Open Data: Datenmanagement und Visualisierung Einführung D3 App Entwicklung und Web Programmierung

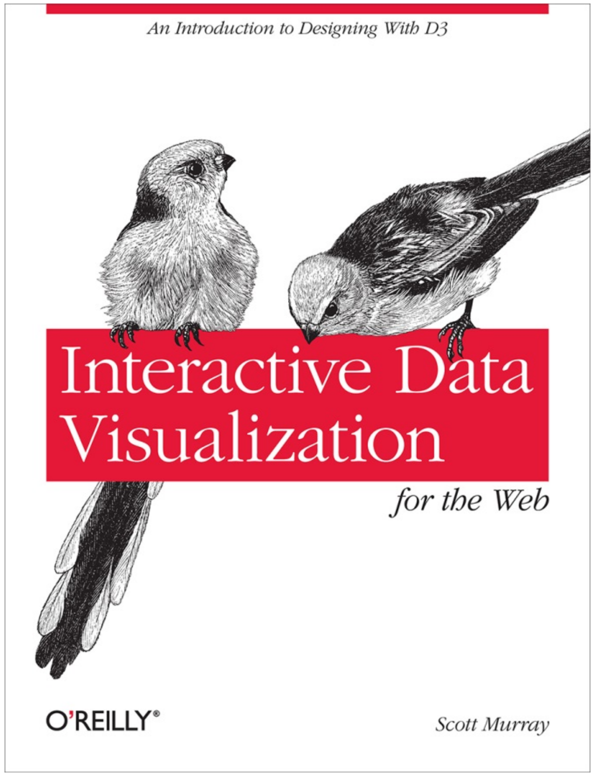
Termin 7, 3. April 2014

Dr. Matthias Stürmer und Prof. Dr. Thomas Myrach
Universität Bern, Institut für Wirtschaftsinformatik
Abteilung Informationsmanagement
Forschungsstelle Digitale Nachhaltigkeit

Interactive Data Visualization for the Web

Quelle:

> O'Reilly Media, von Scott Murray
> März 2013, 272 Seiten, Englisch
> ISBN-10: 1449339735
> **Gratis online als ebook**
> Auf Amazon.de für CHF 22.50
> „Create and publish your own interactive data visualization projects on the Web-even if you have little or no experience with data visualization or web development."
> Total 13 Kapitel, 10 Kapitel davon werden in den Übungen behandelt

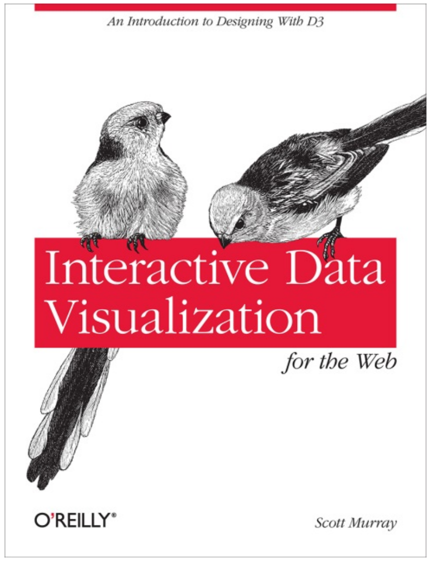

An Introduction to Designing With D3
Interactive Data Visualization
for the Web
O'REILLY
Scott Murray

Agenda

Chapter 3. Technology Fundamentals:

http://chimera.labs.oreilly.com/books/1230000000345/ch03.html

1. **Web Servers**
2. Hypertext Markup Language HTML
3. Cascading Style Sheets CSS
4. JavaScript
5. Scalable Vector Graphics SVG


An Introduction to Designing With D3
Interactive Data Visualization
for the Web
O'REILLY
Scott Murray

Server client architecture of the Internet

- CLIENT: I'd really like to know what's going on over at somewebsite.com. I better call over there to get the latest info. [Silent sound of Internet connection being established.]
- SERVER: Hello, unknown web client! I am the server hosting **somewebsite.com**. What page would you like?
- CLIENT: This morning, I am interested in the page at **somewebsite.com/news/**.
- SERVER: Of course, one moment.
- Code is transmitted from SERVER to CLIENT.
- CLIENT: I have received it. Thank you!
- SERVER: You're welcome! Would love to stay on the line and chat, but I have other requests to process. Bye!

