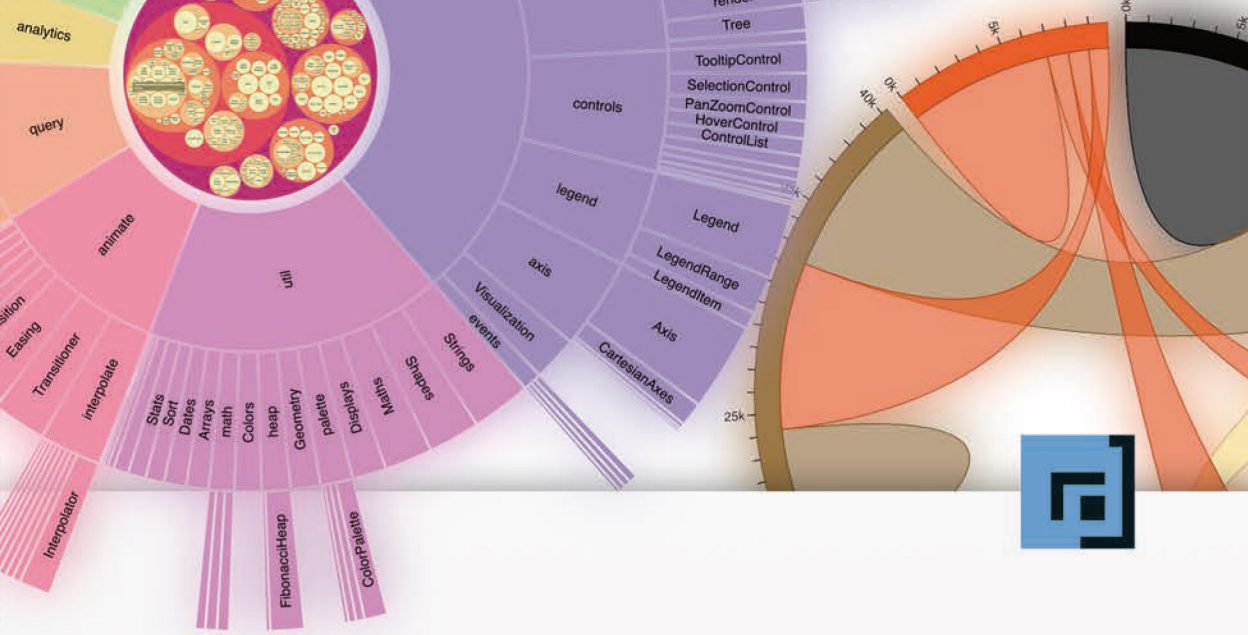

D3 Praxisbuch

Philipp K. Janert

Interaktive JavaScript-Grafiken im Browser

dpunkt.verlag

Der Autor

Philipp K. Janert ist in Deutschland geboren und aufgewachsen. 1997 erwarb er an der Universität von Washington einen Doktorgrad in Theoretischer Physik. Seitdem arbeitet er als Programmierer, Wissenschaftler und Mathematiker im technischen Bereich. Er ist der Autor der Bücher *Data Analysis with Open Source Tools* (O'Reilly), *Feedback Control for Computer Systems* (O'Reilly) und *Gnuplot in Action* (Manning Publications).

Philipp K. Janert

D3-Praxisbuch

Interaktive JavaScript-Grafiken im Browser

Aus dem Amerikanischen von Volkmar Gronau

dpunkt.verlag

Philipp K. Janert

Lektorat: Michael Barabas
Projektkoordinierung/Lektoratsassistenz: Anja Weimer
Übersetzung & Satz: G&U Language & Publishing Services GmbH, Flensburg, *www.GundU.com*
Copy-Editing: Ursula Zimpfer, Herrenberg
Herstellung: Stefanie Weidner
Umschlaggestaltung: Helmut Kraus, *www.excalm.de*
Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-86490-725-8
PDF 978-3-96088-882-6
ePub 978-3-96088-880-2
mobi 978-3-96088-881-9

1. Auflage 2020

Wieblinger Weg 17
69123 Heidelberg

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: hallo@dpunkt.de

5 4 3 2 1 0

Inhalt

Vorbemerkungen

Schreibweisen in diesem Buch

In diesem Buch werden die folgenden speziellen Schreibweisen verwendet:

Kursivschrift
: Kennzeichnet neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateinamenerweiterungen.

`Nichtproportionale Schrift`
: Wird für Programmlistings, aber auch für Programmelemente innerhalb des Fließtextes wie Variablen- und Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter verwendet.

Die Codebeispiele

Ergänzendes Material, wie Codebeispiele, Übungen usw., steht auf *https://github.com/janert/d3-for-the-impatient* zum Download bereit.

Danksagung

Ich möchte Mike Loukides und Scott Murray danken, die dieses Projekt von Anfang an voller Begeisterung unterstützt haben. Giuseppe Verni, Jane Pong, Matt Kirk, Noah Iliinsky, Richard Kreckel, Sankar Rao Bhogi, Scott Murray und Sebastien Martel haben das Manuskript oder Teile davon gelesen, die Beispiele getestet und viele wichtige Anregungen gegeben. Darüber hinaus haben Matt, Scott und Sebastien in einer umfangreichen Korrespondenz Fragen beantwortet und mir ihre Kenntnisse vermittelt. Besonderer Dank gilt Giuseppe Verni, der das gesamte Manuskript mit großem Interesse und Hingabe gelesen und viele hilfreiche Ratschläge gegeben hat.

Der Originaltitel ist eine verspätete Hommage an das Buch *Unix for the Impatient* von Paul W. Abrahams und Bruce R. Larson (Addison-Wesley Professional).

Teil I

1 Einleitung

D3.js (oder kurz D3 für *Data-Driven Documents*, also »datengestützte Dokumente«) ist eine JavaScript-Bibliothek, die dazu dient, den DOM-Baum (Document Object Model) zu bearbeiten, um Informationen grafisch darzustellen. Sie ist zu einem De-facto-Standard für Infografiken im Web geworden.

