JS

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' + seventType] = 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.

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 ...

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.

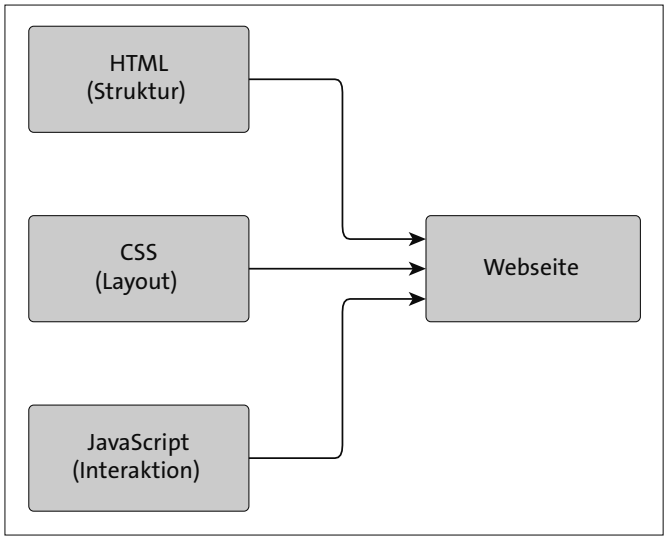


Abbildung 2.5 In der Regel wird innerhalb einer Webseite eine Kombination aus HTML, CSS und JavaScript verwendet.

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.

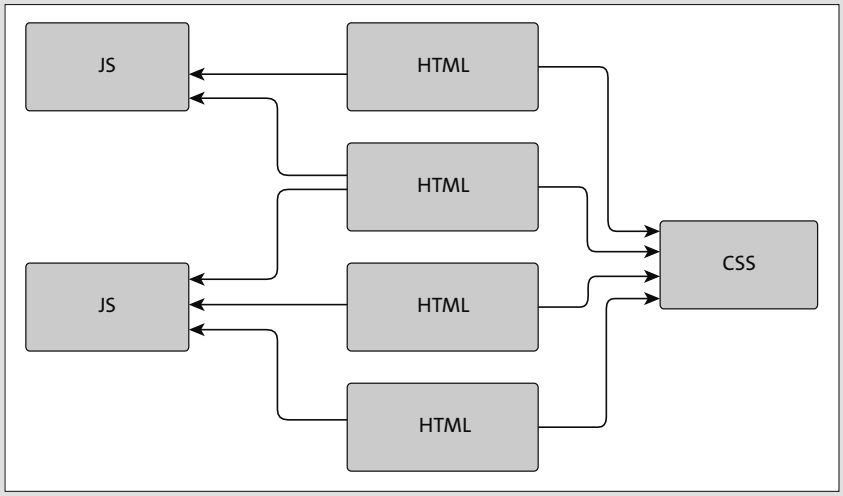


Abbildung 2.6 Wenn Sie CSS- und JavaScript-Code nicht direkt in den HTML-Code schreiben, sondern in separate Dateien, erleichtert das die Wiederverwendbarkeit.

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.