URLs und URIs

- Abkürzungen:
  - URI (Uniform Resource Identifier): identifies a resource
  - URL (Uniform Resource Locator): identifies and locates a resource
- URL-Beispiel: **http://alignedleft.com:80/tutorials/d3/**
- Complete URLs consist of four parts:
  - An indication of the *communication protocol*, such as HTTP or HTTPS
  - The *domain name* of the resource, such as *alignedleft.com*
  - The *port number :80*, indicating over which port the connection to the server should be attempted
  - Any additional locating information */tutorials/d3/*, such as the path of the requested file, or any query parameters

Visiting a website

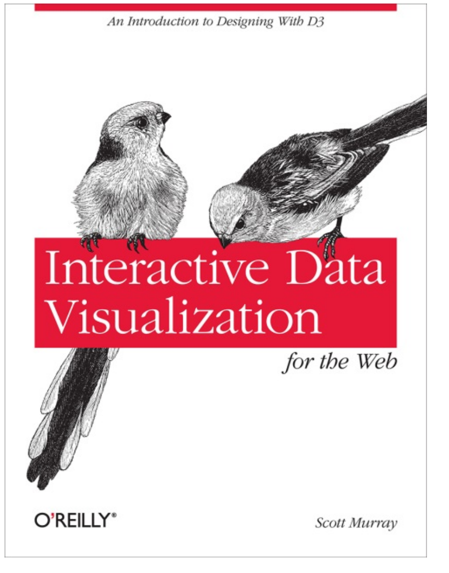
- User runs the web browser of her choice, then types a URL into the address bar, such as *alignedleft.com/tutorials/d3/*. Because she did not specify a protocol, HTTP is assumed, and “http://” is prepended to the URL.
- The browser then attempts to connect to the server behind *alignedleft.com* across the network, via port 80, the default port for HTTP.
- The server associated with *alignedleft.com* acknowledges the connection and is taking requests. (“I’ll be here all night.”)
- The browser sends a request for the page that lives at */tutorials/d3/*.
- The server sends back the HTML content for that page.
- As the client browser receives the HTML, it discovers references to *other files* needed to assemble and display the entire page, including CSS stylesheets and image files. So it contacts the same server again, once per file, requesting the additional information.
- The server responds, dispatching each file as needed.
- Finally, all the web documents have been transferred over. Now the client performs its most arduous task, which is to *render* the content. It first parses through the HTML to understand the structure of the content. Then it reviews the CSS selectors, applying any properties to matched elements. Finally, it plugs in any image files and executes any JavaScript code.

Agenda

Chapter 3. Technology Fundamentals:

http://chimera.labs.oreilly.com/books/1230000000345/ch03.html

1. Web Servers
2. **Hypertext Markup Language HTML**
3. Cascading Style Sheets CSS
4. JavaScript
5. Scalable Vector Graphics SVG