Trotz ihrer Beliebtheit wird D3 eine steile Lernkurve nachgesagt. Meiner Meinung nach liegt das nicht daran, dass D3 kompliziert wäre (was sie nicht ist), und auch nicht an ihrer umfangreichen API (die zwar groß, aber gut strukturiert und sehr gut gestaltet ist). Viele der Schwierigkeiten, mit denen neue Benutzer zu kämpfen haben, sind, so glaube ich, auf *falsche Vorstellungen* zurückzuführen. Da D3 verwendet wird, um eindrucksvolle Grafiken zu erstellen, liegt es nahe, sie als »Grafikbibliothek« anzusehen, die den Umgang mit grafischen Grundelementen erleichtert und es ermöglicht, gängige Arten von Plots zu erstellen, ohne sich um die Einzelheiten kümmern zu müssen. Neulinge, die sich D3 mit dieser Erwartung nähern, sind unangenehm überrascht von der Ausführlichkeit, mit der so grundlegende Dinge wie die die Farbe eines Elements festzulegen sind. Und was hat es

mit diesen ganzen »Selections« auf sich? Warum kann man nicht einfach ein leinwandartiges Element verwenden?

Das Missverständnis beruht darauf, dass D3 eben *keine* Grafikbibliothek ist, sondern eine JavaScript-Bibliothek zur Bearbeitung des DOM-Baums. Ihre Grundbausteine sind keine Kreise und Rechtecke, sondern Knoten und DOM-Elemente. Die typischen Vorgehensweisen bestehen nicht darin, grafische Formen auf eine »Leinwand« (Canvas) zu zeichnen, sondern Elemente durch Attribute zu formatieren. Die »aktuelle Position« wird nicht durch *x*/*y*-Koordinaten auf einer Leinwand angegeben, sondern durch die Auswahl von Knoten im DOM-Baum.

Das führt – aus meiner Sicht – zu dem zweiten Hauptproblem für viele neue Benutzer: Bei D3 handelt es sich um eine Webtechnologie, die sich auf andere Webtechnologien stützt, auf die DOM-API und das Ereignismodell, auf CSS-Selektoren und -Eigenschaften (Cascading Style Sheets), auf das JavaScript-Objektmodell und natürlich auf das SVG-Format (Scalable Vector Graphics). In vielen Aspekten stellt D3 nur eine relativ dünne Schicht über diesen Webtechnologien dar und ihr eigenes Design spiegelt oft die zugrunde liegenden APIs wider. Das führt zu einer sehr umfangreichen und uneinheitlichen Umgebung. Wenn Sie bereits mit dem kompletten Satz moderner Webtechnologien vertraut sind, der als HTML5 bekannt ist, werden Sie sich darin zu Hause fühlen, doch wenn *nicht*, dann kann das Fehlen einer ausgeprägten, einheitlichen Abstraktionsschicht ziemlich verwirrend sein.

Glücklicherweise müssen Sie nicht sämtliche dieser zugrunde liegenden Technologien ausgiebig studieren. D3 erleichtert ihre Nutzung und bietet eine erhebliche Vereinheitlichung und Abstraktion. Der einzige Bereich, in dem es definitiv nicht möglich ist, sich einfach durchzumogeln, ist SVG. Sie müssen auf jeden Fall ein ausreichendes Verständnis von SVG mitbringen, und zwar nicht nur der darstellenden Elemente, sondern auch der Strukturelemente, die steuern, wie die Informationen in einem Graphen gegliedert sind. Alles, was Sie wissen müssen, habe ich in Anhang B zusammengestellt. Wenn Sie mit SVG nicht vertraut sind, sollten Sie diesen Anhang durcharbeiten, bevor Sie sich an den Rest des Buches machen. Sie werden später dafür dankbar sein.

Zielgruppe

Dieses Buch ist für *Programmierer* und *Wissenschaftler* gedacht, die D3 zu ihrem Werkzeugkasten hinzufügen möchten. Ich gehe davon aus, dass Sie ausreichende Erfahrungen als Programmierer aufweisen und ohne Schwierigkeiten mit Daten und Grafiken arbeiten können. Allerdings erwarte ich nicht, dass Sie mehr als oberflächliche Kenntnisse in professioneller Webentwicklung haben.

Folgende Voraussetzungen sollten Sie mitbringen:

- Kenntnisse in einer oder zwei Programmiersprachen (nicht unbedingt JavaScript) und ausreichend Selbstvertrauen, um sich die Syntax einer neuen Sprache aus ihrer Referenz anzueignen.
- Vertrautheit mit modernen Programmierkonzepten, also nicht nur mit Schleifen, Bedingungen und gewöhnlichen Datenstrukturen, sondern auch mit Closures und Funktionen höherer Ordnung.
- Grundkenntnisse in XML und der hierarchischen Struktur von XML-Dokumenten. Ich erwarte, dass Sie das DOM kennen und wissen, dass es die Elemente einer Webseite als Knoten eines Baums behandelt. Allerdings setze ich nicht voraus, dass Sie mit der ursprünglichen DOM-API oder einem ihrer modernen Nachfolger (wie jQuery) vertraut sind.
- Einfache Kenntnisse in HTML und CSS (Sie sollten in der Lage sein, `<body>`- und `<p>`-Tags usw. zu erkennen und zu verwenden) sowie eine gewisse Vertrautheit mit der Syntax und den Mechanismen von CSS.

Insbesondere aber gehe ich davon aus, dass meine Leser *ungeduldig* sind: erfahren und fähig, aber frustriert von früheren Versuchen, mit D3 zurechtzukommen. Wenn Sie sich darin wiedererkennen, dann ist dieses Buch genau das richtige für Sie!

Warum D3?