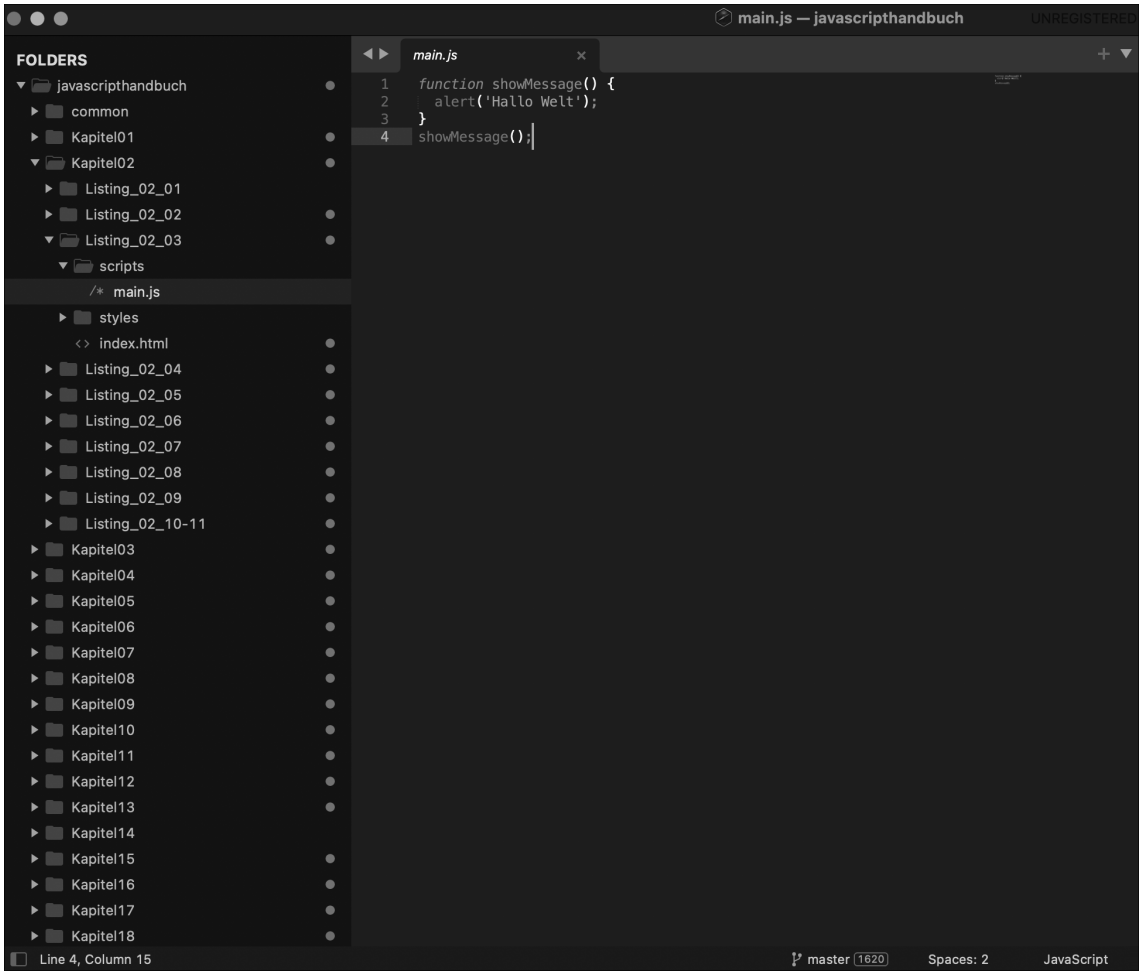


Abbildung 2.7 Der Editor Sublime Text

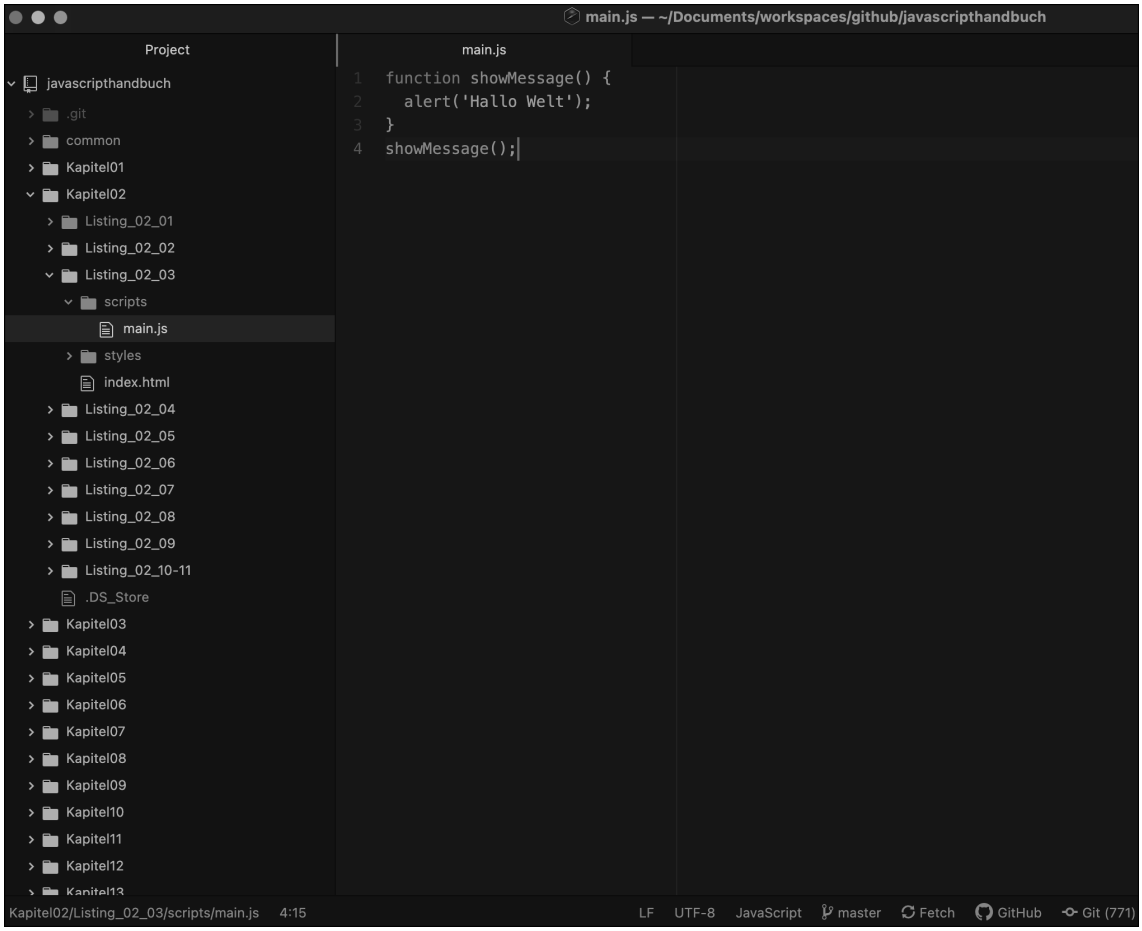


Abbildung 2.8 Der Editor Atom

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.

