

4. Auflage

DOM, Node.js,
ES-Modules,
Web-APIs,
OOP, mobile
Anwendungen,
ECMAScript

```
function addEvent(element, eventType, eventHandler)
{
  if (window.addEventListener) {
    element.addEventListener(
      eventType, eventHandler, false
    );
  }
  else if (window.attachEvent) {
    element.attachEvent('on' + eventType, eventHandler);
  }
  else {
    element['on' + eventType] = eventHandler;
  }
}
```

Philip Ackermann

JavaScript

Das umfassende Handbuch

- ▶ Grundlagen, Anwendung, Referenz
- ▶ OOP, moderne ECMAScript-Features, mobile Anwendungen
- ▶ Web-APIs, Node.js und KI mit TensorFlow.js und LangChain



Mit allen Beispielen zum Download



Rheinwerk
Computing

Kapitel 2

Erste Schritte

Nach wie vor wird JavaScript hauptsächlich für die Erstellung dynamischer Webseiten, sprich innerhalb eines Browsers, eingesetzt. Bevor wir uns in späteren Kapiteln im Detail mit anderen Anwendungsgebieten befassen, werde ich Ihnen in diesem Kapitel zeigen, auf welche Weisen Sie JavaScript in eine Webseite einbinden und einfache Ausgaben erzeugen können. Dieses Kapitel bildet somit die Grundlage für die folgenden Kapitel.

Bevor wir uns ausführlicher mit der Sprache JavaScript an sich beschäftigen, sollten Sie zunächst wissen, in welchem Zusammenhang JavaScript mit *HTML (Hypertext Markup Language)* und *CSS (Cascading Style Sheets)* innerhalb einer Webseite steht, wie man JavaScript in eine Webseite einbindet und wie man Ausgaben erzeugen kann.

2.1 Einführung in JavaScript und die Webentwicklung

Die wichtigsten drei Sprachen für die Erstellung von Web-Frontends sind sicherlich HTML, CSS und JavaScript. Jede dieser Sprachen hat dabei ihre eigene Bestimmung.

2.1.1 Der Zusammenhang zwischen HTML, CSS und JavaScript

Mithilfe von HTML legen Sie über *HTML-Elemente* die *Struktur* einer Webseite und die *Bedeutung* (die *Semantik*) einzelner Komponenten auf einer Webseite fest. Sie beschreiben beispielsweise, welcher Bereich auf der Webseite den Hauptinhalt darstellt, welcher die Navigation, und definieren Komponenten wie Formulare, Listen, Schaltflächen, Eingabefelder oder, wie in Abbildung 2.1 zu sehen, Tabellen.

Artist	Album	Release Date	Genre
Monster Magnet	Powertrip	1998	Spacerock
Kyuss	Welcome to Sky Valley	1994	Stonerrock
Ben Harper	The Will to Live	1997	Singer/Songwriter
Tool	Lateralus	2001	Progrock
Beastie Boys	Ill Communication	1994	Hip Hop

Abbildung 2.1 HTML verwenden Sie, um die Struktur einer Webseite zu definieren.

Über CSS dagegen gestalten Sie mithilfe von speziellen *CSS-Regeln*, wie die einzelnen Komponenten, die Sie zuvor in HTML definiert haben, dargestellt werden sollen, sprich, Sie legen das *Design* und das *Layout* einer Webseite fest. Sie definieren hierbei beispielsweise Textfarbe, Textgröße, Umrandungen, Hintergrundfarben, Farbverläufe etc. In Abbildung 2.2 ist zu sehen, wie CSS dazu genutzt wurde, Schriftart und Schriftgröße der Tabellenüberschriften sowie der Tabellenzellen anzupassen, Rahmen zwischen Tabellenspalten und Tabellenzeilen hinzuzufügen und die Hintergrundfarbe der Tabellenzeilen im Wechsel mit einer jeweils anderen Hintergrundfarbe einzufärben. Das Ganze sieht dann schon um einiges ansprechender aus als die Variante ohne CSS.

Artist	Album	Release Date	Genre
Monster Magnet	Powertrip	1998	Spacerock
Kyuss	Welcome to Sky Valley	1994	Stonerrock
Ben Harper	The Will to Live	1997	Singer/Songwriter
Tool	Lateralus	2001	Progrock
Beastie Boys	Ill Communication	1994	Hip Hop

Abbildung 2.2 Mit CSS definieren Sie das Layout und das Aussehen einzelner Elemente der Webseite.

JavaScript zu guter Letzt dient dazu, der Webseite (bzw. den Komponenten auf einer Webseite) *dynamisches Verhalten* hinzuzufügen bzw. die Interaktivität auf der Webseite zu erhöhen. Beispiele hierfür sind die bereits in Kapitel 1, »Grundlagen und Einführung«, angesprochene Sortierung und Filterung von Tabellendaten (siehe Abbildung 2.3 und Abbildung 2.4). Während CSS also für das Design einer Webseite zuständig ist, kann mithilfe von JavaScript die Nutzerfreundlichkeit und die Interaktivität einer Webseite erhöht werden.

Q	Search artist		
Artist ▼	Album	Release Date	Genre
Beastie Boys	Ill Communication	1994	Hip Hop
Ben Harper	The Will to Live	1997	Singer/Songwriter
Kyuss	Welcome to Sky Valley	1994	Stonerrock
Monster Magnet	Powertrip	1998	Spacerock
Tool	Lateralus	2001	Progrock

Abbildung 2.3 JavaScript ermöglicht Ihnen, eine Webseite nutzerfreundlicher und interaktiver zu gestalten, z. B. um wie hier die Daten in einer Tabelle sortierbar ...

Q	Be		
Artist ▾	Album	Release Date	Genre
Beastie Boys	Ill Communication	1994	Hip Hop
Ben Harper	The Will to Live	1997	Singer/Songwriter

Abbildung 2.4 ... oder, wie hier gezeigt, die Daten filterbar zu machen.

Eine Webseite besteht also (in den allermeisten Fällen) aus einer Kombination von HTML-, CSS- und JavaScript-Code (siehe Abbildung 2.5) – wobei gilt: Auch wenn ich eben gesagt habe, dass JavaScript für das Verhalten einer Webseite zuständig ist, kann man funktionsfähige Webseiten auch gänzlich ohne JavaScript erstellen. Ja, prinzipiell kann man Webseiten auch ohne CSS erstellen. Prinzipiell schon. Dann wird eben nur das HTML vom Browser ausgewertet. In so einem Fall ist die Webseite aber weniger schick (ohne CSS) und weniger interaktiv und nutzerfreundlich (ohne JavaScript), siehe erneut Abbildung 2.1.

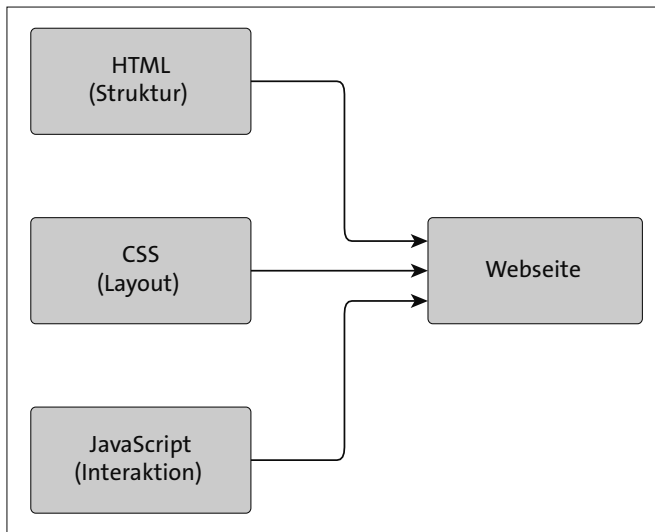


Abbildung 2.5 In der Regel wird innerhalb einer Webseite eine Kombination aus HTML, CSS und JavaScript verwendet.

Merke

HTML dient der Struktur einer Webseite, CSS dem Layout und dem Design, JavaScript dem Verhalten und der Interaktivität.

Definition

Web- und Softwareentwickler sprechen in diesem Zusammenhang auch gerne von drei Schichten: HTML bildet die *Inhaltsschicht*, CSS die *Darstellungsschicht* und JavaScript die *Verhaltensschicht*.

Trennen des Codes für die einzelnen Schichten

Guter Entwicklungsstil sieht vor, die einzelnen Schichten nicht zu vermischen, sprich HTML-, CSS- und JavaScript-Code unabhängig voneinander und in separaten Dateien vorzuhalten. Dies erleichtert den Überblick über ein Webprojekt und sorgt letztendlich dafür, dass Sie effektiver entwickeln können. Darüber hinaus können Sie auf diese Weise ein und dieselben CSS- und JavaScript-Dateien auch in verschiedenen HTML-Dateien einbinden (siehe Abbildung 2.6) und damit dieselben CSS-Regeln bzw. denselben JavaScript-Quelltext in verschiedenen HTML-Dateien wiederverwenden.

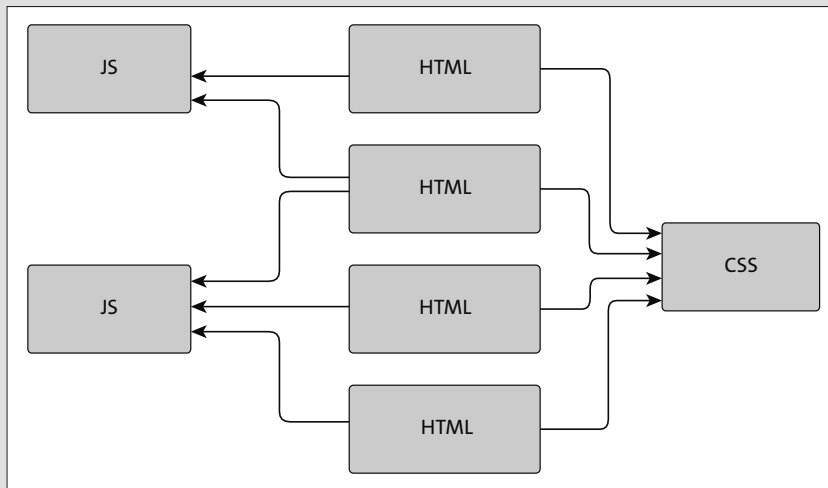


Abbildung 2.6 Wenn Sie CSS- und JavaScript-Code nicht direkt in den HTML-Code schreiben, sondern in separate Dateien, erleichtert das die Wiederverwendbarkeit.

Eine gute Vorgehensweise bei der Entwicklung einer Webseite ist es, sich erst über deren Struktur Gedanken zu machen: Welche Bereiche gibt es auf der Webseite? Welche Überschriften gibt es? Gibt es Daten, die in tabellarischer Form dargestellt werden? Aus welchen Einträgen besteht die Navigation? Welche Informationen sind im Fußbereich der Seite enthalten, welche im Kopfbereich? Hierbei verwendet man ausschließlich HTML. Die Webseite sieht dann zwar noch nicht schön aus und ist nur wenig interaktiv, aber darum soll es in diesem ersten Schritt bewusst nicht gehen, um nicht vom Wesentlichen, dem Inhalt der Webseite, abzulenken.

Aufbauend auf dieser strukturellen Grundlage, setzt man anschließend das Design mit CSS und das Verhalten der Webseite mit JavaScript um. Dabei können diese beiden Schritte prinzipiell parallel auch von verschiedenen Personen vorgenommen werden. Beispielsweise kann sich ein Webdesigner um das Design mit CSS kümmern, während ein Webentwickler die Funktionalität in JavaScript programmiert (in der Praxis sind zwar Webdesigner und Webentwickler häufig ein und dieselbe Person, aber insbesondere in großen Projekten mit vielen, vielen Webseiten ist eine Verteilung der Zuständigkeiten nicht selten).

Phasen der Website-Entwicklung

Bei der Entwicklung professioneller Websites gehen der reinen Entwicklung natürlich mehrere Phasen voraus. Bevor überhaupt mit der Entwicklung begonnen wird, werden in Konzept- und Designphasen Prototypen (entweder digital oder ganz klassisch mit Stift und Papier) entworfen. Das eben beschriebene schrittweise Vorgehen (erst HTML, dann CSS, dann JavaScript) bezieht sich somit nur auf die Entwicklung.

Auszeichnungssprache HTML und Stilsprache CSS

HTML und CSS sind übrigens keine Programmiersprachen! HTML ist eine *Auszeichnungssprache* und CSS eine *Stilsprache*, nur JavaScript ist von den drei genannten eine *Programmiersprache*. Daher sind auch Aussagen wie »Das lässt sich doch mit HTML programmieren« genau genommen nicht korrekt. Vielmehr müsste man sagen: »Das lässt sich doch mit HTML umsetzen.«

Definition

Der Prozess des Darstellens einer Webseite durch den Browser wird *Rendern* genannt. Man sagt unter Entwicklern auch: »Der Browser rendert eine Webseite.« Dabei wird HTML-, CSS- und JavaScript-Code ausgewertet, ein entsprechendes Modell der Webseite erstellt (auf das wir in Kapitel 5, »Webseiten dynamisch verändern«, noch zu sprechen kommen) und die Webseite in das Browserfenster »gezeichnet«. Im Detail ist dieser Prozess recht komplex, und wenn Sie sich mehr für dieses Thema interessieren, kann ich Ihnen den Blogbeitrag unter www.html5rocks.com/de/tutorials/internals/howbrowserswork/ empfehlen.

2.1.2 Das richtige Werkzeug für die Entwicklung

Für das Erstellen von JavaScript-Dateien würde prinzipiell zwar ein einfacher Texteditor ausreichen (und für einfache Codebeispiele ist dies auch durchaus in Ordnung), allerdings sollten Sie besser einen guten Editor verwenden, der Sie beim Schreiben von JavaScript unterstützt und der speziell für die Entwicklung von JavaScript-Programmen ausgelegt ist. Ein solcher Editor unterstützt Sie beispielsweise dahin gehend, dass er den Quelltext farblich

hervorhebt (*Syntax Highlighting*), Ihnen Schreibarbeit beim Schreiben von Code abnimmt (*Code Completion* bzw. *Syntax Completion*), Fehler im Quelltext erkennt (*Bug Detection*) und vieles mehr (das Gleiche gilt natürlich für das Arbeiten mit HTML und CSS).

Editoren

Es gibt mittlerweile eine Reihe wirklich guter Editoren, mit denen sich effektiv arbeiten lässt. In der Entwickler-Community sind beispielsweise Sublime Text (www.sublimetext.com, siehe Abbildung 2.7) und Atom (<https://atom.io>, siehe Abbildung 2.8) beliebt, die beide für Windows, macOS und Linux zur Verfügung stehen. Während Ersterer derzeit 99 US\$ kostet (Stand: September 2025), ist der Editor Atom kostenlos. Im Detail haben beide Editoren ihre eigenen Features und Stärken, sind sich prinzipiell aber doch recht ähnlich. Probieren Sie einfach aus, welcher Ihnen mehr zusagt.

