



Marco Emrich und Christin Marit

JavaScript Grundlagen, Sprachkern, Syntax

Ein Webmasters Press Lernbuch

Version 1.10.10 vom 10.02.2022

Autorisiertes Curriculum für das Webmasters Europe Ausbildungs- und Zertifizierungsprogramm

Inhaltsverzeichnis

Vorwort	15
1 Was Sie schon immer über JavaScript wissen wollten...	16
1.1 JavaScript?	16
1.2 Was diese Class ist	18
1.3 Was diese Class nicht(!) ist	18
1.4 Wo sind die Animationen und die GUI?	19
1.5 Projekt »NerdWorld«	19
2 Los geht's: die »Chrome Webkonsole«	21
2.1 Wo ist denn die IDE?	21
2.2 Hilfreiche Shortcuts für die Webkonsole	22
2.3 Hello World mit console.log	22
2.4 Alarm! Alarm! — Die Konsole brennt	23
2.5 Mehrere Anweisungen	25
2.6 Programmierrichtlinien	27
2.7 Seien Sie »strict«	28
2.8 Kommentare oder: Geben Sie doch Ihren Senf dazu	28
2.9 Testen Sie Ihr Wissen!	29
3 Von Operatoren, Datentypen und anderen Prioritäten	30
3.1 Rechnen mit JavaScript	30
3.1.1 Ausdrücke in der Chrome Webkonsole	30
3.1.2 Notation	31
3.2 JavaScript als Taschenrechner: Arithmetische Operatoren	31
3.3 Zeichen? Strings!	32
3.4 Die Frage nach der Länge	33
3.5 Literale: Sagen was sie meinen	33
3.6 Die Datentypen number & string	34
3.7 Prioritäten & Klammerung	35
3.7.1 Reihenfolge von Operatoren	35
3.7.2 Klammerung	36
3.8 Testen Sie Ihr Wissen	37
3.9 Übung	38
4 Variablen oder die Realität in Schubladen	39
4.1 Ganz schön variabel	39
4.2 Variablen anlegen	39
4.3 Variablen Werte zuweisen	40
4.4 Jetzt aber prompt bitte!	42
4.5 Grundsolide: Konstanten	43
4.6 Kampf der Giganten: Variable vs. Konstante	44
4.7 Zusammengesetzte Zuweisungsoperatoren	45
4.8 Ausdrücklich mehr Ausdruck?	47
4.9 Testen Sie Ihr Wissen!	48

5	Sehr bezeichnend, diese Bezeichner!	49
5.1	Aussagekräftige Bezeichner	49
5.2	Keine Sonderzeichen	50
5.3	Keine eigenen Abkürzungen	50
5.4	Variablen in camelCase	51
5.5	Konstanten in SCREAMING_SNAKE_CASE	52
5.6	JS-Bezeichner sind case-sensitive	52
5.7	Nomen für Variablenbezeichner	52
5.8	Variablenbezeichner im Singular	53
5.9	Testen Sie Ihr Wissen!	53
5.10	Übungen	53
6	Setzen Sie ein Zeichen — mit Strings!	55
6.1	Die Verkettung glücklicher Strings	55
6.2	Template Strings — mehr als nur ein Platzhalter	56
6.3	Zeit des Umbruchs	57
6.4	Backslash oder »Behind the Mask«	58
6.5	Testen Sie Ihr Wissen!	59
6.6	Übungen	59
7	Typkonvertierung und Nummern, die keine sind	61
7.1	Implizite Typkonvertierung	61
7.2	Explizite Typkonvertierung	61
7.3	Implizite Typkonvertierung beim +-Zeichen	63
7.4	Das NaN-Paradoxon	63
7.5	Testen Sie Ihr Wissen!	64
7.6	Übungen	65
8	Math, denn das Ganze ist mehr als die Summe seiner Teile	67
8.1	Das Math-Objekt	67
8.2	Eine runde Sache: floor, round & ceil	67
8.3	Frisch gefixt mit toFixed	68
8.4	Wie es der Zufall will mit random	68
8.5	Referenz	69
8.6	Testen Sie Ihr Wissen!	69
8.7	Übungen	70
9	Von der Wahrheit: Relationale Operatoren & der Datentyp Boolean	71
9.1	Immer diese Entscheidungen	71
9.2	Relationale Operatoren und die Bedeutung der Wahrheit	72
9.3	Der Datentyp Boolean	72
9.4	Number oder nicht Number... das ist hier die Frage	76
9.5	String größer string? Strings, Number und wie sie miteinander können	77
9.6	Testen Sie Ihr Wissen!	79
10	Für alle Fälle ... if & else	80
10.1	Wenn ... dann	80
10.2	Und wenn nicht, was dann?	83
10.3	Besonderheiten bei nur einer Anweisung	84
10.4	Wenn drei zusammenkommen: Der ternäre Operator	86
10.5	Testen Sie Ihr Wissen!	88

11	Exkurs - Denken wie ein Programmierer. Wie löse ich Probleme?	89
12	Logische Operatoren oder der Anfang aller Weisheit	94
12.1	Mehrere Bedingungen mit logischen Operatoren verbinden	94
12.2	Funktionsweise des Oder-Operators	95
12.3	Der Und-Operator	96
12.4	Funktionsweise des Und-Operators	96
12.5	Der Nicht-Operator	97
12.6	Priorität von logischen Operatoren	98
12.7	Testen Sie Ihr Wissen!	98
12.8	Übungen	101
13	Buntes String-Allerlei	102
13.1	Vom Suchen und Finden mit indexOf	102
13.2	Abnehmen mit substr	103
13.3	trim Dich fit	104
13.4	Wählerisch sein mit charAt	105
13.5	Schrumpfen mit toLowerCase	105
13.6	Verflixt und ausgetauscht mit replace	106
13.7	Fokussieren der inneren Werte mit includes	108
13.8	Wehret den Anfängen mit startsWith	109
13.9	Referenz	110
13.10	Testen Sie Ihr Wissen!	111
13.11	Übungen	111
14	Regex — reguläre Ausdrücke	114
14.1	Was sind reguläre Ausdrücke?	114
14.2	Metazeichen	116
14.2.1	definierte Zeichenklassen	119
14.2.2	Modifikatoren	119
14.2.3	Lookahead, Lookbehind, Lookaround — Look... bitte was?	120
14.2.4	greed & lazy Match	121
14.3	Reguläre Ausdrücke und JavaScript	121
14.4	Testen Sie Ihr Wissen!	124
14.5	Übungen	124
15	Und jetzt mal bitte mit Funktion!	126
15.1	A function is a function is a function	126
15.2	Let's function	128
15.3	Hallo \${username},...	128
15.4	Signaturen: Gut dokumentiert ist halb verstanden	131
15.5	Testen Sie Ihr Wissen!	132
15.6	Übungen	133
16	Funktionen geben uns so viel ... zurück	136
16.1	Wenn Kunden in die Röhre schauen...	136
16.2	Ab in die Kiste...	137
16.3	... und wieder zurück	138
16.4	Wenn's mehr Ausdruck braucht...	140
16.5	Wächter für Ihre Funktionen	141
16.6	Testen Sie Ihr Wissen!	142
16.7	Übungen	142