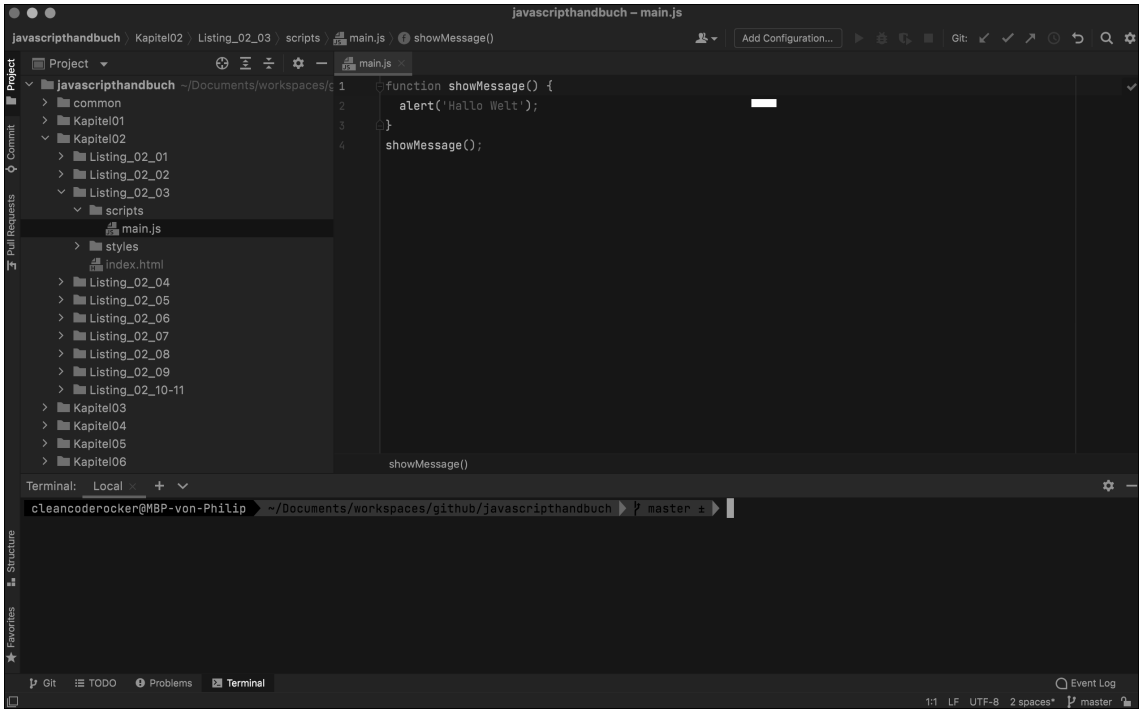


Abbildung 2.9 Die WebStorm-IDE

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.

