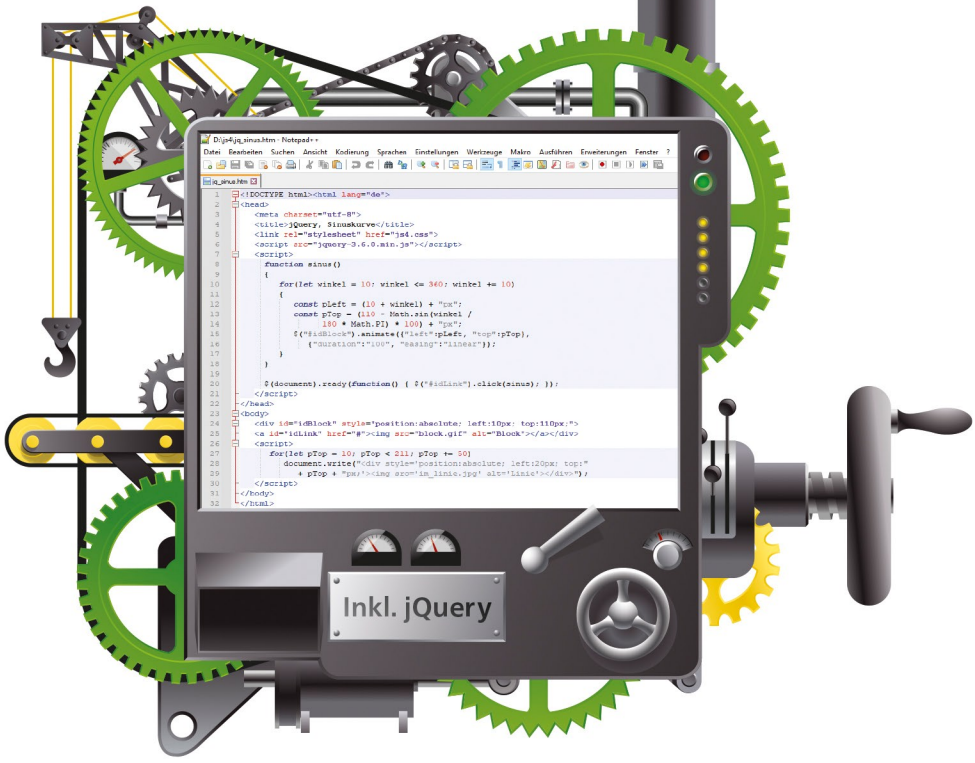
Thomas Theis

Web-Programmierung leicht gemacht

Einstieg in JavaScript



- Dynamische Webanwendungen entwickeln, auch für mobile Geräte
- Programmiergrundlagen, DOM, ECMAScript 2021, Ajax, jQuery, Three.js
- Mit über 200 Beispielprogrammen als Projektvorlagen

Alle Beispielprojekte zum Download

Rheinwerk Computing

Kapitel 1
Einführung

Was ist JavaScript? Was kann ich damit machen und was nicht? Wie baue ich es in meine Internetseite ein? In diesem Kapitel werden erste Fragen geklärt.

ECMAScript

JavaScript ist eine objektorientierte Programmiersprache. Sie wurde 1995 erstmals entworfen und wird mithilfe des Standards ECMAScript ständig aktuell gehalten. Seit 2015 gibt es jährlich eine neue Version von ECMAScript. Die Inhalte dieses Buchs basieren auf ECMAScript 2021.

Einsatz im Browser

Die Sprache JavaScript wurde für den Einsatz in Internetbrowsern entworfen. Das ist das Thema dieses Einsteigerbuchs. Mittlerweile wird JavaScript aber auch außerhalb von Internetseiten eingesetzt. Trotz Ähnlichkeit in einzelnen Aspekten: JavaScript und die ebenfalls weit verbreitete Programmiersprache Java müssen klar voneinander unterschieden werden.

JavaScript gehört zu den interpretierten Sprachen. Das bedeutet: JavaScript-Programme werden Zeile für Zeile übersetzt und ausgeführt.

Document Object Model

Die Sprache bietet viele Elemente, die aus anderen Programmiersprachen bekannt sind, wie z. B. Schleifen zur schnellen Wiederholung von Programmteilen, Verzweigungen zur unterschiedlichen Behandlung verschiedener Situationen und Funktionen zur Zerlegung eines Programms in übersichtliche Bestandteile. Außerdem steht Ihnen eine Vielfalt von Objekten zur Verfügung. Mithilfe des *Document Object Model* (DOM) haben Sie Zugriff auf alle Elemente Ihrer Internetseiten, so dass Sie sie dynamisch verändern können.

1.1 Was mache ich mit JavaScript?

Dynamische Internetseiten

Die Programme werden den Benutzern gemeinsam mit HTML-Code innerhalb von Internetseiten zur Verfügung gestellt. Sie werden auf dem Browser des Benutzers ausgeführt und können die Inhalte einer Internetseite dynamisch verändern. Dies geschieht entweder sofort nach dem Laden der Internetseite oder nach dem Eintreten eines Ereignisses, z. B. der Betätigung einer Schaltfläche durch den Benutzer. JavaScript ermöglicht den Entwurf komplexer Anwendungen mit einer Benutzeroberfläche.