17	Einer für alle und alle für einen: Arrays	143
17.1	Erzeugen von Arrays	143
17.2	Die Frage nach der Länge, Teil 2	144
17.3	Zugreifen mit dem Indexoperator	144
17.4	Drücken und Schubsen mit push	144
17.5	Dezimieren mit pop	145
17.6	Gezielt manipulieren mit splice	145
17.7	Ordnung ist das halbe Leben mit sort	147
17.8	Verbinden mit join	147
17.9	Vom Suchen und Finden mit indexOf, Teil 2	148
17.10	Wenn Strings sich trennen: split	148
17.11	Referenz	151
17.12	Testen Sie Ihr Wissen!	151
17.13	Übungen	152
18	Exkurs - Denken wie ein Programmierer. Wie schreibe ich Funktionen richtig?	157
19	Auf zu Höherem mit map, reduce, ...	162
19.1	Funktionen als Parameter	162
19.2	Funktionen, die auf Funktionen stehen	167
19.3	Arrays im Wandel mit map	167
19.4	Mehr über Higher-Order-Funktionen und Callbacks	169
19.5	Nur sehen, was man sehen will, mit filter	170
19.6	Jeder ist mal dran mit forEach	171
19.7	Aufs Wesentliche reduzieren mit reduce	175
19.8	Manche mögens heiß mit some	178
19.9	Referenz	179
19.10	Testen Sie Ihr Wissen!	181
19.11	Übungen	182
20	›Strict‹ behandelte Variablen und ihre Scopes	186
20.1	Die ganze verheerende Wahrheit über Scopes	186
20.1.1	let	186
20.1.2	const	187
20.1.3	Parameter	187
20.1.4	Globale Variablen	188
20.2	Da bleibe ich strict!	188
20.3	In Zukunft konstant	189
20.4	Testen Sie Ihr Wissen!	192
21	Noch mal von vorne mit Rekursion	194
21.1	Wiederholungen müssen nicht anstrengend sein	194
21.2	Sich selbst ... aufrufen	195
21.3	Testen Sie Ihr Wissen!	196
21.4	Übungen	196
22	Tapeten-Design mit Rekursion — Teil 2	197
22.1	Im Zeichen der Linie	197
22.2	Rechteck: Wenn Linien sich zusammentun	199
22.3	Testen Sie Ihr Wissen!	201
22.4	Übungen	201

23	Verloren in Raum und Zeit? Times & Range	204
23.1	Zum x-ten Mal mit times	204
23.2	times-mal optimiert	206
23.3	Von Anfang bis Ende mit range	208
23.4	Was ist nur los mit den Schleifen?	209
23.5	Übungen	210
24	Das merkwürdige Verhalten reifer Funktionsdefinitionen zur Aufrufzeit	213
24.1	Funktionsparameter — mal ist's zu viel, mal ist's zu wenig	213
24.2	Den Standardfall erschlagen mit Default-Parametern	214
24.3	Nehmen wie es kommt: Rest-Parameter	215
24.4	Higher-Order-Funktionen	216
24.5	Referenz	220
24.6	Testen Sie Ihr Wissen!	220
24.7	Übungen	221
25	In großen Dimensionen denken: Mehrdimensionale Arrays	223
25.1	Schachmatt mit 2D-Arrays	223
25.2	Zug um Zug	226
25.3	Die Stellung entscheidet	228
25.4	Ab jetzt neu: in 3D!	229
25.5	Testen Sie Ihr Wissen!	232
25.6	Übungen	233
26	Objekte der Begierde	236
26.1	Was für'n Ding? Objekte und Eigenschaften	236
26.2	Beschwingt mit dynamischen Properties	239
26.3	JSON — der Typ mit der Maske?	242
26.4	Referenz	245
26.5	Testen Sie Ihr Wissen!	245
26.6	Übungen	245
27	Appetit auf »Zerstörung«? — Destructuring	248
27.1	Arrays auseinandernehmen mit Destructuring	250
27.2	Kurzschreibweise für Objekte	251
27.3	Parameter auseinandernehmen mit Destructuring	252
27.4	Objekte auseinandernehmen mit Object-Destructuring	254
27.5	Ein Standard der »Zerstörung«? Destructuring-Defaults	256
27.6	Genug »kaputt gemacht«? Der unzerstörte Rest	257
27.7	Namen sind viel mehr als Schall und Rauch: benannte Parameter	258
27.8	Benannte Parameter mit Vorgabewerten	260
27.9	Referenz	262
27.10	Testen Sie Ihr Wissen!	263
27.11	Übungen	263
28	Anhang A: Sprachversionen und Transpiler	265
28.1	JavaScript, JScript, LiveScript, ECMAScript, WasZurHölleFürnScript?	265
28.2	Von JavaScript-Maschinen, Umgebungen und Transpilern	267
28.2.1	Die Maschine läuft und läuft ...	267
28.2.2	Transpiling und JS als Transpile Target	267
28.3	Von JS nach JS	268

29	Anhang B: Programmierrichtlinien	269
29.1	ESLint	269
29.2	Allgemein	269
29.3	Variablen & Konstanten	269
29.4	Variablen- & Konstanten-Bezeichner	270
29.5	Codeblöcke	270
29.6	Schlüsselwörter	270
29.7	Funktionen	270
30	Anhang C: Quellen & Literaturhinweise	271
30.1	APA-Style	271
30.2	Quellen	271
	Lösungen der Übungsaufgaben	274
	Lösungen der Wissensfragen	308
	Index	325

Vorwort

JavaScript! JavaScript! JavaScript!

Immer häufiger ist diese Sprache anzutreffen. Webentwickler sind sich einig: An JavaScript führt kein Weg vorbei. Das bestätigen beispielsweise auch die *RedMonk Programming Language Rankings* — eine Seite, die Programmiersprachen nach ihrer Popularität bewertet. Dort hält sich JavaScript momentan auf Platz eins, gefolgt von Java, PHP und Python.

Auch wenn diese Class für Anfänger ist, und wir uns bemühen, Ihnen JavaScript möglichst schonend beizubringen, so legen wir doch auch großen Wert auf gute Codequalität. Denn *noch wichtiger als lauffähiger Code ist gut geschriebener Code* — Code, der leicht zu verstehen, leicht zu ändern und leicht zu erweitern ist. Das sagt schon das berühmte *Software Craftsmanship Manifesto*, das Tausende von Softwareentwicklern weltweit unterzeichnet haben.

Zusammengefasst lässt sich sagen: Unser Ziel ist es, Sie in möglichst kurzer Zeit an einen Punkt zu bringen, an dem Sie sauberen, aktuellen JavaScript-Code programmieren.

Sie haben evtl. schon Vorkenntnisse? Dann lassen Sie sich überraschen, und lernen Sie einen neuen Programmierstil kennen, der von den neusten Sprachfeatures profitiert und zu kürzerem, prägnanterem Code führt.

Beim Durcharbeiten, könnte sich Ihnen der Gedanke aufdrängen, dass wir viel Spaß beim Schreiben hatten. Da könnte etwas dran sein :) Wir hoffen, Sie haben nun mindestens genauso viel Spaß beim Lesen und Programmieren...

In diesem Sinne: Ran an den Code!

Ihre Autoren — Christin & Marco

PS: Wenn Sie Feedback und Anregungen haben, oder einfach verfolgen möchten, woran ich gerade arbeite, folgen oder schreiben Sie mir auf Twitter: [@marcoemrich](https://twitter.com/marcoemrich) — Marco



1 Was Sie schon immer über JavaScript wissen wollten...

In dieser Lektion lernen Sie

- ▶ was JavaScript ist.
- ▶ wo JavaScript eingesetzt wird.
- ▶ was Sie in dieser Class erwartet.

Selbst ein Weg von tausend Meilen beginnt mit einem Schritt.

aus Japan

1.1 JavaScript?

Sie haben sich entschlossen, eine Class über JavaScript zu belegen. Vermutlich haben Sie schon die eine oder andere Vorstellung, was Sie mit JavaScript anfangen können. Vielleicht gelingt es uns dennoch, Sie ein wenig damit zu beeindrucken, wie vielseitig und universell die Einsatzmöglichkeiten von JavaScript tatsächlich sind.

JavaScript ist überall!

