~~x: 0px;~~
   ~~-webkit-transform-origin-~~
      ~~y: 0px;~~
   ~~-webkit-transform-origin-~~
      ~~z: initial;~~
}
```

Figure 1.17 Inspecting the DOM of a web page with an SVG canvas reveals the nested graphical elements as well as the style and attributes that determine their position. Notice that the circle and rectangle exist as child elements of a group.

You can inspect the elements like you would the traditional elements we looked at earlier, as you can see in figure 1.17, and you can manipulate these elements using traditional JavaScript selectors like `document.getElementById` or with D3, removing them or changing the style like so:

```
   d3.select("circle").remove() #a
   d3.select("rect").style("fill", "purple") #b
```

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

#a Deletes the circle
#b Changes the rectangle color to purple

Now, refresh your page and let's take a look at the new elements. You're familiar with divs, and it's useful to put an SVG canvas in a div so you can access the parent container for layout and styling. Let's take a look at each of the elements we've added:

<SVG>

This is your canvas on which everything is drawn. The top-left corner is 0,0, and the canvas clips anything drawn beyond its defined height and width of 500,500 (the rectangle in your example). An <svg> element can be styled with CSS to have different borders and backgrounds. The <svg> element can also be dynamically resized using the `viewBox` attribute, which is more complex and beyond the scope of the overview here.

You can use CSS (which we'll touch on later in this chapter) to style your SVG canvas or use D3 to add inline styles like so:

```
d3.select("svg").style("background", "darkgray"); #a
```

#a Infoviz is always cooler on a dark background.

NOTE: The x-axis is drawn left to right, but the y-axis is drawn top to bottom, so you'll see that the circle is set 200 pixels to the right and 100 pixels down.

<canvas>

There's a second mode of drawing available with HTML5 using <canvas> elements to draw bitmaps. We won't be going into detail here, but you'll see this method used in chapter 8 and again in chapter 11. The <canvas> element creates static graphics drawn in a manner similar to SVG that can then be saved as images. There are four main reasons to use canvas:

Compatibility—We won't worry about this because if you're using D3, then you're coding for a modern browser.

Creating static images—You can draw your data visualization with canvas to save views as snapshots for thumbnail and gallery views (this is what you'll do in chapter 8).

Large amounts of data—SVG creates individual elements in the DOM, and although this is great for attaching events and styling, it can overwhelm a browser and cause significant slowdown (this is what you'll use canvas for in chapter 11).

WebGL—The <canvas> element allows you to use WebGL to draw, so that you can create 3D objects. You can also create 3D objects like globes and polyhedrons using SVG, which we'll get into a bit in chapter 7 as we examine geospatial information visualization.

<CIRCLE> <RECT> <LINE> <POLYGON>

SVG provides a set of common shapes, each of which has *attributes* that determine their size and position to make them easier to deal with than the generic "d" attribute you saw earlier.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

These attributes vary depending on the element that you're dealing with, so that <rect> has "x" and "y" attributes that determine the shape's top-left corner, as well as "height" and "width" attributes that determine its overall form. In comparison, the <circle> element has "cx" and "cy" attributes that determine the center of the circle, and an "r" attribute that determines the radius of the circle. The <line> element has "x1" and "y1" attributes that determine the starting point of the line and "x2" and "y2" attributes that determine its end point. There are other simple shapes that are similar to these, such as the <ellipse>, and there are more complex shapes, like the <polygon> that has a "points" attribute that holds a set of comma-separated xy coordinates in clockwise order that determines the area bounded by the polygon.

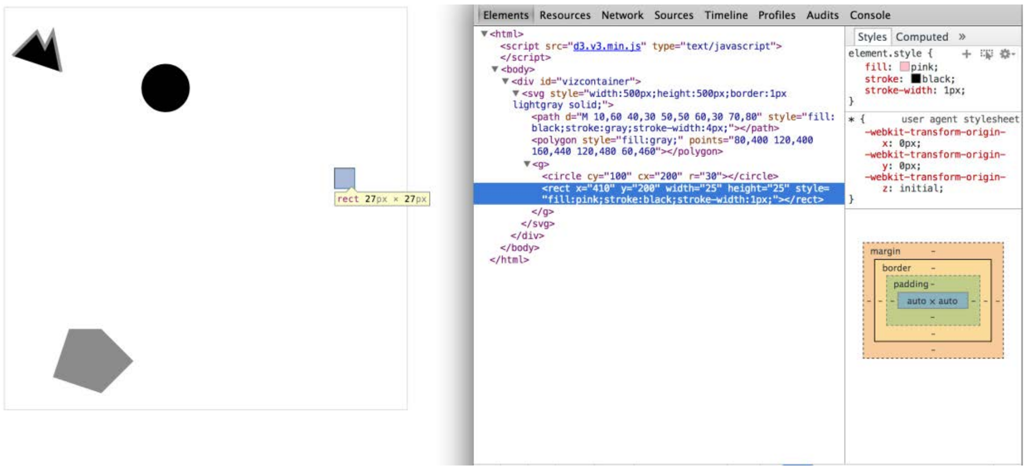
Infoviz vocabulary: geometric primitive

Accomplished artists can draw anything with vector graphics, but you're probably not looking at D3 because you're an artist. Instead, you're dealing with graphics with more pragmatic goals in mind. From that perspective, it's important to understand the concept of geometric primitives (also known as graphical primitives). Geometric primitives are simple shapes such as points, lines, circles, and rectangles. These shapes, which can be combined to make more complex graphics, are particularly useful for visually displaying information.

Primitives are also useful for understanding complex information visualization that you see out in the real world. Dendrograms like the one shown in figure 1.20 are far less intimidating when you realize they're just circles and lines. Interactive timelines are easier to understand and create when you think of them as collections of rectangles and points. Even geographic data, which primarily comes in the form of polygons, points, and lines, is less confusing when you break it down into its most basic graphical structures.

Each of these attributes can be hand-edited in HTML to adjust their size, form, and position. Open up your element inspector, and click the <rect>. Change its "width" to 25 and its "height" to 25 as shown in figure 1.18.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905



```
▼<html>
   <script src="d3.v3.min.js" type="text/javascript">
   </script>
  ▼<body>
    ▼<div id="vizcontainer">
      ▼<svg style="width:500px;height:500px;border:1px
       lightgray solid;">
         <path d="M 10,60 40,30 50,50 60,30 70,80" style="fill:
         black;stroke:gray;stroke-width:4px;"></path>
         <polygon style="fill:gray;" points="80,400 120,400
         160,440 120,480 60,460"></polygon>
        ▼<g>
           <circle cy="100" cx="200" r="30"></circle>
           <rect x="410" y="200" width="25" height="25" style=
           "fill:pink;stroke:black;stroke-width:1px;"></rect>
         </g>
       </svg>
     </div>
   </body>
</html>
```

```
element.style {
   fill: pink;
   stroke: black;
   stroke-width: 1px;
}
* {          user agent stylesheet
   -webkit-transform-origin-
      x: 0px;
   -webkit-transform-origin-
      y: 0px;
   -webkit-transform-origin-
      z: initial;
}
```

Figure 1.18 Modifying the height and width attributes of a `<rect>` element changes the appearance of that element. Inspecting the element also shows how the stroke adds to the computed size of the element.

Now you've learned why there's no SVG <square>. The color, stroke, and transparency of any shape can be changed by adjusting the style of the shapes, with `"fill"` determining the color of the area of the shape and `"stroke"`, `"stroke-width”`, `"stroke-dasharray"` determining its outline.

Notice, though, that the inspected element has a measurement of 27 px x 27 px. That's because the 1-px stroke is drawn on the outside of the shape. That makes sense, once you know the rule, but if you change the stroke-width to “2 px” it will still be 27 px x 27 px. That's because the stroke is drawn evenly over the inside and outside borders. This may not seem too big a deal, but it's something to remember when you're trying to line up your shapes later on.

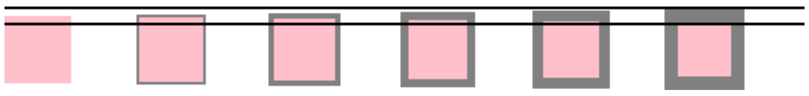


Figure 1.19 The same 25 x 25 `<rect>` with no, 1-px, 2-px, 3-px, 4-px, and 5-px strokes. Though these are drawn on a retina screen using half-pixels, the second and third report the same width and height (27 px x 27 px) as the fourth and fifth (29 px x 29 px).

Change the style parameters of the rectangle to the following:

`"fill:purple;stroke-width:5px;stroke:cornflowerblue;".`

Congratulations! You've now successfully visualized the complex and ambiguous phenomenon known as “ugly.”

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

<TEXT>

SVG provides the capacity to write text as well as shapes. SVG text, though, doesn't have the formatting support found in HTML elements, and so it's primarily used for labels. If you do want to do basic formatting, you can nest `<tspan>` elements in `<text>` elements.

<G>

The <g> or group element is distinct from the SVG elements we've discussed in that it has no graphical representation and doesn't exist as a bounded space. Instead, it's a logical grouping of elements. You'll want to use <g> elements extensively when creating graphical objects that are made up of several shapes and text. For instance, if you wanted to have a circle with a label above it and move the label and the circle at the same time, then you'd place it inside a <g> element:

```
<g>
<circle r="2"/>
<text>This circle’s Label</text>
</g>
```

Moving a <g> around your canvas requires you to adjust the `"transform"` attribute of the <g> element. The `"transform"` attribute is more intimidating than the various xy attributes of shapes, because it accepts a structured description in text of how you want to transform a shape. One of those structures is `translate()`, which accepts a pair of coordinates that move the element to the x and y position defined by the values in `translate(x,y)`. So if you want to move a <g> element 100 pixels to the right and 50 pixels down, then you need to set its `"transform"` attribute to `transform="translate(100,50) "`. The transform attribute also accepts a `scale()` setting, so that you can change the rendered scale of the shape. You can see these settings in action by modifying the previous example with the results shown in figure 1.20.