Warum sollten sich Programmierer und Wissenschaftler – oder überhaupt irgendwelche Personen, die nicht vorrangig Webentwickler sind – mit D3 beschäftigen? Dafür gibt es vor allem die folgenden Gründe:

- D3 bietet eine bequeme Möglichkeit, um Grafiken im Web zu verbreiten. Wenn Sie mit Daten und Visualisierungen arbeiten, kennen Sie ja den Vorgang: Sie erstellen Diagramme in ihrem bevorzugten Plotprogramm, speichern die Ergebnisse als PNG oder PDF und erstellen dann eine Webseite mit `<img>`-Tags, sodass andere Ihre Arbeit einsehen können. Wäre es nicht schöner, wenn Sie Ihre Diagramme in einem Schritt erstellen und veröffentlichen könnten?
- Noch wichtiger ist jedoch der Umstand, dass es mit D3 auf einfache und komfortable Weise möglich ist, *animierte* und *interaktive* Grafiken zu erstellen. Dieser Punkt kann gar nicht deutlich genug herausgestellt werden: Die Visualisierung wissenschaftlicher Daten profitiert genauso von Animation und Interaktivität wie jeder Bereich, allerdings ließ sich dies früher nur schwer erreichen, es erforderte meistens komplizierte und unpassende Technologien (haben Sie sich jemals an Xlib-Programmierung versucht?) oder spezialisierte

und oftmals teure kommerzielle Lösungen. Mit D3 können Sie diese Hürden überwinden und Ihre zeitgemäßen Visualisierungsbedürfnisse erfüllen.

- Ganz abgesehen von Grafiken ist D3 ein gut zugängliches, leicht zu lernenden und leicht zu verwendendes Framework zur DOM-Bearbeitung für alle möglichen Zwecke. Wenn Sie hin und wieder mit dem DOM arbeiten müssen, kann D3 dazu völlig ausreichen, sodass Sie nicht auch noch das Beherrschen aller anderen Frameworks und APIs für die Webprogrammierung erlernen müssen. Der Aufbau der Bibliothek selbst ist außerdem ein bemerkenswertes Modell der Funktionalitäten für gängige Datenbearbeitungs- und Visualisierungsaufgaben, die sie im Lieferzustand mitbringt.

Vor allem aber bin ich der Meinung, dass D3 eine *emanzipierende* Technologie ist, die es ihren Benutzern erlaubt, ihren Bestand an verfügbaren Lösungsmöglichkeiten ganz allgemein zu erweitern. Die bemerkenswertesten Nutzanwendungen von D3 sind wahrscheinlich diejenigen, die noch nicht entdeckt wurden.

Was Sie in diesem Buch finden werden ...

Dieses Buch soll eine möglichst *umfassende* und gleichzeitig *knappe* Einführung in D3 sein, die die wichtigsten Aspekte in ausreichender Tiefe darstellt.

- Es soll als komfortables, *zentrales* Nachschlagewerk dienen, da es sowohl die API-Referenzdokumentation als auch Hintergrundinformationen zu verwandten Themen bietet, mit denen Sie nicht unbedingt vertraut sein müssen (wie SVG, JavaScript und das DOM, aber auch Farbräume und das `canvas`-Element von HTML).
- Der Schwerpunkt liegt auf *Mechanismen* und *Gestaltungsprinzipien* und nicht auf vorgefertigten »Kochrezepten«. Ich gehe davon aus, dass Sie D3 gründlich genug lernen wollen, um es für Ihre eigenen und möglicherweise neuartigen und unvorhergesehenen Zwecke einzusetzen.

Im Grunde genommen wünsche ich mir, dass dieses Buch Sie auf die Dinge vorbereitet, die Sie mit D3 tun können, an die ich aber selbst niemals gedacht hätte.

... und was nicht!

Dieses Buch ist bewusst auf D3 und die Möglichkeiten und Mechanismen dieser Bibliothek beschränkt. Daher fehlt eine ganze Reihe von Dingen:

- Ausführliche Fallstudien und Kochrezepte
- Einführungen in Datenanalyse, Statistik und Grafikdesign

- Andere JavaScript-Frameworks als D3
- Allgemeine Erörterung der modernen Webentwicklung

Ich möchte insbesondere die beiden letzten Punkte betonen. In diesem Buch geht es *ausschließlich* um D3, ohne Nutzung oder Abhängigkeiten von anderen JavaScript-Frameworks und Bibliotheken. Das ist volle Absicht: Ich möchte D3 auch solchen Lesern zugänglich machen, die mit dem reichhaltigen, aber uneinheitlichen Umfeld von JavaScript nicht vertraut sind oder sogar auf Kriegsfuß stehen. Aus demselben Grund werden in diesem Buch auch keine anderen Themen der modernen Webentwicklung besprochen. Insbesondere finden Sie hier keinerlei Diskussionen zur *Browserkompatibilität* und ähnlichen Themen. Ich setze voraus, dass Sie einen modernen, aktuellen JavaScript-fähigen Browser verwenden, der in der Lage ist, SVG darzustellen.[1]

Ein weiterer Aspekt, der nicht behandelt wird, betrifft die Unterstützung von D3 für geografische und raumbezogene Informationen. Diese Themen sind zwar wichtig, aber gut überschaubar, sodass es nicht allzu schwierig sein sollte, sie durch die Lektüre der D3-Referenzdokumentation (*https://github.com/d3/d3/blob/master/API.md*) zu erlernen, wenn Sie mit den Grundlagen von D3 vertraut sind.

Ein Leitfaden durch dieses Buch

Dieses Buch ist kontinuierlich aufgebaut. Von Kapitel zu Kapitel wird neuer Stoff eingeführt. Allerdings können Sie insbesondere die hinteren Kapitel in beliebiger Reihenfolge lesen, nachdem Sie die Grundlagen in der ersten Hälfte des Buches erlernt haben. Ich schlage die folgende Vorgehensweise vor:

1. Sofern Sie nicht bereits solide Kenntnisse in SVG haben, empfehle ich Ihnen dringend, mit Anhang B anzufangen. Ohne diese Kenntnisse ergibt alles andere nicht viel Sinn.
2. Lesen Sie auf jeden Fall Kapitel 2 zur Einführung und um Ihre Erwartungen für die kommenden Themen richtig einzuordnen.
3. Kapitel 3 ist Pflichtlektüre. Selections bilden *das* grundlegende Ordnungsprinzip von D3. Sie bieten nicht nur Zugriff auf den DOM-Baum, sondern kümmern sich auch um die Verknüpfung zwischen den DOM-Elementen und den Datenmengen. Praktisch jedes D3-Programm beginnt mit einer Selection, und das Verständnis, was eine Selection ist und was sie kann, ist für die Arbeit mit D3 unbedingt erforderlich.

