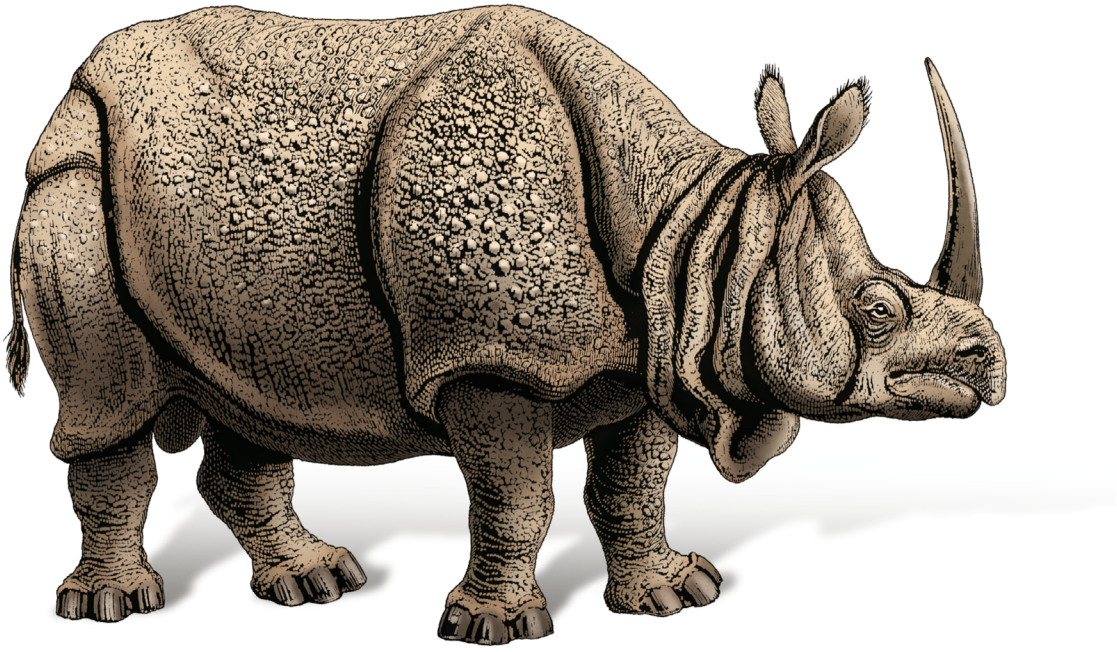
O'REILLY®

Der Klassiker in der 7. Auflage

JavaScript

Das Handbuch für die Praxis

Meistern Sie die beliebte Sprache für Web und Node.js



David Flanagan

Übersetzung von Jørgen W. Lang und Jens Olaf Koch

Lob für *JavaScript: Das Handbuch für die Praxis*, siebte Auflage

»In diesem Buch erfahren Sie Dinge über JavaScript, von denen Sie bisher nicht einmal geahnt hatten, wie dringend Sie sie wissen wollten. Die Qualität Ihres Codes und Ihre Produktivität werden extrem davon profitieren. Davids Wissen über JavaScript, dessen Feinheiten und typische Fehlerquellen ist verblüffend, und seine Qualitäten sind in diesem wirklich außergewöhnlichen Handbuch überall spürbar.«

– Schalk Neethling, Senior Frontend Engineer bei MDN Web Docs

»David Flanagan nimmt die Leser mit auf eine Führung durch die JavaScript-Welt und vermittelt Ihnen ein komplettes Bild aller Sprachfeatures und des gesamten Ökosystems.«

– Sarah Wachs, Frontend Developer und Women Who Code Berlin Lead

»Für alle Entwicklerinnen und Entwickler, die produktiv an Codebasen aus unterschiedlichsten Entwicklungsstadien von JavaScript arbeiten wollen (einschließlich der neuesten Funktionen), wird es sich auszahlen, sich intensiv auf dieses umfassende und maßgebliche Buch einzulassen.«

– Brian Sletten, Präsident von Bosatsu Consulting

7. AUFLAGE

JavaScript – Das Handbuch für die Praxis

Meistern Sie die beliebte Sprache für Web und Node.js

David Flanagan

Deutsche Übersetzung von Jørgen W. Lang und Jens Olaf Koch

O'REILLY®

David Flanagan

Lektorat: Ariane Hesse
Übersetzung: Jørgen W. Lang und Jens Olaf Koch
Korrektorat: Sibylle Feldmann, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Karen Montgomery, Michael Oreal, *www.oreal.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 978-3-96009-157-8
PDF 978-3-96010-491-9
ePub 978-3-96010-492-6
mobi 978-3-96010-493-3

7. Auflage 2021

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: *kommentar@oreilly.de*.

5 4 3 2 1 0

Meinen Eltern, Donna und Matt, voller Liebe und Dankbarkeit.

Inhalt

Vorwort

Dieses Buch behandelt die Programmiersprache JavaScript und die von Webbrowsern und von Node implementierten JavaScript-APIs. Ich habe es für Leser geschrieben, die bereits etwas Programmiererfahrung haben und JavaScript lernen wollen. Es ist aber auch für Programmierer gedacht, die JavaScript bereits verwenden, ihr Wissen darüber aber erweitern und die Sprache wirklich beherrschen wollen. In diesem Buch möchte ich JavaScript umfassend und eindeutig dokumentieren und detailliert in die wichtigsten client- und serverseitigen APIs einführen, die für JavaScript-Programme zur Verfügung stehen. Deshalb ist es ein langes und ausführliches Buch. Ich hoffe aber, dass sein gründliches Studium belohnt wird und sich die Zeit, die Sie mit der Lektüre verbringen, in Form einer höheren Produktivität beim Programmieren schnell wieder bezahlt macht.