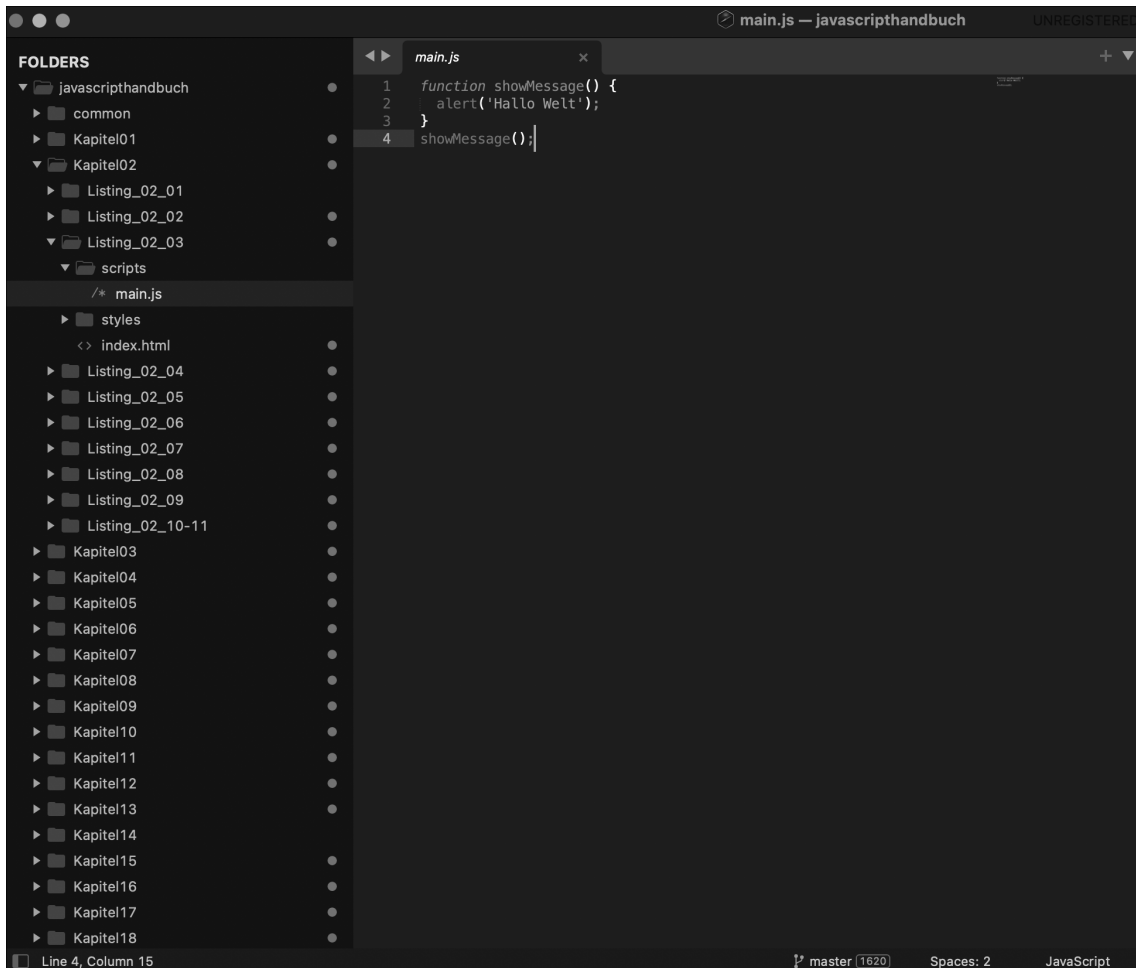


Abbildung 2.7 Der Editor Sublime Text

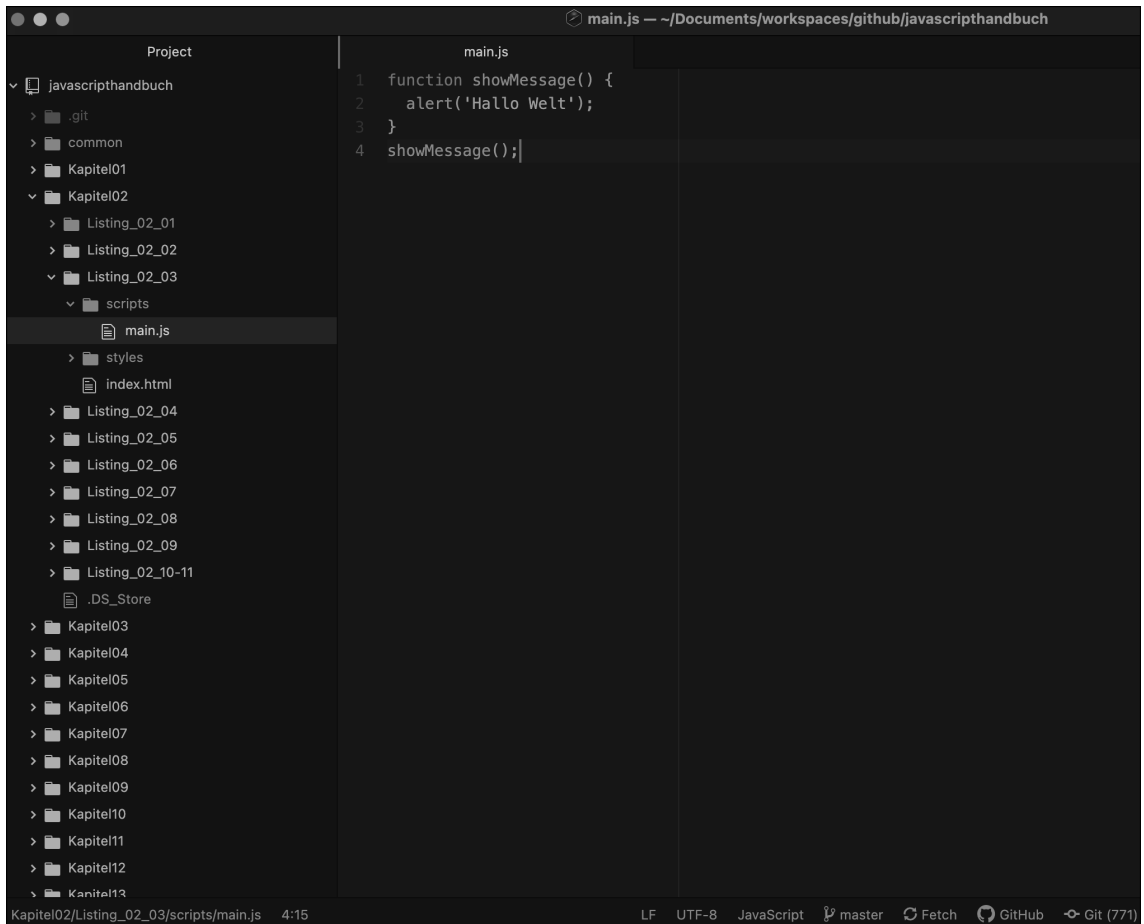


Abbildung 2.8 Der Editor Atom

Entwicklungsumgebungen

Softwareentwickler, die von Sprachen wie Java oder C++ zu JavaScript wechseln, sind von »ihren Programmiersprachen« in den meisten Fällen sogenannte *Entwicklungsumgebungen* gewohnt (im Englischen kurz *IDE* für *Integrated Development Environment*). Eine Entwicklungsumgebung können Sie sich gewissermaßen wie einen sehr, sehr mächtigen Editor vorstellen, der gegenüber einem »normalen« Editor noch diverse andere Features bereitstellt, wie beispielsweise die Synchronisation mit einem *Source-Verwaltungssystem*, das Ausführen von *automatischen Builds* oder die Integration von *Test-Frameworks*. (Wenn Sie jetzt nur verständnislos den Kopf schütteln und sich fragen, was sich hinter all diesen Begriffen verbirgt, warten Sie bis Kapitel 20, »Einen professionellen Entwicklungsprozess aufsetzen«, denn dort gehe ich auf diese fortgeschrittenen Themen der Softwareentwicklung mit JavaScript ein.)

WebStorm von JetBrains (www.jetbrains.com/webstorm/, siehe Abbildung 2.9) ist ein Beispiel für eine sehr beliebte und, wie ich finde, sehr gute Entwicklungsumgebung, die sowohl für Windows als auch für macOS und Linux zur Verfügung steht.

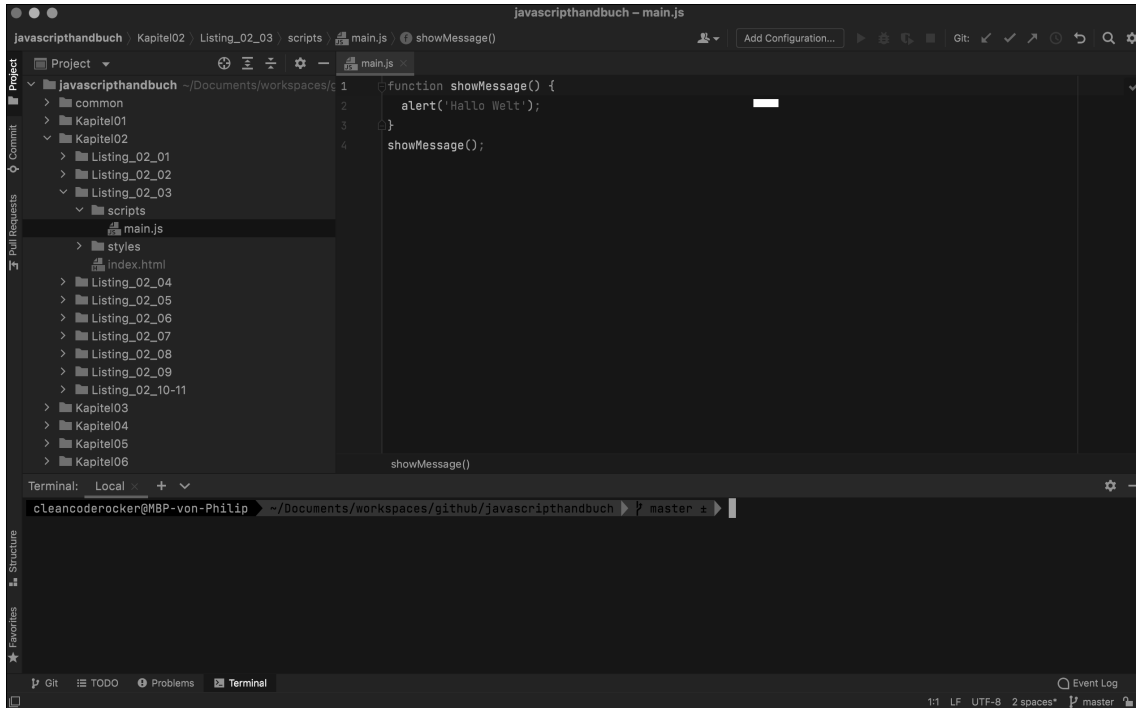


Abbildung 2.9 Die WebStorm-IDE

Mein persönlicher Favorit unter den Entwicklungsumgebungen ist mittlerweile Visual Studio Code von Microsoft (<https://code.visualstudio.com/>, siehe Abbildung 2.10). Es steht kostenlos zum Download bereit, kann flexibel über Plug-ins erweitert werden und ist gefühlt performanter als etwa WebStorm (und selbst übrigens eine JavaScript-Applikation).

Tipp

Für den Anfang – also beispielsweise für das Ausprobieren der Codebeispiele in diesem Buch – empfehle ich Ihnen, einen der genannten Editoren zu verwenden und (noch) keine Entwicklungsumgebung. Letztere haben nämlich den Nachteil, dass sie teils mit Menüs und Funktionalitäten überfrachtet sind, sodass Sie sich – zusätzlich zum Lernen von JavaScript – auch noch mit dem Erlernen der Entwicklungsumgebung beschäftigen müssen. Das möchte ich Ihnen für den Moment zumindest möglichst ersparen.

Zudem sind Entwicklungsumgebungen eigentlich auch erst ab einer gewissen Projektgröße sinnvoll, für kleinere Projekte und die Beispiele in diesem Buch reicht ein Editor allemal (nicht dass wir nicht auch komplexe Themen behandeln werden!). Hinzu kommt, dass die Editoren

in der Regel im Hinblick auf die Ausführungsgeschwindigkeit schneller als die Entwicklungsumgebungen sind.

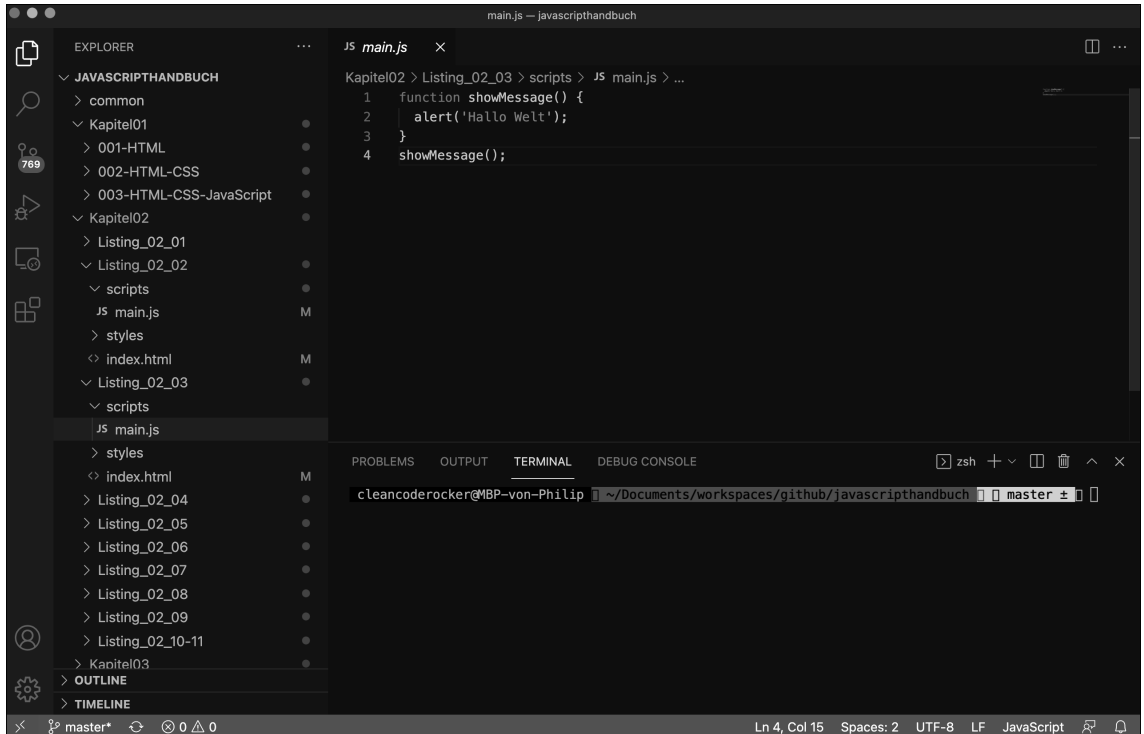


Abbildung 2.10 Microsoft Visual Studio Code

2.2 JavaScript in eine Webseite einbinden

Da ich davon ausgehe, dass Sie bereits wissen, wie man eine HTML-Datei erstellt und wie man eine CSS-Datei einbindet und Sie »nur« hier sind, um JavaScript zu lernen, will ich auch keine weitere Zeit mit Details über HTML und CSS verschwenden, sondern gleich mit JavaScript loslegen. Keine Sorge: Das Einbinden und Ausführen einer JavaScript-Datei gestaltet sich alles andere als schwierig.

Traditionsgemäß starte ich (wie nahezu jedes Buch über Programmiersprachen) mit einem sehr einfachen sogenannten *Hello World*-Beispiel, das lediglich die Ausgabe *Hello World* (bzw. in unserem Fall die Ausgabe *Hallo Welt*) erzeugt. Das ist zwar noch wenig spannend, aber momentan geht es ja darum, Ihnen zu zeigen, wie Sie überhaupt erst mal eine JavaScript-Datei in eine HTML-Datei einbinden und den in der JavaScript-Datei enthaltenen Quelltext ausführen können. Um die komplexen Dinge kümmern wir uns dann später.