1 Das ist auch im Einklang mit dem Geist von D3. So heißt es auf der D3-Website: »D3 ist keine Kompatibilitätsschicht. Wenn Ihr Browser die Standards nicht unterstützt, haben Sie Pech gehabt« (*https://github.com/d3/d3/wiki*).

4. Streng genommen ist Kapitel 4 zum Thema Ereignisbehandlung, Interaktivität und Animation optional. Da diese Dinge jedoch zu den faszinierenden Möglichkeiten gehören, die D3 bietet, wäre es schade, dieses Kapitel zu überspringen.
5. Kapitel 5 ist wichtig, da es die Grundprinzipien des Designs von D3 beschreibt (wie Komponenten und Layouts) und eine Einführung in allgemein nützliche Techniken gibt (wie SVG-Transformationen und benutzerdefinierte Komponenten).
6. Die restlichen Kapitel können Sie im Großen und Ganzen in beliebiger Reihenfolge lesen, wann immer Sie etwas über die jeweiligen Themen wissen müssen. Insbesondere möchte ich jedoch Ihre Aufmerksamkeit auf Kapitel 7 mit seiner ausführlichen Beschreibung der unscheinbaren, aber äußerst vielseitigen Skalierungsobjekte sowie auf die Vielzahl der Funktionen zur Handhabung von Arrays in Kapitel 9 lenken.

Konventionen

In diesem Abschnitt werden einige Vereinbarungen vorgestellt, die in diesem Buch gelten.

Konventionen in der D3-API

In der D3-API gelten einige Konventionen, die die Nützlichkeit der Bibliothek erhöhen. Bei einigen davon handelt es sich nicht um D3-spezifische Regeln, sondern um gängige JavaScript-Idiome, mit denen Sie aber möglicherweise nicht vertraut sind, wenn Sie nicht selbst in JavaScript programmieren. Ich gebe Sie hier einmal gesammelt an, um die nachfolgenden Erörterungen von überflüssigen Wiederholungen frei zu halten.

- D3 ist vor allem eine Schicht für den Zugriff auf den DOM-Baum. In der Regel versucht D3 nicht, die zugrunde liegenden Technologien zu kapseln, sondern bietet stattdessen komfortable, aber allgemein gehaltene Handles dafür. Beispielsweise stellt D3 keine eigenen Abstraktionen für Kreise und Rechtecke bereit, sondern gibt Programmierern unmittelbaren Zugriff auf die Elemente von SVG zum Erstellen von grafischen Formen. Diese Vorgehensweise bietet den Vorteil einer enormen Anpassungsfähigkeit, denn dadurch ist D3 nicht an eine bestimmte Technologie oder Version gebunden. Der Nachteil besteht darin, dass Programmierer neben D3 auch die zugrunde liegenden Technologien kennen müssen, da D3 selbst keine vollständige Abstraktionsschicht bildet.

- Da JavaScript keine formalen Funktionssignaturen erzwingt, sind technisch gesehen alle Funktionsargumente optional. Viele D3-Funktionen nutzen daher das folgende Idiom: Beim Aufruf *mit* geeigneten Argumenten dienen sie als *Set-Methoden* (sie setzen die entsprechende Eigenschaft auf den übergebenen Wert), beim Aufruf *ohne* Argumente dagegen als *Get-Methoden* (sie geben den aktuellen Wert der Eigenschaft zurück). Um eine Eigenschaft komplett zu *entfernen*, rufen Sie die entsprechende Set-Methode mit dem Argument `null` auf.
- Beim Aufruf als Set-Methoden geben Funktionen gewöhnlich einen Verweis auf das aktuelle Objekt zurück und ermöglichen damit eine Methodenverkettung. (Dieses Idiom wird so konsequent verwendet und lässt sich intuitiv nutzen, sodass ich nur noch selten ausdrücklich darauf hinweisen werde.)
- Anstelle eines Wertes können viele Set-Funktionen von D3 auch eine *Zugriffsfunktion* als Argument entgegennehmen, wobei der von ihr zurückgegebene Wert dazu verwendet wird, die betreffende Eigenschaft festzulegen. Nicht alle Zugriffsfunktionen in D3 erwarten die gleichen Parameter, aber miteinander verwandte D3-Funktionen rufen ihre Zugriffsfunktionen immer auf einheitliche Weise auf. Die Einzelheiten zu den Zugriffsfunktionen werden jeweils bei den entsprechenden D3-Funktionen angegeben.
- Einige wichtige Funktionalitäten von D3 sind als *Funktionsobjekte* implementiert. Sie führen ihre Hauptaufgabe aus, wenn sie als Funktionen aufgerufen werden. Gleichzeitig aber sind sie auch Objekte mit Memberfunktionen und einem internen Status. (Beispiele dafür sind die *Skalierungsobjekte* aus Kapitel 7 sowie die *Generatoren* und *Komponenten* aus Kapitel 5.) Ein häufig genutztes Muster besteht darin, ein solches Objekt zu instanziieren, es mithilfe seiner Memberfunktionen zu konfigurieren und es schließlich aufzurufen, damit es seinen Zweck erfüllen kann. Häufig wird für den endgültigen Aufruf nicht die explizite Funktionsaufrufsyntax verwendet, sondern eine der JavaScript-Vorgehensweisen für »synthetische« Funktionsaufrufe: Das Funktionsobjekt wird an eine andere Funktion übergeben (etwa `call()`), die die erforderlichen Argumente bereitstellt und das Funktionsobjekt auswertet.