Frühere Ausgaben dieses Buchs enthielten einen umfassenden Referenzteil. Ich halte es aber nicht mehr für sinnvoll, dieses Material in gedruckter Form bereitzustellen, weil man heutzutage aktuelles Referenzmaterial jederzeit online findet. Wenn Sie etwas nachschlagen müssen, das mit dem Sprachkern oder clientseitigem JavaScript zu tun hat, empfehle ich Ihnen, die MDN-Website (*https://developer.mozilla.org*) zu besuchen. Für serverseitige Node-APIs sollten Sie direkt zur Quelle gehen und die Node.js-Referenzdokumentation (*https://nodejs.org/api*) konsultieren.

Konventionen, die in diesem Buch verwendet werden

In diesem Buch verwende ich die folgenden typografischen Konventionen:

Kursiv
: Wird zur Hervorhebung und zur Angabe der ersten Verwendung eines Begriffs, aber auch für E-Mail-Adressen, URLs und Dateinamen verwendet.

`Nicht proportional`
: Wird in JavaScript-Code sowie in CSS- und HTML-Listings verwendet – und im Allgemeinen für alles, was Sie beim Programmieren in genau der angegebenen Form per Tastatur eingeben würden.

`Nicht proportional kursiv`
: Wird gelegentlich bei der Erläuterung der JavaScript-Syntax verwendet.

`Nicht proportional fett`
: Zeigt Befehle oder anderen Text an, der vom Benutzer wortgetreu eingegeben werden muss.

Dieses Element kennzeichnet einen allgemeinen Hinweis.

Dieses Element weist auf eine Warnung hin.

Beispielcode

Ergänzendes Material (Codebeispiele, Übungen usw.) für dieses Buch stehen zum Download bereit unter:

- *https://oreil.ly/javascript_defgd7*

Dieses Buch soll Sie bei der Erledigung Ihrer Aufgaben unterstützen. Wenn in diesem Buch Beispielcode angeboten wird, können Sie diesen in Ihren Programmen und Ihrer Dokumentation verwenden. Sie brauchen uns nicht um Erlaubnis zu bitten, es sei denn, Sie reproduzieren einen wesentlichen Teil des Codes. Beispielsweise erfordert das Schreiben eines Programms, das mehrere Codeblöcke aus diesem Buch verwendet, keine Genehmigung. Der Verkauf oder Vertrieb von Beispielen aus O'Reilly-Büchern benötigt dagegen eine Genehmigung. Für die Beantwortung einer Frage, indem aus diesem Buch oder dem Beispielcode zitiert wird, ist keine Genehmigung erforderlich. Das Einbinden einer beträchtlichen Menge an Beispielcode aus diesem Buch in die Dokumentation Ihres Produkts erfordert dagegen eine Genehmigung.

Wir freuen uns über eine Quellenangabe, verlangen sie aber im Allgemeinen nicht. Eine Quellenangabe umfasst in der Regel den Titel, den Autor, den Verlag und die ISBN, zum Beispiel: »*JavaScript: The Definitive Guide*, siebte Auflage, von David Flanagan (O'Reilly). Copyright 2020 David Flanagan, ISBN 9781491952023«.

Wenn Sie glauben, dass Ihre Verwendung der Codebeispiele über einen fairen Gebrauch oder die oben erteilten Genehmigungen hinausgeht, können Sie uns unter *permissions@oreilly.com* gern kontaktieren.

O'Reilly Online-Lernen

Seit mehr als 40 Jahren bietet *O'Reilly Media* (*https://oreilly.com*) Technologie- und Businesstraining sowie Wissen und Einsichten, um Unternehmen zum Erfolg zu verhelfen.

Unser einzigartiges Netzwerk aus Experten und Innovatoren teilt sein Wissen und seine Expertise in Büchern, Artikeln und auf unserer Onlinelernplattform. Die Onlinelernplattform von O'Reilly bietet Ihnen On-Demand-Zugriff auf Live-Schulungen, detaillierte Lernpfade, interaktive Codierumgebungen sowie eine umfangreiche Sammlung an Texten und Videos von O'Reilly und über 200 anderen Verlagen. Für weitere Informationen besuchen Sie bitte *https://oreilly.com*.

Danksagungen

An der Entstehung dieses Buchs waren viele Personen beteiligt. Ich möchte meiner Lektorin Angela Rufino dafür danken, dass sie mich auf Kurs gehalten hat, und für ihre Geduld angesichts meiner versäumten Termine. Danke auch an meine technischen Gutachter Brian Sletten, Elisabeth Robson, Ethan Flanagan, Maximiliano Firtman, Sarah Wachs und Schalk Neethling. Ihre Kommentare und Vorschläge haben deutlich zur Qualität dieses Buchs beigetragen.

Das Produktionsteam von O'Reilly hat seine gewohnt gute Arbeit geleistet: Kristen Brown leitete den Produktionsprozess, Deborah Baker fungierte als Produktionsredakteurin, Rebecca Demarest war für die Abbildungen zuständig und Judy McConville für den Index.

Zu den Lektorinnen, Gutachtern und Mitwirkenden früherer Ausgaben dieses Buchs gehören: Andrew Schulman, Angelo Sirigos, Aristoteles Pagaltzis, Brendan Eich, Christian Heilmann, Dan Shafer, Dave C. Mitchell, Deb Cameron, Douglas Crockford, Dr. Tankred Hirschmann, Dylan Schiemann, Frank Willison, Geoff Stearns, Herman Venter, Jay Hodges, Jeff Yates, Joseph Kesselman, Ken Cooper, Larry Sullivan, Lynn Rollins, Neil Berkman, Mike Loukides, Nick Thompson, Norris Boyd, Paula Ferguson, Peter-Paul Koch, Philippe Le Hegaret, Raffaele Cecco, Richard Yaker, Sanders Kleinfeld, Scott Furman, Scott Isaacs, Shon Katzenberger, Terry Allen, Todd Ditchendorf, Vidur Apparao, Waldemar Horwat und Zachary Kessin.