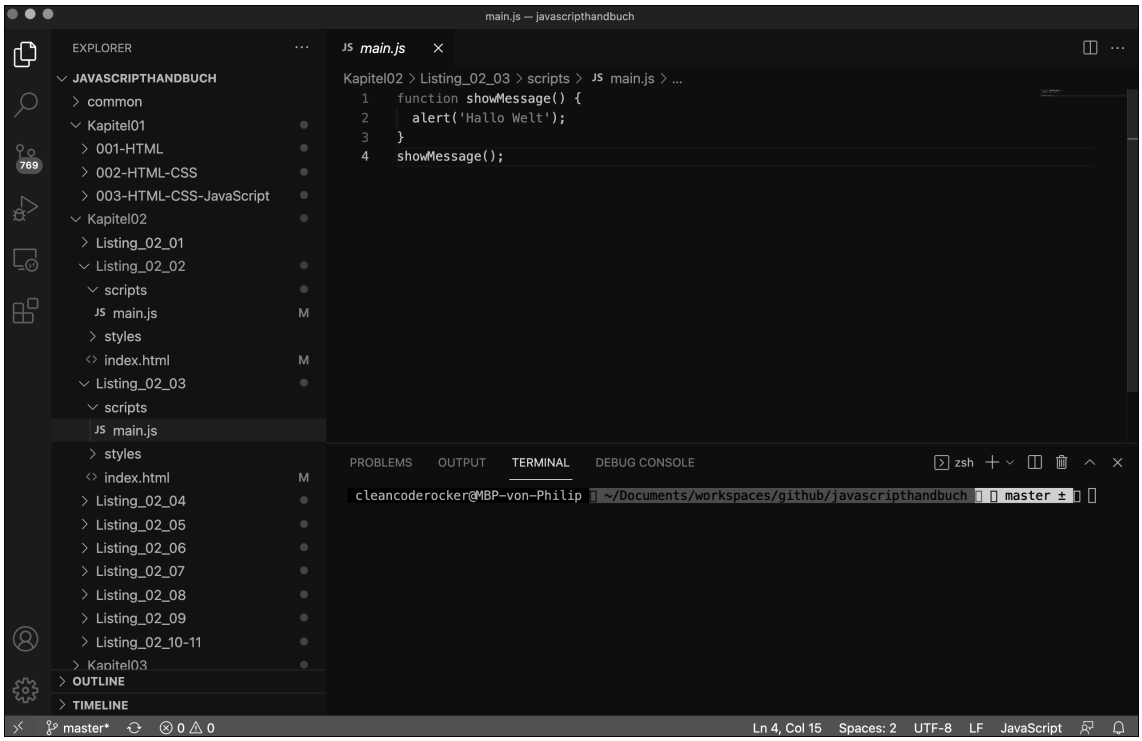


Abbildung 2.10 Microsoft Visual Studio Code

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.

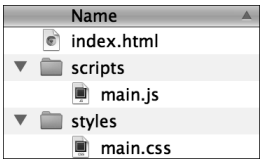


Abbildung 2.11 Exemplarische Ordnerstruktur

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 Dateiendungen 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
`async`	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 `src`-Attribut Sinn.	optional
`charset`	Gibt den Zeichensatz des Quelltextes an, der über das `src`-Attribut eingebunden wird. Ergibt nur in Kombination mit dem `src`-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 `<meta>`-Element über das `charset`-Attribut zu definieren.	optional
`defer`	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 `src`-Attribut Sinn, wird aber insbesondere von älteren Browsern nicht unterstützt.	optional
`language`	Ursprünglich dazu gedacht, die Version des verwendeten JavaScript-Codes anzugeben, wird von Browsern aber weitestgehend nicht beachtet.	veraltet
`src`	Gibt den Pfad zu der JavaScript-Datei an, die eingebunden werden soll.	optional
`type`	Dient der Angabe des *MIME-Types* (siehe Kasten), um die Skriptsprache (in unserem Fall JavaScript) zu identifizieren. Prinzipiell können Sie das Attribut jedoch weglassen, da in diesem Fall standardmäßig `text/javascript` 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).