The screenshot shows a Google Spreadsheet with several data tables. The main title is 'Hearthstone Master Collection (v. 4.3)'. The spreadsheet is divided into several sections:

- Number of Unique Cards by Rarity:** A table with columns for Rarity (Basic, Common, Rare, Epic, Legendary), Unique, Available, and Remaining. Total Unique: 566, Available: 566, Remaining: 0.00%.
- Number of Playable Cards by Rarity:** A table with columns for Rarity, Playable, Available, and Remaining. Total Playable: 1065, Available: 1065, Remaining: 0.00%.
- Number of Unique Cards by Set:** A table with columns for Set (Classic, Promo, Naxxramas, GvG, Blackrock), Unique, Available, and Remaining. Total Unique: 566, Available: 566, Remaining: 0.00%.
- Number of Playable Cards by Set:** A table with columns for Set, Playable, Available, and Remaining. Total Playable: 1065, Available: 1065, Remaining: 0.00%.
- Number of Unique Cards by Class:** A table with columns for Class (Druid, Hunter, Mage, Paladin, Priest, Rogue, Shaman, Warlock, Warrior, Neutral), Unique, Available, and Remaining. Total Unique: 566, Available: 566, Remaining: 0.00%.
- Number of Playable Cards by Class:** A table with columns for Class, Playable, Available, and Remaining. Total Playable: 1065, Available: 1065, Remaining: 0.00%.
- You Should Buy -> Classic Card Packs:** A table comparing Standard Pack Values (87.0), Classic (442.4), and GvG (442.4) across various metrics like Chance, Total Value, D/E Value, #, Missing Cards, etc.
- Time, Money, and Gold Costs For Complete Collection:** A table showing variables (Adjustable, Classic, GvG) and costs for Gold, Cash, Casual, and Dedicated players.
- Monetary Collection Value:** A table showing the total monetary value for each pack type: Standard (\$8.00), Classic (\$23,988), and GvG (\$14,331).
- Hero Levels:** A table showing the number of cards for each class (Warrior, Shaman, Rogue, Paladin, Hunter, Druid, Warlock, Mage, Priest) across different levels.
- Golden Portraits:** A table showing the number of cards for each class (Warrior, Shaman, Rogue, Paladin, Hunter, Druid, Warlock, Mage, Priest) across different levels.
- Occurrence Per Card:** A table showing the percentage of cards for each rarity (Normal, Golden, Common, Rare, Epic, Legendary).
- Dust Needed To Complete Playable Collection:** A table showing the amount of dust needed for each class (Basic, Common, Rare, Epic, Legendary).

Abb. 1 Ein Google-Spreadsheet

JavaScript ist vor allem in Browsern präsent. Mit JavaScript können Sie Programme schreiben, die im Browser ablaufen. Dem Browser kommt dabei die Rolle einer Plattform zu — wie bei einem Betriebssystem, auf dem Sie Programme installieren können. Eine Installation ist aber für JavaScript-Anwendungen nicht nötig.

Die Funktionalität ist dabei sehr weitreichend. Sie fängt an bei einfachen Validierungen von Formularfeldern (z.B. ob eine E-Mail-Adresse richtig geschrieben ist) und reicht bis zu komplexen Anwendungen wie z.B. [Google Sheets](#)¹ (eine Art Tabellenkalkulation à la Microsoft Excel im Browser, [Abb. 1](#)).

Auch moderne E-Shops (z.B. Amazon, eBay), Videoplattformen (z.B. YouTube) oder Social Sites (Facebook, Twitter) setzen massiv JavaScript ein. Ein modernes Internet ist ohne JavaScript nicht vorstellbar.

Browser können neben JavaScript auch andere Sprachen ausführen, z.B. *Dart* (in Google Chrome) oder *VBScript* (im Microsoft Internet Explorer). JavaScript ist aber die einzige Programmiersprache, die alle modernen Browser zuverlässig unterstützen. Zudem lassen sich enorme Ladeverzögerungen, wie Sie sie vielleicht noch von Java-Applets oder Flash kennen, meistens vermeiden.

Browser sind auch längst nicht die einzige Plattform, die JavaScript ausführen können. So finden Sie JavaScript beispielsweise auch innerhalb von PDFs, zum Scripten von Anwendungen wie Adobe Photoshop oder in der Steuerungssoftware von Robotern² (Abb. 2).

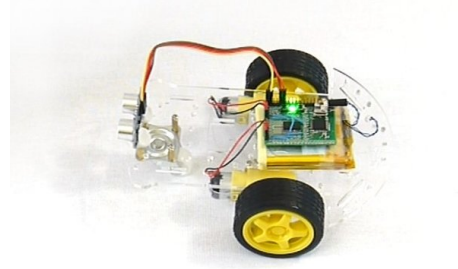


Abb. 2 Roboter mit Espruino-Board, Foto: Pur3 Ltd

Sie sehen: Was Sie in dieser Class lernen, können Sie praktisch überall einsetzen. Dennoch wollen wir uns natürlich vorerst auf den prominentesten Einsatzzweck von JavaScript konzentrieren — den Einsatz im Web.

JavaScript vs. Java

Eine recht hohe Verwechslungsgefahr besteht mit dem Begriff **Java**. *Java* ist auch eine Programmiersprache, hat aber mit JavaScript so wenig gemein wie *car* mit *carpet*. Netscape hat den vorher verwendeten Namen *LiveScript* abgelöst durch den Namen *JavaScript*, den Netscape vom Vertragspartner Sun Microsystems lizenziert hat (siehe [Lektion 28](#)). Das war vermutlich ein Marketing-Trick, um im Fahrwasser der damals sehr erfolgreichen Sprache Java mitzuschwimmen. 2009 wurde Sun dann für 7,4 Milliarden US-Dollar von Oracle gekauft — dadurch ist nun Oracle Besitzer der Marke *JavaScript* [Lektion 30](#).

Deswegen ist der Name *JavaScript* bei vielen Entwicklern nicht sonderlich beliebt. Erfahrene Entwickler sagen oft nur *JS*. Das gilt als »hip« und vermeidet den Java-Bezug (*Java* ist nicht gerade ein Synonym für »schlank und modern«).

JS auf dem Server

Statt im Browser kann JavaScript auch auf dem Server ausgeführt werden — zur dynamischen Generierung von Webseiten, als Alternative zu typischen Serversprachen wie PHP, Ruby oder Java. JavaScript auf dem Server auszuführen ist keine neue Idee. Eine ernsthafte Verbreitung hat die Sprache im Serverbereich aber erst seit ca. 2010/2011 erreicht, im Wesentlichen durch die Plattform [Node.js](#)³.

Damit eröffnen sich für uns neue Möglichkeiten. Früher waren JS-Webentwickler sogenannte **Frontender** (Frontend-Entwickler), die sich vor allem mit der Optik und dem UI-Verhalten im Browser beschäftigt haben. Heutzutage können JS-Entwickler als **FullStack**-Entwickler ihr Geld verdienen und beispielsweise komplexe Finanzregeln auf dem Server implementieren. Wo früher Webanwendungen zwei Programmiersprachen benötigten (JS auf dem Client und z.B. Java auf dem Server), sind heute komplette Anwendungen in JS geschrieben.

1. <https://www.google.com/sheets/about>

2. http://www.espruino.com/distance_sensing_robot

3. <https://nodejs.org>

JS im Embedded-Bereich

Schließlich liegen mit [Espruino](#)⁴ und [Tessel](#)⁵ zwei Boards vor, die JavaScript direkt auf der Hardware ausführen. Damit können Sie Ihre [Pflanzen überwachen](#)⁶, [Ihre Katze beobachten](#)⁷ oder [einen Roboter programmieren](#)⁸. Der Phantasie sind kaum Grenzen gesetzt.

Wir halten es nicht für übertrieben, zu behaupten:



JS ist heutzutage allgegenwärtig!

1.2 Was diese Class ist

Diese Class konzentriert sich auf JavaScript im Browser. Da es sich um Grundlagen handelt, können Sie das Erlernte (mit wenigen Ausnahmen) genauso auf dem Server verwenden — oder in PDFs, zum Scripten von Photoshop, für mobile Anwendungen, um Roboter zu steuern, um Kaffee zu kochen oder das Universum zu retten (beim letzten Punkt sind wir uns nicht zu 100% sicher).