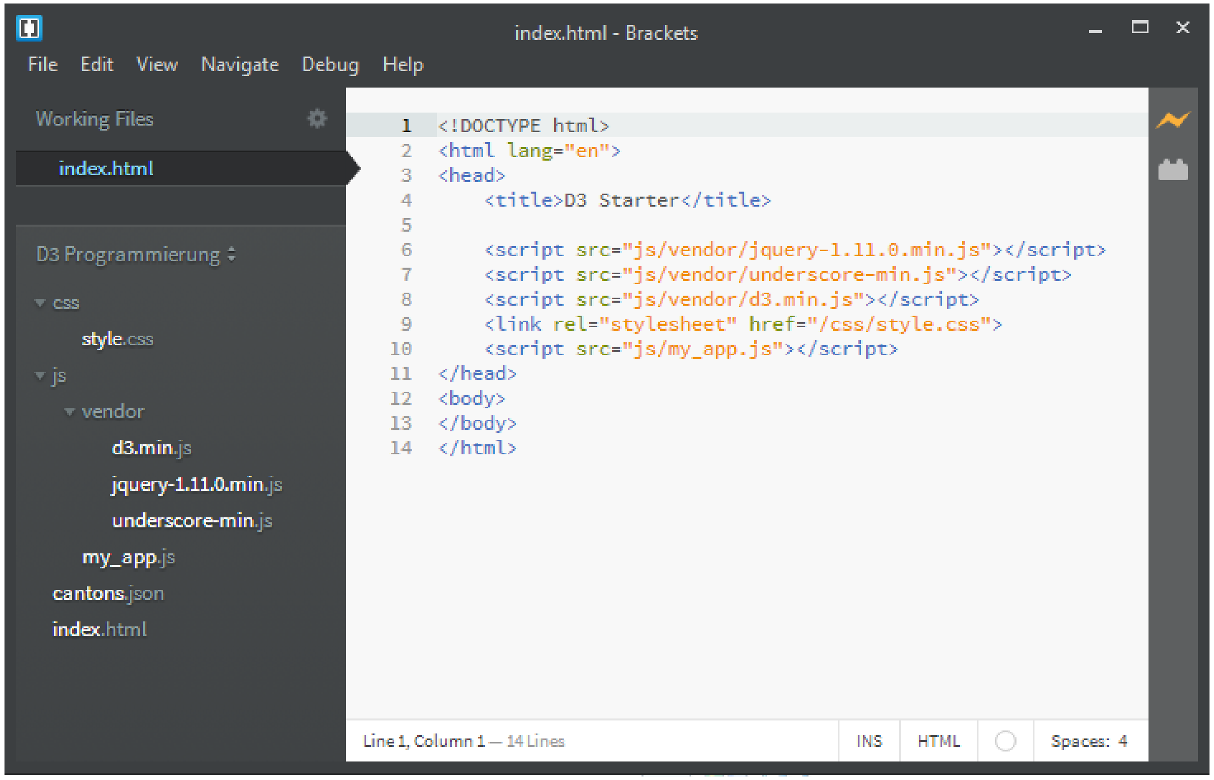

An Introduction to Designing With D3
Interactive Data Visualization
for the Web
O'REILLY
Scott Murray

HTML Dokument

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <h1>Page Title</h1>
        <p>This is a really interesting paragraph.</p>
    </body>
</html>
```

HTML Dokument der D3 App

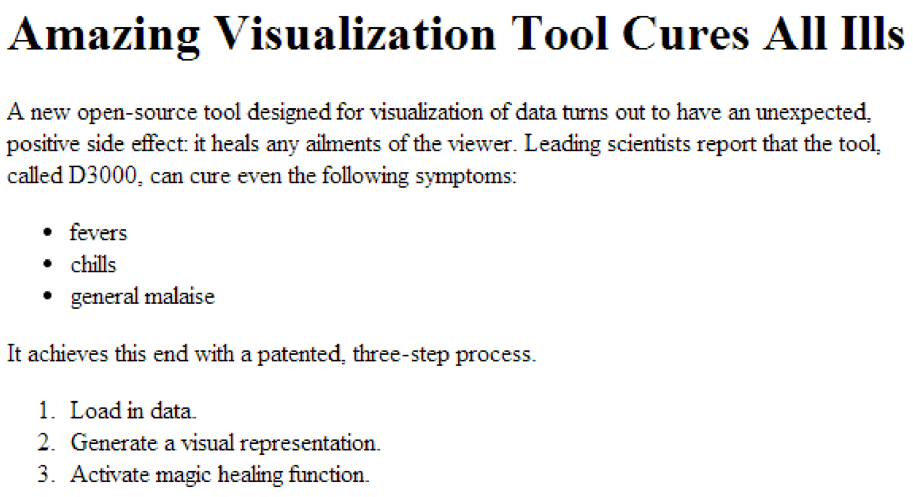
Download open source JavaScript editor Brackets: http://www.brackets.io



```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>D3 Starter</title>

    <script src="js/vendor/jquery-1.11.0.min.js"></script>
    <script src="js/vendor/underscore-min.js"></script>
    <script src="js/vendor/d3.min.js"></script>
    <link rel="stylesheet" href="/css/style.css">
    <script src="js/my_app.js"></script>
</head>
<body>
</body>
</html>
```

Beispiel HTML-Code

```
<h1>Amazing Visualization Tool Cures All Ills</h1>
<p>A new open-source tool designed for visualization
of data turns out to have an unexpected, positive
side effect: it heals any ailments of the viewer.
Leading scientists report that the tool, called
D3000, can cure even the following symptoms:</p>
<ul>
    <li>fevers</li>
    <li>chills</li>
    <li>general malaise</li>
</ul>
<p>It achieves this end with a patented, three-step
process.</p>
<ol>
    <li>Load in data.</li>
    <li>Generate a visual representation.</li>
    <li>Activate magic healing function.</li>
</ol>
```


Amazing Visualization Tool Cures All Ills

A new open-source tool designed for visualization of data turns out to have an unexpected, positive side effect: it heals any ailments of the viewer. Leading scientists report that the tool, called D3000, can cure even the following symptoms:

- fevers
- chills
- general malaise

It achieves this end with a patented, three-step process.

1. Load in data.
2. Generate a visual representation.
3. Activate magic healing function.

Common Elements

<!DOCTYPE html>	The standard document type declaration. Must be the first thing in the document.
html	Surrounds all HTML content in a document.
head	The document head contains all metadata about the document, such as its title and any references to external stylesheets and scripts.
title	The title of the document. Browsers typically display this at the top of the browser window and use this title when bookmarking a page.
body	Everything not in the head should go in the body. This is the primary visible content of the page.
h1, h2, h3, h4	These let you specify headings of different levels. h1 is a top-level heading, h2 is below that, and so on.
p	A paragraph!
ul, ol, li	Unordered lists are specified with ul, most often used for bulleted lists. Ordered lists (ol) are often numbered. Both ul and ol should include li elements to specify list items.
em	Indicates emphasis. Typically rendered in italics.
strong	Indicates additional emphasis. Typically rendered in boldface.
a	A link. Typically rendered as underlined, blue text, unless otherwise specified.
span	An arbitrary span of text, typically within a larger containing element like p.
div	An arbitrary division within the document. Used for grouping and containing related elements.

Attributes

- All HTML elements can be assigned *attributes* by including property/value pairs in the opening tag.
- `<tagname property="value"></tagname>`
- The name of the property is followed by an equals sign, and the value is enclosed within double quotation marks.
- Different kinds of elements can be assigned different attributes. For example, the a link tag can be given an href attribute, whose value specifies the URL for that link. (href is short for “HTTP reference.”)
- `<a href="http://d3js.org/">The D3 website</a>`
- Some attributes can be assigned to *any* type of element, such as class and id.

Classes and IDs

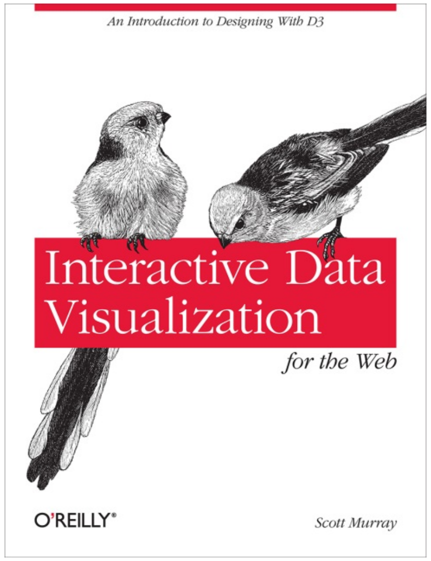
```
<p>Brilliant paragraph</p>
<p>Insightful paragraph</p>
<p class="awesome">Awe-inspiring paragraph</p>