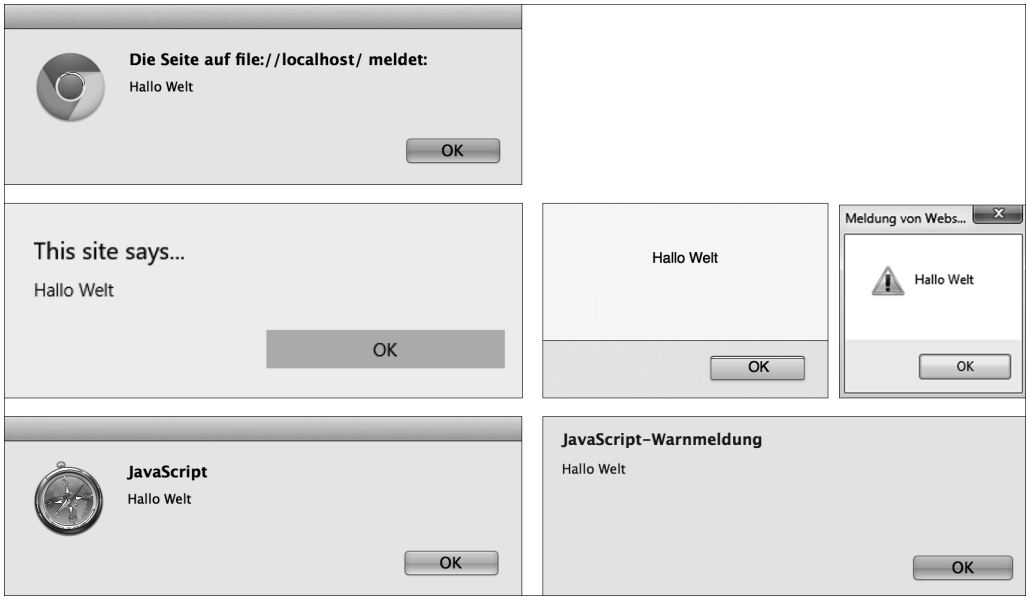


Abbildung 2.12 Hinweisdialoge in den verschiedenen Browsern

```
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 Aufbauen der Webseite gefühlt länger dauert (siehe Abbildung 2.13).

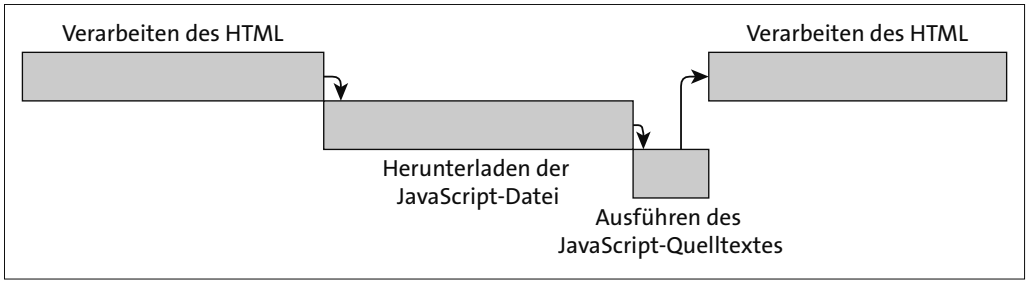


Abbildung 2.13 Standardmäßig wird die Verarbeitung des HTML-Codes gestoppt, wenn der Browser auf ein <script>-Element trifft.

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).

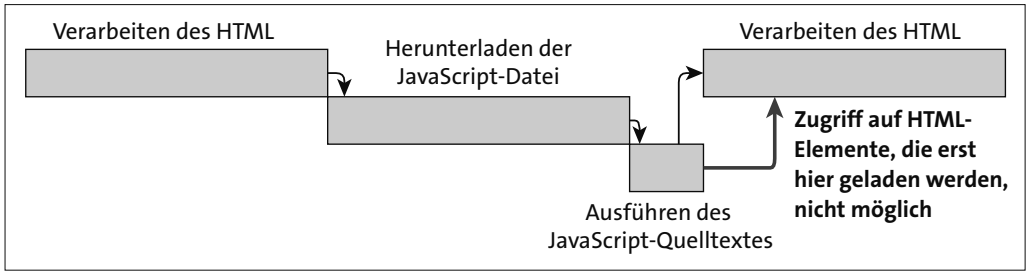


Abbildung 2.14 Wenn JavaScript auf noch nicht geladene HTML-Elemente zugreift, kommt es zu einem Fehler.

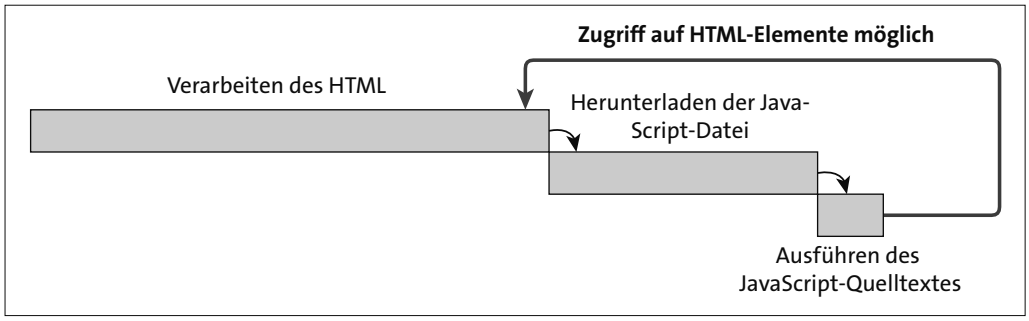


Abbildung 2.15 Wird das <script>-Element vor das schließende </body>-Tag platziert, sind dagegen alle Elemente innerhalb des <body>-Elements geladen.

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.

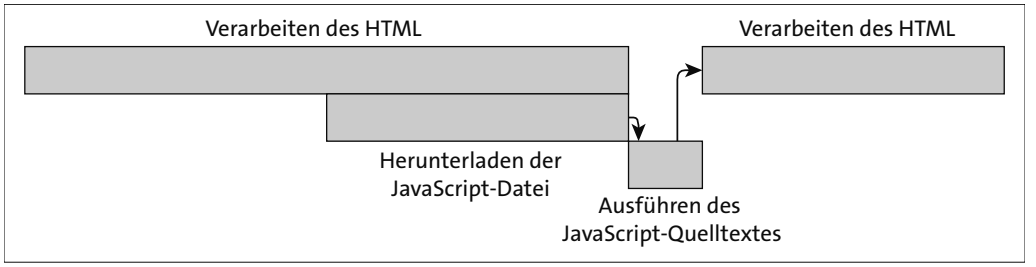


Abbildung 2.16 Über das Attribut »async« wird der HTML-Code so lange weiterverarbeitet, bis das entsprechende JavaScript heruntergeladen wurde.

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`).

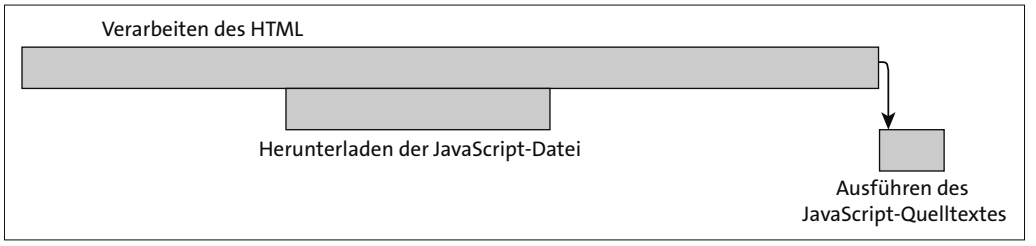