Diese Class behandelt die Sprachversionen **ES6/EcmaScript 2015** und **EcmaScript 2016** von JavaScript. Mehr zu den JavaScript-Sprachversionen finden Sie in [Lektion 28](#).

1.3 Was diese Class nicht(!) ist

Wenn Sie später Software mit JS entwickeln, werden Sie oft Details nachschlagen müssen. Dafür ist diese Class aber nur sehr bedingt geeignet. Wir versuchen Ihnen eher einen Leitfaden an die Hand zu geben, der Ihnen einen schnellen Einstieg in die Sprache JavaScript ermöglicht. Dabei ist die Reihenfolge der Lektionen auf den Lernfortschritt ausgerichtet, und nicht auf schnelles Auffinden von Themen optimiert. Wir erheben auch keinen Anspruch auf Vollständigkeit, sondern beschränken uns bewusst auf die Dinge, die für Sie als Einsteiger zunächst am wichtigsten sind — das »Big Picture« sozusagen.

Wenn Sie später bei Ihrer täglichen Entwicklungsarbeit das volle JS mit all seinen Details benötigen, empfehlen wir Ihnen folgende Nachschlagewerke:

- ▶ <https://developer.mozilla.org> — Mozilla Developer Network (MDN)
Das MDN ist viel mehr als eine Referenz für den Firefox-Browser. Sie finden hier sowohl ausführliche Details zum JS-Sprachkern als auch zur Browser-Funktionalität. Selbst Tabellen, welcher Browser welches Feature ab welcher Version implementiert, sind vorhanden — häufig mit Polyfills (Code, der das Feature nachrüstet) für den Fall der Fälle.
- ▶ www.nodejs.org⁹
Die offizielle Site zur Serverplattform Node.js. Ausführliche Dokumentation zum JS-Sprachkern und allen Node.js-spezifischen Erweiterungen.
- ▶ caniuse.com¹⁰
Kann ich ein bestimmtes Feature benutzen? Diese Site kennt die Antwort. Ausführliche Übersichten zu neuen Features in HTML, CSS und JS mit Angaben zu den Browserversionen und Hinweisen zu Polyfills.

4. <http://www.espruino.com>

5. <https://tessel.io/>

6. <https://www.hackster.io/ryanjgill2/plant-monitoring-system-88ed2b>

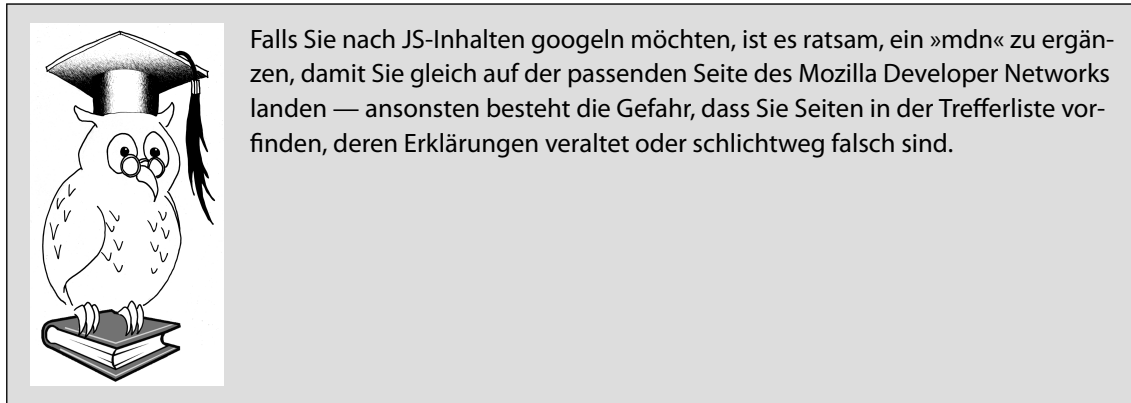
7. https://www.hackster.io/girlie_mac/kittycam-powered-with-raspberry-pi-node-js-dd325b

8. <https://www.hackster.io/29284/ble-bot-9000-c150b8>

9. <http://www.nodejs.org>

10. <http://caniuse.com>

- **Speaking JavaScript** (Rauschmayer 2014, siehe Quellenverzeichnis)
Dieses Buch gilt als *die* Referenz. Es ist modern, ausführlich und präzise.
- **JavaScript: The Definitive Guide** (Flanagan 2011, siehe Quellenverzeichnis)
Das ist das JS-Standardwerk. Es ist zwar nicht mehr ganz aktuell, aber sehr gut strukturiert und ausführlich.
- **Understanding ECMAScript 6** (Zakas, siehe Quellenverzeichnis)
Zakas Werk über alle Neuerungen von ES6/ES2015 ist bei Leanpub erhältlich, und außerdem als [Open Source](#)¹¹.



1.4 Wo sind die Animationen und die GUI?

Wenn Sie schon einmal das Inhaltsverzeichnis überflogen haben, ist Ihnen vielleicht aufgefallen, dass alle Themen recht technisch klingen und scheinbar kaum Bezug zur grafischen Oberfläche (GUI) haben. Die Screenshots sehen auch nicht aus, als könnte sich ein Künstler daran erfreuen. Ist das denn ernst gemeint?

Absolut — wir haben die Erfahrung gemacht, dass es einfacher ist, sich auf den Kern der Sprache zu konzentrieren, statt sich gleichzeitig mit hochglanzpolierten Animationen herumzuschlagen. Konzentrieren Sie sich aufs Wesentliche: auf die Sprache!

Wenn Sie die Sprache erst verstanden haben, ist der Rest einfach. Wie Sie HTML und CSS mit JS manipulieren, erfahren Sie deswegen erst im zweiten Band der Reihe. Wir hoffen, Ihnen den Einstieg dadurch etwas zu erleichtern. Der Code und die Konzepte, die Sie erlernen, sind deswegen nicht weniger praxisrelevant. Alle Beispiele und Übungen sind an Probleme und Situationen angelehnt, die wir tatsächlich so in unserem Programmieralltag erlebt haben.

1.5 Projekt »NerdWorld«

Willkommen zu Ihrem ersten Projekt. Natürlich ist es Ihr erklärtes Ziel, mit der Entwicklung von Software genug Geld zu verdienen, damit Sie sich baldmöglichst in die Karibik absetzen können. Dazu brauchen Sie aber Kunden — und diese sollten Sie optimal zufriedenstellen. Dann kommen weitere, die auch wieder Geld mitbringen.

Ein Kundengespräch

Stellen Sie sich vor, Ihr erster Kunde ist *Björn* — ein Ladenbesitzer, der seine Produkte nun auch im Internet anbieten möchte. Der Laden heißt »NerdWorld« (das Beispiel ist fiktiv — Sie können sich das Googeln sparen).


11. <http://www.nczonline.net/blog/2014/03/26/announcing-understanding-ecmascript-6>

NerdWorld ist ein Online-Shop, bei dem Nerds und andere Technik-Verrückte Produkte des »täglichen Bedarfs« kaufen: Nerfguns, Ufos mit USB-Anschluss, Klingonenwaffen fürs Büro, koffeinhaltige Getränke, stark koffeinhaltige Getränke, noch stärker koffeinhaltige Getränke usw. Eben alles, was der moderne Entwickler zum Überleben im Büro-Dschungel so braucht.

Es gibt zwar schon eine statische HTML-Site, nur können Sie dort bisher keine Produkte kaufen. Die neue Site soll primär aus einem Shop bestehen. Um das Ganze etwas interessanter zu gestalten, sind außerdem weitere Features geplant oder sollen weiterentwickelt werden, wie z.B.

- ▶ ein Chat-Client, über den sich die Kunden mit den Verkäufern/Kundenberatern austauschen
- ▶ eine Newsletter-Funktion, mit der der Shop die Kunden über neue Produkte informiert

Damit dieser Traum auch Realität wird, hat sich der Geschäftsführer an einen Spezialisten gewandt: an Sie! Sie treffen sich also mit ihm und er schildert Ihnen die Situation:



Seit Jahren verkaufen wir cooles Zeug für Nerds — oder solche, die es werden wollen. Unsere Produktpalette reicht von Star-Trek-Fanartikeln über Programmierer-Utensilien bis hin zu Smartphone-gesteuerten Mini-Drohnen und Scherzartikeln wie dem Elektroschock-Kugelschreiber.