2.2.1 Eine geeignete Ordnerstruktur vorbereiten

Für den Anfang und das Durcharbeiten der folgenden Beispiele empfehle ich Ihnen, die in Abbildung 2.11 gezeigte Verzeichnisstruktur für jedes Beispiel zu verwenden. Auf oberster Ebene liegt die HTML-Datei, denn das ist für den Browser der Einstiegspunkt und damit die Datei, die Sie gleich im Browser aufrufen werden.

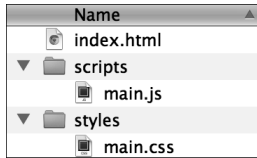


Abbildung 2.11 Exemplarische Ordnerstruktur

Für die CSS- und JavaScript-Dateien dagegen bietet es sich an, jeweils verschiedene Ordner anzulegen. Die Namen *styles* (für CSS-Dateien) und *scripts* (für JavaScript-Dateien) sind dabei recht üblich. Insbesondere wenn Sie es während der Entwicklung mit vielen verschiedenen JavaScript- und CSS-Dateien zu tun haben, erleichtert diese Aufteilung nämlich (bzw. generell eine Aufteilung mit Unterordnern) die Übersicht über Ihr Projekt.

Startpunkt einer JavaScript-Anwendung

Die meisten Beispiele in diesem Buch folgen auch der in Abbildung 2.11 gezeigten Aufteilung, da wir für den Anfang den JavaScript-Code nur im Browser ausführen werden und dazu die HTML-Datei *index.html* gewissermaßen als Einstiegspunkt in das jeweilige Programm verwenden.

Später in Kapitel 17, »Serverseitige Anwendungen mit Node.js erstellen«, werde ich Ihnen zeigen, wie Sie JavaScript auch unabhängig von einem Browser und damit unabhängig von einer entsprechenden HTML-Datei ausführen können. In diesem Fall benötigen Sie dann keine HTML- und damit auch keine CSS-Dateien.

JavaScript im Browser ausführen

Sie können zwar auch innerhalb eines Browsers JavaScript ausführen, ohne dafür eine HTML-Datei zu erstellen, die das entsprechende Skript einbindet (nämlich über spezielle durch die Browser zur Verfügung gestellte Entwicklerwerkzeuge, siehe Abschnitt 2.3.2, »Auf die Konsole schreiben«), für den Anfang wollen wir dieses Feature aber noch nicht verwenden.

2.2.2 Eine JavaScript-Datei erstellen

JavaScript-Code sollte man also besser in einer separaten Datei (oder mehreren separaten Dateien) speichern und diese dann in den HTML-Code einbinden. Sie benötigen also als Erstes eine JavaScript-Datei. Dazu öffnen Sie einfach den Editor Ihrer Wahl (oder falls Sie nicht

auf meinen Rat gehört haben: die Entwicklungsumgebung Ihrer Wahl), erstellen eine neue Datei, geben folgende Zeilen Quelltext dort hinein und speichern die Datei anschließend unter dem Namen *main.js*.

```
function showMessage() {  
    alert('Hallo Welt');  
}
```

Listing 2.1 Ein ganz einfaches JavaScript-Beispiel, in dem eine Funktion definiert wird

Merke

JavaScript-Dateien haben die Endung *.js*. Prinzipiell sind zwar auch andere Dateierendungen möglich, allerdings hat die Endung *.js* den Vorteil, dass Editoren, Entwicklungsumgebungen und Browser direkt wissen, worum es sich bei dem Inhalt handelt. Sie sollten also alle JavaScript-Dateien immer mit der Endung *.js* abspeichern (Browser erkennen JavaScript-Dateien, die von einem Webserver geliefert werden, übrigens an dem sogenannten *Content-Type-Header*, einer Information, die mit der jeweiligen Datei vom Server mitgeliefert wird).

Definiert ist in Listing 2.1 eine sogenannte *Funktion* mit Namen `showMessage`, die ihrerseits eine andere Funktion (mit Namen `alert`) aufruft und dieser die Meldung `Hallo Welt` übergibt. Die Funktion `alert` ist eine Standardfunktion von JavaScript, auf die ich später in diesem Kapitel noch mal kurz zu sprechen komme. Mit Funktionen im Allgemeinen werden wir uns dagegen detailliert in Kapitel 3, »Sprachkern«, beschäftigen.

Downloadbereich zum Buch

Dieses und alle folgenden Codebeispiele finden Sie auch im Downloadbereich zum Buch (siehe www.rheinwerk-verlag.de/6132). Von dort können Sie den Code bequem herunterladen und in Ihrem Editor oder direkt im Browser öffnen (wobei ich ja der Meinung bin, dass man am effektivsten lernt, wenn man die Beispiele selbst abtippt und dabei Schritt für Schritt nachvollzieht).

2.2.3 Eine JavaScript-Datei in eine HTML-Datei einbinden

Um den JavaScript-Quelltext jetzt innerhalb einer Webseite verwenden zu können, müssen Sie die JavaScript-Datei mit der Webseite verknüpfen bzw. die JavaScript-Datei in die HTML-Datei einbinden. Das geschieht über das HTML-Element `<script>`.

Dieses Element kann prinzipiell auf zwei verschiedene Arten genutzt werden: Zum einen können – wie ich direkt im Anschluss zeigen werde – externe JavaScript-Dateien in das HTML eingebunden werden, zum anderen kann JavaScript-Quelltext auch direkt zwischen das öffnende `<script>`-Tag und das schließende `</script>`-Tag geschrieben werden.

Zu Letzterem zeige ich Ihnen später noch ein Beispiel, allerdings ist diese Vorgehensweise eher nur in Ausnahmefällen sinnvoll, weil dann JavaScript-Code und HTML-Code vermischt, sprich in einer Datei gespeichert werden (was wiederum aus genannten Gründen keine Best Practice ist). Schauen wir uns also erst an, wie man es richtig macht und eine separate Datei einbindet.

Das `<script>`-Element hat insgesamt sechs Attribute, von denen das `src`-Attribut mit Sicherheit das wichtigste ist: Hierüber wird der Pfad zu der JavaScript-Datei angegeben, die eingebunden werden soll (eine Übersicht darüber, wofür auch die anderen Attribute gedacht sind, zeigt Tabelle 2.1).

Attribut	Bedeutung	Anmerkung
<code>async</code>	Gibt an, ob das Herunterladen der verlinkten JavaScript-Datei asynchron stattfinden soll und das Herunterladen anderer Dateien nicht unterbrochen wird (siehe Abschnitt 2.2.5). Ergibt nur in Kombination mit dem <code>src</code> -Attribut Sinn.	optional
<code>charset</code>	Gibt den Zeichensatz des Quelltextes an, der über das <code>src</code> -Attribut eingebunden wird. Ergibt nur in Kombination mit dem <code>src</code> -Attribut Sinn, wird aber selten verwendet, weil die meisten Browser dieses Attribut nicht beachten. Zudem ist es besserer Stil, innerhalb einer Website überall UTF-8 zu verwenden und dies im <code><meta></code> -Element über das <code>charset</code> -Attribut zu definieren.	optional
<code>defer</code>	Gibt an, ob mit dem Ausführen der verlinkten JavaScript-Datei bis zu dem Zeitpunkt gewartet werden soll, zu dem der Inhalt der Webseite komplett verarbeitet wurde (siehe Abschnitt 2.2.5). Ergibt nur in Kombination mit dem <code>src</code> -Attribut Sinn, wird aber insbesondere von älteren Browsern nicht unterstützt.	optional
<code>language</code>	Ursprünglich dazu gedacht, die Version des verwendeten JavaScript-Codes anzugeben, wird von Browsern aber weitestgehend nicht beachtet.	veraltet
<code>src</code>	Gibt den Pfad zu der JavaScript-Datei an, die eingebunden werden soll.	optional
<code>type</code>	Dient der Angabe des <i>MIME-Types</i> (siehe Kasten), um die Skriptsprache (in unserem Fall JavaScript) zu identifizieren. Prinzipiell können Sie das Attribut jedoch weglassen, da in diesem Fall standardmäßig <code>text/javascript</code> verwendet wird, das von den meisten Browsern unterstützt wird.	optional

Tabelle 2.1 Die Attribute des `<script>`-Elements

Erstellen Sie nun also eine HTML-Datei mit Namen *index.html* und fügen Sie dort den folgenden in Listing 2.2 gezeigten Inhalt ein.

```
<!DOCTYPE html>
<html>
<head lang="de">
  <meta charset="UTF-8">
  <title>Beispiel</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css">
</head>
<body>
<!--Hier wird die JavaScript-Datei eingebunden -->
<script src="scripts/main.js"></script>
</body>
</html>
```

Listing 2.2 Einbinden von JavaScript in HTML

Wenn Sie jetzt diese HTML-Datei im Browser öffnen, passiert noch nichts, denn die Funktion, die wir in Listing 2.1 definiert haben, wird noch an keiner Stelle aufgerufen. Ergänzen Sie daher am Ende der JavaScript-Datei den Aufruf `showMessage()` und laden Sie die Webseite im entsprechenden Browser neu. Dann sollte sich ein kleiner Hinweisdialog öffnen, der die Meldung *Hallo Welt* enthält und je nach Browser ein etwas anderes Aussehen hat (siehe Abbildung 2.12).



Abbildung 2.12 Hinweisdialoge in den verschiedenen Browsern

```
function showMessage() {  
    alert('Hallo Welt');  
}  
showMessage();
```

Listing 2.3 Funktionsdefinition und Funktionsaufruf**Definition**

MIME-Types (*Multipurpose Internet Mail Extension*, auch *Internet Media Type* oder *Content Type* genannt) waren ursprünglich dafür gedacht, innerhalb von E-Mails, die verschiedene Inhalte (wie Bilder, PDF-Dateien etc.) enthalten, zwischen den einzelnen Inhaltstypen zu unterscheiden. Mittlerweile werden MIME-Types aber nicht nur im Zusammenhang mit E-Mails verwendet, sondern immer, wenn Daten über das Internet übertragen werden. Sendet ein Server eine Datei mit einem speziellen MIME-Type, weiß der Client (z. B. der Browser) direkt, um welchen Typ es sich bei den übertragenen Daten handelt.

Für JavaScript war der MIME-Type lange nicht standardisiert, sodass es direkt mehrere MIME-Types gab, wie beispielsweise `application/javascript`, `application/ecmascript`, `text/javascript` und `text/ecmascript`. Seit 2006 gibt es aber einen offiziellen Standard (www.rfc-editor.org/rfc/rfc4329.txt), der die erlaubten MIME-Types für JavaScript definiert. Demnach sind `text/javascript` und `text/ecmascript` beide veraltet, stattdessen sollten `application/javascript` und `application/ecmascript` verwendet werden. Paradoxerweise ist es am sichersten, im Fall von JavaScript (im `<script>`-Element) keinen MIME-Type anzugeben, da das `type`-Attribut ohnehin von den meisten Browsern ignoriert wird.

Mehrere JavaScript-Dateien einbinden

Sie können innerhalb einer HTML-Datei selbstverständlich auch mehrere JavaScript-Dateien einbinden. Dann verwenden Sie für jede Datei, die eingebunden werden soll, einfach ein eigenes `<script>`-Element.

2.2.4 JavaScript direkt innerhalb des HTML definieren

Der Vollständigkeit halber zeige ich Ihnen noch, wie Sie JavaScript auch direkt innerhalb einer HTML-Datei definieren können. Das ist zwar in der Regel nicht ratsam, weil man auf diese Weise HTML- und JavaScript-Code in einer Datei mischt, zu wissen, dass es trotzdem geht, schadet aber nicht.

Dazu schreiben Sie den entsprechenden JavaScript-Code einfach innerhalb des `<script>`-Elements, statt ihn über das `src`-Attribut zu verlinken. Listing 2.4 zeigt das Beispiel von eben,

verwendet aber keine separate JavaScript-Datei für den JavaScript-Code, sondern bindet diesen direkt in das HTML ein. Das `src`-Attribut fällt daher komplett weg.

```
<!DOCTYPE html>
<html>
<head lang="de">
  <meta charset="UTF-8">
  <title>Beispiel</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css">
</head>
<body>
<script>
  function showMessage() {
    alert('Hallo Welt');
  }
  showMessage();
</script>
</body>
</html>
```

Listing 2.4 Nur in Ausnahmefällen sinnvoll: Definition von JavaScript direkt in einer HTML-Datei

Hinweis

Beachten Sie, dass `<script>`-Elemente, die das `src`-Attribut verwenden, keinen Quelltext zwischen `<script>` und `</script>` haben dürfen. Sollte dies dennoch der Fall sein, wird dieser Quelltext ignoriert.

Tipp

Verwenden Sie besser separate JavaScript-Dateien für Ihren Quelltext, statt ihn direkt in ein `<script>`-Element zu schreiben. Das schafft eine saubere Trennung zwischen der Struktur (HTML) und dem Verhalten (JavaScript) einer Webseite.

Das `<noscript>`-Element

Über das `<noscript>`-Element können Sie einen HTML-Abschnitt definieren, der angezeigt wird, wenn JavaScript im Browser nicht unterstützt wird oder vom Nutzer deaktiviert wurde. Wird dagegen JavaScript unterstützt bzw. ist es aktiviert, wird der Inhalt des `<noscript>`-Elements nicht angezeigt.

```
<noscript>
  JavaScript ist nicht verfügbar oder es ist deaktiviert. <br />
  Bitte verwenden Sie einen Browser, der JavaScript unterstützt,
  oder aktivieren Sie JavaScript in Ihrem Browser.
</noscript>
```

Listing 2.5 Beispiel für die Verwendung des `<noscript>`-Elements

2.2.5 Platzierung und Ausführung der `<script>`-Elemente

Hätten Sie vor einigen (vielen) Jahren einen Webentwickler gefragt, an welcher Stelle ein `<script>`-Element innerhalb einer Webseite einzubinden ist, hätte dieser wahrscheinlich dazu geraten, es im `<head>`-Bereich der Webseite unterzubringen. In den Anfangstagen der Webentwicklung war man nämlich der Ansicht, verlinkte Dateien wie CSS-Dateien und eben JavaScript-Dateien sollten an einer zentralen Stelle innerhalb des HTML-Codes platziert werden.

Mittlerweile ist man allerdings wieder davon abgekommen. CSS-Dateien werden zwar weiterhin im `<head>`-Bereich platziert, JavaScript-Dateien dagegen sollten vor dem schließenden `</body>`-Tag eingebunden werden. Der Grund dafür ist folgender: Wenn der Browser eine Webseite lädt, lädt er neben dem HTML-Code auch die eingebundenen Dateien wie beispielsweise Bilder, CSS-Dateien und JavaScript-Dateien. Je nach Browserimplementierung sind moderne Browser dazu in der Lage, mehrere solcher Dateien parallel herunterzuladen. Wenn der Browser allerdings ein `<script>`-Element vorfindet, fängt er sofort damit an, den entsprechenden Quelltext zu verarbeiten und mithilfe des JavaScript-Interpreters auszuwerten. Um dies aber zu können, muss der entsprechende JavaScript-Quelltext zunächst vollständig heruntergeladen werden. Während das passiert, pausiert der Browser jedoch das Herunterladen aller anderen Dateien und das *Parsen* (also das Verarbeiten) des HTML-Codes, was wiederum zur Folge hat, dass für den Nutzer das Aufbau der Webseite gefühlt länger dauert (siehe Abbildung 2.13).