Konventionen für die API-Referenztabellen

In diesem Buch finden Sie Tabellen mit Erklärungen zu einzelnen Teilen der D3-API. Die Einträge in diesen Tabellen sind nach Relevanz geordnet, wobei zusammengehörige Funktionen auch zusammen aufgeführt werden.

- D3-Funktionen werden entweder für das globale Objekt `d3` oder als Memberfunktionen von D3-Objekten aufgerufen. Bei einigen Funktionen ist beides

möglich. Wird eine Funktion durch ein Objekt aufgerufen, so wird dieses Objekt als *Empfänger* des Methodenaufrufs bezeichnet. Innerhalb der Memberfunktion zeigt die Variable `this` auf dieses Objekt.

- In allen API-Referenztabellen ist jeweils der Typ des Empfängers in der Tabellenunterschrift angegeben. Die Tabellen verweisen nicht ausdrücklich auf den Prototyp des Objekts.
- Bei den Funktionssignaturen wird *versucht*, die Typen der einzelnen Argumente anzugeben, aber da viele Funktionen eine breite Palette an Argumenttypen akzeptieren, lässt sich eine eindeutige Schreibweise nicht umsetzen. Um alle Einzelheiten zu erfassen, müssen Sie die Beschreibung lesen. *Eckige Klammern* stehen für Arrays. Optionale Funktionsargumente werden nicht ausdrücklich angegeben.

Konventionen für die Codebeispiele

Die Codebeispiele sollen die Merkmale und Mechanismen von D3 veranschaulichen. Um den Kernpunkt jeweils möglichst deutlich zu zeigen, wurden die Beispiele auf das Notwendige reduziert. Ich habe auf die meisten Feinheiten wie ansprechende Farben und interessante Datenmengen verzichtet. Meistens verwende ich Primärfarben und kleine und einfache Datenmengen.

Dafür ist jedes Beispiel in sich abgeschlossen, kann wie gezeigt ausgeführt werden und erstellt den zugehörigen Graphen. Bis auf wenige Ausnahmen gebe ich keine Codefragmente an. Meiner Erfahrung nach ist es besser, einfache Beispiele komplett darzustellen, anstatt nur die »interessanten Teile« längerer Beispiele zu zeigen. Auf diese Weise besteht keine Gefahr, dass der Gesamtzusammenhang verloren geht. Alle Beispiele sind ausführbar und können nach Belieben erweitert und ausgeschmückt werden.

Namenskonventionen

In den Beispielen gilt die folgende Namenskonvention für Variablen:

- Für einzelne Objekte wird der erste Buchstabe der englischen Bezeichnung verwendet: `c` für »circle« (Kreis), `p` für »point« (Punkt) usw. Bei Sammlungen wird ein Plural-s angehängt. So ist etwa `cs` ein Array aus Kreisen und `ps` ein Array aus Punkten.
- Häufig verwendete Größen haben davon abweichende Bezeichnungen. Pixel heißen `px` und Skalierungsobjekte `sc` (»scale«). Generationen und Komponenten sind Funktionsobjekte, die etwas erzeugen, und werden daher `mkr` (»maker«) genannt.

- Der Buchstabe `d` wird allgemein verwendet, um in anonymen Funktionen »das aktuelle Etwas« zu bezeichnen. Bei der Arbeit mit D3-Selections ist `d` gewöhnlich ein einzelner Datenpunkt, der an ein DOM-Element gebunden ist, bei der Arbeit mit Arrays dagegen ein Arrayelement (z. B. in `ds.map( d => +d )`).
- Datenmengen werden als `data` oder `ds` bezeichnet.
- Selections, die für ein `<svg>`- oder `<g>`-Element stehen, kommen häufig vor. Werden sie einer Variablen zugewiesen, so werden sie als `svg` bzw. `g` bezeichnet.

Aufbau der Quelldateien

Ab Kapitel 3 setze ich bei jedem Codelisting voraus, dass die Seite bereits ein `<svg>`-Element mit einem eindeutigen `id`-Attribut und ordnungsgemäß festgelegten `width`- und `height`-Attributen enthält. Der Beispielcode wählt dann dieses SVG-Element anhand seines `id`-Attributs aus und weist diese Selection häufig einer Variablen zu, um später darauf verweisen zu können:

```
var svg = d3.select( "#fig" );
```

Dadurch wird die Mehrdeutigkeit vermieden, die sich bei der Verwendung eines allgemeineren Selektors (wie `d3.select( "svg" )`) einstellen würde, und es erleichtert, mehrere Beispiele in eine einzige HTML-Seite aufzunehmen.

Zu jedem Graphen gibt es eine JavaScript-Funktion, die die SVG-Elemente des Diagramms dynamisch erstellt. Vereinbarungsgemäß beginnen die Funktionsnamen mit `make`, worauf der Wert des `id`-Attributs für das SVG-Zielelement folgt.

Bis auf Kapitel 2 gibt es in jedem Kapitel nur eine HTML-Seite und eine JavaScript-Datei. (Von wenigen Ausnahmen abgesehen nehme ich den JavaScript-Code nicht unmittelbar in die HTML-Seite auf.)

Plattform, JavaScript-Version und Browser

Um die Beispiele auszuführen, brauchen Sie einen lokalen oder gehosteten Webserver (siehe Anhang A). Die Beispiele sollten in jedem modernen Browser mit aktiviertem JavaScript laufen. Zurzeit sind verschiedene Versionen von JavaScript im Umlauf.[2] In den Codebeispielen wird fast ausschließlich »klassisches« JavaScript (ES5, freigegeben 2009/2011) ohne weitere Frameworks und Bibliotheken verwendet. Für die folgenden drei Merkmale ist jedoch eine jüngere Version von JavaScript (ES6, veröffentlicht 2015) erforderlich:

- Die knappe *Pfeilschreibweise* für anonyme Funktionen (siehe Anhang C), die in den Beispielen durchgehend verwendet wird.

2 Siehe *https://en.wikipedia.org/wiki/ECMAScript*.

- Destrukturierende Zuweisungen (`[a, b] = [b, a]`), die an einigen Stellen verwendet werden.
- In mehreren Beispielen wird mithilfe von D3-Wrappern für die Fetch-API (siehe Kapitel 6) auf Remoteressourcen zugegriffen. Dafür ist das JavaScript-Objekt `Promise` erforderlich.