Das Schreiben dieser siebten Ausgabe hielt mich viele lange Nächte von meiner Familie fern, bei der ich mich in Liebe dafür bedanken möchte, dass sie meine Abwesenheiten ausgehalten hat.

– David Flanagan, im März 2020

KAPITEL 1

Einführung in JavaScript

JavaScript ist *die* Programmiersprache des Webs. Der weitaus größte Teil moderner Websites nutzt JavaScript, und alle modernen Webbrowser – auf Desktops, Tablets und Smartphones – besitzen JavaScript-Interpreter, was JavaScript zur am häufigsten eingesetzten Programmiersprache überhaupt macht. In den letzten zehn Jahren hat Node.js die JavaScript-Programmierung auch außerhalb von Webbrowsern ermöglicht, und der dramatische Erfolg von Node hat dazu geführt, dass JavaScript nun auch die unter Softwareentwicklern am häufigsten verwendete Programmiersprache ist. Egal ob Sie bei null anfangen oder JavaScript bereits professionell einsetzen, dieses Buch wird Ihnen dabei helfen, diese Sprache zu beherrschen.

Wenn Sie schon mit anderen Programmiersprachen vertraut sind, ist es vielleicht hilfreich, zu wissen, dass es sich bei JavaScript um eine dynamische, interpretierte Hochsprache handelt, die für objektorientierte wie auch funktionale Programmierstile sehr gut geeignet ist. Die Variablen von JavaScript sind untypisiert. Ihre Syntax basiert lose auf Java, aber die Sprachen sind ansonsten nicht miteinander verwandt. JavaScript leitet seine erstklassigen Funktionen von Scheme und seine prototypenbasierte Vererbung von der wenig bekannten Sprache Self ab. Sie müssen aber diese Sprachen weder kennen noch auch nur mit den Begriffen vertraut sein, um dieses Buch verwenden und JavaScript erlernen zu können.

Der Name »JavaScript« ist leicht irreführend. Abgesehen von einer oberflächlichen Ähnlichkeit der Syntax unterscheidet sich JavaScript komplett von Java. Darüber hinaus ist JavaScript längst über seine Wurzeln als Skriptsprache hinausgewachsen und zu einer robusten und effizienten Allzwecksprache geworden, die sich für ernsthaftes Softwareengineering und für Projekte mit riesigen Codebasen eignet.

Damit man eine Sprache nutzen kann, muss sie eine Plattform- bzw. Standardbibliothek besitzen, um zum Beispiel die Ein- und Ausgabe zu ermöglichen. Der Sprachkern von JavaScript legt eine minimale API für den Umgang mit Zahlen, Text, Arrays, Sets, Maps usw. fest, besitzt aber keine Ein- und Ausgabefunktionalität. Diese (und fortgeschrittenere Features der Bereiche Netzwerk, Speicher und Grafik) liegen ganz in der Verantwortung der *Hostumgebung*, in die JavaScript eingebettet ist.

JavaScript: Namen, Versionen und Modi

JavaScript wurde in den frühen Tagen des Webs bei Netscape entwickelt, und die Bezeichnung *JavaScript* selbst ist ein von Sun Microsystems (jetzt Oracle) lizenziertes Warenzeichen, das zur Beschreibung der Implementierung der Sprache durch Netscape (jetzt Mozilla) verwendet wird. Netscape hatte die Sprache zur Standardisierung bei der ECMA eingereicht – der *European Computer Manufacturers Association* –, und aufgrund von Markenrechtsproblemen erhielt die standardisierte Version der Sprache den sperrigen Namen »ECMAScript«. In der Praxis spricht aber eigentlich jeder von JavaScript. In diesem Buch werden der Name *ECMAScript* und die Abkürzung *ES* benutzt, um speziell auf den Sprachstandard und auf bestimmte Versionen dieses Standards hinzuweisen.

In den 2010er-Jahren wurde von allen Webbrowsern überwiegend Version 5 des ECMAScript-Standards unterstützt. Deshalb wird in diesem Buch *ES5* als die Kompatibilitäts-Baseline betrachtet, sodass frühere Versionen der Sprache nicht behandelt werden. ES6 wurde 2015 veröffentlicht und fügte wichtige neue Funktionen – darunter die Klassen- und Modulsyntax – hinzu, die JavaScript von einer Skriptsprache in eine ernst zu nehmende, allgemein einsetzbare Sprache verwandelten, die sich auch für die Softwareentwicklung in großem Maßstab eignet. Seit ES6 wird die ECMAScript-Spezifikation jährlich überarbeitet, und die Versionen der Sprache – ES2016, ES2017, ES2018, ES2019 und ES2020 usw. – werden jetzt nach dem Jahr der Veröffentlichung benannt.