```
<g>
<circle r="2"/>
<text>This circle’s Label</text>
</g>
<g transform="translate(100,50)">
<circle r="2" />
<text>This circle’s Label</text>
</g>
<g transform="translate(100,400) scale(2.5)">
<circle r="2"/>
<text>This circle’s Label</text>
</g>
```

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

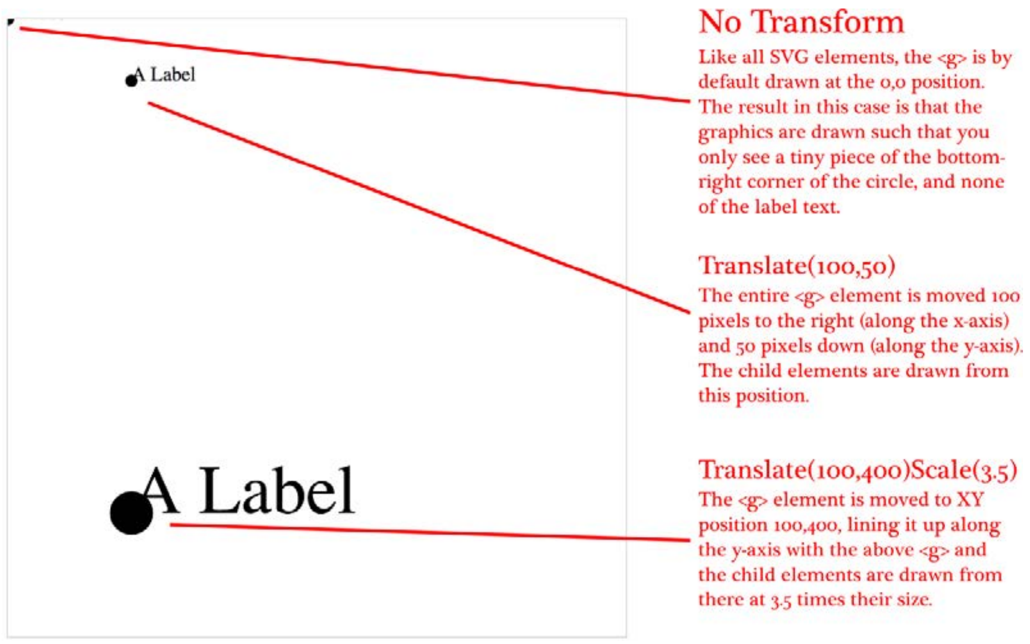

A Label
A Label
No Transform
Like all SVG elements, the <g> is by default drawn at the 0,0 position. The result in this case is that the graphics are drawn such that you only see a tiny piece of the bottom-right corner of the circle, and none of the label text.
Translate(100,50)
The entire <g> element is moved 100 pixels to the right (along the x-axis) and 50 pixels down (along the y-axis). The child elements are drawn from this position.
Translate(100,400)Scale(3.5)
The <g> element is moved to XY position 100,400, lining it up along the y-axis with the above <g> and the child elements are drawn from there at 3.5 times their size.

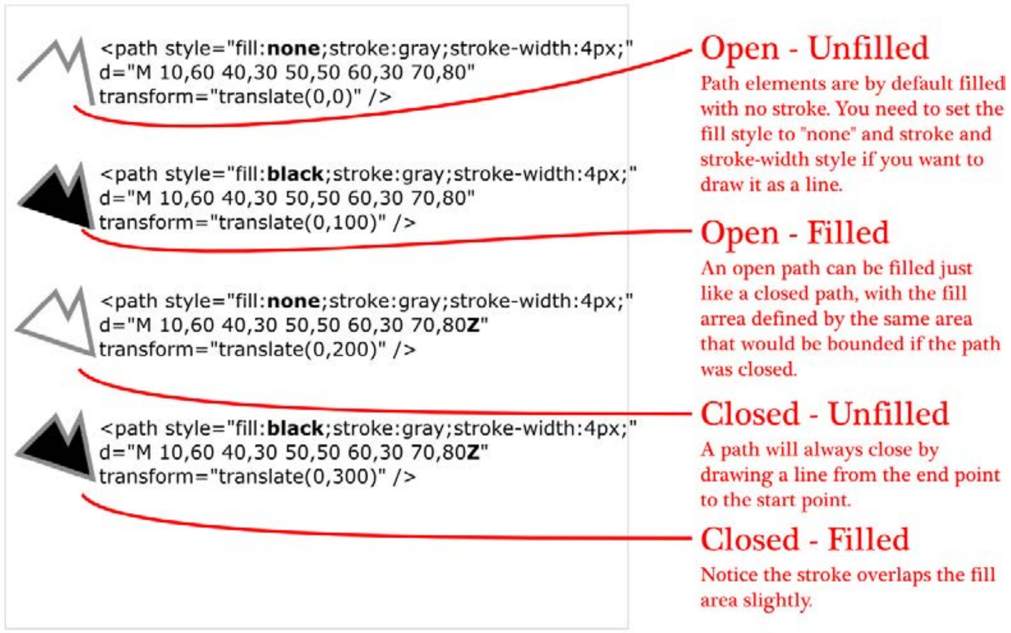
Figure 1.20 All SVG elements can be affected by the `transform` attribute, but this is particularly salient when working with <g> elements, which require this approach to adjust their position. The child elements are drawn by using the position of their parent <g> as their relative 0,0 position. The `scale()` setting in the `transform` attribute then affects the scale of any of the size and position attributes of the child elements.

<PATH>

A path is an area determined by its "d" attribute. Paths can be open or closed, meaning the last point connects to the first if closed and doesn’t if open. The open or closed nature of a path is determined by the absence or presence of the letter “Z” at the end of the text string in the "d" attribute. It can still be filled either way. You can see the difference in figure 1.21.

```
<path style="fill:none;stroke:gray;stroke-width:4px;"
     d="M 10,60 40,30 50,50 60,30 70,80" transform="translate(0,0)" />
<path style="fill:black;stroke:gray;stroke-width:4px;"
     d="M 10,60 40,30 50,50 60,30 70,80" transform="translate(0,100)" />
<path style="fill:none;stroke:gray;stroke-width:4px;"
     d="M 10,60 40,30 50,50 60,30 70,80Z" transform="translate(0,200)" />
<path style="fill:black;stroke:gray;stroke-width:4px;"
     d="M 10,60 40,30 50,50 60,30 70,80Z" transform="translate(0,300)" />
```

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905


```
<path style="fill:none;stroke:gray;stroke-width:4px;"
d="M 10,60 40,30 50,50 60,30 70,80"
transform="translate(0,0)" />

<path style="fill:black;stroke:gray;stroke-width:4px;"
d="M 10,60 40,30 50,50 60,30 70,80"
transform="translate(0,100)" />

<path style="fill:none;stroke:gray;stroke-width:4px;"
d="M 10,60 40,30 50,50 60,30 70,80Z"
transform="translate(0,200)" />