Im Laufe dieses Kurses werden wir immer wieder auf Anforderungen von *Projekt NerdWorld* zu sprechen kommen und Ihnen zeigen, wie Ihnen JS dabei weiterhilft. Am Anfang werden das nur kleine Versatzstücke sein, die Sie leider noch nicht komplett in einem echten Projekt nutzen können. Zumindest sollte Ihnen der Bezug verdeutlichen, dass alles, was wir behandeln, relevant für die Praxis ist.

Welche der folgenden Aussagen sind richtig?

Bitte ankreuzen:

- JavaScript läuft nur im Browser.
- JavaScript kann auf Client und Server eingesetzt werden.
- Mit JavaScript lassen sich Roboter steuern.
- Die Bildbearbeitungssoftware Adobe Photoshop kann mit JavaScript gescripted werden.
- JavaScript wird häufig als Textauszeichnungssprache eingesetzt.

Los geht's: die »Chrome Webkonsole«

2

In dieser Lektion lernen Sie

- › die Chrome Webkonsole kennen.
- › Ihre ersten Zeilen Code zu schreiben.
- › grundlegende Programmierrichtlinien kennen.

2.1 Wo ist denn die IDE?

Viele Entwickler schreiben ihren JS-Code in einer speziellen Entwicklungsumgebung, einer sogenannten **IDE** (Integrated Development Environment). Installation und Konfiguration ist allerdings mit viel Aufwand verbunden — den möchten wir Ihnen ersparen. Für Ihre ersten Schritte in JS benötigen Sie nicht viel. Es genügt, die aktuelle Version von **Google Chrome** zu installieren. Chrome bringt ausreichend Werkzeuge mit, um kleine JS-Anwendungen komplett zu entwickeln. Dazu zählt beispielsweise die **Chrome Webkonsole**, die bereits viele Features mitbringt, die sonst eher IDEs vorbehalten sind.

Chrome erhalten Sie unter https://www.google.com/intl/de_de/chrome/ für Ihr jeweiliges Betriebssystem. Die Sprache können Sie auf der Seite umstellen — in diesem Kurs verwenden wir die englische Version (Vereinigte Staaten).

Schritt für Schritt 1:

- 1 Installieren und starten Sie Chrome auf Ihrem Betriebssystem.
- 2 Öffnen Sie die Chrome **Webkonsole** aus dem Developer-Menü (Abb. 3 4) oder per Tastaturkürzel: **ctrl-shift-j** bzw. **cmd-alt-j**.

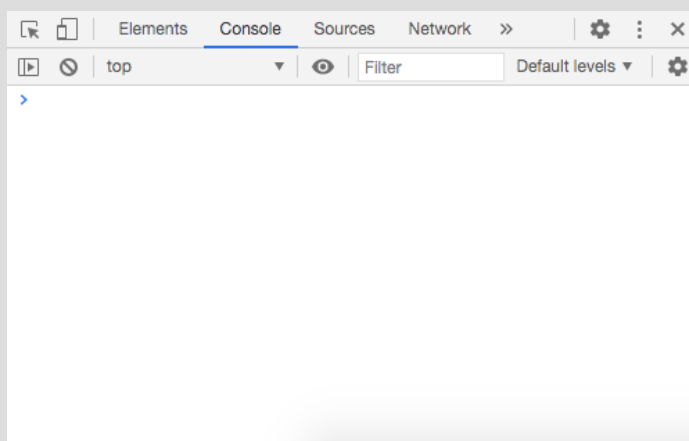


Abb. 3 Google Chrome: Webkonsole

- 3 Wenn Sie mehrere Zeilen schreiben möchten müssen Sie **shift-enter** drücken. Alleiniges drücken von **enter** führt den Code aus.

Alternativ können Sie sich auch ein Snippet erstellen. In einem Snippet können Sie Code in mehrere Zeilen schreiben. Wenn Sie Ihre Konsole bereits offen haben, dann können Sie über den Reiter *Source* in das Source-Menü wechseln. Innerhalb des Source-Menüs finden Sie auf der linken Seite weitere Reiter. Klicken Sie auf die zwei Pfeile nach rechts und wählen Sie *Snippets* aus.

Nun können Sie auf *New snippet* klicken und beliebig viele Snippets erstellen, in welche Sie ganz normal Code schreiben und diesen mit **ctrl-enter** bzw. **cmd-enter** ausführen können.

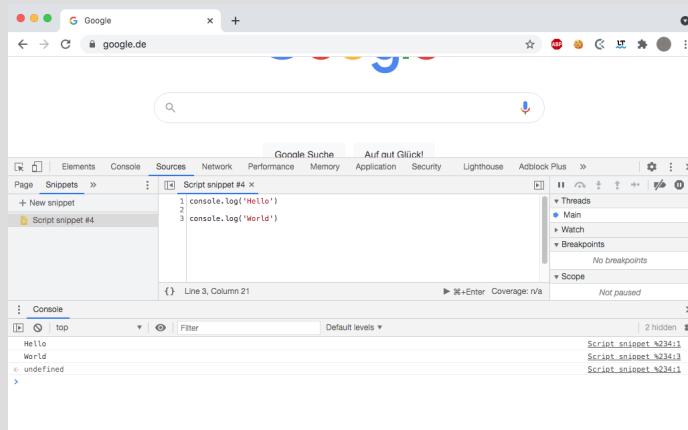


Abb. 4 Google Chrome: Snippets

2.2 Hilfreiche Shortcuts für die Webkonsole

Die folgende Tabelle zeigt einige hilfreiche Tastaturkürzel für die Webkonsole.

Windows/Linux	macOS	Funktion
ctrl-shift-j	cmd-alt-j	Konsole öffnen
enter	enter	Run — den Code in der Konsole ausführen

Tabelle 2.1 Webkonsole Tastaturkürzel

2.3 Hello World mit console.log

Traditionell beginnt das Erlernen einer neuen Programmiersprache mit der Ausgabe von `Hello World`. Sie müssen dazu lediglich die folgende Anweisung in der Webkonsole eingeben.

```
1 console.log('Hello World')
```

Codebeispiel 1 *accompanying_files/02/examples/hello.js*

Sobald Sie die Taste **enter** drücken, erscheint die Ausgabe in der nächsten Zeile.

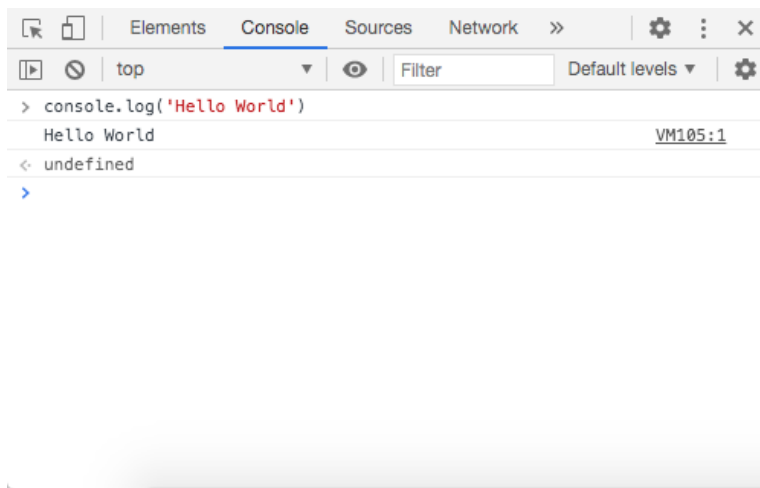


Abb. 5 Ausgabe im Ausgabefenster

Übung 1: In die Konsole geloggt

Geben Sie Ihren Namen aus.