Während der Entwicklung von JavaScript versuchten die Sprachdesigner, Fehler in den frühen Versionen (vor ES5) zu korrigieren. Um die Abwärtskompatibilität aufrechtzuerhalten, ist es nicht möglich, ältere Sprachmerkmale (sogenannte *Legacy-Features*) zu entfernen, egal wie mangelhaft sie sind. Aber in ES5 und späteren Versionen können Programme in den *Strict Mode* – den *strikten* oder *`strict`-Modus* – von JavaScript wechseln, in dem eine Reihe von frühen Sprachfehlern korrigiert wurde. Um diesen Modus zu aktivieren, benutzt man die »use strict«-Anweisung, die in 5.6.3 besprochen wird. Dieser Abschnitt fasst auch die Unterschiede zwischen Legacy-JavaScript und JavaScript im `strict`-Modus zusammen. In ES6 und später aktiviert die Verwendung neuer Sprachfunktionen oft implizit den `strict`-Modus. Wenn Sie beispielsweise das ES6-Schlüsselwort `class` verwenden oder ein ES6-Modul erstellen, ist der gesamte Code innerhalb der Klasse oder des Moduls automatisch »strict«, und die alten, mangelhaften Funktionen sind in diesen Kontexten nicht verfügbar. In diesem Buch behandle ich auch die Legacy-Features von JavaScript, weise aber darauf hin, dass diese im `strict`-Modus nicht verfügbar sind.

Die ursprüngliche Hostumgebung für JavaScript war ein Webbrowser, und das ist immer noch die am häufigsten verwendete Ausführungsumgebung für JavaScript-Code. In einer Webbrowserumgebung kann JavaScript-Code Input durch Maus und Tastatur des Benutzers sowie durch HTTP-Anfragen erhalten, dem Nutzer aber auch per HTML und CSS Ausgaben anzeigen.

Seit 2010 ist eine weitere Hostumgebung für JavaScript-Code verfügbar. Anstatt JavaScript auf die Arbeit mit den APIs zu beschränken, die von einem Webbrowser bereitgestellt werden, gibt Node JavaScript Zugriff auf das gesamte Betriebssystem, sodass JavaScript-Programme Dateien lesen und schreiben, Daten über das Netzwerk senden und empfangen sowie HTTP-Anfragen stellen und bedienen können. Node ist eine beliebte Wahl zur Implementierung von Webservern und außerdem ein bequemes Werkzeug zum Schreiben einfacher Hilfsskripte als Alternative zu Shell-Skripten.

Der größte Teil dieses Buchs konzentriert sich auf die Programmiersprache JavaScript selbst, Kapitel 11 dokumentiert die JavaScript-Standardbibliothek, Kapitel 15 und 16 stellen Webbrowser sowie Node als Hostumgebungen vor.

In diesem Buch werden zunächst die Grundlagen beschrieben, um dann darauf aufbauend zu fortgeschritteneren Abstraktionen auf höherer Ebene überzugehen. Die Kapitel sind dazu gedacht, mehr oder weniger der Reihe nach gelesen zu werden. Aber das Erlernen einer neuen Programmiersprache ist kein linearer Prozess und ebenso wenig das Beschreiben einer Sprache: Jedes Sprachfeature hängt mit anderen Merkmalen zusammen, und dieses Buch ist mit Querverweisen gespickt – manchmal auf zurückliegende Kapitel, manchmal auf Material, das Sie noch nicht gelesen haben. Dieses Einführungskapitel gibt Ihnen einen ersten Überblick über die Sprache und führt in die wichtigsten Merkmale ein, die das Verständnis einer detaillierteren Behandlung in den folgenden Kapiteln erleichtern werden. Falls Sie bereits JavaScript programmieren, können Sie dieses Kapitel wahrscheinlich überspringen. (Obwohl Sie sicher gern Beispiel 1-1 am Ende des Kapitels lesen möchten, bevor Sie zu einem anderen Kapitel übergehen.)

1.1 JavaScript erkunden

Wenn man eine neue Programmiersprache lernt, ist es wichtig, die Beispiele auszuprobieren, sie dann selbst zu modifizieren und erneut auszuprobieren, um das erworbene Verständnis der Sprache zu testen. Dazu benötigen Sie einen JavaScript-Interpreter.

Am einfachsten lassen sich ein paar Zeilen JavaScript ausprobieren, indem man – mit F12 bzw. Strg+Umschalt+I in Windows- bzw. Befehlstaste+Optionstaste+I in Mac-Umgebungen – die Entwicklertools in einem Webbrowser öffnet und auf die Registerkarte *Konsole* wechselt. Sie können dann in der Eingabeaufforderung Code eingeben und während der Eingabe die Ergebnisse sehen. Die Entwicklerwerkzeuge erscheinen oft als Fenster am unteren oder rechten Rand des Browserfensters, aber Sie können sie normalerweise als separate Fenster abkoppeln (wie in Abbildung 1-1 gezeigt), was oft recht bequem ist.

Abbildung 1-1: Die JavaScript-Konsole in den Entwicklertools von Firefox

Um JavaScript-Code auszuprobieren, können Sie auch Node von *https://nodejs.org* herunterladen und installieren. Sobald Node auf Ihrem System eingerichtet ist, öffnen Sie einfach ein Terminalfenster und geben **node** ein, um eine interaktive JavaScript-Sitzung wie diese zu starten:

```
$ node
Welcome to Node.js v14.15.0.
Type ".help" for more information.
> .help
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the repl
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a file

Press ^C to abort current expression, ^D to exit the repl
> let x = 2, y = 3;
undefined
> x + y
5
> (x === 2) && (y === 3)
true
> (x > 3) || (y < 3)
false
```

1.2 Hello World

Wenn Sie bereit sind, mit längeren Codeblöcken zu experimentieren, eignen sich diese zeilenbasierten interaktiven Umgebungen normalerweise nicht mehr besonders, und Sie werden es wahrscheinlich vorziehen, Ihren Code in einem Texteditor zu schreiben. Von dort aus können Sie den Code in die JavaScript-Konsole oder in eine Node-Sitzung kopieren. Oder Sie speichern Ihren Code in einer Datei (die herkömmliche Dateinamenserweiterung für JavaScript-Code lautet *.js*) und führen dann diese Datei mit Node aus:

```
$ node snippet.js
```

Wenn Sie Node in einer solchen nicht interaktiven Weise verwenden, wird der Wert des ausgeführten Codes nicht automatisch ausgegeben, sodass Sie das selbst vornehmen müssen. Verwenden Sie die Funktion `console.log()`, um Text und andere JavaScript-Werte in Ihrem Terminalfenster oder in der Konsole der Entwicklerwerkzeuge eines Browsers anzuzeigen – zum Beispiel indem Sie eine Datei *hello.js* erstellen, die folgende Codezeile enthält:

```
console.log("Hello World!");
```

Wenn Sie diese Datei mit `node hello.js` ausführen, wird die Meldung »Hello World!« ausgegeben.