<p class="uplifting">Brilliant paragraph</p>
<p class="uplifting">Insightful paragraph</p>
<p class="uplifting awesome">Awe-inspiring para</p>

<div id="content">
    <div id="visualization"></div>
    <div id="button"></div>
</div>
```

Agenda

Chapter 3. Technology Fundamentals:

http://chimera.labs.oreilly.com/books/1230000000345/ch03.html

1. Web Servers
2. Hypertext Markup Language HTML
3. **Cascading Style Sheets CSS**
4. JavaScript
5. Scalable Vector Graphics SVG


An Introduction to Designing With D3
Interactive Data Visualization
for the Web
O'REILLY
Scott Murray

CSS-Beispiele

```
selector {
    property: value;
    property: value;
    property: value;
}

selectorA,
selectorB,
selectorC {
    property: value;
    property: value;
    property: value;
}
```

```
body {
    background-color: white;
    color: black;
    font-size: 10px;
}

p,
li,
a {
    font-size: 12px;
    line-height: 14px;
    color: orange;
}
```

Selectors

Type selectors

```
h1          /* Selects all level 1 headings */
p           /* Selects all paragraphs */
strong      /* Selects all strong elements */
em          /* Selects all em elements */
div         /* Selects all divs */
```

Descendant selectors

```
h1 em       /* Selects em elements contained in an h1 */
div p       /* Selects p elements contained in a div */
```

Selectors

Class selectors

```
.caption  /* Selects elements with class "caption" */
.label    /* Selects elements with class "label"   */
.axis     /* Selects elements with class "axis"    */
```

Multiple class selectors

```
.bar.highlight  /* Could target highlighted bars   */
.axis.x         /* Could target an x-axis          */
.axis.y         /* Could target a y-axis           */
```

Selectors

ID selectors

```
#header     /* Selects element with ID "header"    */
#nav        /* Selects element with ID "nav"       */
#export     /* Selects element with ID "export"    */
```

Target specific elements

```
div.sidebar /* Selects divs with class "sidebar", but
               not other elements with that class   */
#button.on  /* Selects element with ID "button", but
               only when the class "on" is applied  */
```

Properties and Values

Groups of property/value pairs cumulatively form the styles:

```
margin: 10px;
padding: 25px;
background-color: yellow;
color: pink;
font-family: Helvetica, Arial, sans-serif;
```

Colors can be specified in several different formats:

- Named colors: **`orange`**
- Hex values: **`#3388aa or #38a`**
- RGB values: **`rgb(10, 150, 20)`**
- RGB with alpha transparency: **`rgba(10, 150, 20, 0.5)`**

Inline CSS

```
<p style="color: blue; font-size: 48px; font-style:
italic;">Inline styles are kind of a hassle</p>
```

> Because inline styles are attached directly to elements, there is no need for selectors.

> Inline styles are **messy and hard to read,** but they are useful for giving special treatment to a single element, when that style information doesn't make sense in a larger stylesheet. We'll learn how to **apply inline styles programmatically with D3** (which is much easier than typing them in by hand, one at a time).

UNIVERSITÄT BERN

Embedded CSS