Abbildung 2.17 Über das Attribut »defer« erreicht man, dass das entsprechende JavaScript erst dann ausgeführt wird, nachdem der gesamte HTML-Code der Webseite geladen wurde.

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-Handlern und Ereignis-Listenern, 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-Handlern und Ereignis-Listenern 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-Listenern

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).

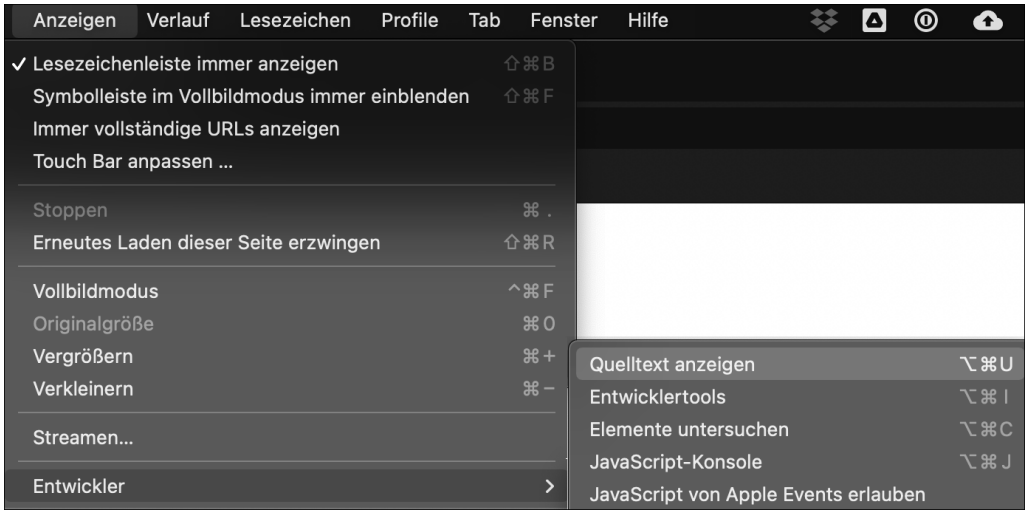


Abbildung 2.18 Quelltext anzeigen in Chrome

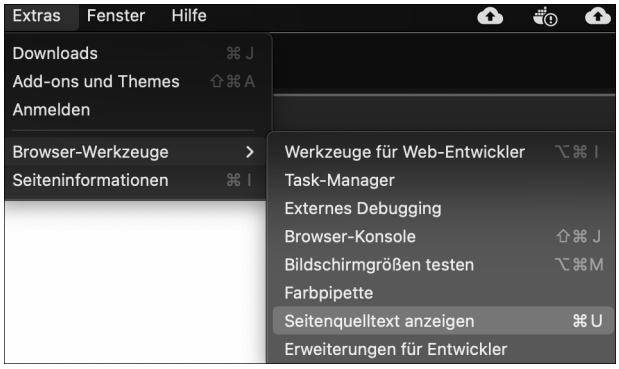


Abbildung 2.19 Quelltext anzeigen in Firefox

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.

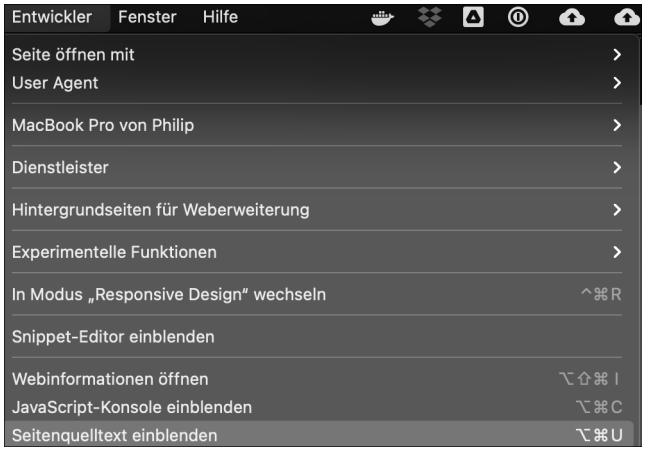


Abbildung 2.20 Quelltext anzeigen in Safari

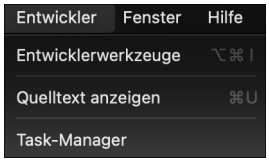


Abbildung 2.21 Quelltext anzeigen in Opera

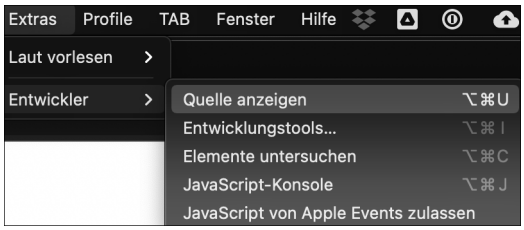


Abbildung 2.22 Quelltext anzeigen in Microsoft Edge

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).

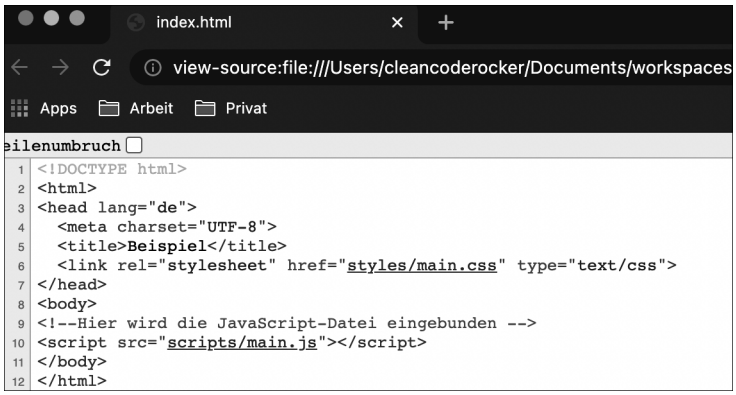


Abbildung 2.23 Quelltextansicht für HTML in Chrome