2.4 Alarm! Alarm! — Die Konsole brennt

Betrachten Sie den JS-Code etwas genauer: `log` ist eine sogenannte **Funktion**. Ihre Aufgabe ist es, in die Konsole (`console`) zu loggen, d.h. dort eine nachvollziehbare Ausgabe von Log-Meldungen einzutragen — wie in einem Logbuch.

Im Log lässt sich die Ausgabe des Programmes Schritt für Schritt nachvollziehen. Tatsächlich ist dieses Log nicht für Anwender bestimmt, sondern soll uns Programmierern helfen, den Ablauf eines Programmes nachzuvollziehen.

Ersetzen Sie `console.log` durch `alert`:

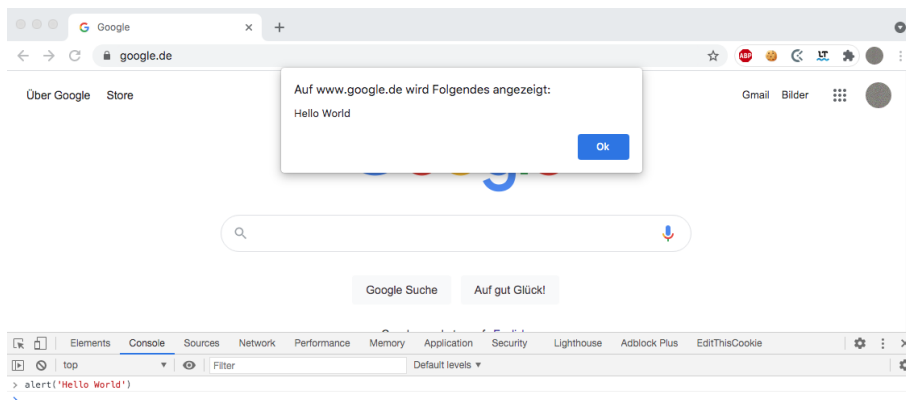


Abb. 6 Ausgabe des Programms im Log

Die Funktion `alert` öffnet eine sogenannte **Alert-Box** im Browserfenster. Sie ist dafür gedacht, Anwendern wichtige Warnmeldungen anzuzeigen. Die Alert-Box ist ein **modales** Popup-Fenster, d.h. der Anwender muss sie wegklicken, um die Website weiter benutzen zu können. Aus Usability-Gründen ist von solchen modalen Popup-Fenstern abzuraten. Setzen Sie sie nur in wenigen begründeten Fällen

ein. Außerdem handelt es sich dabei um eine Funktion, die nur in Browserumgebungen zur Verfügung steht. Wenn Sie z.B. auf dem Server unter *Node.js* programmieren, können Sie sie nicht verwenden.

Sowohl `alert` als auch `log` sind Funktionen. Den Aufruf einer Funktion erkennen Sie an den runden Klammern `()`. Innerhalb der Klammern hinterlegen Sie ein **Argument** — wie hier `'Hello World'`. Was mit dem Argument passiert, ist von der Funktion abhängig. Während `alert` einen Warnhinweis für den Anwender ausgibt, erzeugt `console.log` einen Logeintrag für uns Entwickler.

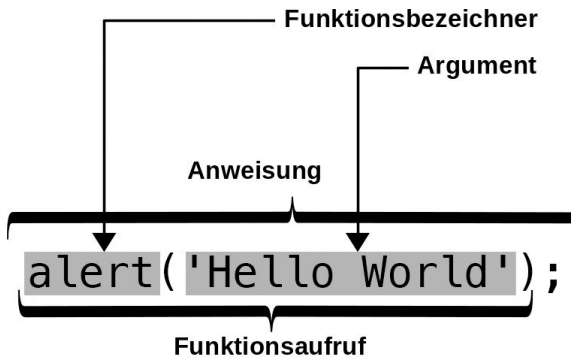


Abb. 7 Begriffserläuterung zum Funktionsaufruf

`alert` ist eine Funktion, die bereits im Browser implementiert ist. Der Browser bringt außerdem `console` mit, ein sogenanntes *Objekt*. Auf `console` steht Ihnen dann wiederum die Funktion `log` zu Verfügung. Vorerst werden Sie nur solche implementierten Funktionen verwenden. In [Lektion 15](#) lernen Sie schließlich, Ihre eigenen Funktionen zu definieren.

Übung 2: Alarm!

Geben Sie den folgenden Text in einer Alert-Box aus: »Alert! Your console is on fire! Hurry! Write an email to the firefighters or tweet with the hashtag #helpMyComputerIsOnFire«

In dieser Class werden Sie der Einfachheit halber mit `console.log` arbeiten. In der Praxis gibt es weitere Möglichkeiten zur Ausgabe. Wir werden sie allerdings hier nicht behandeln, da sie nicht Teil des JS-Sprachkerns sind.



S-Words Shootout: Syntax oder Semantik?

Zwei wichtige Begriffe, denen Sie immer mal wieder in der Softwareentwicklung begegnen, sind **Syntax** und **Semantik**. Es ist wichtig, den Unterschied zu verstehen. Beide Begriffe gibt es nicht erst bei Programmiersprachen. Sprachwissenschaftler prägten die Begriffe zunächst für natürliche Sprachen. Sie lassen sich aber durchaus mit der gleichen Bedeutung auf Programmiersprachen übertragen.

Bei der Syntax geht es um die Regeln, die den Aufbau von Sätzen einer Sprache beschreiben, oder im Fall von JavaScript den Aufbau von Anweisungen. Ist eine Anweisung syntaktisch falsch, meldet die JavaScript-Engine einen Syntaxfehler.

Beispiel

```
alert('Hello')
```

Codebeispiel 2

Hier fehlt die schließende runde Klammer. JavaScript quittiert das mit:

```
Uncaught SyntaxError: missing ) after argument list
```

Codebeispiel 3

Bei der Semantik dagegen geht es um die Bedeutung von Wörtern oder Sätzen. Im Fall von JavaScript also die Bedeutung des Codes. Was soll der Code bewirken?

Beispiel

Beispielsweise könnte der Auftraggeber sich eine Alert-Box mit einer Verabschiedung wünschen. Die Anweisung bewirkt aber eine Begrüßung:

```
1 alert('Welcome to our website')
```

Codebeispiel 4

Dann ist der Code zwar syntaktisch korrekt, aber semantisch falsch. Ein anderes Beispiel wäre ein Additionsprogramm, das falsch rechnet. Auch wenn es zwar grundsätzlich (im Sinne der JavaScript-Engine) funktioniert, so tut es dennoch das Falsche.

2.5 Mehrere Anweisungen

Selbstverständlich können Sie auch mehrere Anweisungen nacheinander loggen:

```
1 console.log('a log message')
2 console.log('... and another one')
```