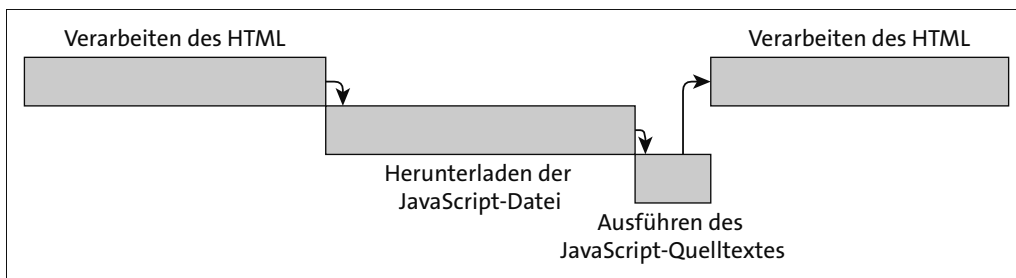


Abbildung 2.13 Standardmäßig wird die Verarbeitung des HTML-Codes gestoppt, wenn der Browser auf ein `<script>`-Element trifft.

Hinzu kommt, dass man innerhalb des JavaScript-Quelltextes häufig auf HTML-Elemente der jeweiligen Webseite zugreifen möchte (wie das genau funktioniert, zeige ich Ihnen in Kapitel 5, »Webseiten dynamisch verändern«). Wenn dann der JavaScript-Code ausgeführt wird, bevor die Verarbeitung dieser HTML-Elemente stattgefunden hat, kommt es zu einem Zugriffsfehler (siehe Abbildung 2.14). Platzieren Sie dagegen das `<script>`-Element vor dem schließenden `</body>`-Tag, sind Sie diesbezüglich auf der sicheren Seite (siehe Abbildung 2.15), da in dem Fall alle Elemente, die sich innerhalb des `<body>`-Elements befinden, bereits geladen sind (mit Ausnahme anderer `<script>`-Elemente selbstverständlich).

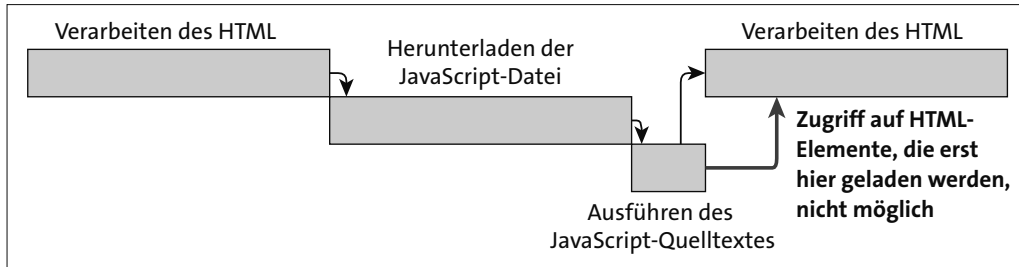


Abbildung 2.14 Wenn JavaScript auf noch nicht geladene HTML-Elemente zugreift, kommt es zu einem Fehler.

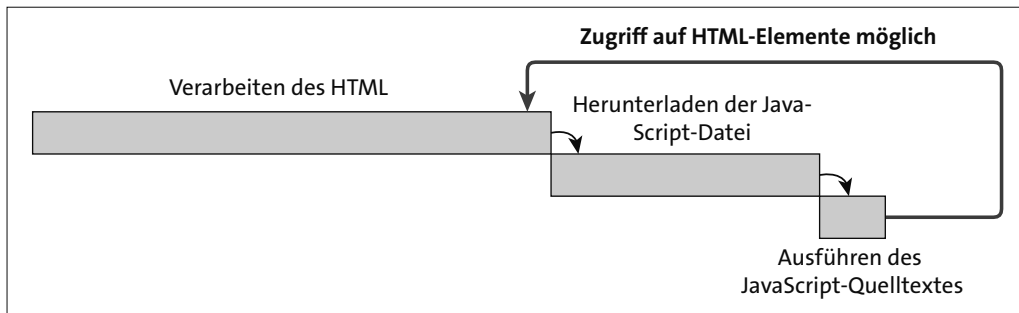


Abbildung 2.15 Wird das `<script>`-Element vor das schließende `</body>`-Tag platziert, sind dagegen alle Elemente innerhalb des `<body>`-Elements geladen.

Merke

In der Regel sollten Sie `<script>`-Elemente am Ende des `<body>`-Elements positionieren. Das liegt daran, dass der Browser bei jedem `<script>`-Element zunächst den dort enthaltenen bzw. eingebundenen JavaScript-Quelltext auswertet, bevor er mit dem Laden weiterer HTML-Elemente fortfährt.

Zwei Attribute, über die man das Ladeverhalten von JavaScript beeinflussen kann, sind die Attribute `async` und `defer`, die ich ja eben schon kurz erwähnt hatte (siehe Tabelle 2.1). Erste-

res sorgt dafür, dass das Verarbeiten des HTML-Codes nicht pausiert wird, wenn der Browser auf ein `<script>`-Element trifft. Das Herunterladen der JavaScript-Datei passiert quasi asynchron (daher der Name `async`). Das Prinzip davon zeigt Abbildung 2.16.

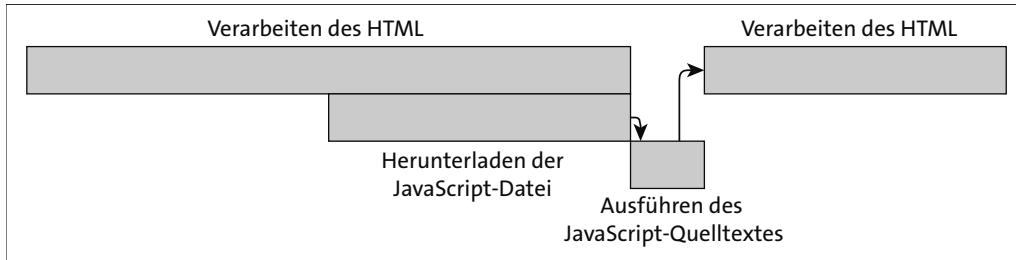


Abbildung 2.16 Über das Attribut »`async`« wird der HTML-Code so lange weiterverarbeitet, bis das entsprechende JavaScript heruntergeladen wurde.

Wie Sie sehen können, wird der JavaScript-Code auch hier direkt ausgeführt, sobald die entsprechende JavaScript-Datei vollständig heruntergeladen wurde.

Einen Schritt weiter geht das Attribut `defer`. Dieses Attribut sorgt nämlich zum einen dafür, dass – wie bei `async` – das Verarbeiten des HTML-Codes nicht pausiert wird. Zum anderen wird der JavaScript-Quelltext erst ausgeführt, nachdem der HTML-Code vollständig verarbeitet wurde (siehe Abbildung 2.17). Die Ausführung des JavaScript-Codes wird quasi verschoben (daher der Name `defer`).

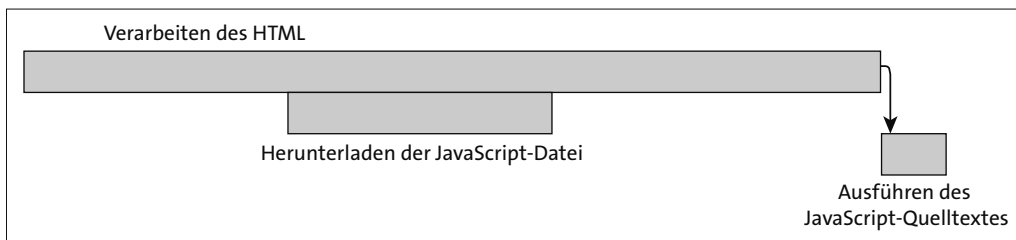


Abbildung 2.17 Über das Attribut »`defer`« erreicht man, dass das entsprechende JavaScript erst dann ausgeführt wird, nachdem der gesamte HTML-Code der Webseite geladen wurde.

Wann sollten Sie also welches Attribut einsetzen? Für den Moment können Sie sich merken, dass Sie wahrscheinlich am besten bedient sind, wenn Sie standardmäßig keines der beiden Attribute verwenden. Das Attribut `async` eignet sich eigentlich nur für solche Skripte, die komplett eigenständig funktionieren und quasi nichts mit dem HTML auf der Webseite »zu tun haben«. Ein Beispiel hierfür wäre die Verwendung von Google Analytics. Das Attribut `defer` dagegen wird momentan noch nicht von allen Browsern unterstützt, sodass Sie auch hier eine Verwendung mit Vorsicht abwägen sollten.

Definition

Eine weitere Möglichkeit, sicherzustellen, dass der gesamte Inhalt der Webseite geladen wurde, bevor JavaScript-Code ausgeführt wird, ist die Verwendung sogenannter *Ereignis-Handler* und *Ereignis-Listener* (auch *Event-Handler* und *Event-Listener* genannt). Beide werde ich Ihnen in Kapitel 6, »Ereignisse verarbeiten und auslösen«, detailliert vorstellen, an dieser Stelle zeige ich Ihnen aber schon mal grob, wie beide verwendet werden, weil sie in den Quelltextbeispielen im Buch schon vor den Beispielen zu Kapitel 6 auftauchen.

Beide, sowohl Ereignis-Handler als auch Ereignis-Listener, dienen allgemein gesagt dazu, auf bestimmte Ereignisse, die bei der Ausführung eines Programms auftreten, zu reagieren und bestimmten Code auszuführen (es gibt zwar einen kleinen, feinen Unterschied zwischen Ereignis-Handleern und Ereignis-Listnern, der für den Moment aber nicht wichtig ist und den ich Ihnen dann in Kapitel 6 erklären werde). Ereignisse können Mausklicks, Tastatureingaben, Änderungen der Fenstergröße und vieles mehr sein. Auch für Webseiten gibt es verschiedene Ereignisse, die ausgelöst werden und auf die man mit solchen Ereignis-Handleern und Ereignis-Listnern reagieren kann. So wird ebenfalls ein Ereignis ausgelöst, wenn der Inhalt einer Webseite vollständig geladen wurde.

Um einen Ereignis-Handler für dieses Ereignis zu definieren, können Sie das Attribut `onload` verwenden: Der Code, den Sie hier als Wert für ein solches Attribut angeben, wird aufgerufen, wenn die Webseite vollständig geladen wurde. Als Wert kann man hier eine JavaScript-Anweisung angeben, z. B. den Aufruf einer Funktion, wie in Listing 2.6 gezeigt:

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>Beispiel</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css">
</head>
<body onload="showMessage()">
<script src="scripts/main.js"></script>
</body>
</html>
```

Listing 2.6 Verwenden eines Event-Handlers

Ereignis-Listener dagegen können nicht über HTML definiert werden, stattdessen verwendet man die Funktion `addEventListener()` des Objekts `document` (dazu später mehr), der man den Namen des Ereignisses übergibt sowie die Funktion, die ausgeführt werden soll, wenn das Ereignis ausgelöst wird. Listing 2.7 zeigt ein entsprechendes Beispiel.