```
<html>
    <head>
        <style type="text/css">
            p {
                font-size: 24px;
                font-weight: bold;
                background-color: red;
                color: white;
            }
        </style>
    </head>
    <body>
        <p>If I were to ask you, as a mere paragraph, would you
       say that I have style?</p>
    </body>
</html>
```

Linked CSS

```
<html>
    <head>
        <link rel="stylesheet" href="style.css">
    </head>
    <body>
        <p>If I were to ask you, as a mere paragraph, would you say that I have style?</p>
    </body>
</html>
```

style.css

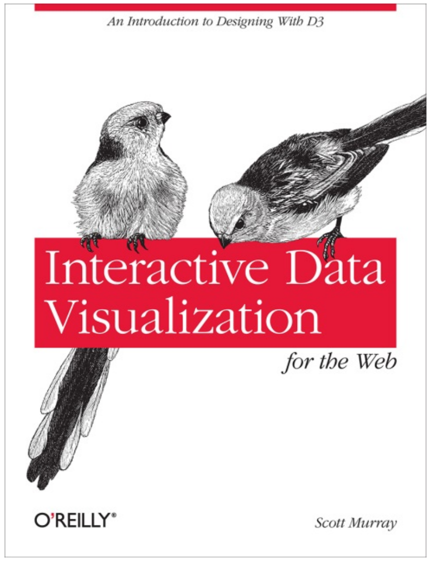
```
p {
  font-size: 24px;
  font-weight: bold;
  background-color: red;
  color: white;
}
```

Agenda

Chapter 3. Technology Fundamentals:

http://chimera.labs.oreilly.com/books/1230000000345/ch03.html

1. Web Servers
2. Hypertext Markup Language HTML
3. Cascading Style Sheets CSS
4. **JavaScript**
5. Scalable Vector Graphics SVG


An Introduction to Designing With D3
Interactive Data Visualization
for the Web
O'REILLY
Scott Murray

UNIVERSITÄT BERN

Referencing JavaScript files

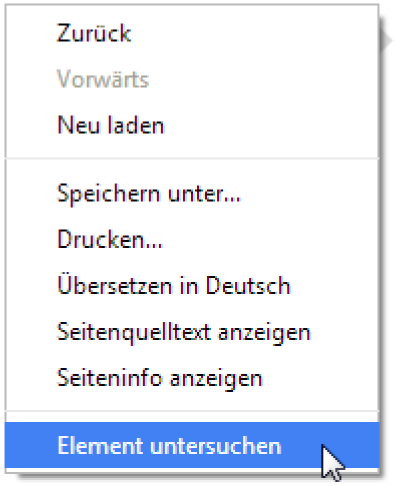
> Scripts can be included directly in HTML between two script tags:

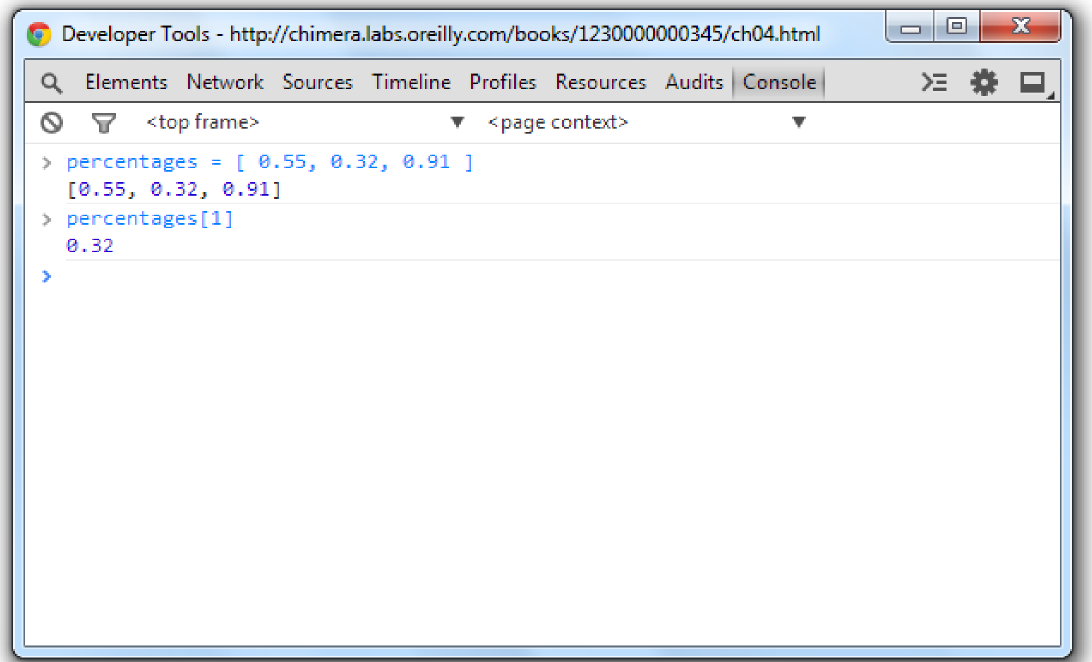
```
<body>
    <script type="text/javascript">
        alert("Hello, world!");
    </script>
</body>
```

> or stored in a separate file with a .js suffix, and then referenced somewhere in the HTML (could be in the head, as shown here, or also just before the end of the closing body tag):