2

Los geht's: erste Graphen mit D3

Zu Anfang wollen wir einige Beispiele durcharbeiten, die die Möglichkeiten von D3 demonstrieren und Sie in die Lage versetzen, selbst Aufgaben aus der Praxis zu lösen – und sei es auch nur, indem Sie diese Beispiele entsprechend anpassen. Die ersten beiden Beispiele in diesem Kapitel zeigen, wie Sie aus Datendateien die gebräuchlichen *Streu-* und *XY-Diagramme* erstellen, und zwar komplett mit Achsen. Die Diagramme sehen nicht sehr hübsch aus, erfüllen aber ihren Zweck, und es ist auf einfache Weise möglich, sie zu verschönern und die Prinzipien auf andere Datenmengen zu übertragen. Das dritte Beispiel ist nicht vollständig, sondern dient mehr der Veranschaulichung, um Ihnen zu zeigen, wie einfach es ist, Ereignisbehandlung und Animationen in Ihre Dokumente aufzunehmen.

Erstes Beispiel: eine einzige Datenmenge

Um D3 kennenzulernen, betrachten wir die kleine, einfache Datenmenge aus Beispiel 2–1. Bei der grafischen Ausgabe dieser Daten mithilfe von D3 kommen wir schon mit vielen der Grundprinzipien in Berührung.

Beispiel 2–1: Eine einfache Datenmenge (examples-simple.tsv)

```
x       y1
100     50
200     100
300     150
400     200
500     250
```

Wie bereits in Kapitel 1 erwähnt, ist D3 eine JavaScript-Bibliothek zur Bearbeitung des DOM-Baums, um Informationen grafisch darzustellen. Das deutete schon darauf hin, dass jeder D3-Graph über mindestens *zwei* oder *drei* »bewegliche Teile« verfügt:

- Eine HTML-Datei mit dem DOM-Baum
- Eine JavaScript-Datei oder ein Skript mit den Befehlen zur Bearbeitung des DOM-Baums
- Häufig eine Datei oder eine andere Ressource mit der Datenmenge

Beispiel 2–2 zeigt die gesamte HTML-Datei.

Beispiel 2–2: Eine HTML-Datei, die ein SVG-Element definiert

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <script src="d3.js"></script>    ❶
  <script src="examples-demo1.js"></script>    ❷
</head>

<body onload="makeDemo1()">    ❸
  <svg id="demo1" width="600" height="300"
       style="background: lightgrey" />    ❹
</body>
</html>
```

Ja, Sie sehen richtig – die HMTL-Datei ist im Grunde genommen leer! Alle Aktionen finden im JavaScript-Code statt. Sehen wir uns kurz die wenigen Dinge an, die in dem HTML-Dokument geschehen:

❶ Als Erstes lädt das Dokument *d3.js*, also die Bibliothek D3.

❷ Danach lädt das Dokument unsere JavaScript-Datei. Sie enthält alle Befehle zur Definition des Graphen, den wir anzeigen wollen.

❸ Das `<body>`-Tag definiert einen `onload`-Ereignishandler, der ausgelöst wird, wenn der Browser das `<body>`-Element komplett geladen hat. Die Ereignishand-

lerfunktion `makeDemo1()` ist in unserer JavaScript-Datei *examples-demo1.js* definiert.[1]

❹ Am Ende enthält das Dokument ein SVG-Element mit einer Größe von 600 × 300 Pixeln. Es hat einen hellgrauen Hintergrund, damit Sie es erkennen können, ist sonst aber leer.

Der dritte und letzte Bestandteil ist die JavaScript-Datei, die Sie in Beispiel 2–3 sehen.

Beispiel 2–3: Befehle für Abbildung 2–1