```
function showMessage() {
    alert('Hallo Welt');
}
document.addEventListener('DOMContentLoaded', showMessage);
```

Listing 2.7 Verwenden von Event-Listnern

Den Aufruf `showMessage()`, den Sie eben an das Ende der Datei *main.js* angefügt hatten, müssten Sie in beiden Fällen wieder entfernen, sonst wird die Funktion zweimal aufgerufen (einmal durch das Skript selbst und einmal durch den Ereignis-Handler/Ereignis-Listener), und entsprechend wird zweimal hintereinander ein Hinweisdialog angezeigt.

2.2.6 Den Quelltext anzeigen

Alle Browser bieten in der Regel eine Möglichkeit, sich den Quelltext einer Webseite anzeigen zu lassen. Dies kann in vielen Fällen hilfreich sein, beispielsweise um zu schauen, wie ein bestimmtes Feature auf einer Website, die Sie entdeckt haben, implementiert ist.

Unter Chrome können Sie den Quelltext einsehen, indem Sie den Menüpunkt ANZEIGEN • ENTWICKLER • QUELLTEXT ANZEIGEN aufrufen (siehe Abbildung 2.18), in Firefox über EXTRAS • BROWSER-WERKZEUGE • SEITENQUELLTEXT ANZEIGEN (siehe Abbildung 2.19), in Safari über ENTWICKLER • SEITENQUELLTEXT EINBLENDEN (siehe Abbildung 2.20), in Opera über ENTWICKLER • QUELLTEXT ANZEIGEN (siehe Abbildung 2.21) und in Microsoft Edge über EXTRAS • ENTWICKLER • QUELLE ANZEIGEN (siehe Abbildung 2.22).

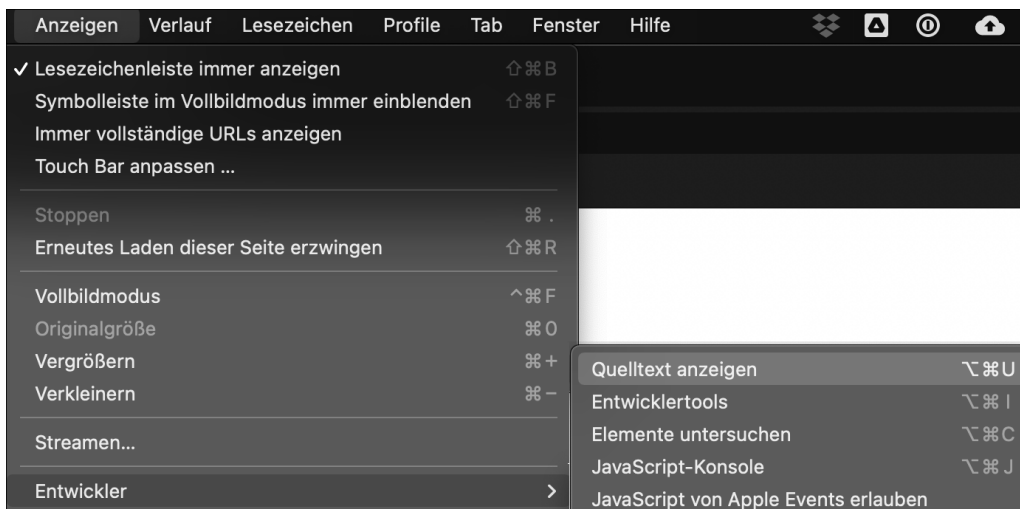


Abbildung 2.18 Quelltext anzeigen in Chrome

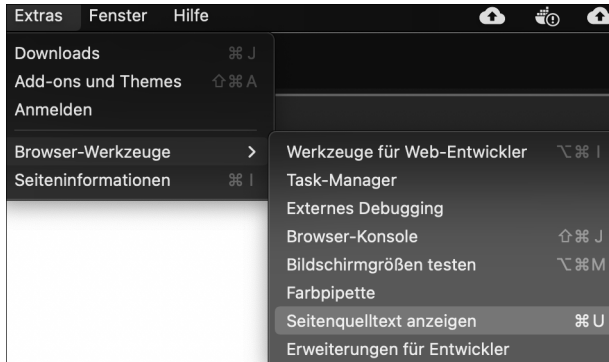


Abbildung 2.19 Quelltext anzeigen in Firefox

Quelltext von umfangreicheren Webseiten

Wenn Sie sich den Quelltext von umfangreicheren Webseiten anschauen, ist dieser häufig sehr unleserlich. Das hat in der Regel verschiedene Gründe: Zum einen werden Inhalte häufig dynamisch generiert, zum anderen wird JavaScript durch Webentwickler oft bewusst komprimiert und unkenntlich gemacht – Ersteres, um Platz zu sparen, Letzteres, um den Quelltext vor fremden Blicken zu schützen. Mit Komprimierung und Unkenntlichmachung des Quelltextes beschäftigen wir uns in diesem Buch nicht. Wenn Sie Interesse daran haben, kann ich Ihnen mein Buch *Professionell entwickeln mit JavaScript: Design, Patterns und Praxistipps* empfehlen, das sich mit solch fortgeschrittenen Themen befasst und ebenfalls beim Rheinwerk Verlag erschienen ist.

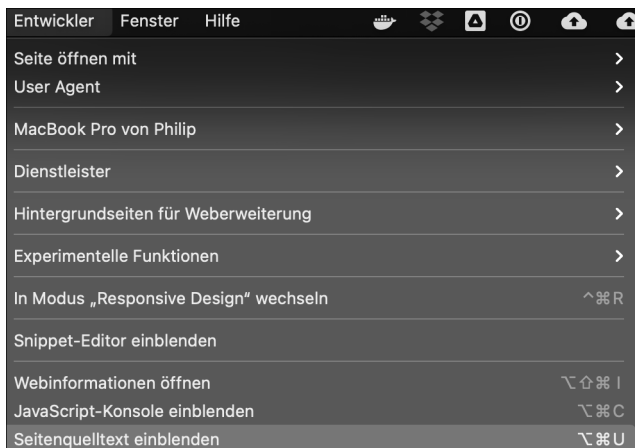


Abbildung 2.20 Quelltext anzeigen in Safari

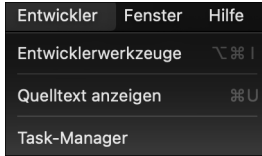


Abbildung 2.21 Quelltext anzeigen in Opera

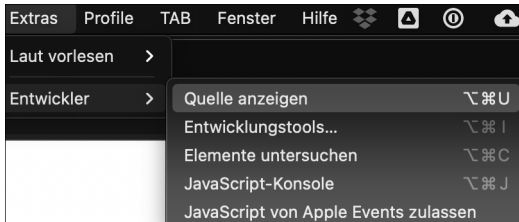


Abbildung 2.22 Quelltext anzeigen in Microsoft Edge

Wenn man sich (egal in welchem Browser) den Quelltext einer Webseite anzeigen lässt, befindet man sich natürlich erst mal im entsprechenden HTML-Code der Webseite. Praktischerweise sind eingebundene Dateien wie beispielsweise CSS-Dateien oder JavaScript-Dateien aber in dieser Quelltextansicht verlinkt (siehe Abbildung 2.23), sodass Sie hierüber bequem auch zum Quelltext der verlinkten Datei gelangen (siehe Abbildung 2.24).

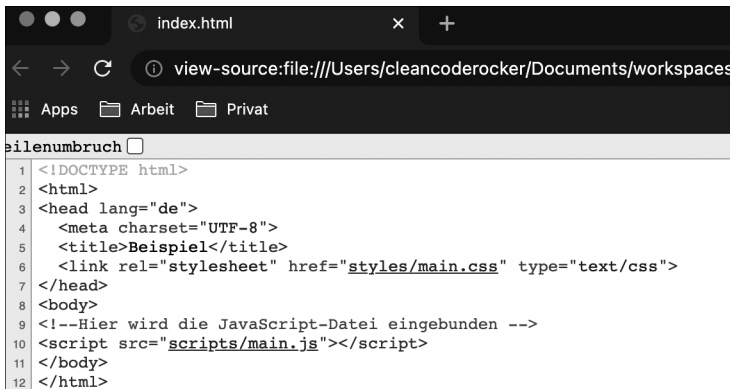


Abbildung 2.23 Quelltextansicht für HTML in Chrome

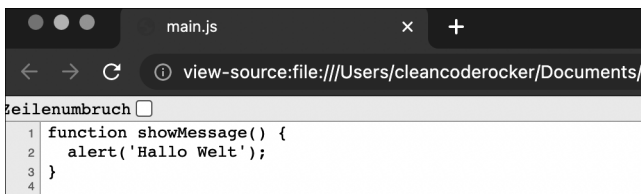


Abbildung 2.24 Quelltextansicht für JavaScript in Chrome

2.3 Eine Ausgabe erzeugen

Im *Hello World*-Beispiel haben Sie ja bereits gesehen, wie Sie über einen Aufruf der Funktion `alert()` eine einfache Ausgabe erzeugen können. Es gibt aber noch verschiedene andere Möglichkeiten.

2.3.1 Standarddialogfenster anzeigen

Neben dem bereits bekannten Hinweisdialog über den Aufruf der Funktion `alert()` (siehe Abbildung 2.25) gibt es noch zwei weitere im Sprachumfang von JavaScript enthaltene Standardfunktionen zur Darstellung von Dialogfenstern. Die erste ist die Funktion `confirm()`. Sie dient der Darstellung von *Bestätigungsdialogen*, sprich Ja/Nein-Entscheidungen (siehe Abbildung 2.27). Im Gegensatz zum Hinweisdialog verfügt der Bestätigungsdialog über zwei Schaltflächen: eine zum Bestätigen der entsprechenden Meldung, eine zum Abbrechen. Die zweite ist die Funktion `prompt()`. Diese öffnet einen *Eingabedialog*, in dem Nutzer einen Text eingeben können (siehe Abbildung 2.26).

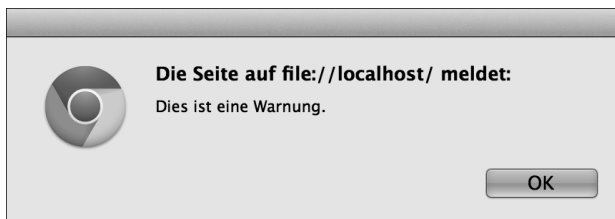


Abbildung 2.25 Ein einfacher Hinweisdialog



Abbildung 2.26 Ein einfacher Eingabedialog



Abbildung 2.27 Ein einfacher Bestätigungsdialog

In der Praxis kommen diese Standarddialoge für Hinweise, Bestätigungen und Eingaben jedoch eher selten zum Einsatz, da sie zum einen in den Ausdrucksmöglichkeiten begrenzt sind, zum anderen aber auch – wie Sie beim Hinweisdialog bereits gesehen haben – optisch dem Layout des jeweiligen Browsers entsprechen und in der Regel nicht zum Layout der Webseite passen. Darüber hinaus haben sie die unschöne Eigenschaft, dass sie die Ausführung des JavaScript-Codes so lange anhalten, bis sie durch den Nutzer weggeklickt werden.

Aus diesem Grund greift man als Webentwickler gerne auf eine der diversen JavaScript-Bibliotheken zurück, die schickere und funktionalere Dialoge anbieten (siehe Abbildung 2.28). Eine dieser Bibliotheken ist jQuery UI, die auf der bekannten Bibliothek jQuery aufbaut und diese um verschiedene UI-Komponenten erweitert. Die Hauptbibliothek jQuery sowie jQuery UI nehmen wir in Kapitel 10, »Aufgaben vereinfachen mit jQuery«, genauer unter die Lupe.

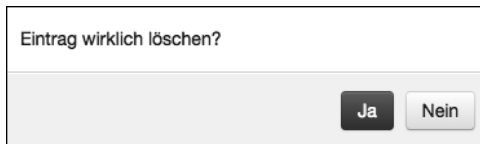


Abbildung 2.28 Ein individueller Bestätigungsdialog mit JavaScript

2.3.2 Auf die Konsole schreiben

Häufig ist es so, dass Sie bei der Entwicklung von JavaScript-Anwendungen eine Ausgabe nur zu Testzwecken für sich selbst erzeugen möchten, beispielsweise um ein Zwischenergebnis auszugeben. Für solche nur zu Testzwecken gedachten Ausgaben ergibt es natürlich keinen Sinn, diese in Dialogen zu zeigen, die auch Nutzer zu Gesicht bekommen würden. Aus diesem Grund bieten mittlerweile alle aktuellen Browser eine sogenannte *Konsole* an, die für genau solche Zwecke geeignet ist und auf die Sie innerhalb eines JavaScript-Programms zugreifen können, um Meldungen auszugeben. Standardmäßig ist diese Konsole ausgeblendet, da Nutzer einer Webseite in der Regel wenig damit anfangen können.

Die Konsole anzeigen

Um die Konsole zu aktivieren, gehen Sie je nach Browser wie folgt vor (auf Screenshots habe ich an dieser Stelle verzichtet, weil die Menüpunkte jeweils an ähnlicher Stelle zu finden sind wie weiter oben in diesem Kapitel die Menüpunkte, um den Quelltext anzuzeigen):

- ▶ Unter Chrome wählen Sie ANZEIGEN • ENTWICKLER • JAVASCRIPT-KONSOLE.
- ▶ In Firefox öffnen Sie die Konsole über EXTRAS • BROWSER-WERKZEUGE • BROWSER-KONSOLE.
- ▶ Unter Safari öffnen Sie die Konsole über ENTWICKLER • JAVASCRIPT-KONSOLE EINBLENDEN.

- ▶ In Opera müssen Sie zunächst ENTWICKLER • ENTWICKLERWERKZEUGE auswählen und anschließend den Tab CONSOLE.
- ▶ In Microsoft Edge öffnen Sie die Konsole über EXTRAS • ENTWICKLER • JAVASCRIPT-KONSOLE.

Abbildung 2.29 zeigt exemplarisch für den Browser Chrome, wie die Konsole aussieht: Wie Sie sehen, nichts wirklich Besonderes, allerdings wird dies eines Ihrer Hauptwerkzeuge sein, wenn Sie JavaScript für die Webentwicklung einsetzen möchten. Neben Ausgaben können Sie über die Konsole nämlich auch Eingaben tätigen (dazu in wenigen Momenten mehr). Mehr oder weniger handelt es sich bei der Konsole so gesehen um eine Art Terminal (oder Eingabeaufforderung, wenn Sie Windows-Nutzer sind), über das Sie JavaScript-Befehle absetzen können, die dann im Kontext der jeweils geladenen Webseite ausgeführt werden.

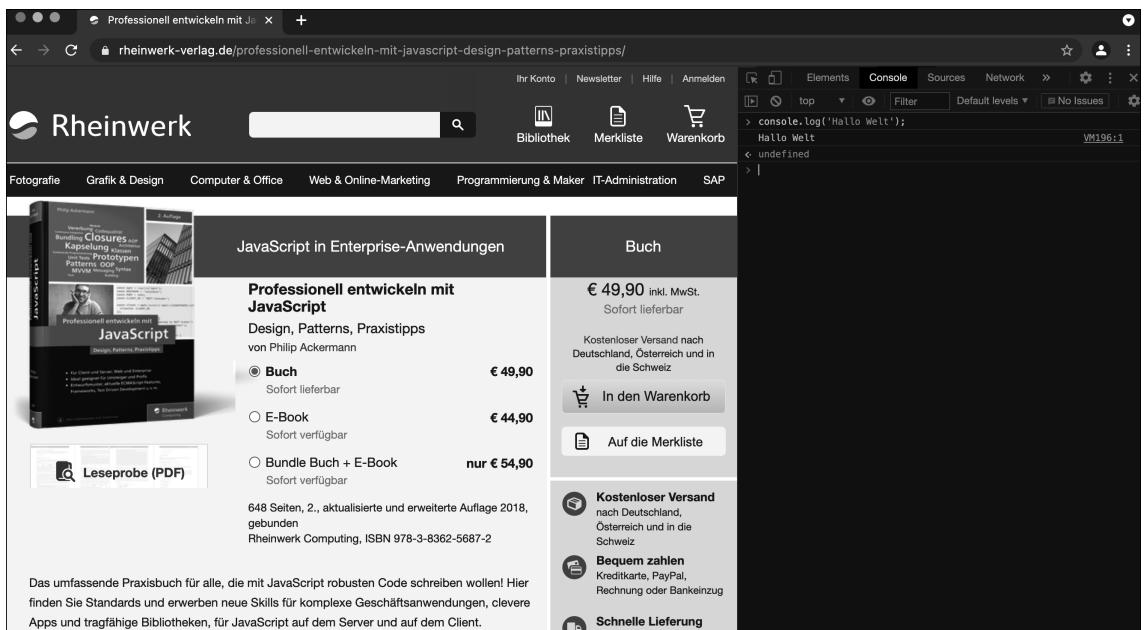


Abbildung 2.29 Standardmäßig wird die Konsole am rechten oder am unteren Rand des Browserfensters angezeigt (hier Google Chrome).

Ausgaben auf die Konsole schreiben

Um auf die Konsole schreiben zu können, stellen Browser das `console`-Objekt zur Verfügung. Dabei handelt es sich um ein JavaScript-Objekt, das erstmals durch das Firefox-Plug-in Firebug (<https://getfirebug.com>) eingeführt wurde und verschiedene Möglichkeiten bietet, Ausgaben auf der Konsole zu erzeugen. Mittlerweile steht das `console`-Objekt in nahezu jeder JavaScript-Laufzeitumgebung zur Verfügung und ist mittlerweile durch die WHATWG, der *Web Hypertext Application Technology Working Group*, standardisiert (<https://console.spec.whatwg.org>).

Um eine einfache Konsolenausgabe zu erzeugen, steht die Methode `log()` zur Verfügung, der Sie einfach die entsprechende Meldung als String übergeben. Um die Verwendung der Konsole auszuprobieren, ersetzen Sie den Quelltext der Datei `main.js` einfach mit folgendem Quelltext und rufen die Webseite erneut auf.

```
// scripts/main.js
function showMessage() {
  console.log('Hallo Entwicklerwelt');
}
```

Listing 2.8 Ein einfaches JavaScript-Beispiel

Das Ergebnis sollte – je nach Browser – wie in folgender Abbildung aussehen:

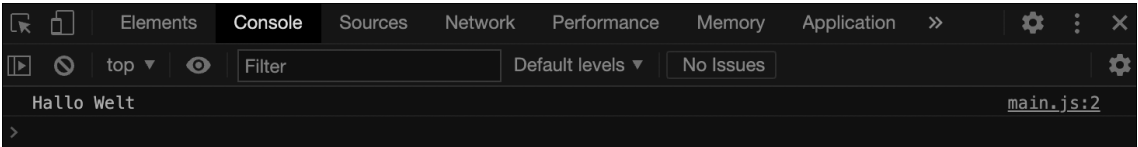


Abbildung 2.30 Ausgabe auf der Konsole in Chrome

Neben der Methode `log()` bietet `console` noch einige weitere Methoden an, von denen Tabelle 2.2 eine Übersicht der wichtigsten zeigt.

Methode	Beschreibung
<code>clear()</code>	Leert die Konsole.
<code>debug()</code>	Dient dazu, eine für das <i>Debuggen</i> (sprich die <i>Fehlerbehebung</i>) gedachte Meldung auszugeben (eventuell müssen Sie in den entsprechenden Entwicklertools erst einstellen, dass diese Art der Ausgaben ausgegeben werden sollen).
<code>error()</code>	Dient dazu, eine Fehlermeldung auszugeben. In manchen Browsern wird innerhalb der Konsole ein Fehlersymbol neben der ausgegebenen Meldung dargestellt.
<code>info()</code>	Hierdurch wird eine Infomeldung auf der Konsole ausgegeben. Chrome beispielsweise gibt zusätzlich ein Infosymbol mit aus.
<code>log()</code>	Die wohl am häufigsten verwendete Methode von <code>console</code> . Erzeugt eine normale Ausgabe auf der Konsole.

Tabelle 2.2 Die wichtigsten Methoden des »console«-Objekts

Methode	Beschreibung
<code>trace()</code>	Gibt den sogenannten <i>Stack-Trace</i> , also den <i>Methodenaufruf-Stack</i> (siehe auch Kapitel 3, »Sprachkern«), auf der Konsole aus.
<code>warn()</code>	Dient dazu, eine Warnung auf der Konsole auszugeben. Auch hier wird in den meisten Browsern ein entsprechendes Symbol neben der Meldung ausgegeben.

Tabelle 2.2 Die wichtigsten Methoden des »console«-Objekts (Forts.)

Listing 2.9 zeigt den entsprechenden Quelltext für die Verwendung des `console`-Objekts. Die Ausgaben für die einzelnen Methoden werden je nach Browser farblich oder durch Symbole hervorgehoben (siehe Abbildung 2.31).