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.

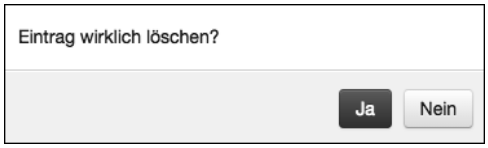


Abbildung 2.28 Ein individueller Bestätigungsdialog mit JavaScript

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.

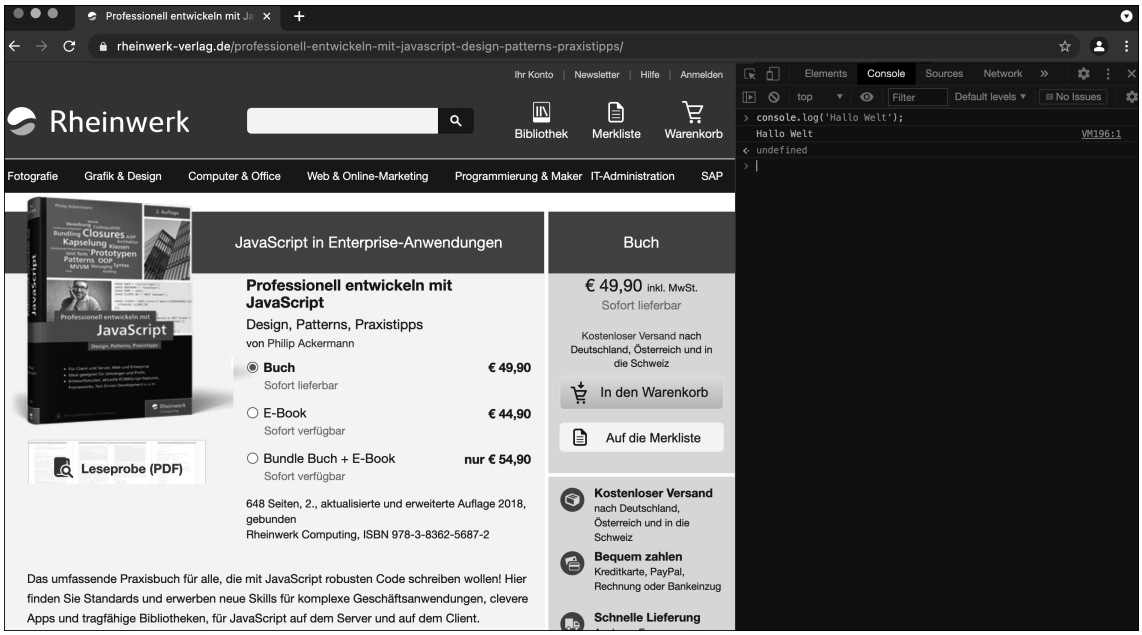


Abbildung 2.29 Standardmäßig wird die Konsole am rechten oder am unteren Rand des Browserfensters angezeigt (hier Google Chrome).

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:

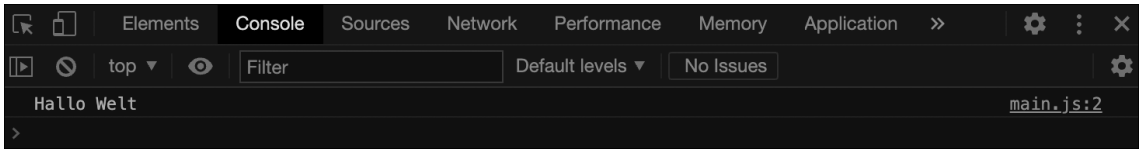


Abbildung 2.30 Ausgabe auf der Konsole in Chrome

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

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

Tabelle 2.2 Die wichtigsten Methoden des »console«-Objekts

Methode	Beschreibung
`trace()`	Gibt den sogenannten *Stack-Trace*, also den *Methodenaufruf-Stack* (siehe auch Kapitel 3, »Sprachkern«), auf der Konsole aus.
`warn()`	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

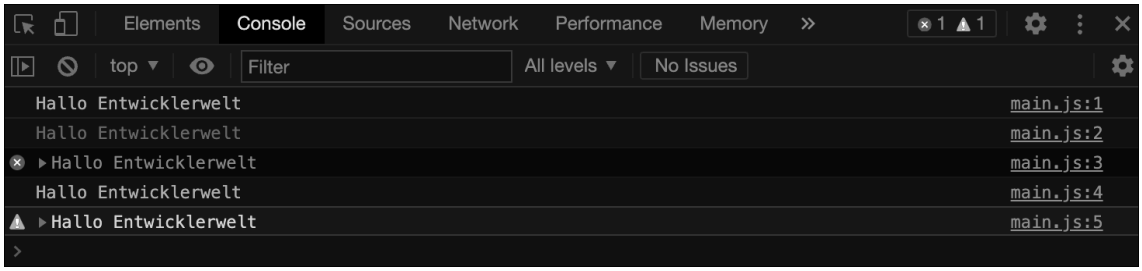


Abbildung 2.31 Die verschiedenen Meldungstypen werden farblich oder durch Symbole hervorgehoben.

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.