```
function makeDemo1() {    ❶
    d3.tsv( "examples-simple.tsv" )    ❷
        .then( function( data ) {    ❸ ❹
            d3.select( "svg" )    ❺
                .selectAll( "circle" )    ❻
                .data( data )    ❼
                .enter()    ❽
                .append( "circle" )    ❾
                .attr( "r", 5 ).attr( "fill", "red" )    ❿
                .attr( "cx", function(d) { return d["x"] } )    ⓫
                .attr( "cy", function(d) { return d["y"] } );
        } );
}
```

Wenn Sie diese drei Dateien (die Datendatei, die HTML-Seite und die JavaScript-Datei) zusammen mit der Bibliotheksdatei *d3.js* in ein gemeinsames Verzeichnis stellen und die Seite im Browser laden, sollte der Browser den Graphen aus Abbildung 2–1[2] anzeigen.

Sehen wir uns nun nacheinander die einzelnen JavaScript-Befehle an:

❶ Das Skript definiert nur eine einzige Funktion, nämlich den Callback `makeDemo1()`, der aufgerufen wird, wenn die HTML-Seite vollständig geladen ist.

❷ Die Funktion lädt die Datendatei mithilfe der `Fetch`-Funktion `tsv()` (sie »ruft sie ab«, daher der Bezug zu »to fetch«). In D3 sind mehrere Funktionen definiert, um Dateiformate mit Trennzeichen zu lesen. `tsv()` ist für tabulatorgetrennte Dateien bestimmt.

---

1 Die Definition eines Ereignishandlers über das `onload`-Tag wird oft missbilligt, weil dadurch JavaScript-Code in HTML eingebettet wird. Moderne Alternativen finden Sie in Anhang A und C.

2 Sie sollten in der Lage sein, die Seite und die zugehörige JavaScript-Datei zu laden, indem Sie dem Browser das lokale Verzeichnis als Zieladresse angeben. Allerdings kann es sein, dass der Browser sich weigert, die Datendatei auf diese Weise zu laden. Daher ist es gewöhnlich notwendig, bei der Arbeit mit D3 einen Webserver auszuführen. Mehr darüber erfahren Sie in Anhang A.

❸ Wie alle Funktionen aus der JavaScript-API `Fetch` gibt auch `tsv()` ein `Promise`-Objekt zurück. Ein solches Objekt enthält ein Resultset und einen Callback und ruft Letzteren auf, wenn das Resultset vollständig und zur Verarbeitung bereit ist. Anschließend stellt das `Promise`-Objekt die Funktion `then()` bereit, um den gewünschten Callback zu registrieren. (Mehr über JavaScript-Promises erfahren Sie im Abschnitt »*Promises in JavaScript*« auf *S. 120*.)

❹ Der Callback, der nach dem Laden der Datei aufgerufen werden soll, wird als anonyme Funktion definiert, die den Inhalt der Datendatei als Argument erhält. Die Funktion `tsv()` gibt den Inhalt der tabulatorgetrennten Datei als Array von JavaScript-Objekten zurück. Jede Zeile in der Datei ergibt ein Objekt, wobei die Eigenschaftennamen in der Kopfzeile der Eingabedatei definiert sind.

❺ Wir wählen das `<svg>`-Element als die Position im DOM-Baum, die den Graphen enthalten soll. Die Funktionen `select()` und `selectAll()` nehmen einen CSS-Selektorstring entgegen (siehe »*CSS-Selektoren*« auf *S. 38* in Kapitel 3) und geben die übereinstimmenden Knoten zurück – `select()` nur die erste Übereinstimmung, `selectAll()` eine Sammlung aller Übereinstimmungen.

❻ Als Nächstes wählen wir *alle* `<circle>`-Elemente innerhalb des `<svg>`-Knotens aus. Das mag absurd erscheinen, denn schließlich gibt es darin gar keine `<circle>`-Elemente! Dieser merkwürdige Aufruf stellt jedoch kein Problem dar, da `selectAll("circle")` lediglich eine *leere* Sammlung (von `<circle>`-Elementen) zurückgibt, sondern erfüllt sogar eine wichtige Funktion, da er mit dieser leeren Sammlung einen *Platzhalter* erstellt, den wir anschließend füllen werden. Das ist ein gängiges D3-Idiom, um Graphen mit neuen Elementen zu versehen.

❼ Als Nächstes verknüpfen wir die Sammlung der `<circle>`-Elemente mit der Datenmenge. Dazu verwenden wir den Aufruf `data(data)`. Es ist wichtig, zu erkennen, dass die beiden Sammlungen (DOM-Elemente auf der einen Seite und Datenpunkte auf der anderen) nicht im Ganzen miteinander verbunden werden. Stattdessen versucht D3 eine 1:1-Beziehung zwischen den DOM-Elementen und den Datenpunkten herzustellen: *Jeder Datenpunkt wird durch ein eigenes DOM-Element* dargestellt, das wiederum seine Eigenschaften (wie Position, Farbe und Erscheinungsbild) aus den Informationen über den Datenpunkt bezieht. Der Aufbau und die Pflege dieser 1:1-Beziehungen zwischen einzelnen Datenpunkten und den zugehörigen DOM-Elementen ist ein grundlegendes Merkmal von D3. (In Kapitel 3 werden wir uns diesen Vorgang noch ausführlicher ansehen.)

❽ Die Funktion `data()` gibt die Sammlung der Elemente zurück, die mit den einzelnen Datenpunkten verknüpft wurden. Zurzeit kann D3 die einzelnen Datenpunkte nicht mit `<circle>`-Elementen verknüpfen, da es (noch) keine

gibt. Die zurückgegebene Sammlung ist daher leer. Allerdings bietet D3 mithilfe der (merkwürdig benannten) Funktion `enter()` auch Zugriff auf alle restlichen Datenpunkte, für die keine übereinstimmenden DOM-Elemente gefunden werden konnten. Die übrigen Befehle werden für die einzelnen Elemente in dieser »Restsammlung« aufgerufen.

❾ Als Erstes wird der Sammlung der `<circle>`-Elemente innerhalb des in Zeile 6 ausgewählten SVG-Elements ein `<circle>`-Element angehängt.

❿ Anschließend werden einige fixe (also nicht datenabhängige) Attribute und Formatierungen festgelegt, nämlich der Radius (das Attribut `r`) und die Füllfarbe.

⓫ Schließlich wird die Position der einzelnen Kreise aufgrund des Werts bestimmt, den der zugehörige Datenpunkt hat. Die Attribute `cx` und `cy` der einzelnen `<circle>`-Elemente werden jeweils auf der Grundlage der Einträge in der Datendatei festgelegt. Statt eines festen Werts stellen wir *Zugriffsfunktionen* bereit, die zu einem gegebenen Eintrag der Datendatei (also zu einem einzeiligen Datensatz) den entsprechenden Wert zurückgeben.

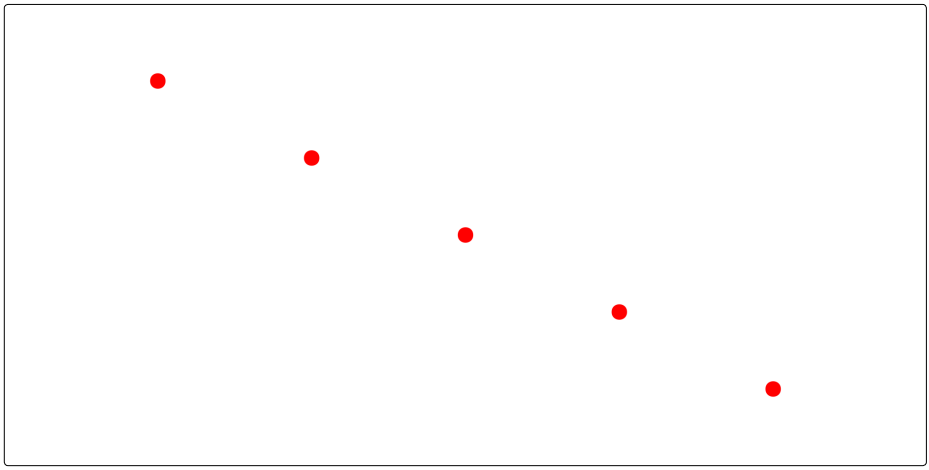

Abb. 2–1 *Grafische Darstellung einer einfachen Datenmenge (siehe Beispiel 2–3)*

Um ehrlich zu sein: Für einen so einfachen Graphen ist das ein erheblicher Aufwand! Hieran können Sie schon etwas erkennen, was im Laufe der Zeit noch deutlicher wird: D3 ist keine Grafikbibliothek und erst recht kein Werkzeug zur Diagrammerstellung, sondern eine Bibliothek, um den DOM-Baum zu bearbeiten und dadurch Informationen grafisch darzustellen. Sie werden die ganze Zeit damit beschäftigt sein, Operationen an Teilen des DOM-Baums vorzunehmen (die Sie mit Selections auswählen). Außerdem werden Sie feststellen, dass bei D3 nicht

gerade wenig Tipparbeit anfällt, da Sie Attributwerte bearbeiten und eine Zugriffsfunktion nach der anderen schreiben müssen. Andererseits ist der Code meiner Meinung nach zwar sehr wortreich, aber auch sauber und unkompliziert.

Wenn Sie ein wenig mit den Beispielen herumspielen, werden Sie womöglich noch einige weitere Überraschungen erleben. Beispielsweise ist die Funktion `tsv()` ziemlich wählerisch: Spalten *müssen* durch Tabulatoren getrennt sein, Weißraum wird *nicht* ignoriert, eine Kopfzeile *muss* vorhanden sein usw. Bei genauerer Untersuchung der Datenmenge und des Graphen werden Sie schließlich auch feststellen, dass das Diagramm nicht korrekt ist, sondern auf dem Kopf steht! Das liegt daran, dass SVG »grafische Koordinaten« verwendet, bei denen die horizontale Achse zwar wie üblich von links nach rechts, die vertikale aber von oben nach unten läuft.

Um diese ersten Eindrücke besser einordnen zu können, fahren wir mit einem zweiten Beispiel fort.

Zweites Beispiel: zwei Datenmengen

Für unser zweites Beispiel verwenden wir die Datenmenge aus Beispiel 2–4. Sie sieht zwar fast genauso harmlos aus wie die vorherige, aber bei genauerer Betrachtung werden einige zusätzliche Schwierigkeiten deutlich. Sie enthält nämlich nicht nur *zwei* Datenmengen (in den Spalten `y1` und `y2`), sondern umfasst auch Datenbereiche, die unserer Aufmerksamkeit bedürfen. Im vorherigen Beispiel konnten wir die Datenwerte unmittelbar als Pixelkoordinaten verwenden, doch die Werte der neuen Datenmenge erfordern eine Transformation, bevor sie als Bildschirmkoordinaten genutzt werden können. Wir müssen also etwas mehr Arbeit aufwenden.

Beispiel 2–4: Eine kompliziertere Datenmenge (examples-multiple.tsv)