```
console.log('Hallo Entwicklerwelt');    // Ausgabe einer normalen Meldung
console.debug('Hallo Entwicklerwelt');  // Ausgabe einer Debug-Meldung
console.error('Hallo Entwicklerwelt');  // Ausgabe einer Fehlermeldung
console.info('Hallo Entwicklerwelt');   // Ausgabe einer Infomeldung
console.warn('Hallo Entwicklerwelt');   // Ausgabe einer Warnung
```

Listing 2.9 Verwendung des »console«-Objekts

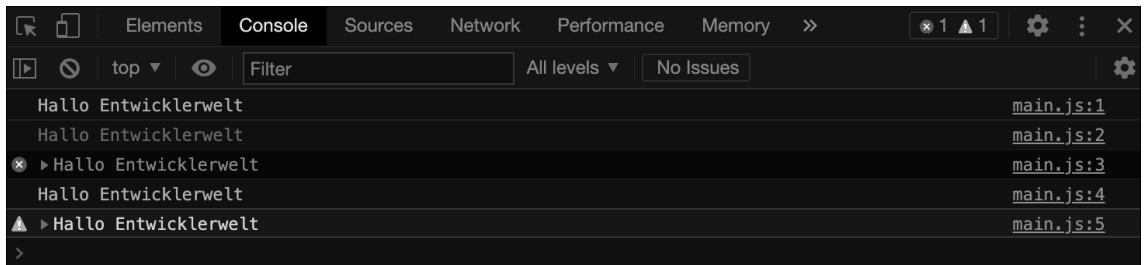


Abbildung 2.31 Die verschiedenen Meldungstypen werden farblich oder durch Symbole hervorgehoben.

Eingaben auf der Konsole schreiben

Wenn Sie sich die Screenshots genauer ansehen, werden Sie vielleicht unterhalb der Ausgabe das `>`-Zeichen bemerkt haben. An dieser Stelle können Sie beliebigen JavaScript-Code eingeben und direkt ausführen lassen. Ein prima Weg, um schnell einfache Skripte auszuprobieren, und für die Webentwicklung eigentlich unersetzlich. Probieren Sie es aus: Schreiben Sie den Befehl `showMessage()` in die Eingabe und drücken Sie anschließend die `[↵]`-Taste, um den Befehl auszuführen. Das Ergebnis sehen Sie in Abbildung 2.32.

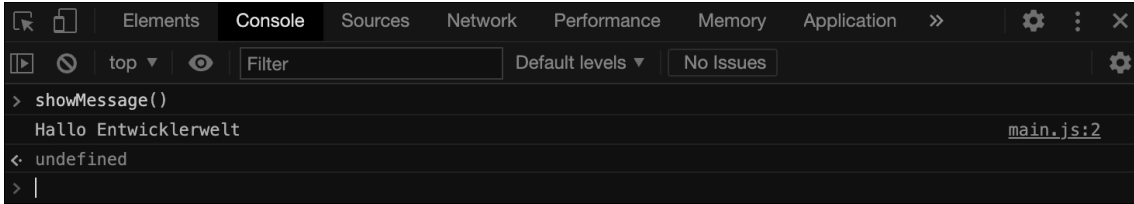


Abbildung 2.32 Über die Konsole können Sie auch Quelltext ausführen.

Merke

Das Konsolenfenster und das `console`-Objekt sind wichtige Werkzeuge für Webentwickler. Machen Sie sich mit beidem bei Gelegenheit gut vertraut.

Logging-Bibliotheken

Das `console`-Objekt eignet sich gut für die schnelle Ausgabe während der Entwicklung. Wird eine Webseite dagegen live geschaltet bzw. eine JavaScript-Anwendung produktiv eingesetzt, will man eigentlich keine `console`-Aufrufe mehr haben (auch wenn der Nutzer sie in der Regel ohnehin nicht sehen würde). In der Praxis verwendet man daher häufig spezielle *Logging-Bibliotheken*, mit deren Hilfe sich Konsolenausgaben über bestimmte Konfigurationseinstellungen einschalten (für die Entwicklung), aber auch wieder abschalten lassen (für den Produktiveinsatz). Für den Anfang und auch die Beispiele in diesem Buch soll uns aber die Verwendung des `console`-Objekts ausreichen.

2.3.3 Bestehende UI-Komponenten verwenden

Während der Einsatz von `alert()`, `confirm()` und `prompt()` eher veraltet und nur zum schnellen Testen sinnvoll und die Ausgabe über das `console`-Objekt ohnehin nur Entwicklern vorbehalten ist, braucht man natürlich noch einen Weg, um eine ansprechende Ausgabe für den Nutzer einer Webseite zu erzeugen. Dazu können Sie die Ausgabe eines Programms in bestehende UI-Komponenten wie Textfelder etc. hineinschreiben.

Listing 2.10, Listing 2.11 und Abbildung 2.33 zeigen hierzu ein Beispiel. Sie sehen hier ein einfaches Formular, über das sich das Ergebnis der Addition zweier Zahlen ermitteln lässt. Die beiden Zahlen können dabei in zwei Textfelder eingegeben werden, die Addition wird durch Betätigen der Schaltfläche ausgelöst und das Ergebnis in das dritte Textfeld hineingeschrieben.

Den Code für dieses Beispiel müssen Sie jetzt noch nicht verstehen, und ich gehe an dieser Stelle auch noch gar nicht auf die Details ein. Für den Moment müssen Sie sich nur merken, dass man bei der Webentwicklung mit JavaScript relativ häufig auf HTML-Komponenten zurückgreifen muss, um Ausgaben eines Programms an den Nutzer zu »senden«.

```
// scripts/main.js
function calculateSum() {
  const x = parseInt(document.getElementById('field1').value);
  const y = parseInt(document.getElementById('field2').value);
  const result = document.getElementById('result');
  console.log(x + y);
  result.value = x + y;
}
```

Listing 2.10 Der JavaScript-Code der Datei »main.js«