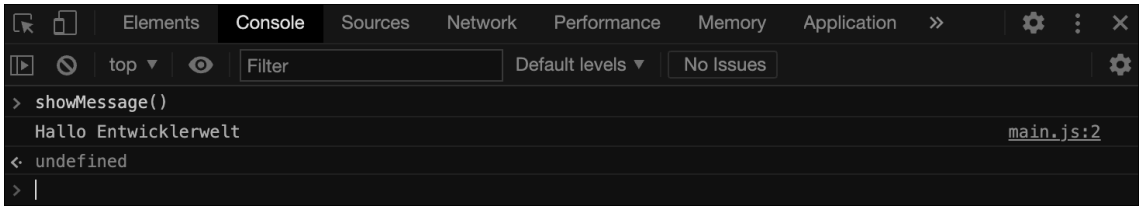


Abbildung 2.32 Über die Konsole können Sie auch Quelltext ausführen.

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

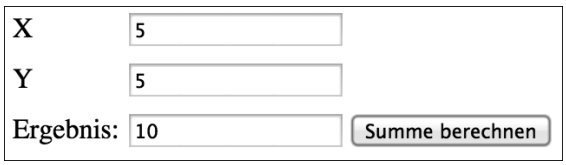


Abbildung 2.33 Beispielanwendung

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

Inhalt

3 Sprachkern 85

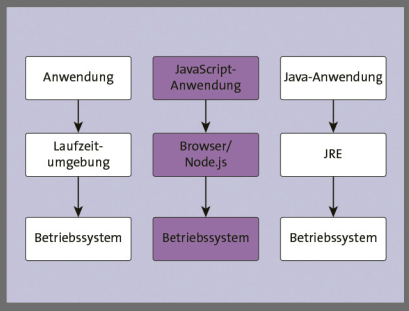
Anhang 1043

JavaScript

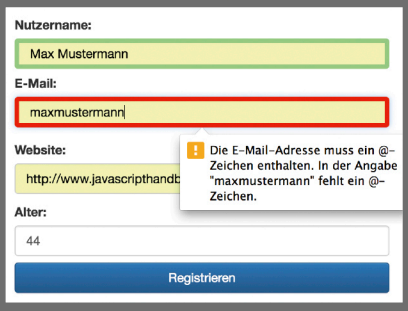
»Die JavaScript-Bibel!«

Das Lehr- und Nachschlagewerk zu JavaScript

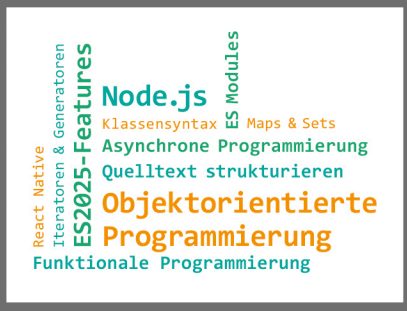
Moderne Websites, mobile und serverseitige Anwendungen, KI-Integration und Machine Learning: Lernen Sie JavaScript von Grund auf und für alle Plattformen. Philip Ackermann liefert Ihnen praxisnahe Beispiele, die Sie sofort einsetzen können. Dabei macht er Sie mit zahlreichen Features und APIs vertraut.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



Grundlagen der Programmierung



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

Rheinwerk Computing

Gedruckt in Deutschland
Zertifiziertes Papier

€ 49,90 [D] € 51,30 [A]
Für Windows, macOS und Linux
Programmierung
ISBN 978-3-367-10916-6