Möchten Sie die gleiche Nachricht in der JavaScript-Konsole eines Webbrowsers ausgeben, erstellen Sie eine neue Datei mit dem Namen *hello.html* und fügen diesen Text darin ein:

```
<script src="hello.js"></script>
```

Dann laden Sie *hello.html* in Ihren Webbrowser mit einer *file://*-URL wie dieser:

```
file:///Users/username/javascript/hello.html
```

Öffnen Sie nun das Fenster der Entwicklertools, um die Begrüßung in der Konsole zu sehen.

1.3 Ein Rundgang durch JavaScript

Dieser Abschnitt enthält eine kurze Einführung in die JavaScript-Sprache anhand von Codebeispielen. Nach diesem einführenden Kapitel steigen wir richtig tief in JavaScript ein und beginnen damit ganz weit unten – Kapitel 2 erklärt Dinge wie JavaScript-Kommentare, Semikola und den Unicode-Zeichensatz. In Kapitel 3 wird es schon ein bisschen spannender: Es werden JavaScript-Variablen erklärt sowie die Werte, die man ihnen zuweisen kann.

Hier ist etwas Beispielcode, der die Höhepunkte dieser beiden Kapitel veranschaulicht:

```
// Alles nach einem doppelten Schrägstrich ist ein Kommentar in normaler Sprache.
// Lesen Sie Kommentare sorgfältig: Sie erläutern den JavaScript-Code.

// Eine Variable ist ein symbolischer Name für einen Wert.
// Variablen werden mit dem Schlüsselwort let deklariert:
let x;                      // Eine Variable namens x deklarieren.

// Mit dem =-Zeichen können Variablen Werte zugewiesen werden.
x = 0;                      // Jetzt hat die Variable x den Wert 0.
x                           // => 0: Eine Variable wird zu ihrem Wert ausgewertet.

// JavaScript unterstützt eine ganze Reihe von Wertetypen.
x = 1;                      // Zahlen.
x = 0.01;                   // Zahlen können ganze oder reelle Zahlen sein.
x = "hello world";          // Textsequenzen in Anführungszeichen.
x = 'JavaScript';           // Auch einfache Anführungszeichen sind möglich.
x = true;                   // Ein boolescher Wert.
x = false;                  // Der andere boolesche Wert.
x = null;                   // null ist ein spezieller Wert, der für "kein Wert"
                            // steht.
x = undefined;              // undefined ist ein weiterer spezieller Wert wie null.
```

Zwei weitere sehr wichtige Datentypen, mit denen man in JavaScript-Programmen arbeiten kann, sind Objekte und Arrays. Diese Typen behandeln wir in den Kapiteln 6 und 7, aber sie sind so wichtig, dass sie Ihnen in diesem Buch bereits viele Male zuvor begegnen werden:

```
// Der wichtigste Datentyp in JavaScript ist das Objekt.
// Ein Objekt ist eine Sammlung von Name/Wert-Paaren
// (bzw. eine Map, in der Zeichenketten auf Werte abgebildet werden).
let book = {                // Objekte sind von geschweiften Klammern umschlossen.
    topic: "JavaScript",    // Die Eigenschaft "topic" hat den Wert "JavaScript".
    edition: 7              // Die Eigenschaft "edition" hat den Wert 7.
};                          // Die geschweifte Klammer markiert das Ende
                            // des Objekts.

// Zugriff auf die Eigenschaften eines Objekts mit . oder []:
book.topic                  // => "JavaScript"
book["edition"]             // => 7: Eine weitere Möglichkeit, auf
                            // Eigenschaftswerte zuzugreifen.
book.author = "Flanagan";   // Neue Eigenschaften durch Zuweisung erstellen.
book.contents = {};         // {} ist ein leeres Objekt ohne Eigenschaften.

// Bedingter Zugriff auf Eigenschaften mit ?. (ES2020):
book.contents?.ch01?.sect1 // => undefined: book.contents hat keine
                            // Eigenschaft ch01.

// JavaScript unterstützt auch Arrays (numerisch indexierte Listen) mit Werten:
let primes = [2, 3, 5, 7]; // Ein Array mit 4 Werten, begrenzt durch [ und ].
primes[0]                   // => 2: Das erste Element (Index 0) des Arrays.
primes.length               // => 4: Anzahl der Elemente im Array.
primes[primes.length-1]     // => 7: Das letzte Element des Arrays.
primes[4] = 9;              // Ein neues Element durch Zuweisen hinzufügen.
primes[4] = 11;             // Oder ein vorhandenes Element durch Zuweisen
                            // verändern.
```

```
let empty = [];            // [] ist ein leeres Array ohne Elemente.
empty.length               // => 0

// Arrays und Objekte können andere Arrays und Objekte enthalten:
let points = [             // Ein Array mit 2 Elementen.
    {x: 0, y: 0},          // Jedes Element ist ein Objekt.
    {x: 1, y: 1}
];
let data = {                 // Ein Objekt mit 2 Eigenschaften.
    trial1: [[1,2], [3,4]],  // Der Wert jeder Eigenschaft ist ein Array.
    trial2: [[2,3], [4,5]]   // Die Elemente der Arrays sind Arrays.
};
```

Kommentarsyntax in Codebeispielen

Sie haben vielleicht im vorhergehenden Code bemerkt, dass einige der Kommentare mit einem Pfeil (=>) beginnen. Diese zeigen den Wert, den der Code vor dem Kommentar erzeugt, und sind mein Versuch, eine interaktive JavaScript-Umgebung wie etwa eine Webbrowserkonsole in einem gedruckten Buch nachzubilden.