Codebeispiel 5 *accompanying_files/02/examples/logmessage.js*

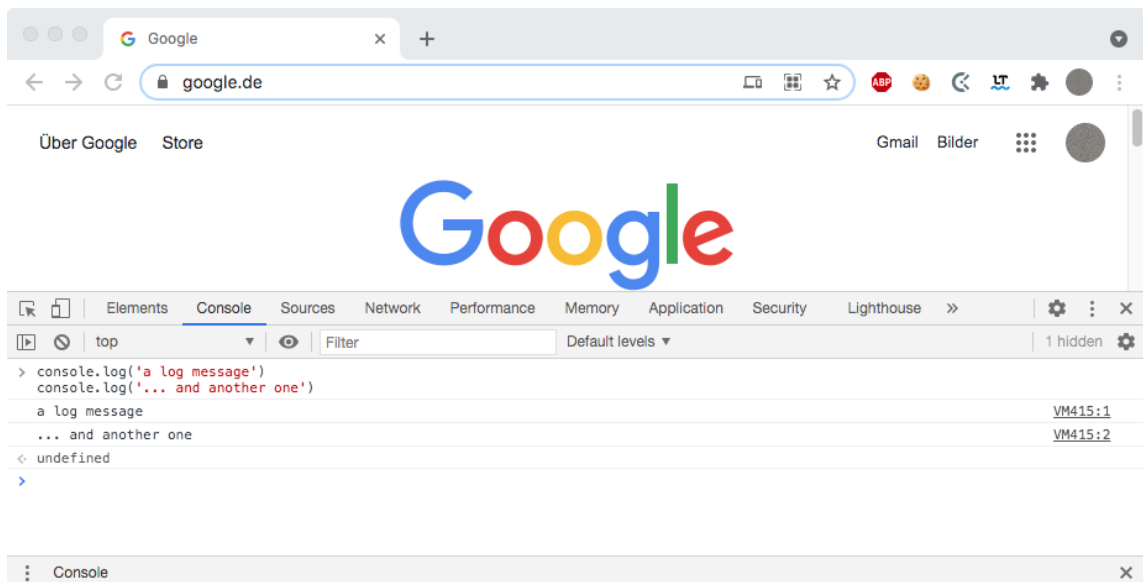



Abb. 8 Zwei Ausgaben mit `console.log`

Schreiben Sie jede Anweisung in eine eigene Zeile. Die Verwendung von Semikolon ist in JavaScript nicht nötig. Sie können das Semikolon also weglassen. In diesem Fall greift ein Mechanismus, der sich **Automatic Semicolon Insertion (ASI)** (dt.: automatische Semikolon-Ergänzung) nennt und das Semikolon für Sie automatisch setzt. Das ist einerseits recht praktisch, andererseits folgt die automatische Semikolon-Ergänzung relativ komplexen Regeln, die Sie erst einmal lernen sollten.

In der Praxis gibt es deswegen zwei Möglichkeiten, damit umzugehen:

1. Den semikolonlosen Stil
2. Den Semikolon-Stil

Beide Stile haben ihre Vor- und Nachteile und es gibt fanatische Verfechter auf beiden Seiten. Wir haben uns in dieser Class für den ersten Stil entschieden — aber nicht etwa, weil wir der Meinung wären, dass es sich dabei um den eindeutig besseren Stil handelt. Letztendlich ist das eher Geschmackssache. Was für den ersten Stil spricht, ist vor allem die größere Verbreitung.



Oder doch mal ohne?

Argumente und Hinweise der Schule des semikolonlosen Stils finden Sie bei:

- ▶ [Marohnić 2010](#)¹²
- ▶ [Fuch 2013](#)¹³
- ▶ [Schlueter 2010](#)¹⁴

Falls Sie den Umgang mit Semikolons üben möchten, können Sie Ihre Fähigkeiten gerne auf den [ASI Scanning Test Grounds](#)¹⁵ unter Beweis stellen.

12. <https://web.archive.org/web/20120501062915/http://mislav.uniqpath.com/2010/05/semicolons>

13. <https://github.com/madrobby/zepto/blob/master/CONTRIBUTING.md>

14. <http://blog.izs.me/post/2353458699/an-open-letter-to-javascript-leaders-regarding>

15. <http://asi.qfox.nl>

2.6 Programmierrichtlinien

Wie Sie sich vielleicht schon vorstellen können, gibt es bei der Programmierung viele Freiheiten, z.B. wie Sie den Code formatieren, wo Sie Semikolons setzen usw. Damit nicht völliges Chaos ausbricht und Ihr Code konsistent bleibt, sollten Sie spezifische Regeln zur Schreibweise verwenden. Eine Sammlung solcher Regeln heißt **Programmierrichtlinien**.

Softwarehäuser haben meist ihre eigenen Programmierrichtlinien, die sich von Firma zu Firma oder sogar von Projekt zu Projekt unterscheiden. Dabei ist vor allem entscheidend, dass Dinge einheitlich geregelt sind und jedes Team-Mitglied diese Regeln befolgt. Nur so lassen sich Wildwuchs und Chaos verhindern. Der exakte Wortlaut jeder Regel ist dagegen eher sekundär.

Gute Beispiele für ausführliche Programmierrichtlinien-Dokumente sind z.B. der [jQuery-Style-Guide](#)¹⁶, den die Entwickler der jQuery-Bibliothek befolgen oder der [JavaScript Styleguide von Airbnb](#)¹⁷.

Programmierrichtlinien sind auch wichtig, weil Sie nie wissen können, wo oder von wem Ihr Code einmal weiterentwickelt wird. Nachdem ein Projekt abgeschlossen ist, ist die entstandene Software oftmals noch jahrelang im Einsatz. Die Entwicklung steht nie still. Fehler sind zu verbessern, Änderungswünsche umzusetzen, Anpassungen durchzuführen — die sogenannte **Wartung** der Software. Der Programmierer, der den Code nach Ihnen pflegen, verbessern und abändern muss, ist Ihr **Wartungsprogrammierer**. Seien Sie nett zu ihm! Helfen Sie ihm, Ihren Code zu verstehen, indem Sie sich an gängige Programmierrichtlinien halten. (Und wenn Ihnen nur nett sein nicht ausreicht, stellen Sie sich den Wartungsprogrammierer einfach als Serienkiller vor, der genau weiß, wo Sie wohnen...)

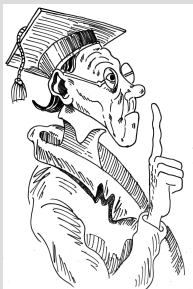
Vielleicht sind Sie selbst derjenige, der den Code Jahre später noch mal erweitern muss — seien Sie nett zu sich selbst!

Jeder Narr kann Code schreiben, den ein Computer versteht. Gute Programmierer schreiben Code, den Menschen verstehen.

Fowler (1999)



Wir werden ab sofort unsere eigenen Programmierrichtlinien entwickeln, dabei orientieren wir uns am [JavaScript Styleguide von Airbnb](#)¹⁸. Zu der jeweils aktuellen Thematik bestimmen wir passende Regeln, die uns das Programmiererleben erleichtern. Airbnb bietet seine [Programmierrichtlinien](#)¹⁹ für JavaScript auf GitHub an. In dieser Sammlung finden Sie zu jedem Gebiet in JavaScript Anweisungen, wie Sie Ihren Code sauber schreiben. Folgende Regeln wollen wir ab sofort einhalten:



Programmierrichtlinien

- Schreiben Sie immer genau eine Anweisung in genau eine Zeile.
- Verzichten Sie beim Beenden einer Anweisung auf das Semikolon.

Weitere Programmierrichtlinien folgen in Kürze.

16. <http://contribute.jquery.org/style-guide/js>

17. <https://github.com/airbnb/javascript>

18. <https://github.com/airbnb/javascript>

19. <https://github.com/airbnb/javascript>

2.7 Seien Sie »strict«

Neben den Programmierrichtlinien gibt es noch eine weitere Möglichkeit, wie Sie sich selbst und Ihrem Wartungsprogrammierer das Leben etwas erleichtern können:

Seien Sie strikt!

Keine Sorge, dazu müssen Sie nicht gleich einen Besen verschlucken oder Ihre Kollegen mit strenger Stimme schimpfen, wenn Ihnen ihr Code nicht gefällt.

Es gibt vielmehr die Möglichkeit, JS in den sogenannten **strict mode** zu versetzen. In diesem Modus verhält sich JS tatsächlich strikter. Das bedeutet:

- ▶ Einige Richtlinien, die der *standard mode* als reine Konvention betrachtet, führen im *strict mode* zu einer Fehlermeldung.
- ▶ Viele veraltete Sprachkonstrukte werden nicht mehr akzeptiert (z.B. die `with`-Anweisung)
- ▶ Fehlerhafter oder problematischer Code, den JS im *standard mode* ohne Murren akzeptiert, führt nun zu einer (meist) aussagekräftigen Fehlermeldung. Damit bemerken Sie Probleme frühzeitig und sparen sich die oft aufwendige Fehlersuche.
- ▶ Einige der neueren Sprachkonstrukte (z.B. `let`) lassen sich in den meisten Umgebungen überhaupt nur im *strict mode* verwenden.