```
<head>
    <title>Page Title</title>
    <script type="text/javascript" src="myscript.js"></script>
</head>
```

JavaScript Developer Console




Developer Tools - http://chimera.labs.oreilly.com/books/1230000000345/ch04.html
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
<page context>

```
> percentages = [ 0.55, 0.32, 0.91 ]
  [0.55, 0.32, 0.91]
> percentages[1]
  0.32
>
```

Variables

- Variables are containers for data. A simple variable holds one value:

  ```
  var number = 5;
  ```

- In that statement, **`var`** indicates you are declaring a new variable, the name of which is number. The equals sign is an *assignment operator* because it takes the value on the right (5) and *assigns* it to the variable on the left (number).
- A variable is a datum, the smallest building block of data. The variable is the foundation of all other data structures, which are simply different configurations of variables.
- More examples:

  ```
  var defaultColor = "hot pink";
  var thisMakesSenseSoFar = true;
  ```

my_app.js

Variablen-Zuweisungen aus der Open Data App:

```
var svg = d3.select('body').append('svg').append('g')
.attr('transform', 'translate(400,400)');

var cantons = svg.selectAll('g').data(data);

var g = cantons.enter()
.append('g');

var colors = d3.scale.category10();
```

Arrays

- Keeping track of related values in separate variables is inefficient:

```
var numberA = 5;
var numberB = 10;
var numberC = 15;
var numberD = 20;
var numberE = 25;
```

- Rewritten as an array, those values are much simpler. Hard brackets [] indicate an array, and each value is separated by a comma:

```
var numbers = [ 5, 10, 15, 20, 25 ];
```

- You can access a value in an array by using *bracket notation*:

```
numbers[0]  //Returns 5
numbers[1]  //Returns 10
numbers[2]  //Returns 15
```

Arrays

> Arrays can contain any type of data, not just integers:

```
var percentages = [ 0.55, 0.32, 0.91 ];
var names = [ "Ernie", "Bert", "Oscar" ];

percentages[1]  //Returns 0.32
names[1]        //Returns "Bert"
```

> Although I don’t recommend it, different types of values can even be stored within the same array:

```
var mishmash = [ 1, 2, 3, 4.5, 5.6, "oh boy",
"say it isn't", true ];
```

Objects

- Think of a JavaScript object as a custom data structure. We use curly brackets **{}** to indicate an object. In between the brackets, we include **properties** and **values**. A colon **:** separates each property and its value, and a comma separates each property/value pair:

```
var fruit = {
    kind: "grape",
    color: "red",
    quantity: 12,
    tasty: true
};
```

- To reference each value, we use dot notation, specifying the name of the property:

```
fruit.kind      //Returns "grape"
fruit.color     //Returns "red"
fruit.quantity  //Returns 12
fruit.tasty     //Returns true
```

UNIVERSITÄT BERN

Objects and Arrays

- You can combine these two structures to create arrays of objects, or objects of arrays, or objects of objects or, well, basically whatever structure makes sense for your dataset.
- Let’s say we have acquired a couple more pieces of fruit, and we want to expand our catalog accordingly. We use hard brackets [] on the outside, to indicate an array, followed by curly brackets{} and object notation on the inside, with each object separated by a comma:

```
var fruits = [
    {
        kind: "grape",
        color: "red",
        quantity: 12,
        tasty: true
    },
    {
        kind: "kiwi",
        color: "brown",
        quantity: 98,
        tasty: true
    },
    {
        kind: "banana",
        color: "yellow",
        quantity: 0,
        tasty: true
    }
];
```

Objects and Arrays

> To access this data, we just follow the trail of properties down to the values we want. Remember, **[] means array,** and **{} means object.** fruits is an array, so first we use bracket notation to **specify an array** index:

```
fruits[1]
```

> Next, each array element is an object, just add a dot and a property:

```
fruits[1].quantity   //Returns 98
```

> Access values in the fruits array of objects:

```
fruits[0].kind       ==  "grape"
fruits[0].color      ==  "red"
fruits[0].quantity   ==  12
fruits[0].tasty      ==  true
```

JSON

- JSON = JavaScript Object Notation:

```
{
    "kind": "grape",
    "color": "red",
    "quantity": 12,
    "tasty": true
}
```

- The only difference here is that our property names are now surrounded by double quotation marks "", making them string values.
- JSON objects, like all other JavaScript objects, can be stored in variables like so:

```
var jsonFruit = {
    "kind": "grape",
    "color": "red",
    "quantity": 12,
    "tasty": true
};
```

GeoJSON

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [ 150.1282427, -24.471803 ]
            },
            "properties": {
                "type": "town"
            }
        }
    ]
}
```

cantons.json

```
[
    {
        "name": "Bern",
        "numbers": [
            {"name": "Einwohnerzahl", "value": 1408575},
            {"name": "Fläche", "value": 5959.1},
            {"name": "Wähleranteile in % FDP", "value": 8.7},
            {"name": "Wähleranteile in % CVP", "value": 2.1},
            {"name": "Wähleranteile in % SP", "value": 19.3},
            {"name": "Wähleranteile in % SVP", "value": 29.0},
            {"name": "Wähleranteile in % EVP/CSP", "value": 4.2},
            {"name": "Wähleranteile in % GLP", "value": 5.3},
            {"name": "Wähleranteile in % BDP", "value": 14.9},
            {"name": "Wähleranteile in % PdA/Sol.", "value": 0.3},
            {"name": "Wähleranteile in % GPS", "value": 9.4},
            {"name": "Wähleranteile in % kleine Rechtsparteien", "value": 3.7}
        ]
    },
         ...
```

Mathematical and Comparison Operators

```
1 + 2      //Returns 3
10 - 0.5   //Returns 9.5
33 * 3     //Returns 99
3 / 4      //Returns 0.75

==         //Equal to
!=         //Not equal to
<          //Less than
>          //Greater than
<=         //Less than or equal to
>=         //Greater than or equal to
```

Übung:

```
3 == 3
3 == 5
3 >= 3
3 >= 2
100 < 2
298 != 298
```

Control Structure: if()

> If the test between parentheses is **true,** then the code between the curly brackets is run. If the test turns up **false,** then the bracketed code is ignored, and life goes on. (Technically, life goes on either way.)

```
if (3 < 5) {
     Eureka! Three is less than five!";
}
```

> In the preceding example, the bracketed code will always be executed, because 3 < 5 is always true.if statements are more useful when comparing variables or other conditions that change.

Control Structure: for()

> You can use for loops to repeatedly execute the same code, with slight variations.

> They are so-called because they loop through the code *for* as many times as specified. First, the initialization statement is run. Then, the test is evaluated, like a mini if statement. If the test is true, then the bracketed code is run. Finally, the update statement is run, and the test is reevaluated.

> The most common application of a for loop is to increase some variable by 1 each time through the loop. The test statement can then control how many times the loop is run by referencing that value. (The variable is often named i, purely by convention, because it is short and easy to type.)

```
for (var i = 0; i < 5; i++) {
    console.log(i);  //Prints value to console
}
```

What arrays are made for()

> An array organizes lots of data values in one convenient place. Then **for()** can quickly “loop” through every value in an array and perform some action with it—such as, express the value as a visual form. D3 often manages this looping for us, such as with its magical **data()** method.

```
var numbers = [ 8, 100, 22, 98, 99, 45 ];
for (var i = 0; i < numbers.length; i++) {
    console.log(numbers[i]);  //Print value to console
}
```

> **length** is a property of every array. In this case,numbers contains six values, so **numbers.length** resolves to 6, and the loop runs six times. If numberswere 10 positions long, the loop would run 10 times.

Functions

> Functions can take arguments or parameters as input, and then return values as output. Parentheses are used to ***call*** **(execute)** a function. If that function requires any **arguments (input values),** then they are *passed* to the function by including them in the parentheses.

```
var calculateTip = function(bill) {
    return bill * 0.2;
};
calculateTip(38);
```

> Beispiel einer anonymen Funktion aus der Open Data App:

```
d3.json('cantons.json',function(err, data){
        var cantons = svg.selectAll('g').data(data);
```

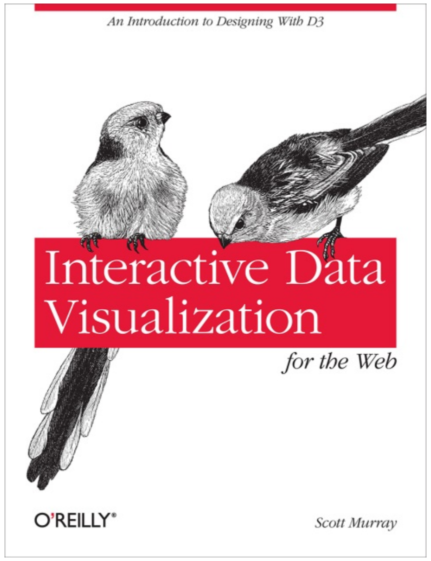
UNIVERSITÄT BERN

Agenda

Chapter 3. Technology Fundamentals:

http://chimera.labs.oreilly.com/books/1230000000345/ch03.html

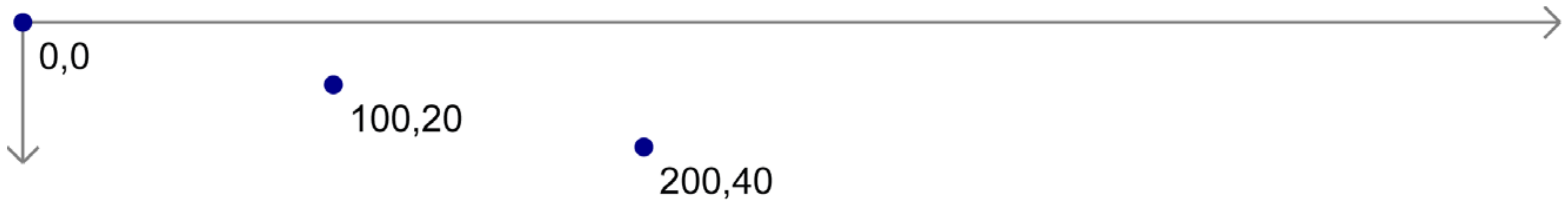
1. Web Servers
2. Hypertext Markup Language HTML
3. Cascading Style Sheets CSS
4. JavaScript
5. **Scalable Vector Graphics SVG**


An Introduction to Designing With D3
Interactive Data Visualization
for the Web
O'REILLY
Scott Murray

The SVG Element

- Before you can draw anything, you must create an SVG element. Think of the SVG element as a canvas on which your visuals are rendered. (In that respect, SVG is conceptually similar to HTML's canvas element.)
- At a minimum, it's good to specify width and height values. If you don't specify these, the SVG will behave like a typically greedy, block-level HTML element and take up as much room as it can within its enclosing element:

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



Simple Shapes

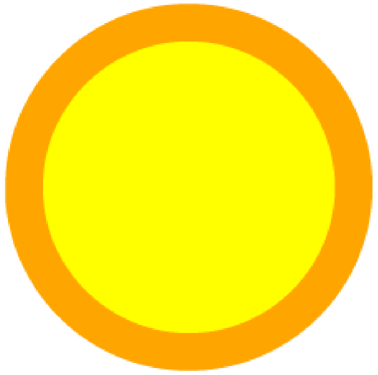
Einige Beispiele:

```
<rect x="0" y="0" width="500" height="50"/>

<circle cx="250" cy="25" r="25"/>

<ellipse cx="250" cy="25" rx="100" ry="25"/>

<line x1="0" y1="0" x2="500" y2="50"
stroke="black"/>

<text x="250" y="25" font-family="serif" font-
size="25" fill="gray">Easy-peasy</text>
```

Styling SVG Elements

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

```
<circle cx="25" cy="25"
r="22"  fill="yellow"
stroke="orange" stroke-width="5"/>
```



SVG und CSS

Alternatively, we could strip the style attributes and assign the circle a class (just as if it were a normal HTML element):

```
<circle cx="25" cy="25" r="22" class="pumpkin"/>
```

and then put the fill, stroke, and stroke-width rules into a CSS style that targets the new class:

```
.pumpkin {
    fill: yellow;
    stroke: orange;
    stroke-width: 5;
 }
```

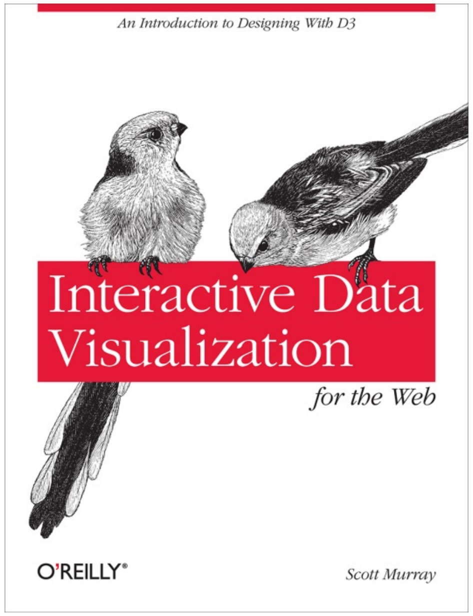
The CSS approach has a few obvious benefits:

> You can specify a style once and have it applied to multiple elements.

> CSS code is easier to read than inline attributes.

> For those reasons, the CSS approach might be more maintainable and make design changes faster to implement.

Interactive Data Visualization for the Web


An Introduction to Designing With D3
Interactive Data Visualization
for the Web
O'REILLY
Scott Murray

Zum nachlesen:

http://chimera.labs.oreilly.com/books/1230000000345/ch03.html