Diese `// =>`-Kommentare dienen ebenfalls als *Zusicherungen* in Softwaretests (auch als *Behauptungen* bzw. *Assertionen* bezeichnet), und ich habe ein Tool geschrieben, um den Code zu testen und dabei zu verifizieren, dass er den im Kommentar angegebenen Wert erzeugt – was, so hoffe ich, dazu beiträgt, die Fehlerzahl in diesem Buch zu verringern.

Es gibt zwei verwandte Arten von Kommentaren/Zusicherungen. Ein Kommentar der Form `// a == 42` bedeutet, dass die Variable `a` den Wert 42 haben wird, nachdem der Code, der vor dem Kommentar steht, ausgeführt wurde. Ein Kommentar der Form `// !` zeigt an, dass der Code in der Zeile vor dem Kommentar eine Ausnahme auslöst (und der Rest des Kommentars nach dem Ausrufezeichen erklärt normalerweise, welche Art von Ausnahme auftritt).

Solche Kommentare werden Ihnen im gesamten Buch begegnen.

Die hier dargestellte Syntax zur Auflistung von Array-Elementen innerhalb eckiger Klammern oder zur Abbildung von Bezeichnungen von Objekteigenschaften auf Eigenschaftswerte innerhalb geschweifter Klammern wird als *Initialisierungsausdruck* bezeichnet – und ist nur eines der Themen von Kapitel 4. Ein *Ausdruck* ist eine Phrase in JavaScript, die zu einem Wert *ausgewertet* werden kann. Beispielsweise ist die Verwendung von `.` und `[]`, um auf den Wert einer Objekteigenschaft oder eines Array-Elements zu verweisen, ein Ausdruck.

Am häufigsten bildet man Ausdrücke in JavaScript, indem man *Operatoren* verwendet:

```
// Operatoren agieren auf Werten (den Operanden), um einen neuen Wert zu erzeugen.
// Arithmetische Operatoren gehören zu den einfachsten Operatoren:
3 + 2                      // => 5: Addition
```

```
3 - 2                       // => 1: Subtraktion
3 * 2                       // => 6: Multiplikation
3 / 2                       // => 1.5: Division
points[1].x - points[0].x   // => 1: Auch komplexere Operanden sind möglich.
"3" + "2"                   // => "32": + addiert Zahlen, verkettet Strings.

// JavaScript bietet einige arithmetische Kurzoperatoren:
let count = 0;              // Variable definieren.
count++;                    // Variable inkrementieren.
count--;                    // Variable dekrementieren.
count += 2;                 // 2 addieren, entspricht count = count + 2;.
count *= 3;                 // Mit 3 multiplizieren, entspricht count = count * 3;.
count                       // => 6: Variablennamen sind ebenfalls Ausdrücke.

// Gleichheits- und Relationsoperatoren prüfen, ob zwei Werte gleich, ungleich,
// kleiner als, größer als usw. sind. Sie werden zu true oder false ausgewertet.
let x = 2, y = 3;           // Diese =-Zeichen sind Zuweisungen, keine
                            // Gleichheitstests.
x === y                     // => false: Gleichheit.
x !== y                     // => true: Ungleichheit.
x < y                       // => true: kleiner als.
x <= y                      // => true: kleiner als oder gleich.
x > y                       // => false: größer als.
x >= y                      // => false: größer als oder gleich.
"two" === "three"           // => false: Die beiden Strings sind verschieden.
"two" > "three"             // => true: "tw" ist alphabetisch größer als "th".
false === (x > y)           // => true: false ist gleich false.

// Logische Operatoren kombinieren oder invertieren boolesche Werte:
(x === 2) && (y === 3)      // => true: Beide Vergleiche sind wahr. && ist AND (UND).
(x > 3) || (y < 3)          // => false: Keiner der Vergleiche ist wahr.
                            // || ist OR (ODER).
!(x === y)                  // => true: ! invertiert einen booleschen Wert.
```

Betrachtet man JavaScript-*Ausdrücke* als Phrasen, wären JavaScript-*Anweisungen* ganze Sätze. Anweisungen sind das Thema von Kapitel 5. Ein Ausdruck ist, vereinfacht gesagt, etwas, das einen Wert liefert, aber nichts tut. Anweisungen hingegen liefern keinen Wert (zumindest keinen, der uns interessiert), ändern aber den Zustand. Oben haben Sie Variablendeklarationen und Zuweisungsanweisungen gesehen. Die andere große Kategorie von Anweisungen sind *Kontrollstrukturen* wie z.B. Bedingungen und Schleifen. Beispiele finden Sie weiter unten, nachdem wir die Funktionen behandelt haben.