<path style="fill:black;stroke:gray;stroke-width:4px;"
d="M 10,60 40,30 50,50 60,30 70,80Z"
transform="translate(0,300)" />
```

Open - Unfilled
Path elements are by default filled with no stroke. You need to set the fill style to "none" and stroke and stroke-width style if you want to draw it as a line.

Open - Filled
An open path can be filled just like a closed path, with the fill arrea defined by the same area that would be bounded if the path was closed.

Closed - Unfilled
A path will always close by drawing a line from the end point to the start point.

Closed - Filled
Notice the stroke overlaps the fill area slightly.

Figure 1.21 Each path shown here uses the same coordinates in its `"d"` attribute, with the only differences between them being the presence or absence of “Z” at the end of the text string defining the `"d"` attribute, the settings for fill and stroke, and the position via the `“transform”` attribute.

Although sometimes you may want to write that "d" attribute yourself, it’s more likely that your experience crafting SVG will come in one of three ways: using geometric primitives like circles, rectangles, or polygons; drawing SVG using a vector graphics editor like Adobe Illustrator or Inkscape; or drawing SVGs parametrically using handwritten constructors or built-in constructors in D3. Most of this book will focus on using D3 to create SVG, but don’t overlook the possibility of creating SVG in another and then manipulating it using D3 like you’ll do using d3.html in chapter 3.

1.3.4 CSS

CSS are used to style the elements in the DOM. A style sheet can exist as a separate .css file that you include in your HTML page or can be embedded directly in the HTML page. Style sheets refer to an ID, class, or type of element and determine the appearance of that element. The terminology used to define the style is known as a CSS selector and is the same type of selector used in the `d3.select()` syntax. You can set inline styles (that are applied to only a single element) by using `d3.select("#someElement").style("opacity", .5)` to set the opacity of an element to 50%. Let’s update your d3ia.html to include a style sheet as shown in the following listing.

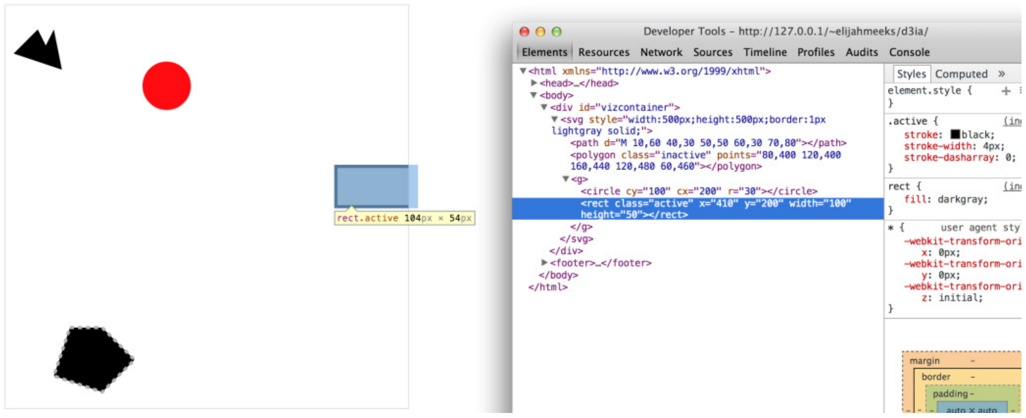
©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

Listing 1.2 A sample web page with a style sheet

```
<!doctype html>
<html>
<script src="d3.v3.min.js" type="text/JavaScript"></script>
<style>
.inactive, .tentative {
  stroke: darkgray;
  stroke-width: 2px;
  stroke-dasharray: 5 5;
}

.tentative {
  opacity: .5;
}

.active {
  stroke: black;
  stroke-width: 4px;
  stroke-dasharray: 1;
}

circle {
  fill: red;
}

rect {
  fill: darkgray;
}
</style>
<body>
<div id="infovizDiv">
<svg style="width:500px;height:500px;border:1px lightgray solid;">
  <path d="M 10,60 40,30 50,50 60,30 70,80" />
  <polygon class="inactive" points="80,400 120,400 160,440 120,480 60,460"  />
<g>
<circle class="active tentative" cy="100" cx="200" r="30"/>
<rect class="active" x="410" y="200" width="100" height="50" />
</g>
</svg>
</div>
</body>
</html>
```

The results stack on each other, so when you examine the rectangle element, as shown in figure 1.22, you see that its style is set by the reference to `rect` in the style sheet as well as the class attribute of `active`.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

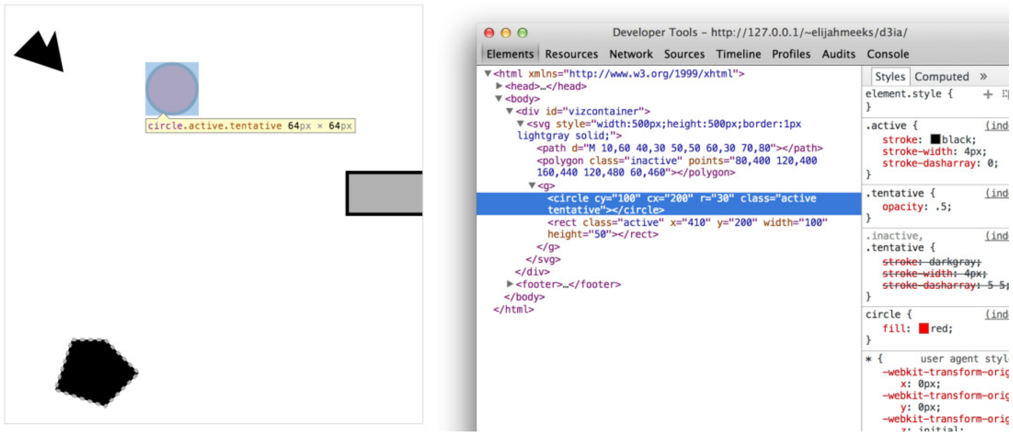


```
▼<html xmlns="http://www.w3.org/1999/xhtml">
  ▶<head>…</head>
  ▼<body>
    ▼<div id="vizcontainer">
      ▼<svg style="width:500px;height:500px;border:1px
      lightgray solid;">
          <path d="M 10,60 40,30 50,50 60,30 70,80"></path>
          <polygon class="inactive" points="80,400 120,400
          160,440 120,480 60,460"></polygon>
        ▼<g>
            <circle cy="100" cx="200" r="30"></circle>
            <rect class="active" x="410" y="200" width="100"
            height="50"></rect>
          </g>
        </svg>
      </div>
    ▶<footer>…</footer>
    </body>
  </html>
```

```
element.style {
}
.active {
   stroke: ■black;
   stroke-width: 4px;
   stroke-dasharray: 0;
}
rect {
   fill: darkgray;
}
* {       user agent sty
   -webkit-transform-ori
      x: 0px;
   -webkit-transform-ori
      y: 0px;
   -webkit-transform-ori
      z: initial;
}
```

Figure 1.22 Examining an SVG rectangle in the console shows that it inherits its fill style from the CSS style applied to `<rect>` types and its stroke style from the `.active` class.

Style sheets can also refer to a state of the element, so that with `:hover` you can change the way an element looks when the user mouses over that element. You can learn about other complex CSS selectors in more detail in a book devoted to that subject. For this book, we'll focus mostly on using CSS classes and IDs for selection and to change style. The most useful way to do this is to have CSS classes associated with particular stylistic changes and then change the class of an element. You can change the class of an element, which is an attribute of an element, by selecting and modifying the class attribute. The circle shown in figure 1.23 is affected by two overlapping classes: `.active` and `.tentative`.


circle.active.tentative 64px × 64px
Developer Tools – http://127.0.0.1/~elijahmeeks/d3ia/
Elements
Resources
Network
Sources
Timeline
Profiles
Audits
Console
Styles
Computed

```
▼<html xmlns="http://www.w3.org/1999/xhtml">
  ▶<head>…</head>
  ▼<body>
    ▼<div id="vizcontainer">
      ▼<svg style="width:500px;height:500px;border:1px
      lightgray solid;">
          <path d="M 10,60 40,30 50,50 60,30 70,80"></path>
          <polygon class="inactive" points="80,400 120,400
          160,440 120,480 60,460"></polygon>
        ▼<g>
            <circle cy="100" cx="200" r="30" class="active
            tentative"></circle>
            <rect class="active" x="410" y="200" width="100"
            height="50"></rect>
          </g>
        </svg>
      </div>
    ▶<footer>…</footer>
    </body>
  </html>
```

```
element.style {
}
.active {
   stroke: ■black;
   stroke-width: 4px;
   stroke-dasharray: 0;
}
.tentative {
   opacity: .5;
}
.inactive,
.tentative {
   ~~stroke: darkgray;~~
   ~~stroke-width: 4px;~~
   ~~stroke-dasharray: 5 5;~~
}
circle {
   fill: 🟥red;
}
* {       user agent styl
   -webkit-transform-orig
      x: 0px;
   -webkit-transform-orig
      y: 0px;
   -webkit-transform-orig
```

Figure 1.23 The SVG circle has its fill value set by its type in the style sheet, with its opacity set by its membership in the `.tentative` class and its stroke set by its membership in the `.active` class. Notice that the stroke settings from the `.tentative` class are overwritten by the stroke settings in the later declared `.active` class.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

Here you defined a couple of overlapping classes, with tentative, active, and inactive all applying different style changes to your shape. When an element is in only one of these classes, you can overwrite the class attribute entirely:

```
d3.select("circle").attr("class", "tentative");
```

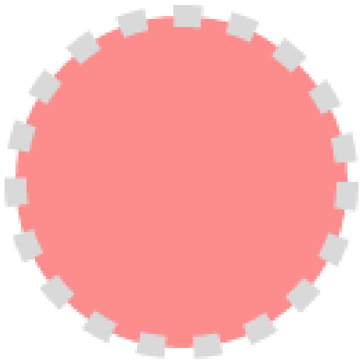


Figure 1.24 An SVG circle with fill style determined by its type and its opacity and stroke settings determined by its membership in the `tentative` class.

The results, as shown in figure 1.24, are what we would expect. This overwrites the entire class attribute to the value you set. But elements can have multiple classes, and sometimes an element is both active and tentative or inactive and tentative, so let’s reload the page and take advantage of the helper function `d3.classed()`, which allows you to add or remove a class from the classes in an element:

```
d3.select("circle").classed("active", true);
```

By using `.classed()`, you don’t overwrite the existing attribute, but rather append or remove the named class from the list. You can see the results of two classes with conflicting styles defined. The active style overwrites the tentative style because it occurs later in the style sheet. Another rule to remember is that more specific rules overwrite more general rules. There’s more to CSS, but this book won’t go into that that.

By defining style in your style sheet and changing appearance based on class membership, you create code that’s more maintainable and readable. You’ll need to use inline styles to set the graphical appearance of a set of elements to a variety of different values, for example, changing the fill color to correspond to a color ramp based on the data bound to that set of elements. You’ll see that functionality in action later when you deal with bound data. But as a general rule, setting inline styles should only be used when you can’t use traditional classes and states defined in a style sheet.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

1.3.5 JavaScript

D3, like many information visualization libraries in JavaScript, provides functions to abstract the process of creating and modifying web page elements. On top of that, it provides mechanisms to link data and web page elements in a way that makes the drawing and updating of these SVG elements reusable and maintainable. But these mechanisms are also applicable to more traditional HTML elements like paragraphs and divs.

As a result, a web application written in D3 can accomplish much of the UI functionality that users expect without relying on libraries like jQuery. This is because the latest version of JavaScript has built-in functionality that used to be available only with jQuery. If you read the solutions on StackOverflow, you may think that being a JavaScript developer requires being a jQuery developer, but unless you're developing for an audience that uses a browser that won't support the key features of D3, or you need to use a plugin that requires jQuery, then you might just as easily write the same functionality in JavaScript.

When writing JavaScript with D3, you should be familiarize yourself with two subjects: method chaining and arrays.

METHOD CHAINING

D3 examples, like many examples written in JavaScript, use method chaining extensively. Method chaining, also known as function chaining, is facilitated by returning the method itself with the successful completion of functions associated with a method. One way to think of method chaining is to think of how we talk and refer to each other. Imagine you were talking to someone at a party, and you asked about another guest:

"What's her name?"

"Her name is Lindsay."

"Where does she work?"

"She works at Tesla."

"Where does she live?"

"She lives in Cupertino."

"Does she have any children?"

"Yes, she has a daughter."

"What's her name?"

Do you think the answer to that last question would be "Lindsay"? Of course not. You'd expect the answer to refer to Lindsay's daughter, even though all the previous questions referred to Lindsay. Method chaining is like that. It returns the same function as long as you use getter and setter methods of that function, and returns the new function when you call a method that creates something new. Method chaining is used a lot in D3 examples, which means you'll see something like this written on one line or formatted (but functionally identical) to something written on multiple lines:

```
d3.selectAll("div").data(someData).enter().append("div").html("Wow").append("span"
       ).html("Even More Wow").style("font-weight", "900");
```

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

That line is the same as the following code. The only change is in the use of line breaks, which JavaScript ignores.

```
d3.selectAll("div") #a
.data(someData) #b
.enter() #c
.append("div") #d
.html("Wow") #e
.append("span") #f
.html("Even More Wow") #g
.style("font-weight", "900"); #h
```

#a Returns the Function 1, a selection
#b Sets the data on Function 1, and returns Function 1
#c Returns Function 2, the selection.enter() function
#d Sets .append() behavior on function 2, and returns function 3, a selection
#e Sets .html() for function 3, and returns function 3
#f Sets the append() behavior on function 3, and returns function 4
#g Sets the html() for function 4, and returns function 4
#h Sets the font-weight style on function 4, and returns function 4

You could write each line separately, declaring the different variables as you go, and achieve the same effect. That might make more sense if you haven't been exposed to method chaining before.

```
var function1 = d3.selectAll("div");
var function1withData = function1.data(someData);
var function2 = function1withData.enter();
var function3 = function2.append("div");
function3.html("Wow");
var function4 = function3.append("span");
function4.html("Even More Wow");
function4.style("font-weight", "900");
```

You can see this when you run the code in your console. This is the first time you've used the `.data()` function, which along with `.select()` is at the core of developing with D3. When you use `.data()`, you bind each element in your selection to each item in an array. If you have more items in your array than elements in your selection, then you can use the `.enter()` function to define what to do with each extra element. In the previous function, you select all the <div> elements in the <body> and the `.enter()` function tells D3 to `.append()` a new div when there are more elements in the array than elements in the selection. Given that your d3ia.html page already has one div, if you bind an array with more than one value, D3 appends, or adds, a div for each value in the array beyond the first.

A corresponding `.exit()` function defines how to respond when an array has fewer values than a selection. For now, you're going to run the code as it appears in the examples, and in later chapters you'll get into much more detail on the way selections and binding work.

With this example, you're not doing anything with the data in the array and only creating elements based on the size of the array (one <div> for each element in the array). This

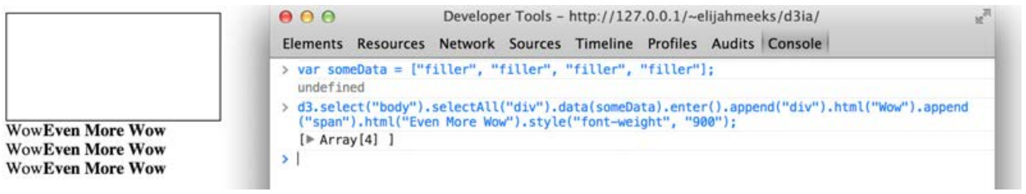
©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

example assumes that you already have a <div> in your html with a black border (as seen in Figure 1.25). Here’s the HTML that would get that done:

```
<!doctype html>
<html>
<script src="d3.v3.min.js" type="text/JavaScript"></script>
<style>
#borderdiv {
width: 200px;
height: 50px;
border: 1px solid gray;
}
</style>
<body>
<div id="borderdiv"></div>
</body>
</html>
```

For this to work, you need to give `someData` a value. With that in place, you can run your code:

```
var someData = ["filler", "filler", "filler", "filler"];
d3.select("body").selectAll("div")
.data(someData)
.enter()
.append("div")
.html("Wow")
.append("span")
.html("Even More Wow")
.style("font-weight", "900");
```


Developer Tools – http://127.0.0.1/~elijahmeeks/d3ia/
Elements Resources Network Sources Timeline Profiles Audits Console

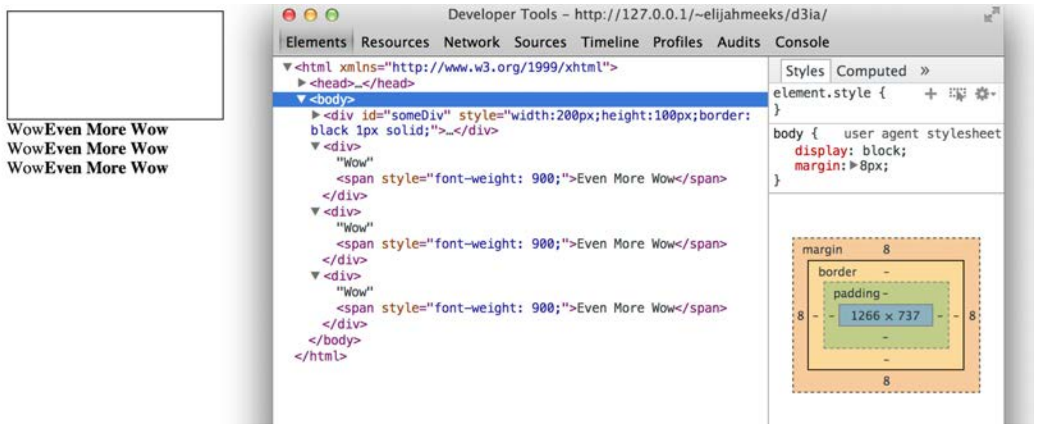
```
WowEven More Wow
WowEven More Wow
WowEven More Wow
```