```
<!DOCTYPE html>
<html>
<head lang="de">
  <meta charset="UTF-8">
  <title>Beispiel</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css">
</head>
<body>
<div class="container">
  <div class="row">
    <label for="field1">X</label> <input id="field1" type="text" value="5">
  </div>
  <div class="row">
    <label for="field2">Y</label> <input id="field2" type="text" value="5">
  </div>
  <div class="row">
    <label for="result">Ergebnis: </label> <input id="result" type="text">
    <button onclick="calculateSum()">Summe berechnen</button>
  </div>
</div>
<script src="scripts/main.js"></script>
</body>
</html>
```

Listing 2.11 Der HTML-Code für die Beispielanwendung

X	5
Y	5
Ergebnis:	10 Summe berechnen

Abbildung 2.33 Beispielanwendung

DOM-Manipulation

Am komplexesten wird es, wenn Sie eine Webseite dynamisch ändern, um eine Ausgabe zu erzeugen, beispielsweise dynamisch eine Tabelle, um tabellarisch strukturierte Daten darzustellen. Dieses Thema der sogenannten DOM-Manipulation werden wir noch detailliert in Kapitel 5, »Webseiten dynamisch verändern«, besprechen.

2.4 Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie Sie JavaScript-Dateien erzeugen und in HTML einbinden. Sie besitzen nun die Grundlage dafür, die Beispiele aus den nächsten Kapiteln ausführen zu können. Die wichtigsten Punkte aus diesem Kapitel sind:

- ▶ Für die Frontend-Entwicklung sind drei Sprachen wichtig: HTML als *Auszeichnungssprache*, um die Struktur einer Webseite festzulegen, CSS als *Stilsprache*, um Design und Layout zu definieren, und JavaScript als *Programmiersprache*, um einer Webseite zusätzliches Verhalten und Interaktivität hinzuzufügen.
- ▶ Sie können JavaScript entweder direkt innerhalb des `<script>`-Elements angeben oder über das `src`-Attribut des `<script>`-Elements eine separate JavaScript-Datei einbinden. Ich empfehle Ihnen Letzteres, da so eine saubere Trennung zwischen Struktur (HTML) und Verhalten (JavaScript) der Webseite sichergestellt ist.
- ▶ Sie sollten `<script>`-Elemente immer vor dem schließenden `</body>`-Tag platzieren, da so sichergestellt ist, dass der Inhalt der Webseite vollständig geladen ist.
- ▶ JavaScript bietet von Haus aus drei Funktionen für das Erzeugen einer Ausgabe: `alert()` für das Erstellen von Hinweisdialogen, `confirm()` für das Erzeugen von Bestätigungsdialogen und `prompt()` für das Erzeugen von Eingabedialogen.
- ▶ In der Praxis verwendet man aber statt dieser (mehr oder weniger veralteten) Funktionen schickere Dialoge, wie sie beispielsweise die Bibliothek jQuery anbietet.
- ▶ Darüber hinaus bieten alle aktuellen Browser über eine Konsole die Möglichkeit, Ausgaben zu erzeugen, die eher für Sie als Entwickler gedacht sind.

Auf einen Blick

1	Grundlagen und Einführung	29
2	Erste Schritte	55
3	Sprachkern	85
4	Mit Referenztypen arbeiten	243
5	Webseiten dynamisch verändern	381
6	Ereignisse verarbeiten und auslösen	439
7	Mit Formularen arbeiten	491
8	Browser steuern und Browserinformationen auslesen	521
9	Inhalte einer Webseite dynamisch nachladen	549
10	Aufgaben vereinfachen mit jQuery	595
11	Bilder und Grafiken dynamisch erstellen	641
12	Moderne Web-APIs verwenden	673
13	Objektorientierte Programmierung	773
14	Funktionale Programmierung	817
15	Den Quelltext richtig strukturieren	831
16	Asynchrone Programmierung und weitere fortgeschrittene Features verwenden	853
17	Serverseitige Anwendungen mit Node.js erstellen	887
18	Mobile Anwendungen mit JavaScript erstellen	935
19	KI-Anwendungen mit JavaScript	959
20	Einen professionellen Entwicklungsprozess aufsetzen	1001

Inhalt

Materialien zum Buch	23
Vorwort	25

1 Grundlagen und Einführung 29

1.1 Grundlagen der Programmierung	29
1.1.1 Mit dem Computer kommunizieren	30
1.1.2 Programmiersprachen	31
1.1.3 Hilfsmittel für den Programmentwurf	39
1.2 Einführung in JavaScript	45
1.2.1 Historie	45
1.2.2 Anwendungsgebiete	47
1.3 Zusammenfassung	53

2 Erste Schritte 55

2.1 Einführung in JavaScript und die Webentwicklung	55
2.1.1 Der Zusammenhang zwischen HTML, CSS und JavaScript	55
2.1.2 Das richtige Werkzeug für die Entwicklung	59
2.2 JavaScript in eine Webseite einbinden	63
2.2.1 Eine geeignete Ordnerstruktur vorbereiten	64
2.2.2 Eine JavaScript-Datei erstellen	64
2.2.3 Eine JavaScript-Datei in eine HTML-Datei einbinden	65
2.2.4 JavaScript direkt innerhalb des HTML definieren	68
2.2.5 Platzierung und Ausführung der <script>-Elemente	70
2.2.6 Den Quelltext anzeigen	74
2.3 Eine Ausgabe erzeugen	77
2.3.1 Standarddialogfenster anzeigen	77
2.3.2 Auf die Konsole schreiben	78
2.3.3 Bestehende UI-Komponenten verwenden	82
2.4 Zusammenfassung	84

3	Sprachkern	85
3.1	Werte in Variablen speichern	85
3.1.1	Variablen definieren	85
3.1.2	Gültige Variablennamen verwenden	88
3.1.3	Konstanten definieren	96
3.2	Die verschiedenen Datentypen verwenden	96
3.2.1	Zahlen	97
3.2.2	Zeichenketten	100
3.2.3	Boolesche Werte	106
3.2.4	Arrays	106
3.2.5	Objekte	112
3.2.6	Besondere Datentypen	114
3.2.7	Symbole	115
3.3	Die verschiedenen Operatoren einsetzen	116
3.3.1	Operatoren für das Arbeiten mit Zahlen	117
3.3.2	Operatoren für das einfachere Zuweisen	118
3.3.3	Operatoren für das Arbeiten mit Zeichenketten	120
3.3.4	Operatoren für das Arbeiten mit booleschen Werten	121
3.3.5	Operatoren für das Arbeiten mit Bits	128
3.3.6	Operatoren für das Vergleichen von Werten	129
3.3.7	Der Optional Chaining Operator	132
3.3.8	Die Logical Assignment Operatoren	134
3.3.9	Operatoren für spezielle Operationen	136
3.4	Den Ablauf eines Programms steuern	137
3.4.1	Bedingte Anweisungen definieren	137
3.4.2	Verzweigungen definieren	139
3.4.3	Den Auswahloperator verwenden	146
3.4.4	Mehrfachverzweigungen definieren	147
3.4.5	Zählschleifen definieren	154
3.4.6	Kopfgesteuerte Schleifen definieren	162
3.4.7	Fußgesteuerte Schleifen definieren	165
3.4.8	Schleifen und Schleifeniterationen vorzeitig abbrechen	167
3.5	Wiederverwendbare Codebausteine erstellen	176
3.5.1	Funktionen definieren	176
3.5.2	Funktionen aufrufen	180
3.5.3	Funktionsparameter übergeben und auswerten	180
3.5.4	Rückgabewerte definieren	189
3.5.5	Standardwerte für Parameter definieren	191

3.5.6	Elemente aus einem Array als Parameter verwenden	193
3.5.7	Funktionen über Kurzschreibweise definieren	195
3.5.8	Zeichenketten über Funktionen verändern	197
3.5.9	Funktionen im Detail	199
3.5.10	Funktionen aufrufen durch Nutzerinteraktion	207
3.6	Auf Fehler reagieren und sie richtig behandeln	208
3.6.1	Syntaxfehler	209
3.6.2	Laufzeitfehler	209
3.6.3	Logische Fehler	210
3.6.4	Das Prinzip der Fehlerbehandlung	211
3.6.5	Fehler fangen und behandeln	213
3.6.6	Fehler auslösen	215
3.6.7	Fehler und der Funktionsaufruf-Stack	220
3.6.8	Bestimmte Anweisungen unabhängig von aufgetretenen Fehlern aufrufen	222
3.7	Den Quelltext kommentieren	228
3.8	Den Code debuggen	229
3.8.1	Einführung	229
3.8.2	Ein einfaches Codebeispiel	229
3.8.3	Haltepunkte definieren	230
3.8.4	Variablenbelegungen einsehen	233
3.8.5	Ein Programm schrittweise ausführen	233
3.8.6	Mehrere Haltepunkte definieren	236
3.8.7	Die verschiedenen Typen von Haltepunkten verwenden	237
3.8.8	Den Funktionsaufruf-Stack einsehen	237
3.9	Zusammenfassung	239
4	Mit Referenztypen arbeiten	243

4.1	Unterschied zwischen primitiven Datentypen und Referenztypen	243
4.1.1	Das Prinzip von primitiven Datentypen	243
4.1.2	Das Prinzip von Referenztypen	244
4.1.3	Primitive Datentypen und Referenztypen als Funktionsargumente	246
4.1.4	Den Typ einer Variablen ermitteln	247
4.1.5	Ausblick	250
4.2	Zustand und Verhalten in Objekten kapseln	251
4.2.1	Einführung objektorientierte Programmierung	251
4.2.2	Objekte erstellen über die Literal-Schreibweise	252

4.2.3	Objekte erstellen über Konstruktorfunktionen	254
4.2.4	Objekte erstellen unter Verwendung von Klassen	257
4.2.5	Objekte erstellen über die Funktion »Object.create()«	261
4.2.6	Auf Eigenschaften zugreifen und Methoden aufrufen	265
4.2.7	Objekteigenschaften und Objektmethoden hinzufügen oder überschreiben	271
4.2.8	Objekteigenschaften und Objektmethoden löschen	276
4.2.9	Objekteigenschaften und Objektmethoden ausgeben	278
4.2.10	Symbole zur Definition eindeutiger Objekteigenschaften verwenden	282
4.2.11	Änderungen an Objekten verhindern	283
4.3	Mit Arrays arbeiten	286
4.3.1	Arrays erzeugen und initialisieren	287
4.3.2	Auf Elemente eines Arrays zugreifen	290
4.3.3	Elemente einem Array hinzufügen	292
4.3.4	Elemente aus einem Array entfernen	296
4.3.5	Einen Teil der Elemente aus einem Array kopieren	299
4.3.6	Arrays sortieren	302
4.3.7	Arrays als Stack verwenden	305
4.3.8	Arrays als Queue verwenden	307
4.3.9	Elemente in Arrays finden	308
4.3.10	Elemente innerhalb eines Arrays kopieren	311
4.3.11	Arrays in Zeichenketten umwandeln	312
4.4	Werte aus Arrays und Objekten extrahieren	313
4.4.1	Werte aus Arrays extrahieren	313
4.4.2	Werte aus Objekten extrahieren	317
4.4.3	Werte innerhalb einer Schleife extrahieren	320
4.4.4	Argumente einer Funktion extrahieren	322
4.4.5	Objekteigenschaften in ein anderes Objekt kopieren	323
4.4.6	Objekteigenschaften aus einem anderen Objekt kopieren	324
4.5	Mit Zeichenketten arbeiten	325
4.5.1	Der Aufbau einer Zeichenkette	326
4.5.2	Die Länge einer Zeichenkette ermitteln	326
4.5.3	Innerhalb einer Zeichenkette suchen	327
4.5.4	Teile einer Zeichenkette extrahieren	330
4.6	Maps verwenden	333
4.6.1	Maps erstellen	333
4.6.2	Grundlegende Operationen	334
4.6.3	Über Maps iterieren	336
4.6.4	Weak Maps verwenden	338

4.7	Sets verwenden	340
4.7.1	Sets erstellen	340
4.7.2	Grundlegende Operationen von Sets	341
4.7.3	Über Sets iterieren	343
4.7.4	Weak Sets verwenden	344
4.8	Sonstige globale Objekte	345
4.8.1	Mit Datum und Zeit arbeiten	345
4.8.2	Komplexe Berechnungen durchführen	348
4.8.3	Wrapperobjekte für primitive Datentypen	348
4.9	Mit regulären Ausdrücken arbeiten	349
4.9.1	Reguläre Ausdrücke definieren	349
4.9.2	Zeichenketten gegen einen regulären Ausdruck testen	350
4.9.3	Zeichenklassen verwenden	353
4.9.4	Anfang und Ende begrenzen	356
4.9.5	Quantifizierer verwenden	359
4.9.6	Nach Vorkommen suchen	364
4.9.7	Alle Vorkommen innerhalb einer Zeichenkette suchen	365
4.9.8	Auf einzelne Teile eines Vorkommens zugreifen	366
4.9.9	Nach bestimmten Zeichenketten suchen	368
4.9.10	Vorkommen innerhalb einer Zeichenkette ersetzen	368
4.9.11	Nach Vorkommen suchen	369
4.9.12	Zeichenketten zerteilen	369
4.10	Funktionen als Referenztypen	370
4.10.1	Funktionen als Argumente verwenden	370
4.10.2	Funktionen als Rückgabewert verwenden	373
4.10.3	Standardmethoden jeder Funktion	375
4.11	Zusammenfassung	379
5	Webseiten dynamisch verändern	381
5.1	Aufbau einer Webseite	381
5.1.1	Document Object Model	381
5.1.2	Die verschiedenen Knotentypen	382
5.1.3	Der Dokumentknoten	386
5.2	Elemente selektieren	387
5.2.1	Elemente per ID selektieren	390
5.2.2	Elemente per Klasse selektieren	392
5.2.3	Elemente nach Elementnamen selektieren	396

5.2.4	Elemente nach Namen selektieren	397
5.2.5	Elemente per Selektor selektieren	399
5.2.6	Das Elternelement eines Elements selektieren	405
5.2.7	Die Kindelemente eines Elements selektieren	408
5.2.8	Die Geschwisterelemente eines Elements selektieren	412
5.2.9	Selektionsmethoden auf Elementen aufrufen	415
5.2.10	Elemente nach Typ selektieren	417
5.3	Mit Textknoten arbeiten	417
5.3.1	Auf den Textinhalt eines Elements zugreifen	418
5.3.2	Den Textinhalt eines Elements verändern	419
5.3.3	Das HTML unterhalb eines Elements verändern	420
5.3.4	Textknoten erstellen und hinzufügen	421
5.4	Mit Elementen arbeiten	421
5.4.1	Elemente erstellen und hinzufügen	422
5.4.2	Elemente und Knoten entfernen	425
5.4.3	Die verschiedenen Typen von HTML-Elementen	426
5.5	Mit Attributen arbeiten	431
5.5.1	Den Wert eines Attributs auslesen	432
5.5.2	Den Wert eines Attributs ändern oder ein neues Attribut hinzufügen	434
5.5.3	Attributknoten erstellen und hinzufügen	435
5.5.4	Attribute entfernen	435
5.5.5	Auf CSS-Klassen zugreifen	435
5.6	Zusammenfassung	437
6	Ereignisse verarbeiten und auslösen	439
6.1	Das Konzept der ereignisgesteuerten Programmierung	439
6.2	Auf Ereignisse reagieren	440
6.2.1	Einen Event-Handler per HTML definieren	443
6.2.2	Einen Event-Handler per JavaScript definieren	445
6.2.3	Event-Listener definieren	447
6.2.4	Mehrere Event-Listener definieren	449
6.2.5	Argumente an Event-Listener übergeben	451
6.2.6	Event-Listener entfernen	453

6.2.7	Event-Handler und Event-Listener per Helferfunktion definieren	454
6.2.8	Auf Informationen eines Ereignisses zugreifen	455
6.3	Die verschiedenen Typen von Ereignissen	457
6.3.1	Ereignisse bei Interaktion mit der Maus	458
6.3.2	Ereignisse bei Interaktion mit der Tastatur	463
6.3.3	Ereignisse beim Arbeiten mit Formularen	465
6.3.4	Ereignisse bei Fokussieren von Elementen	466
6.3.5	Allgemeine Ereignisse der Nutzerschnittstelle	467
6.3.6	Ereignisse bei mobilen Endgeräten	470
6.4	Den Ereignisfluss verstehen und beeinflussen	471
6.4.1	Die Event-Phasen	471
6.4.2	Den Ereignisfluss unterbrechen	478
6.4.3	Standardaktionen von Events verhindern	484
6.5	Ereignisse programmatisch auslösen	486
6.5.1	Einfache Ereignisse auslösen	486
6.5.2	Ereignisse mit übergebenen Argumenten auslösen	487
6.5.3	Standardereignisse auslösen	488
6.6	Zusammenfassung	489
7	Mit Formularen arbeiten	491

7.1	Auf Formulare und Formularfelder zugreifen	492
7.1.1	Auf Formulare zugreifen	492
7.1.2	Auf Formularelemente zugreifen	495
7.1.3	Den Wert von Textfeldern und Passwortfeldern auslesen	497
7.1.4	Den Wert von Checkboxes auslesen	499
7.1.5	Den Wert von Radiobuttons auslesen	499
7.1.6	Den Wert von Auswahllisten auslesen	501
7.1.7	Die Werte von Mehrfachauswahllisten auslesen	503
7.1.8	Auswahllisten per JavaScript mit Werten befüllen	504
7.2	Formulare programmatisch abschicken und zurücksetzen	505
7.3	Formulareingaben validieren	508
7.4	Zusammenfassung	519

8 Browser steuern und Browserinformationen auslesen 521

8.1	Das Browser Object Model	521
8.2	Auf Fensterinformationen zugreifen	523
8.2.1	Die Größe und Position eines Browserfensters ermitteln	523
8.2.2	Die Größe und Position eines Browserfensters ändern	525
8.2.3	Auf Anzeigeinformationen der Browserleisten zugreifen	527
8.2.4	Allgemeine Eigenschaften ermitteln	528
8.2.5	Neue Browserfenster öffnen	529
8.2.6	Browserfenster schließen	531
8.2.7	Dialoge öffnen	532
8.2.8	Funktionen zeitgesteuert ausführen	533
8.3	Auf Navigationsinformationen der aktuellen Webseite zugreifen	535
8.3.1	Auf die einzelnen Bestandteile der URL zugreifen	535
8.3.2	Auf Querystring-Parameter zugreifen	536
8.3.3	Eine neue Webseite laden	536
8.4	Den Browserverlauf einsehen und verändern	538
8.4.1	Im Browserverlauf navigieren	538
8.4.2	Browserverlauf bei Single Page Applications	539
8.4.3	Einträge in den Browserverlauf hinzufügen	540
8.4.4	Auf Änderungen im Browserverlauf reagieren	543
8.4.5	Den aktuellen Eintrag im Browserverlauf ersetzen	543
8.5	Browser erkennen und Browserfeatures bestimmen	544
8.6	Auf Informationen des Bildschirms zugreifen	547
8.7	Zusammenfassung	548

9 Inhalte einer Webseite dynamisch nachladen 549

9.1	Das Prinzip von Ajax	549
9.1.1	Synchrone Kommunikation	549
9.1.2	Asynchrone Kommunikation	550
9.1.3	Typische Anwendungsfälle für die Verwendung von Ajax	552
9.1.4	Verwendete Datenformate	555
9.2	Das XML-Format	555
9.2.1	Der Aufbau von XML	555
9.2.2	XML und die DOM API	557

9.2.3	Zeichenketten in XML-Objekte umwandeln	558
9.2.4	XML-Objekte in Zeichenketten umwandeln	559
9.3	Das JSON-Format	561
9.3.1	Der Aufbau von JSON	561
9.3.2	Unterschied zwischen JSON und JavaScript-Objekten	563