Eine Funktion ist ein benannter und parametrisierter Block mit JavaScript-Code, den Sie einmal definieren und dann immer wieder aufrufen können. Funktionen werden formal erst in Kapitel 8 behandelt, aber wie Objekte und Arrays werden sie Ihnen auch vorher schon immer wieder mal begegnen. Hier ein paar einfache Beispiele:

```
// Funktionen sind parametrisierte Blöcke mit JavaScript-Code, die wir aufrufen
// können.
function plus1(x) {         // Funktion mit dem Namen "plus1" und dem
                            // Parameter "x" definieren.
```

```
    return x + 1;          // Wert zurückgeben, der um eins größer ist
                           // als der übergebene Wert.
}                          // Funktionen sind in geschweifte Klammern
                           // eingeschlossen.

plus1(y)                   // => 4: y ist (immer noch) 3, also gibt
                           // dieser Aufruf 3+1 zurück.

let square = function(x) { // Funktionen sind Werte und können Variablen
                           // zugewiesen werden.
    return x * x;          // Funktionswert berechnen.
};                         // Semikola kennzeichnen das Ende der Zuweisung.

square(plus1(y))           // => 16: Zwei Funktionen in einem Ausdruck aufrufen.
```

In ES6 und späteren Versionen gibt es eine Kurzschriftsyntax für die Definition von Funktionen. Diese prägnante Syntax verwendet =>, um die Argumentliste vom Funktionskörper zu trennen, daher werden auf diese Weise definierte Funktionen als *Pfeilfunktionen* bezeichnet. Pfeilfunktionen werden am häufigsten eingesetzt, wenn eine unbenannte Funktion als Argument an eine andere Funktion übergeben werden soll. Verwendet man Pfeilfunktionen, sieht der vorhergehende Code so aus:

```
const plus1 = x => x + 1;   // Die Eingabe x wird auf die Ausgabe x + 1 abgebildet.
const square = x => x * x;  // Die Eingabe x wird auf die Ausgabe x * x abgebildet.
plus1(y)                    // => 4: Funktionen werden wie sonst auch aufgerufen.
square(plus1(y))            // => 16
```

Wenn wir Funktionen mit Objekten verwenden, erhalten wir *Methoden*:

```
// Werden Funktionen den Eigenschaften eines Objekts zugewiesen, nennen wir sie
// "Methoden". Alle JavaScript-Objekte (einschließlich Arrays) besitzen Methoden:
let a = [];                // Ein leeres Array anlegen.
a.push(1,2,3);             // Die Methode push() fügt einem Array Elemente hinzu.
a.reverse();               // Eine weitere Methode: die Reihenfolge der Elemente
                           // umkehren.

// Wir können auch eigene Methoden definieren. Das Schlüsselwort "this" bezieht
// sich auf das Objekt, für das die Methode definiert ist: hier das Array points
// aus einem vorhergehenden Beispiel.
points.dist = function() { // Methode zum Berechnen des Abstands zwischen
                           // Punkten definieren.
    let p1 = this[0];      // Erstes Element des Arrays, auf dem die Methode
                           // aufgerufen wird.
    let p2 = this[1];      // Zweites Element des Arrays "this".
    let a = p2.x-p1.x;     // Differenz der x-Koordinaten.
    let b = p2.y-p1.y;     // Differenz der y-Koordinaten.
    return Math.sqrt(a*a + // Der Satz des Pythagoras.
                     b*b); // Math.sqrt() zieht die Wurzel.
};
points.dist()              // => Math.sqrt(2): Abstand zwischen unseren
                           // zwei Punkten.
```

Hier erhalten Sie nun wie versprochen einige Funktionen, in denen als Anweisungen häufig genutzte JavaScript-Kontrollstrukturen vorkommen:

```
// Zu den JavaScript-Anweisungen gehören auch bedingte Anweisungen und Schleifen,
// die eine aus C, C++, Java und anderen Sprachen bekannte Syntax nutzen.
function abs(x) {          // Eine Funktion zur Berechnung des Absolutbetrags.
    if (x >= 0) {          // Die if-Anweisung ...
        return x;          // ... führt diesen Code aus, wenn der Vergleich
                           // wahr ist.
    }                      // Das Ende der if-Anweisung.
    else {                 // Die optionale else-Anweisung führt den in ihr
                           // enthaltenen Code
        return -x;         // aus, wenn der Vergleich falsch ist.
    }                      // Geschweifte Klammern sind optional, wenn es nur
                           // eine Anweisung gibt.
}                          // Beachten Sie die return-Anweisungen innerhalb
                           // von if/else.
abs(-10) === abs(10)       // => true

function sum(array) {      // Berechnet die Summe der Elemente eines Arrays.
    let sum = 0;           // Beginnt mit einer Anfangssumme von 0.
    for(let x of array) {  // Schleife über das Array, wobei jedes Element x
                           // zugeordnet wird.
        sum += x;          // Aktuellen Elementwert der Summe hinzufügen.
    }                      // Das Ende der for-Schleife.
    return sum;            // Die Summe zurückgeben.
}
sum(primes)                // => 28: Summe der ersten 5 Primzahlen 2+3+5+7+11.

function factorial(n) {    // Eine Funktion zur Berechnung von Fakultäten.
    let product = 1;       // Beginne mit dem Produkt von 1.
    while(n > 1) {         // Wiederhole Anweisungen in {}, solange der Ausdruck
                           // in () wahr ist.
        product *= n;      // Kurzform für product = product * n.
        n--;               // Kurzform für n = n - 1.
    }                      // Das Ende der while-Schleife.
    return product;        // Rückgabe des Produkts.
}
factorial(4)               // => 24: 1*4*3*2

function factorial2(n) {   // Eine alternative Version, die eine andere
                           // Schleife verwendet.
    let i, product = 1;    // Beginne mit 1.
    for(i=2; i <= n; i++)  // i automatisch von 2 bis n inkrementieren.
        product *= i;      // Wird jedes Mal ausgeführt. {} wird für einzeilige
                           // Schleifen nicht benötigt.
    return product;        // Rückgabe der Fakultät.
}
factorial2(5)              // => 120: 1*2*3*4*5
```