```
> var someData = ["filler", "filler", "filler", "filler"];
  undefined
> d3.select("body").selectAll("div").data(someData).enter().append("div").html("Wow").append
  ("span").html("Even More Wow").style("font-weight", "900");
  [▶ Array[4] ]
> |
```

Figure 1.25 By binding an array of four values to a selection of `<div>` elements on the page, the `.enter()` function created three new `<div>` elements to reflect the size mismatch between the data array and the selection.

The result, as shown in figure 1.25, is the addition of three lines of text. It might surprise you that this code is three lines, given that the array has four values. Although the data was bound to the existing <div> element on the page, the actions that changed the contents were only applied to the `.enter()` function. They were only applied to the newly created <div> elements that were “entering” the DOM for the first time.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905


Developer Tools – http://127.0.0.1/~elijahmeeks/d3ia/
Elements Resources Network Sources Timeline Profiles Audits Console
Styles Computed »
margin 8
border -
padding -
1266 × 737

WowEven More Wow
WowEven More Wow
WowEven More Wow

```
▼<html xmlns="http://www.w3.org/1999/xhtml">
  ▶<head>…</head>
  ▼<body>
    ▶<div id="someDiv" style="width:200px;height:100px;border:
    black 1px solid;">…</div>
    ▼<div>
        "Wow"
        <span style="font-weight: 900;">Even More Wow</span>
     </div>
    ▼<div>
        "Wow"
        <span style="font-weight: 900;">Even More Wow</span>
     </div>
    ▼<div>
        "Wow"
        <span style="font-weight: 900;">Even More Wow</span>
     </div>
   </body>
 </html>
```

```
element.style {
}
body {    user agent stylesheet
   display: block;
   margin:▶8px;
}
```

Figure 1.26 Inspecting the DOM shows that the new `<div>` elements have been created with unformatted content followed by the child `<span>` element with style and content set by your code.

When you inspect the DOM, as shown in figure 1.26, you see that the method chaining operated in the manner just described. A `<div>` was added, and its HTML content was set to “Wow”. A `<span>` element with a different style was appended to the `<div>`, and its HTML content was set to “Even More Wow”. There’s more you can do, but first you need to examine the array object you’re binding, and focus on JavaScript arrays and array functions.

ARRAYS AND ARRAY FUNCTIONS

D3 is all about arrays, and so it’s important to understand the structure of arrays and the options available to you to prepare those arrays for binding to data. Your array might be an array of string or number literals, such as this:

```
someNumbers = [17, 82, 9, 500, 40];
someColors = ["blue", "red", "chartreuse", "orange"];
```

Or it may be an array of JSON objects, which will become more common as you do more interesting things with D3:

```
somePeople = [{name: "Peter", age: 27}, {name: "Sulayman", age: 24},
{name: "K.C.", age: 49}];
```

One example of a useful array function is `.filter()`, which returns an array whose elements satisfy a test you provide. For instance, here’s how to create an array out of `someNumbers` that had values greater than 40:

```
someNumbers.filter(function(el) {return el >= 40});
```

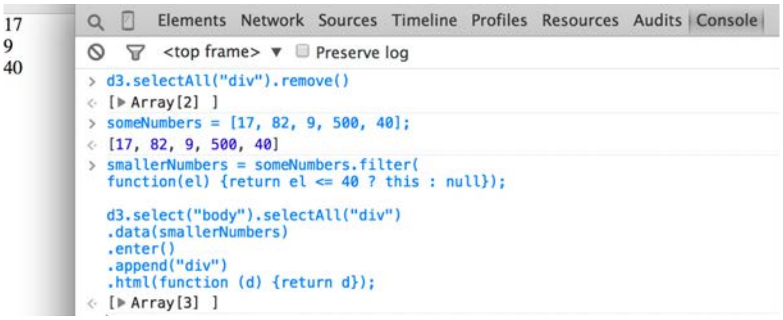
Likewise, here’s how you could create an array out of `someColors` with words shorter than five letters:

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

```
someColors.filter(function(el) {return el.length < 5});
```

The function `.filter()` is a method of an array and accepts a function that iterates through the array with the variable you've named. In this function, you name that variable "el", and the function runs a test on each value by testing on "el". When that test evaluates `true`, the element is kept in our new array.

The result of this `.filter()` function, which you can see in figure 1.27, returns either the element or nothing, depending on if it satisfies the test, building a new array consisting only of the elements that do.



```
> d3.selectAll("div").remove()
< [▶ Array[2] ]
> someNumbers = [17, 82, 9, 500, 40];
< [17, 82, 9, 500, 40]
> smallerNumbers = someNumbers.filter(
  function(el) {return el <= 40 ? this : null});

  d3.select("body").selectAll("div")
  .data(smallerNumbers)
  .enter()
  .append("div")
  .html(function (d) {return d});
< [▶ Array[3] ]
```

Figure 1.27 Running JavaScript in the console allows you to test your code. Here you've created a new array called `smallerNumbers` that consists of only three values, which you can then use as your data in a selection to update and create new elements.

```
smallerNumbers = someNumbers.filter(
function(el) {return el <= 40 ? this : null});

d3.select("body").selectAll("div")
.data(smallerNumbers)
.enter()
.append("div")
.html(function (d) {return d});
```

The resulting code creates two new divs from your three-value array `smallerNumbers`. (Remember that one div already exists and so the `.enter()` function doesn't trigger though data is bound to that existing div.). The contents of the div are the values in your array. This is done through an anonymous function (sometimes referred to in D3 examples as an "accessor") in your `.html()` function and is another key aspect of D3. Any anonymous function called when setting the `.style()`, `.attr()`, `.property()`, `.html()`, or other function of a selection can provide you with the data bound to that selection. As you explore examples, you'll see this function deployed over and over again:

```
.style("background", function(d,i) {return d})
.attr("cx", function(d,i) {return i})
.html(function(d,i) {return d})
```

In every case, the first variable (typically represented with the letter "d", but you can declare it as whatever you want) contains the data value bound to that element, and the

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

second variable returns the array position (known as an index, hence the variable name “i”) of the value bound to that element. This may seem a bit strange, but you’ll get used to it as you see it used in a variety of ways in the upcoming chapters.

JavaScript has many other array functions, and you can do much more than we covered here, but that’s the subject of several other books. It’s time to look at the kinds of data you’ll work with.

1.4 Data standards

Standardization of methods of displaying data has been fed by and feeds into standardization of methods of formatting that data. Data can be formatted in a variety of manners for a variety of purposes, but it tends to fall in a few recognizable classes: tabular data, nested data, network data, geographic data, raw data, and objects.

1.4.1 Tabular data

Tabular data appears in columns and rows typically found in a spreadsheet or a table in a database. Although you invariably end up creating arrays of objects in D3, it’s often more efficient and easier to pull in data in tabular format. Tabular data is delimited with a particular character, and that delimiter determines its format. You can have Comma-Separated Values (CSV), where the delimiter is a comma, or tab-delimited values, or a semicolon or a pipe symbol acting as the delimiter. For instance, you may have a spreadsheet of user information that includes age and salary. If you export it in a delimited form, it will look like table 1.1.

Table 1.1 Delimited data can be expressed in different forms. Here a dataset stores name, age, and a salary of two people using commas, spaces, or the bar symbol to delimit the different fields.

name,age,salary	name age salary	name|age|salary
Sal,34,50000	Sal 34 50000	Sal|34|50000
Nan,22,75000	Nan 22 75000	Nan|22|75000

D3 provides three different functions to pull in tabular data: `d3.csv()`, `d3.tsv()`, and `d3.dsv()`. The only difference between them is that `d3.csv()` is built for comma-delimited files, `d3.tsv()` is built for tab-delimited files, and `d3.dsv()` allows you to declare the delimiter. You’ll see them in action throughout the book.

1.4.2 Nested data

Data that is nested, with objects existing as children of objects recursively, is very common. Many of the most intuitive layouts in D3 are based on nested data, which can be represented as trees, such as the one in figure 1.28, or packed in circles or boxes. Data isn’t often output in such a format, and requires a bit of scripting to organize it as such, but the flexibility of this representation is worth the effort. You’ll see hierarchical data in detail in chapter 5 when we look at various popular D3 layouts.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

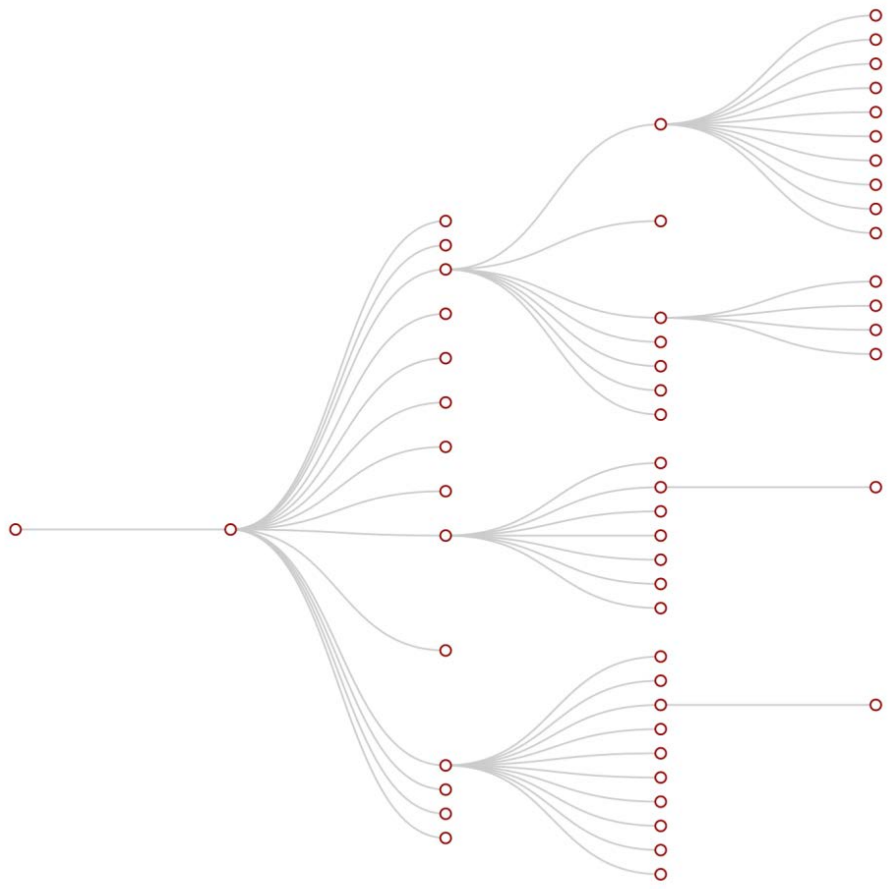


Figure 1.28 Nested data represents parent/child relationships of objects, typically with each object having an array of child objects, and is represented in a number of forms, such as this dendrogram. Notice that each object can have only one parent.

1.4.3 Network data

Networks are everywhere. Whether they're the raw output of social networking streams, transportation networks, or a flowchart, networks are a powerful method of delivering an understanding of complex systems. Networks are often represented as node-link diagrams as shown in figure 1.29. Like geographic data, network data has many standards, but this text focuses only on two forms: node/edge lists and connected arrays. Network data can also be easily transformed into these data types by using a freely available network analysis tool like Gephi (available at gephi.org). We'll examine network data and network data standards when we deal with network visualization in chapter 6.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

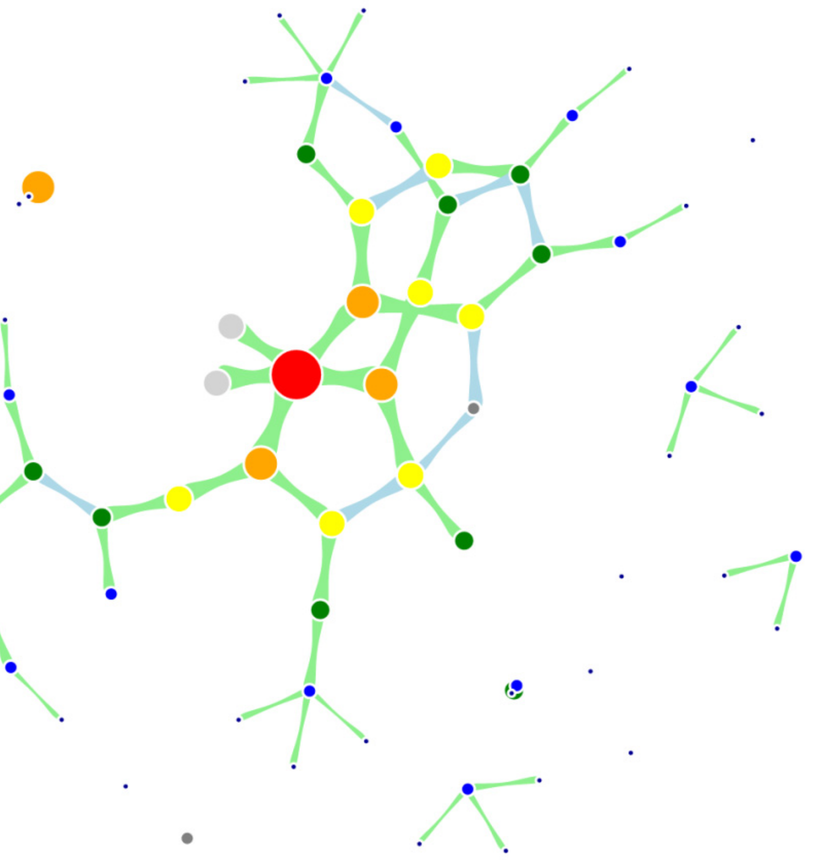


Figure 1.29 Network data consists of objects and the connections between them. The objects are typically referred to as nodes or vertices, while the connections are referred to as edges or links. Networks are often represented using force-directed algorithms, such as the example here, that arrange the network in such a way to pull connected nodes toward each other.

1.4.4 Geographic data

Geographic data refers to locations either as points or shapes, and is used to create the variety of online maps seen on the web today, such as the map of the United States in figure 1.30. The incredible popularity of web mapping means that you can get access to a massive amount of publicly accessible geodata for any project. Geographic data has a few standards, but the focus in this book will be on two: the GeoJSON and TopoJSON standards. Although geodata may come in many forms, readily available geographic information systems (GIS) tools like Quantum GIS allow developers to transform it into GIS format for ready delivery to the web. We'll look at geographic data closely in chapter 7.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

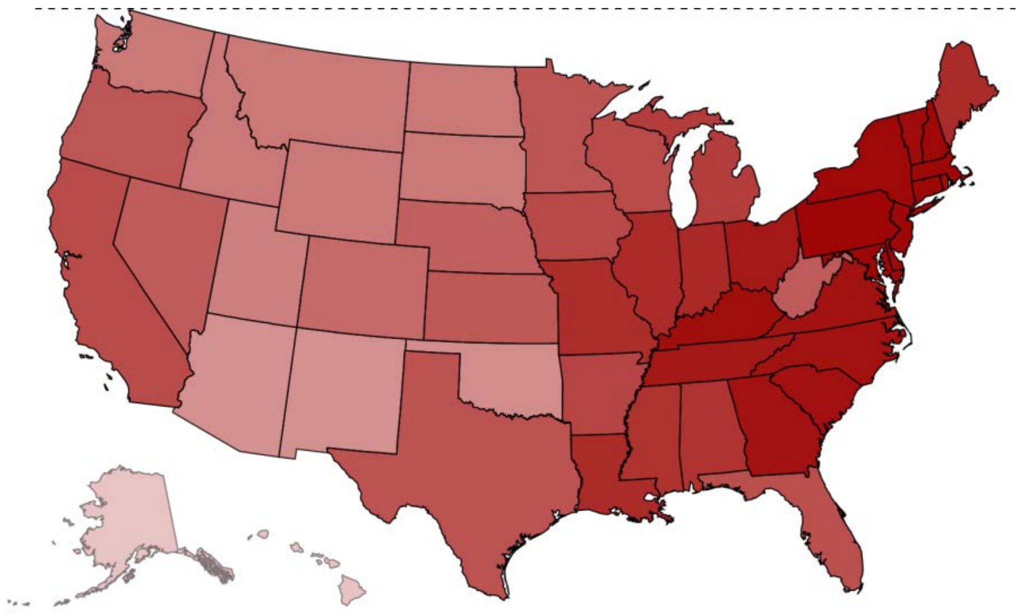


Figure 1.30 Geographic data stores the spatial geometry of objects, such as states. Each of the states in this image is represented as a separate feature with an array of values indicating its shape. Geographic data can also consist of points, such as for cities, or lines, such as for roads.

1.4.5 Raw data

As you'll see in chapter 2, everything is data, including images or blocks of text. Although information visualization typically uses shapes encoded by color and size to represent data, sometimes the best way to represent it in D3 is with linear narrative text, an image, or a video. If you develop applications for an audience that needs to understand complex systems but considers the manipulation of text or images to be somehow separate from the representation of numerical or categorical data as shapes, then you arbitrarily reduce your capability to communicate. The layouts and formatting used when dealing with text and images, typically tied to older modes of web publication, are possible in D3, and we'll deal with that throughout this book, but especially in chapter 8.

1.4.6 Objects

You'll be using two types of data points with D3: literals and objects. A literal, such as a string literal like `"Apple"` or `"beer"` or a number literal like 64273 or 5.44, is straightforward. A JavaScript object, expressed using JavaScript Object Notation (JSON), isn't so straightforward but something that you need to understand if you plan to do sophisticated data visualization.

Let's say you have a dataset that consists of individuals from an insurance database, and you need to know how old someone is, whether they're employed, their name, and their

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

children, if any. A JSON object that represents each individual in such a database would be expressed as follows:

```
{name: "Charlie", age: 55, employed: true, childrenNames: ["Ruth", "Charlie Jr."]}
```

Each object is surround by { } and has attributes that have a string, number, array, boolean, or object as their value. You can assign an object to a variable and access its attributes by referring to them, like so:

```
var person = {name: "Charlie", age: 55, employed: true, childrenNames: ["Ruth",
      "Charlie Jr."]};

person.name // Charlie
person["name"] // Charlie
person.name = "Charles" // Sets name to Charles
person["name"] = "Charles" // Sets name to Charles
person.age < 65 // true
person.childrenNames // ["Ruth", "Charlie Jr."]
person.childrenNames[0] // "Ruth"
```

Objects can be stored in arrays and associated with elements using `d3.select()` syntax. But objects can also be iterated through like arrays using a `for` loop:

```
for (x in person) {console.log(x); console.log(person[x]);}
```

The `x` in the loop represents each attribute in the `person` object. Each `x` will be one of the attributes such as `"name"`, `"age"`, and so on This allows you to iterate through the attributes using `person[x]` to show the value of that attribute of the object.

If your data is stored as JSON, then you can import it using `d3.json()`, which you'll see many times in later chapters. But remember that whenever you use `d3.csv()`, D3 imports the data as an array of JSON objects. We'll look at objects more extensively as we use them later.

1.5 Infoviz standards expressed in D3

Information visualization has never been so popular as it is today. The wealth of maps, charts, and complex representations of systems and datasets isn't just present in the workplace but also in our entertainment and everyday lives. With this popularity comes a growing library of classes and subclasses of representation of data and information using visual means, as well as aesthetic rules to promote legibility and comprehension. Your audience, whether the general public, academics, or decision makers, has grown accustomed to what we once considered incredibly abstract and complicated representations of trends in data. This is why libraries like D3 aren't just popular among data scientists but also among journalists, artists, scholars, IT professionals, and even fan communities.

But the wealth of options can seem overwhelming, and the relative ease of modifying a dataset to appear in a streamgraph, treemap, or histogram tends to promote the idea that information visualization is more about style than substance. Fortunately, well-established rules dictate what charts and methods to use for different types of data from different systems. Although we can't cover every rule in the book, we'll touch on ones that are useful to

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

consider as we create more complicated information visualizations. Although some developers use D3 to revolutionize the use of color and layout, most simply want to create visual representations of data that support practical concerns. Because D3 is being developed in this mature information visualization environment, it contains numerous helper functions to let developers worry about interface and design rather than color and axes.

Still, to properly deploy information visualization, you should know what to do and what not to do. The best way to learn this is to review the work of established designers and information visualization practitioners, and you need to have a firm understanding not only of your data but of your audience. Although an entire library of works deals with these issues, here are a few that I've found useful and can get you oriented on the basics:

Edward Tufte	*The Visual Display of Quantitative Information* *Envisioning Information*
Isabel Meirelles	*Designing for Information*
Christian Swinehart	*Pattern Recognition*

These are by no means the only or most applicable texts for learning data visualization, but I've found them useful for getting started. You should pare down and establish fundamental, even basic, data visualization practices that clearly represent the trends that are salient to your audience. When in doubt, simplify—it's often better to present a histogram than a streamgraph, or a hierarchical network layout (like a dendrogram) than a force-directed one. The more visually complex methods of displaying data tend to inspire more excitement, but can also lead an audience to see what they want to see or focus on the aesthetics of the graphics rather than the data.

Infoviz tip: kill your darlings

One of the best pieces of advice when it comes to working in information visualization comes from the practice of writing: "Kill your darlings." Just as writers may become enamored of certain scenes or characters, you can become enamored of a particularly elegant or sophisticated-looking graphics. Your love of a cool chart or animation can distract you from the goal of communicating the structure and patterns in the data. Remember to save your harshest criticism for your most beloved pieces, because you may find, much to your chagrin, that they're not as useful and informative as you think they are.

One thing to keep in mind while reading about data visualization is that the literature is often focused on static charts. With D3 you'll be making interactive and dynamic visualizations and not just static ones. You'll make a dynamic (or animated) data visualization before you finish this chapter, and using D3 to make a chart interactive is incredibly simple. A few interactive

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

touches can make a visualization not only more readable but significantly more engaging. Users who feels like they're exploring rather than reading, even if only with a few mouseover events or a simple click to zoom, will find the content of the visualization more compelling than in a static page. But this added complexity requires an investment in learning principles of interface design and user experience. We'll get into this in more detail in chapters 9 and 12.

1.6 Your first D3 app

Throughout this chapter, you've seen various lines of code and the effect of those lines of code on the growing d3ia.html sample page you've been building. But I've avoided explaining the code in too much detail so that you could concentrate on the principles at work in D3. It's simple to build an application from scratch that uses D3 to create and modify elements. Let's put it all together and see how it works. First, let's start with a clean HTML page that doesn't have any styles defined or existing divs.

```
<!doctype html>
<html>
<head>
<script src="d3.v3.min.js" type="text/JavaScript"></script>
</head>
<body>
</body>
</html>
```

1.6.1 Hello world with divs

You can use D3 as an abstraction layer for adding traditional content to the page. Although you can write JavaScript inside the your .html file or in its own .js file, let's put code in the console and see how it works. Later, we'll focus on the various commands and get into them in more detail for layouts and interfaces. You can get started with a piece of code that uses D3 to write to your web page:

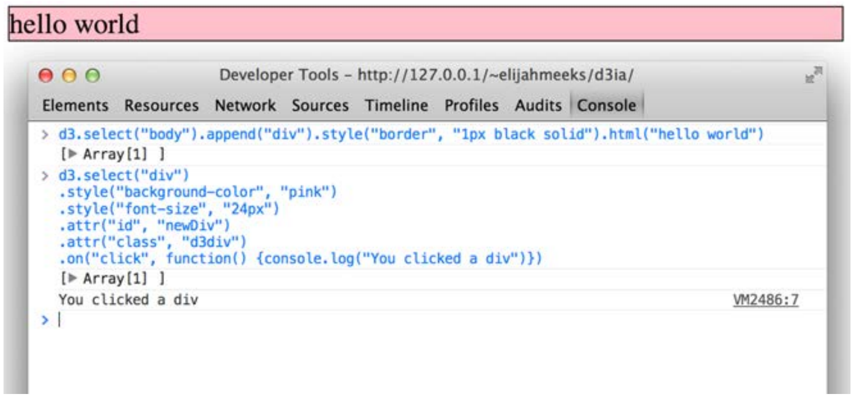
```
d3.select("body").append("div")
.style("border", "1px black solid")
.html("hello world");
```

You can adjust the element on the page and give it interactivity with the inclusion of the `.on()` function:

```
d3.select("div")
         .style("background-color", "pink")
         .style("font-size", "24px")
         .attr("id", "newDiv")
         .attr("class", "d3div")
         .on("click", function() {console.log("You clicked a div")});
```

The `.on()` method allows you to create an event listener for the currently selected element or set of elements. It accepts the variety of events that can happen to an element, such as "click", "mouseover", "mouseout" and so on. If you click your div, you'll notice that it gives a response in your console, as shown in figure 1.31.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905


hello world

Developer Tools – http://127.0.0.1/~elijahmeeks/d3ia/
Elements Resources Network Sources Timeline Profiles Audits Console

```
> d3.select("body").append("div").style("border", "1px black solid").html("hello world")
  [▶ Array[1] ]
> d3.select("div")
  .style("background-color", "pink")
  .style("font-size", "24px")
  .attr("id", "newDiv")
  .attr("class", "d3div")
  .on("click", function() {console.log("You clicked a div")})
  [▶ Array[1] ]
  You clicked a div                                                    VM2486:7
> |
```

Figure 1.31 Using `console.log()`, you can test to see if an event is properly firing. Here you create a `<div>` and assign an onclick event handler using the `.on()` syntax. When you click that element and fire the event, the action is noted in the console.

1.6.2 Hello World with circles

You didn't pick up this book to learn how to add divs to a web page, but you likely want to deal with graphics like lines and circles. To append shapes to a page with D3, you need to have an SVG canvas element somewhere in your page's DOM. You could either add this SVG canvas to the page as you write the HTML, or you could append it using the D3 syntax you've learned:

```
d3.select("body").append("svg");
```

Let's adjust your d3ia.html page to start with an SVG canvas:

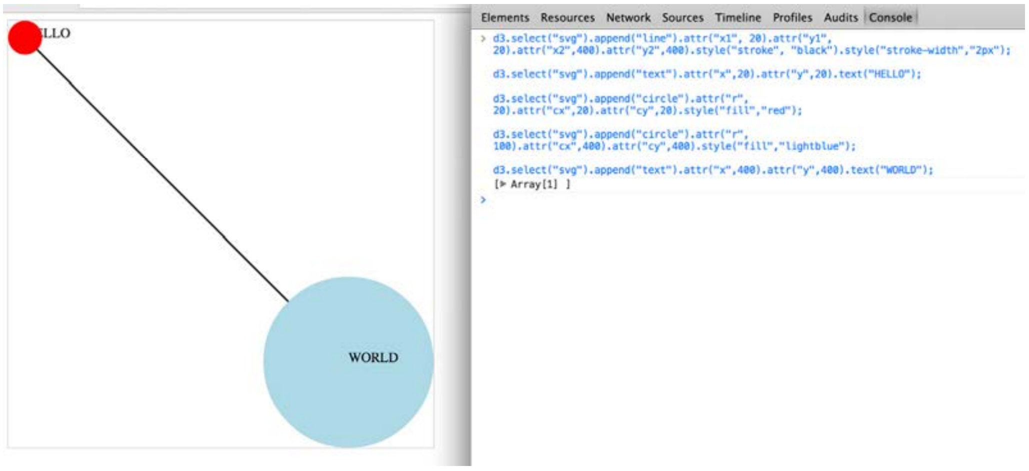
```
<!doctype html>
<html>
<head>
   <script src="d3.v3.min.js" type="text/JavaScript"></script>
</head>
<body>
<div id="vizcontainer">
<svg style="width:500px;height:500px;border:1px lightgray solid;" />
</div>
</body>
</html>
```

After you have an SVG canvas on your page, you can append various shapes to it using the same `select()` and `append()` syntax you've been using in 1.5.1 for <div> elements:

```
d3.select("svg")
.append("line")
.attr("x1", 20)
.attr("y1", 20)
.attr("x2",400)
.attr("y2",400)
```

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

```
.style("stroke", "black")
.style("stroke-width","2px");

      d3.select("svg")
.append("text")
.attr("x",20)
.attr("y",20)
.text("HELLO");

d3.select("svg")
.append("circle")
.attr("r", 20)
.attr("cx",20)
.attr("cy",20)
.style("fill","red");

      d3.select("svg")
.append("circle")
.attr("r", 100)
.attr("cx",400)
.attr("cy",400)
.style("fill","lightblue");

      d3.select("svg")
.append("text")
.attr("x",400)
.attr("y",400)
.text("WORLD");
```



```
> d3.select("svg").append("line").attr("x1", 20).attr("y1",
  20).attr("x2",400).attr("y2",400).style("stroke", "black").style("stroke-width","2px");

  d3.select("svg").append("text").attr("x",20).attr("y",20).text("HELLO");

  d3.select("svg").append("circle").attr("r",
  20).attr("cx",20).attr("cy",20).style("fill","red");

  d3.select("svg").append("circle").attr("r",
  100).attr("cx",400).attr("cy",400).style("fill","lightblue");

  d3.select("svg").append("text").attr("x",400).attr("y",400).text("WORLD");
  [▶ Array[1] ]
>
```

Figure 1.32 The result of running the previous code in the console is the creation of two circles, a line, and two text elements. The order in which these elements are drawn results in the first label covered by the circle drawn later.

Notice that your circles are drawn over the line and the text is drawn above or below the circle, depending on the order that you run your commands, as you can see in figure 1.32.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

This is because the draw order of SVG is based on its DOM order. Later you'll learn some methods to adjust that order.

1.6.3 A conversation with D3

Writing Hello World with languages is such a common example that I thought we should give the world a chance to respond. Let's add the same big circle and little circle from before, but this time, when you add text, you'll include the .style("opacity") setting that makes your text invisible. You'll also give each of them a .attr("id") setting so that the text near the small circle has an "id" attribute with the value of "a" and the text near the large circle has an "id" attribute with the value of "b".

```
d3.select("svg")
.append("circle")
.attr("r", 20)
.attr("cx",20)
.attr("cy",20)
.style("fill","red");

      d3.select("svg")
.append("text")
.attr("id", "a")
.attr("x",20)
.attr("y",20)
.style("opacity", 0)
.text("HELLO WORLD");

      d3.select("svg")
.append("circle")
.attr("r", 100)
.attr("cx",400)
.attr("cy",400)
.style("fill","lightblue");

      d3.select("svg")
.append("text")
.attr("id", "b")
.attr("x",400)
.attr("y",400)
.style("opacity", 0)
.text("Uh, hi.");
```

Two circles, no line, and no text. Now you make the text appear using the .transition() method with the .delay() method, and you should have an end state like the one shown in figure 1.33.

```
d3.select("#a").transition().delay(1000).style("opacity", 1);
d3.select("#b").transition().delay(3000).style("opacity", .75);
```

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

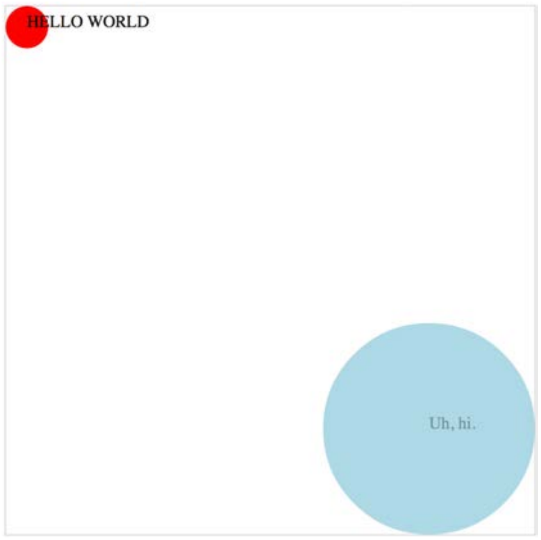


Figure 1.33 Transition behavior when associated with a delay results in a pause before the application of the attribute or style.

Congratulations! You've made your first dynamic data visualization. The `.transition()` method indicates that you don't want your change to be instantaneous. By chaining it with the `.delay()` method, you indicate how many milliseconds to wait before implementing the style or attribute changes that appear in the chain after that `.delay()` setting.

We'll get a bit more ambitious later on, but before we finish here, let's look at another `.transition()` setting. You can set a `.delay()` before applying the new style or attribute, but you can also set a `.duration()` over which the change is applied. The results in your browser should move the shapes in the direction of the arrows in figure 1.34.

```
d3.selectAll("circle").transition().duration(2000).attr("cy", 200);
```

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

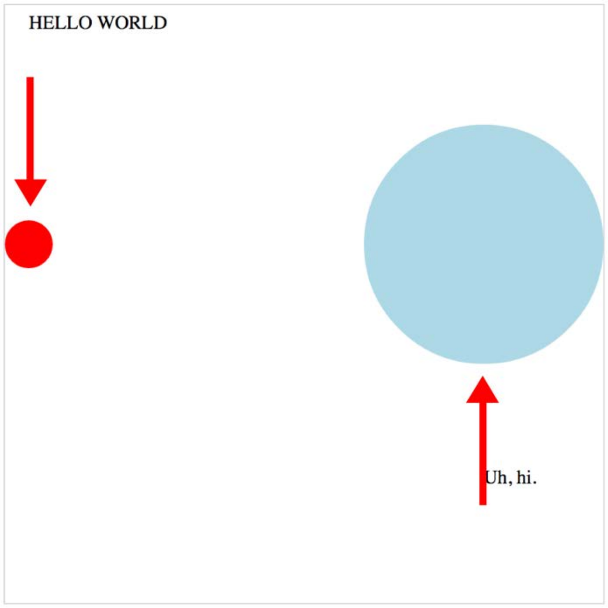


Figure 1.34 Transition behavior when associated with position makes the shape graphically move to its new position over the course of the assigned duration. Because you used the same “y” position for both circles, the first circle moves down and the second circle moves up the “y” position you set, which is between the two circles.

The `.duration()` method, as you can see, adjusts the setting over the course of the amount of time (again, in milliseconds) that you set it for.

That covers the basics of how D3 works and how it’s designed, and these fundamental concepts will come back again and again throughout the following chapters, where you’ll learn more complicated variations on representing and manipulating data.

1.7 Summary

In this chapter you’ve had an overview of D3 with a focus on how well suited it is for developers building web applications for the modern browser. We’ve highlighted the standardizations and advances that allow this to happen. This included

- A few examples of the kinds of data visualization you can create with D3.
- A process map to show how to go from data to data visualization to interactivity, noting where you can find each step in this book.
- An overview of the DOM, SVG, and CSS.
- A first look at data-binding and selection to create and change elements on the page.
- An overview of the different types of data you’ll encounter when planning and creating your data visualizations.
- Some simple animations using D3 transitions.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905

D3.js is another JavaScript library, one of thousands, but it’s also indicative of a change in our expectations of what a web page can do. Although you may initially use it to build one-off data visualizations, D3 has much more power and functionality than that. Throughout this book, you’ll explore the ways that you can use D3 to create rich data-driven documents that will enthrall and impress.

©Manning Publications Co. We welcome reader comments about anything in the manuscript - other than typos and other simple mistakes. These will be cleaned up during production of the book by copyeditors and proofreaders.
http://www.manning-sandbox.com/forum.jspa?forumID=905