9.3.3	Objekte in das JSON-Format umwandeln	564
9.3.4	Objekte aus dem JSON-Format umwandeln	565
9.4	Anfragen per Ajax stellen	566
9.4.1	Das »XMLHttpRequest«-Objekt	566
9.4.2	HTML-Daten per Ajax laden	573
9.4.3	XML-Daten per Ajax laden	577
9.4.4	JSON-Daten per Ajax laden	581
9.4.5	Daten per Ajax an den Server schicken	584
9.4.6	Formulare per Ajax abschicken	585
9.4.7	Daten von anderen Domains laden	586
9.4.8	Die neuere Alternative zu »XMLHttpRequest«: die Fetch API	589
9.5	Zusammenfassung	593

10 Aufgaben vereinfachen mit jQuery 595

10.1	Einführung	595
10.1.1	jQuery einbinden	596
10.1.2	jQuery über ein CDN einbinden	597
10.1.3	jQuery verwenden	598
10.1.4	Aufgaben mit jQuery vereinfachen	599
10.2	Mit dem DOM arbeiten	601
10.2.1	Elemente selektieren	601
10.2.2	Auf Inhalte zugreifen und diese verändern	606
10.2.3	Ausgewählte Elemente filtern	610
10.2.4	Auf Attribute zugreifen	612
10.2.5	Auf CSS-Eigenschaften zugreifen	613
10.2.6	Zwischen Elementen navigieren	614
10.2.7	Effekte und Animationen verwenden	616
10.3	Auf Ereignisse reagieren	618
10.3.1	Event-Listener registrieren	618
10.3.2	Auf allgemeine Ereignisse reagieren	619
10.3.3	Auf Mausereignisse reagieren	620

10.3.4	Auf Tastaturereignisse reagieren	621
10.3.5	Auf Formularereignisse reagieren	622
10.3.6	Auf Informationen von Ereignissen zugreifen	623
10.4	Ajax-Anfragen erstellen	625
10.4.1	Ajax-Anfragen erstellen	625
10.4.2	Auf Ereignisse reagieren	628
10.4.3	HTML-Daten per Ajax laden	629
10.4.4	XML-Daten per Ajax laden	631
10.4.5	JSON-Daten per Ajax laden	632
10.5	Zusammenfassung	633

11 Bilder und Grafiken dynamisch erstellen 641

11.1	Bilder zeichnen	641
11.1.1	Die Zeichenfläche	641
11.1.2	Der Rendering-Kontext	642
11.1.3	Rechtecke zeichnen	644
11.1.4	Pfade verwenden	647
11.1.5	Texte zeichnen	653
11.1.6	Farbverläufe zeichnen	654
11.1.7	Speichern und Wiederherstellen des Canvas-Zustands	656
11.1.8	Transformationen anwenden	658
11.1.9	Animationen erstellen	661
11.2	Vektorgrafiken einbinden	663
11.2.1	Das SVG-Format	663
11.2.2	SVG in HTML einbinden	665
11.2.3	Das Aussehen von SVG-Elementen mit CSS beeinflussen	668
11.2.4	Das Verhalten von SVG-Elementen mit JavaScript beeinflussen	669
11.3	Zusammenfassung	671

12 Moderne Web-APIs verwenden 673

12.1	Über JavaScript kommunizieren	675
12.1.1	Unidirektionale Kommunikation mit dem Server	675
12.1.2	Bidirektionale Kommunikation mit einem Server	677
12.1.3	Vom Server ausgehende Kommunikation	679

12.2 Nutzer wiedererkennen	684
12.2.1 Cookies verwenden	684
12.2.2 Cookies anlegen	686
12.2.3 Cookies auslesen	687
12.2.4 Ein Beispiel: Einkaufswagen auf Basis von Cookies	689
12.2.5 Nachteile von Cookies	693
12.3 Den Browserspeicher nutzen	693
12.3.1 Werte im Browserspeicher speichern	694
12.3.2 Werte aus dem Browserspeicher lesen	695
12.3.3 Werte im Browserspeicher aktualisieren	695
12.3.4 Werte aus dem Browserspeicher löschen	696
12.3.5 Auf Änderungen im Browserspeicher reagieren	696
12.3.6 Die verschiedenen Typen von Browserspeichern	697
12.3.7 Ein Beispiel: Einkaufswagen auf Basis des Browserspeichers	699
12.4 Die Browserdatenbank nutzen	700
12.4.1 Öffnen einer Datenbank	701
12.4.2 Erstellen einer Datenbank	702
12.4.3 Erstellen eines Objektspeichers	703
12.4.4 Hinzufügen von Objekten zu einem Objektspeicher	704
12.4.5 Lesen von Objekten aus einem Objektspeicher	708
12.4.6 Löschen von Objekten aus einem Objektspeicher	709
12.4.7 Aktualisieren von Objekten in einem Objektspeicher	710
12.4.8 Verwendung eines Cursors	711
12.5 Auf das Dateisystem zugreifen	712
12.5.1 Auswählen von Dateien per Dateidialog	713
12.5.2 Auswählen von Dateien per Drag and Drop	714
12.5.3 Lesen von Dateien	716
12.5.4 Den Lesefortschritt überwachen	719
12.6 Komponenten einer Webseite mit Drag and Drop verschieben	721
12.6.1 Ereignisse einer Drag-and-Drop-Operation	721
12.6.2 Verschiebbare Elemente definieren	722
12.6.3 Verschieben von Elementen	724
12.7 Aufgaben parallelisieren	726
12.7.1 Das Prinzip von Web Workern	727
12.7.2 Web Worker verwenden	728
12.8 Den Standort von Nutzern ermitteln	730
12.8.1 Auf Standortinformationen zugreifen	730
12.8.2 Kontinuierlich auf Standortinformationen zugreifen	732

12.9 Den Batteriestand eines Endgeräts auslesen	733
12.9.1 Auf Batterieinformationen zugreifen	733
12.9.2 Auf Ereignisse reagieren	734
12.10 Sprache ausgeben und Sprache erkennen	736
12.10.1 Sprache ausgeben	737
12.10.2 Sprache erkennen	739
12.11 Animationen erstellen	741
12.11.1 Verwendung der API	741
12.11.2 Steuern einer Animation	744
12.12 Mit der Kommandozeile arbeiten	745
12.12.1 Auswahl und Inspektion von DOM-Elementen	745
12.12.2 Analyse von Events	747
12.12.3 Debugging, Monitoring und Profiling	750
12.13 Mehrsprachige Anwendungen entwickeln	754
12.13.1 Begriffserklärungen	755
12.13.2 Die Internationalization API	756
12.13.3 Vergleich von Zeichenketten	758
12.13.4 Formatierung von Datums- und Zeitangaben	761
12.13.5 Formatierung von Zahlenwerten	764
12.14 Übersicht über verschiedene Web-APIs	767
12.15 Zusammenfassung	771

13 Objektorientierte Programmierung 773

13.1 Die Prinzipien der objektorientierten Programmierung	773
13.1.1 Klassen, Objektinstanzen und Prototypen	774
13.1.2 Prinzip 1: Abstraktes Verhalten definieren	776
13.1.3 Prinzip 2: Zustand und Verhalten kapseln	777
13.1.4 Prinzip 3: Zustand und Verhalten vererben	778
13.1.5 Prinzip 4: Verschiedene Typen annehmen	779
13.1.6 JavaScript und die Objektorientierung	780
13.2 Prototypische Objektorientierung	780
13.2.1 Das Konzept von Prototypen	781
13.2.2 Von Objekten ableiten	781
13.2.3 Methoden und Eigenschaften vererben	782
13.2.4 Methoden und Eigenschaften im erbenden Objekt definieren	783

13.2.5	Methoden überschreiben	784
13.2.6	Die Prototypenkette	785
13.2.7	Methoden des Prototyps aufrufen	786
13.2.8	Prototypische Objektorientierung und die Prinzipien der Objektorientierung	787
13.3	Pseudoklassische Objektorientierung	787
13.3.1	Konstruktorfunktionen definieren	788
13.3.2	Objektinstanzen erzeugen	788
13.3.3	Methoden definieren	788
13.3.4	Vererbung mit Prototypen	789
13.3.5	Konstruktor der »Oberklasse« aufrufen	793
13.3.6	Methoden überschreiben	793
13.3.7	Methoden der »Oberklasse« aufrufen	793
13.3.8	Pseudoklassische Objektorientierung und die Prinzipien der Objektorientierung	794
13.4	Objektorientierung mit Klassensyntax	794
13.4.1	Klassen definieren	795
13.4.2	Objektinstanzen erzeugen	797
13.4.3	Getter und Setter definieren	797
13.4.4	Private Eigenschaften und private Methoden definieren	799
13.4.5	Von »Klassen« ableiten	803
13.4.6	Methoden überschreiben	807
13.4.7	Methoden der »Oberklasse« aufrufen	809
13.4.8	Statische Methoden definieren	810
13.4.9	Statische Eigenschaften definieren	812
13.4.10	Klassensyntax und die Prinzipien der Objektorientierung	814
13.5	Zusammenfassung	815

14 Funktionale Programmierung 817

14.1	Prinzipien der funktionalen Programmierung	817
14.1.1	Prinzip 1: Funktionen sind Objekte erster Klasse	817
14.1.2	Prinzip 2: Funktionen arbeiten mit unveränderlichen Datenstrukturen	818
14.1.3	Prinzip 3: Funktionen haben keine Nebeneffekte	818
14.1.4	Prinzip 4: Funktionale Programme sind deklarativ	818
14.2	Imperative Programmierung und funktionale Programmierung	819
14.2.1	Iterieren mit der Methode »forEach()«	819
14.2.2	Werte abbilden mit der Methode »map()«	822

14.2.3	Werte filtern mit der Methode »filter()«	823
14.2.4	Mehrere Werte zu einem Wert reduzieren mit der Methode »reduce()«	825
14.2.5	Kombination der verschiedenen Methoden	828
14.3	Zusammenfassung	829

15 Den Quelltext richtig strukturieren 831

15.1	Namenskonflikte vermeiden	831
15.2	Module definieren und verwenden	835
15.2.1	Das Module-Entwurfsmuster	835
15.2.2	Das Revealing-Module-Entwurfsmuster	839
15.2.3	AMD	843
15.2.4	CommonJS	845
15.2.5	Native Module	847
15.3	Zusammenfassung	850

16 Asynchrone Programmierung und weitere fortgeschrittene Features verwenden 853

16.1	Asynchrone Programmierung verstehen und anwenden	853
16.1.1	Das Callback-Entwurfsmuster verwenden	854
16.1.2	Promises verwenden	858
16.1.3	Async Functions verwenden	867
16.2	Das Iterieren über Datenstrukturen kapseln	871
16.2.1	Das Prinzip von Iteratoren	871
16.2.2	Iteratoren verwenden	872
16.2.3	Einen eigenen Iterator erstellen	872
16.2.4	Ein iterierbares Objekt erstellen	874
16.2.5	Über iterierbare Objekte iterieren	875
16.3	Funktionen anhalten und fortsetzen	876
16.3.1	Eine Generatorfunktion erstellen	876
16.3.2	Einen Generator erstellen	876
16.3.3	Über Generatoren iterieren	877
16.3.4	Unendliche Generatoren erstellen	878
16.3.5	Generatoren mit Parametern steuern	878

16.4	Den Zugriff auf Objekte abfangen	879
16.4.1	Das Prinzip von Proxies	880
16.4.2	Proxies erstellen	880
16.4.3	Handler für Proxies definieren	880
16.5	Zusammenfassung	885
17	Serverseitige Anwendungen mit Node.js erstellen	887
17.1	Einführung Node.js	887
17.1.1	Die Architektur von Node.js	887
17.1.2	Installation von Node.js	889
17.1.3	Eine einfache Anwendung	890
17.2	Node.js-Packages verwalten	891
17.2.1	Den Node.js Package Manager installieren	891
17.2.2	Packages installieren	891
17.2.3	Eigene Packages initialisieren	894
17.3	Ereignisse verarbeiten und auslösen	898
17.3.1	Ein Event auslösen und abfangen	898
17.3.2	Ein Event mehrfach auslösen	901
17.3.3	Auf ein Event genau einmal reagieren	901
17.3.4	Mehrere Event-Listener für ein Event registrieren	902
17.4	Auf das Dateisystem zugreifen	903
17.4.1	Dateien lesen	903
17.4.2	Dateien schreiben	904
17.4.3	Dateiinformatoren auslesen	905
17.4.4	Dateien löschen	906
17.4.5	Mit Verzeichnissen arbeiten	906
17.5	Einen Webserver erstellen	908
17.5.1	Einen Webserver starten	908
17.5.2	Dateien per Webserver zur Verfügung stellen	909
17.5.3	Einen Client für einen Webserver erstellen	910
17.5.4	Routen definieren	910
17.5.5	Das Web-Framework Express.js verwenden	911
17.6	Auf Datenbanken zugreifen	916
17.6.1	Installation von MongoDB	917
17.6.2	MongoDB-Treiber für Node.js installieren	917
17.6.3	Verbindung zur Datenbank herstellen	917

17.6.4	Eine Collection erstellen	918
17.6.5	Objekte speichern	919
17.6.6	Objekte lesen	920
17.6.7	Objekte aktualisieren	922
17.6.8	Objekte löschen	923
17.7	Mit Streams arbeiten	924
17.7.1	Einführung und Arten von Streams	925
17.7.2	Anwendungsfälle von Streams	925
17.7.3	Daten mit Streams lesen	926
17.7.4	Daten mit Streams schreiben	928
17.7.5	Streams mithilfe von Piping kombinieren	929
17.7.6	Fehlerbehandlung beim Piping	932
17.8	Zusammenfassung	933
18	Mobile Anwendungen mit JavaScript erstellen	935
18.1	Die unterschiedlichen Arten mobiler Anwendungen	935
18.1.1	Native Anwendungen	935
18.1.2	Mobile Webanwendungen	936
18.1.3	Hybridanwendungen	938
18.1.4	Vergleich der verschiedenen Ansätze	939
18.2	Mobile Anwendungen mit React Native erstellen	941
18.2.1	Das Prinzip von React Native	942
18.2.2	Installation und Projektinitialisierung	942
18.2.3	Die Anwendung starten	945
18.2.4	Das Grundgerüst einer React-Native-Anwendung	948
18.2.5	UI-Komponenten verwenden	950
18.2.6	Kommunikation mit dem Server	955
18.2.7	Bauen und Veröffentlichen von Anwendungen	956
18.3	Zusammenfassung	956
19	KI-Anwendungen mit JavaScript	959
19.1	Grundlagen und Konzepte	960
19.1.1	Wie arbeitet ein LLM?	962
19.1.2	Tokens – die Währung eines LLMs	962

19.2 Arbeiten mit LLMs auf dem Server	965
19.2.1 Vorbereitung: Lokales LLM installieren	967
19.2.2 Über Bibliotheken mit dem LLM kommunizieren	968
19.2.3 Direkt über die REST-Schnittstelle mit einem LLM kommunizieren	972
19.3 Praktische Anwendung von LLMs	977
19.3.1 Chatbot mit JavaScript	977
19.3.2 Bilderkennung mit TensorFlow.js	983
19.4 LangChain – Ein Architekturframework für KI-Applikationen	991
19.4.1 Eine erste Kette mit LangChain	991
19.5 Zusammenfassung	999

20 Einen professionellen Entwicklungsprozess aufsetzen

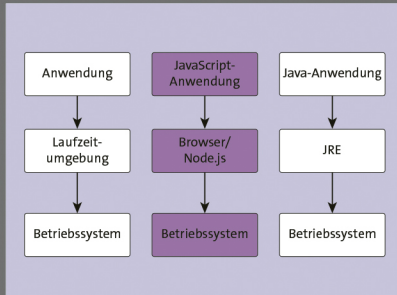
1001

20.1 Den Workflow automatisieren	1001
20.1.1 Überblick: Moderne Tools für die Automatisierung	1002
20.1.2 Codequalität prüfen mit ESLint	1004
20.1.3 Quelltext automatisiert testen mit Vitest	1007
20.1.4 Quelltext bündeln und optimieren mit esbuild	1020
20.1.5 Zusammenspiel im npm-Workflow	1023
20.2 Versionsverwaltung des Quelltexts	1024
20.2.1 Einführung in die Versionsverwaltung	1024
20.2.2 Das Versionsverwaltungssystem Git installieren und konfigurieren	1028
20.2.3 Ein neues lokales Repository anlegen	1030
20.2.4 Ein bestehendes Repository klonen	1030
20.2.5 Änderungen in den Staging-Bereich übertragen	1031
20.2.6 Änderungen in das lokale Repository übertragen	1031
20.2.7 Die verschiedenen Zustände in Git	1033
20.2.8 Änderungen in das Remote Repository übertragen	1034
20.2.9 Änderungen aus dem Remote Repository übertragen	1035
20.2.10 In einem neuen Branch arbeiten	1036
20.2.11 Änderungen aus einem Branch übernehmen	1037
20.2.12 Übersicht über die wichtigsten Befehle und Begriffe	1038
20.3 Zusammenfassung	1042

Anhang	1043
A JavaScript-Referenz	1045
B DOM-Referenz und HTML-Erweiterungen	1103
C BOM und Ajax	1193
D HTML5-Web-APIs-Referenz	1231
 Index	 1281

Das Lehr- und Nachschlagewerk zu JavaScript

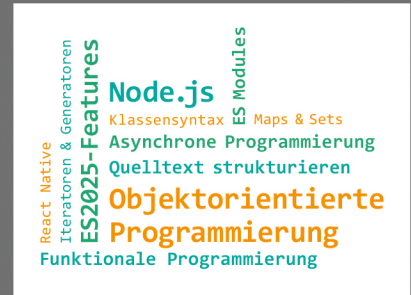
Moderne Websites, mobile und serverseitige Anwendungen, KI-Integration und Machine Learning: Lernen Sie JavaScript von Grund auf und für alle Plattformen. Philip Ackermann liefert Ihnen praxisnahe Beispiele, die Sie sofort einsetzen können. Dabei macht er Sie mit zahlreichen Features und APIs vertraut. So schöpfen Sie die Möglichkeiten von JavaScript voll aus.



Grundlagen der Programmierung

Das Screenshot zeigt ein Registrierungsformular mit den Feldern: 'Nutzername:' (Max Mustermann), 'E-Mail:' (maxmustermann), 'Website:' (http://www.javascripthandb...) und 'Alter:' (44). Ein Fehlerhinweis ist über das E-Mail-Feld gelegt: 'Die E-Mail-Adresse muss ein @-Zeichen enthalten. In der Angabe "maxmustermann" fehlt ein @-Zeichen.' Ein 'Registrieren' Button befindet sich am unteren Rand.

JavaScript in der Webentwicklung



Weiterführende Konzepte

Einsteigen, programmieren, nachschlagen

Mit diesem Buch haben Sie alles in der Hand, um Projekte mit JavaScript erfolgreich umzusetzen: von der kompletten Spracheinführung für JavaScript im Browser, auf dem Server oder für mobile Anwendungen bis zum Einbau des Codes in Ihre Website.

Websites mit beeindruckenden Features

Lernen Sie moderne Web-APIs, asynchrone Datenabfragen mit der Fetch-API sowie aktuelle JavaScript-Techniken kennen, um Ihre Website mit dynamischen Inhalten und interaktiven Funktionen auszustatten.

Professionelle Anwendungen für alle Plattformen

Ob Sie serverseitige Anwendungen mit Node.js erstellen möchten, einen KI-Chatbot aufsetzen oder elegante GUIs für mobile und Desktop-Plattformen bauen: Alle Codebeispiele aus dem Buch lassen sich ideal für Ihre eigenen Projekte nutzen.



Alle Codebeispiele zum Download



Philip Ackermann ist CTO und CPO der Cedalo GmbH und Autor zahlreicher IT-Bücher und Fachartikel. Bei heise Developer bloggte er viele Jahre zum Thema Webentwicklung. Sein Expertenwissen zu JavaScript gibt er in diesem Buch weiter.

Aus dem Inhalt

Wichtige Grundlagen

Einführung in JavaScript
Grundlagen der Programmierung
Einstieg in die Webentwicklung

JavaScript in der Praxis

Webseiten dynamisch verändern
Mit Formularen arbeiten
Browser steuern
Bilder dynamisch erstellen
Web-APIs verwenden
Professionelle Entwicklung

Weiterführende Konzepte

Objektorientierte und funktionale Programmierung
ECMAScript-Features nutzen
Serverseitige Anwendungen
KI und Machine Learning
Umfassende Referenz