```
x       y1      y2
1.0     0.001   0.63
3.0     0.003   0.84
4.0     0.024   0.56
4.5     0.054   0.22
4.6     0.062   0.15
5.0     0.100   0.08
6.0     0.176   0.20
8.0     0.198   0.71
9.0     0.199   0.65
```

Symbole und Linien darstellen

Die HTML-Seite für dieses Beispiel ist im Großen und Ganzen identisch mit derjenigen, die wir zuvor benutzt haben (Beispiel 2–2). Allerdings müssen wir die folgende Zeile ersetzen:

```
<script src="examples-demo1.js"></script>
```

Für dieses Beispiel muss sie wie folgt lauten, damit sie auf das neue Skript verweist:

```
<script src="examples-demo2.js"></script>
```

Auch der `onload`-Ereignishandler muss den neuen Funktionsnamen angeben:

```
<body onload="makeDemo2()">
```

Das Skript sehen Sie in Beispiel 2–5, das resultierende Diagramm in Abbildung 2–2.

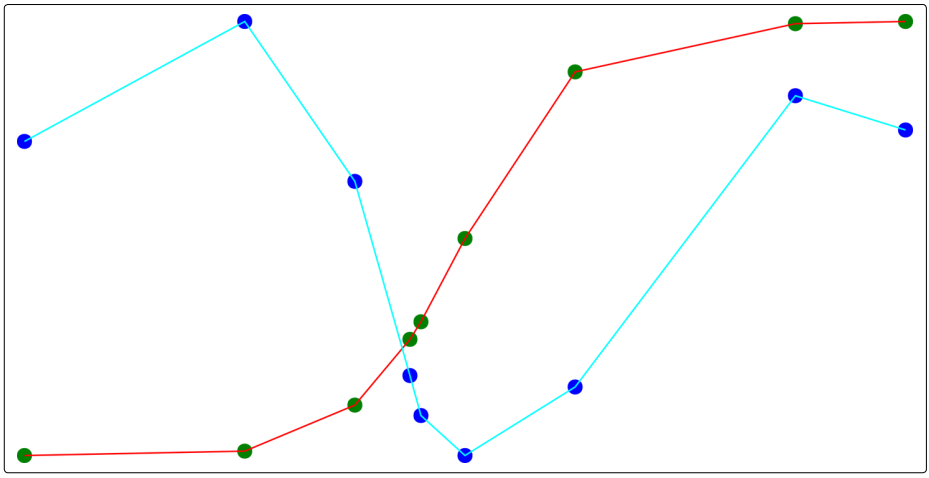

Abb. 2–2 *Einfaches Diagramm der Datenmenge aus Beispiel 2–4 (siehe auch Beispiel 2–5)*

Beispiel 2–5: Befehle für Abbildung 2–2

```
function makeDemo2() {
    d3.tsv( "examples-multiple.tsv" )
        .then( function( data ) {
            var pxX = 600, pxY = 300;    ❶

            var scX = d3.scaleLinear()    ❷
                .domain( d3.extent(data, d => d["x"] ) )    ❸
                .range( [0, pxX] );
```