JavaScript unterstützt einen objektorientierten Programmierstil, unterscheidet sich aber deutlich von »klassischen« objektorientierten Programmiersprachen. Kapitel 9 behandelt die objektorientierte Programmierung in JavaScript im Detail anhand vieler Beispiele. Es folgt ein sehr einfaches Beispiel, das zeigt, wie Sie eine JavaScript-Klasse erstellen, die einen zweidimensionalen Punkt repräsentiert. Objekte,

die Instanzen dieser Klasse sind, besitzen genau eine Methode namens `distance()`, die den Abstand des Punkts vom Ursprung berechnet:

```
class Point {               // Gemäß Konvention werden Klassennamen großgeschrieben.
    constructor(x, y) {     // Konstruktorfunktion zur Initialisierung neuer
                            // Instanzen.
        this.x = x;         // Das Schlüsselwort "this" bezeichnet das neue Objekt,
                            // das initialisiert wird.
        this.y = y;         // Funktionsargumente als Objekteigenschaften speichern.
    }                       // In Konstruktorfunktionen ist keine Rückgabe
                            // erforderlich.

    distance() {            // Methode zur Berechnung der Entfernung zwischen
                            // Ursprung und Punkt.
        return Math.sqrt(   // Liefert die Quadratwurzel aus x² + y².
            this.x * this.x +  // "this" bezieht sich auf das Punktobjekt, auf dem
            this.y * this.y    // die Methode distance() aufgerufen wird.
        );
    }
}

// Verwenden Sie die Konstruktorfunktion Point() mit "new", um Punktobjekte
// zu erstellen.
let p = new Point(1, 1);   // Der geometrische Punkt (1,1).

// Verwenden Sie jetzt eine Methode des Punktobjekts p.
p.distance()               // => Math.SQRT2
```

So weit die Einführung in die grundlegende Syntax und die Fähigkeiten von JavaScript. Es folgen in sich abgeschlossene Kapitel, in denen weitere Sprachmerkmale behandelt werden:

Kapitel 10, Module
: Zeigt, wie JavaScript-Code in einer Datei oder einem Skript Funktionen und Klassen verwenden kann, die in anderen Dateien oder Skripten definiert sind.

Kapitel 11, Die JavaScript-Standardbibliothek
: Deckt die eingebauten Funktionen und Klassen ab, die allen JavaScript-Programmen zur Verfügung stehen. Dazu gehören wichtige Datenstrukturen wie Maps und Sets, eine Klasse für reguläre Ausdrücke für die Suche nach Textmustern, Funktionen zur Serialisierung von JavaScript-Datenstrukturen und vieles mehr.

Kapitel 12, Iteratoren und Generatoren
: Erklärt, wie die `for/of`-Schleife funktioniert und wie Sie Ihre eigenen Klassen mit `for/of` iterierbar machen können. Daneben werden Generatorfunktionen und die `yield`-Anweisung behandelt.

Kapitel 13, Asynchrones JavaScript
: Untersucht eingehend die asynchrone Programmierung in JavaScript, wobei Callbacks und Events, Promise-basierte APIs und die Schlüsselwörter `async` und `await` behandelt werden. Obwohl JavaScript im Kern nicht asynchron ist,

sind asynchrone APIs Standard sowohl in Webbrowsern als auch in Node, und in diesem Kapitel werden die Techniken für die Arbeit mit diesen APIs erläutert.

Kapitel 14, Metaprogrammierung
: Stellt eine Reihe fortgeschrittener Funktionen von JavaScript vor, die für Programmierer interessant sind, die Codebibliotheken schreiben, um diese anderen JavaScript-Programmierern zur Verfügung zu stellen.

Kapitel 15, JavaScript im Webbrowser
: Stellt die Hostumgebung des Webbrowsers vor, erklärt, wie Webbrowser JavaScript-Code ausführen, und behandelt die wichtigsten der vielen von Webbrowsern definierten APIs. Dies ist bei Weitem das längste Kapitel des Buchs.

Kapitel 16, Serverseitiges JavaScript mit Node
: Stellt die Hostumgebung von Node vor und behandelt das grundlegende Programmiermodell sowie die für das Verständnis wichtigsten Datenstrukturen und APIs.

Kapitel 17, JavaScript-Werkzeuge und -Erweiterungen
: Deckt weitverbreitete Werkzeuge und Spracherweiterungen ab, die man kennen sollte und die Sie zu einem produktiveren Programmierer machen können.

1.4 Beispiel: Häufigkeitshistogramme

Dieses Kapitel schließt mit einem kurzen, aber nicht trivialen JavaScript-Programm. Beispiel 1-1 ist ein Node-Programm, das Text von der Standardeingabe liest, aus diesem Text ein Häufigkeitshistogramm der vorkommenden Zeichen berechnet und dieses Histogramm ausgibt. Sie können das Programm wie folgt aufrufen, um die Zeichenhäufigkeit seines eigenen Quelltexts zu analysieren:

```
$ node charfreq.js < charfreq.js
T: ########## 11.22%
E: ########## 10.15%
R: ####### 6.68%
S: ###### 6.44%
A: ###### 6.16%
N: ###### 5.81%
O: ##### 5.45%
I: ##### 4.54%
H: #### 4.07%
C: ### 3.36%
L: ### 3.20%
U: ### 3.08%
/: ### 2.88%
```

Dieses Beispiel verwendet eine Reihe fortgeschrittener JavaScript-Funktionen und soll demonstrieren, wie JavaScript-Programme in der realen Welt aussehen können. Falls Sie den Code noch nicht vollständig verstehen, macht das nichts – ich verspreche Ihnen, dass in den folgenden Kapiteln alles erklärt wird.