Um JS in den *strict mode* zu versetzen, müssen Sie nicht viel tun: Es genügt, wenn Sie die Anweisung

```
1 'use strict'
```

Codebeispiel 6

an den Anfang Ihrer JS-Datei schreiben.

Ab sofort werden wir das immer tun, da es für uns praktisch keine Nachteile hat. Wir müssen schließlich keinen Legacy-Code (genauer: Code in alter JS-Syntax) unterstützen. In [Lektion 20](#) erfahren Sie dann noch mal genauer, was es mit `'use strict'` auf sich hat. Wenn Sie es noch genauer wissen möchten, finden Sie auf MDN (Mozilla Development Network) einen [Eintrag über Strict Mode](#)²⁰, der alle Unterschiede zwischen den Modi erläutert.

2.8 Kommentare oder: Geben Sie doch Ihren Senf dazu

Mit *Kommentaren* ist es möglich, hilfreiche Erläuterungen zum Code zu schreiben oder auch Codezeilen zu deaktivieren. Kommentare können helfen, Code schneller zu verstehen, den Ihr früheres Ich oder ein anderer Programmierer geschrieben hat.



Richtig platzierte Kommentare verbessern die Lesbarkeit Ihres Codes.

Der JavaScript-Interpreter ignoriert alles, was sich in Kommentaren befindet. Das gilt sogar für vollständige JavaScript-Anweisungen. In JavaScript gibt es zwei Möglichkeiten, Kommentare einzubinden. Dazu wird zwischen

- ▶ *einzeiligen* und
- ▶ *mehrzeiligen* Kommentaren

unterschieden.

20. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode/Transitioning_to_strict_mode#Differences_from_non-strict_to_strict

Einzeilige Kommentare binden Sie mit zwei Slashes `//` ein.

```
1 //The following code contains a meaningful string.
2 console.log('I am a meaningful string')
```

Codebeispiel 7

Sie sollten die Kommentare in der Praxis nur für komplexere Zusammenhänge verwenden und keine Trivialitäten kommentieren, wie im obigen Beispiel. Wenn Sie aufgrund zu vieler Kommentare Ihren Code kaum noch erkennen können, haben Sie nichts gewonnen.

Mehrzeilige Kommentare erlauben Ihnen, längere — eben mehrzeilige — Texte in den Code einzubetten. Ein solcher Kommentar beginnt mit den Zeichen `/*` und endet mit `*/`.

```
1 /*
2   TODO: Add a more spicy greeting here. Candidates are:
3   - Speak Friend and Enter
4   - Live long and prosper
5   - Welcome to the Future (in mystery Futurama Voice)
6 */
7 alert('Welcome to NerdWorld')
```

Codebeispiel 8 *accompanying_files/02/examples/comment.js*

Kommentare, die mit `//` beginnen, müssen am Ende einer Zeile stehen oder komplett alleine in der Zeile. Kommentare der Form `/* ein Kommentar */` dürfen Sie dagegen fast überall platzieren.

Beispiel

```
1 console.log(/* some comment inside */ 'I am a meaningful string')
```

Codebeispiel 9

2.9 Testen Sie Ihr Wissen!

1. Welche der folgenden Zeilen sind syntaktisch korrekt?

Bitte ankreuzen:

- `console.log 'Hello'`
- `console.log("Hello")`
- `console.log 'Hello';`
- `console.log('Hello')`

2. Wie versetzen Sie JS in den strikten Modus?

Bitte ankreuzen:

- Durch `'use strict'` am Anfang des Codes.
- Durch den Aufruf der Funktion `useStrict()` am Anfang des Codes.
- Starten Sie Chrome mit der Kommandozeilen-Option `--strict`.
- Durch `'strict'` am Anfang des Codes.
- Wählen Sie *strict mode* aus dem *Developer*-Menü des Browsers.

Inhalt von b: **1**

Inhalt von c: **2**

2. Bestimmen Sie den Inhalt von `a` und `b`, nach dem Ausführen der folgenden Anweisung:

```
1 const [b, a] = [3, 2, 1]
```

Inhalt von a: **2**

Inhalt von b: **3**

3. Bestimmen Sie den Inhalt von `a` und `b`, nach dem Ausführen der folgenden Anweisung:

```
1 const [b, a] = [1, 2, 3]
```

Inhalt von a: **2**

Inhalt von b: **1**

Index

A

APA-Style 271
API 67 102 110 147 151
arithmetische Operatoren 31
61

B

Babel 268
Bezeichner 39-43 48-54
128-131 270 275 309-311
benannte Parameter 248
258-260
Boolean 71-76 151 243

C

camelCase 51-52 192 270
ceil 67-69
charAt 105-106 110-111 142
152 317-319
Codeblock 126
coercion 61
concat 151

D

Debugger 190
default 208 213-215 257
Duck Typing 236
destructuring 248-262

E

ECMAScript 18-19 31 39 110
151 179 236 245 265-273
endsWith 110
Engine 25 126 164-165 267
else 80-88 92 101 270 314
EVA-Prinzip 138
every 167 180 184 219-220

explizite Typkonvertierung
61-62

F

fill 151 205
filter 158-160 170 174 180
190 210 219-220
find 180 219-220
findIndex 181 219-220
Firefox 18 165 265-267
Firefox Webkonsole
first class 162
floor 67-70 313
forEach 171-174 180 205
216-220
function expression 126-127
169
function scope 187
Funktionsausdruck 127 169

G

Gruppierungsoperator 36 46
GUI 19

H

Higher-Order-Funktion
167-169 174-176 181 210
320
Hoisting 127 187

I

IDE 21
identifier 39
if 80-95 106 158-162 179
186 194 209 270 314-315
implizite Typkonvertierung
61-63 76
indexOf 102-105 110 148-153
319
Indexoperator 144 206 219
239
includes 108-111 151 317
IPO-Model 138

J

join 147-151 205 220
322-323

K

key 238-243 251 255
Klammerung 35-36
Kommentar 29-31
Konstante 32 44-45 67-69 81
186 233 240 245 250 276
313
Kontrollstrukturen 81 90 158

L

lastIndexOf 103-104 110-111
151 317
Literal 33-38 47 63 76 118
189 240 309
local scope 186-187
logische Operatoren 94 98

M

Magic Number 44
Modulo 31-36 46
Mozilla 18-19 28 131-132
265-267

N

named parameters 258-259
263

O

oder-Operator 95
Operand 61-64 78

P

Paradigma 162
Pi 54 67-69 276

Präzedenz 35-36 46 75 98
Priorität 35-37 46 75 98
Programmierrichtlinie 32 37
47 188
pop 145 150-151
Property 238-240
Prototyp 266
push 144-145 151 206

R

random 68-69
reduce 162 175-180 190 210
219-220 228 245
regex 114 119-121
regular Expression 107 115
248
Regulärer Ausdruck 107
114-119 242
relationale Operatoren 71-72
77-78
repeat 110-111 317
replace 106-110 122 242
rest 19 31 153 160 211
215-216 224 228 257
return 31 140-141 160
253-255 270
reverse 149-151
round 67-69

S

Scopes 186
Scoping 39
SCREAMING_SNAKE_CASE 52
192 270
shift 21-22 31 56 151
slice 150-153 201 231 248
321
Signatur 132
some 178-180 219-220
sort 147-151 163-167 179
203 220
splice 145-146 150-151
split 148-151 171 184 220
248-252 322-323
startsWith 109-110
substr 103-106 110-111
131-132 152 317-319

T

Template Strings 55-57
toFixed 68-69
toString 151
toLowerCase 105 109-111
317

toUpperCase 105-106
110-111 152 317-319
trim 104 110 248
Typkonvertierung 61-63 76

U

und-Operator 96
unshift 151

V

Verkettungsoperator 55-56 63
Verzweigung 81-83

W

Webkonsole 21-22 30 42 165

Z

Zuweisungsoperator 35-36 40
45-47 75 269