Zusätzliche Möglichkeiten

JavaScript wurde entworfen, um dem Benutzer zusätzliche Möglichkeiten und Hilfen zu bieten, die er allein mit HTML nicht hat. Sie sollten diese nicht dazu nutzen, den Benutzer in irgendeiner Form einzuschränken. Er wird sich z. B. ärgern, wenn er beim Surfen auf eine Internetseite geleitet wird, die er nicht mehr verlassen kann. Nach dieser Erfahrung wird er sich hüten, diese Website jemals wieder aufzusuchen.

Daten senden

Formulare spielen im Zusammenhang mit JavaScript eine wichtige Rolle:

- Sie dienen der Übermittlung von Daten an einen Webserver. Vor dem Absenden können die Inhalte der Formate durch JavaScript auf Gültigkeit hin überprüft werden. Auf diese Weise wird unnötiger Netzverkehr vermieden.
- **Interaktion** Sie ermöglichen eine Interaktion mit dem Benutzer, ähnlich wie er dies von anderen Anwendungen auf seinem Rechner gewohnt ist. Er kann Eingaben vornehmen und eine Verarbeitung auslösen. Das Programm liefert ihm anschließend ein Ergebnis.

Notepad++

Zum Schreiben Ihrer Programme genügt ein Texteditor, der die Markierungen von HTML und die Schlüsselwörter von JavaScript hervorheben kann. Das erleichtert Ihnen die Programmierung sehr. Der Editor Notepad++ ist einer von vielen Editoren, der das beherrscht.

1.2 Was kann JavaScript nicht?

Einschalten nicht möglich

JavaScript kann sich selbst nicht einschalten. Es wird immer einzelne Benutzer geben, die JavaScript in ihrem Browser ausgeschaltet haben. Allerdings ist der Anteil an Internetseiten, die diese Benutzer dann nicht mehr vollständig nutzen können, recht hoch. Wir können aber zumindest erkennen, ob JavaScript eingeschaltet ist oder nicht, und entsprechend reagieren, siehe Abschnitt 1.11, »Kein JavaScript möglich«.

Keine Speicherung auf Server

JavaScript kann (ohne Zusätze) nichts auf dem Webserver speichern. JavaScript-Programme werden im Browser des Benutzers ausgeführt und nicht auf dem Webserver, von dem sie geladen werden. Daher ist es z. B. nicht möglich, Daten auf dem Webserver speichern.

Wenig Speicherung im Browser

JavaScript kann nur wenige Daten auf dem Endgerät des Benutzers speichern. Es kann dort keine Schäden verursachen. Ein Zugriff auf Daten des Benutzers auf seiner Festplatte ist nur in geringem Umfang, in einem eingeschränkten Bereich und mit Zustimmung des Benutzers möglich. Beispiele dazu sehen Sie in Kapitel 14, »Cookies«.

1.3 Browser und mobile Browser

Internetseiten, ob mit oder ohne JavaScript, werden mithilfe von unterschiedlichen Browsern unter verschiedenen Betriebssystemen auf diversen Endgeräten empfangen und für den Benutzer umgesetzt.

Häufigkeit der Browser

Im Internet werden verschiedene Browserstatistiken geführt. Mit ihrer Hilfe kann festgestellt werden, welche Browser von den Benutzern in welcher Häufigkeit verwendet werden. Zurzeit (im April 2021) hat der Browser *Google Chrome* den höchsten Anteil, mit steigender Tendenz. Daher dient er als wichtigste, aber nicht einzige Referenz für die Beispielprogramme in diesem Buch.

Standardform

Manche JavaScript-Anweisung kann für bestimmte Browser eventuell anders formuliert werden. Ich empfehle allerdings, immer die Standardform zu benutzen, damit die Anweisung für möglichst viele Browser geeignet ist. Dieser Grundsatz gilt auch für die Beispielprogramme in diesem Buch.

Mobilgeräte, Sensoren

Der Anteil an mobilen Endgeräten mit den dafür zugeschnittenen mobilen Browsern wird immer größer. Mobilgeräte bieten einige zusätzliche Möglichkeiten, wie z. B. Empfänger bzw. Sensoren für Standortdaten, Lage und Beschleunigung des Mobilgeräts. Die dabei ermittelten Daten können von JavaScript weiterverarbeitet werden, siehe Abschnitt 16.3 und folgende.

1.4 ECMAScript

Klassen und Konstruktoren

JavaScript basiert auf *ECMAScript*. Sie können in JavaScript objektorientiert programmieren. Dies geschieht auf der Basis von sogenannten Prototypen. Seit der Einführung des Standards ECMAScript 2015 gibt es eine zusätzliche Schreibweise, in der mit Klassen, Konstruktoren, Eigenschaften und Methoden gearbeitet wird, wie Sie es eventuell bereits aus anderen objektorientierten Sprachen kennen. Alle modernen Browser setzen diese Schreibweise um. Mehr dazu in aller Ausführlichkeit in Kapitel 3, »Eigene Objekte«.

Übersicht

Seit dem Jahr 2015 wird der Standard jährlich aktualisiert. Zurzeit (im April 2021) ist bereits der Entwurf für den Standard ECMAScript 2021 fertiggestellt und wird in diesem Buch genutzt. Auf der Internetseite mit der Adresse *http://kangax.github.io/compat-table/es2016plus* finden Sie eine Übersicht der standardisierten Elemente und ihrer Umsetzung in den einzelnen Browsern.

Viele Elemente der neueren Standards sind auch für Einsteiger interessant und an der passenden Stelle im Buch zu finden. Ich weise jeweils gesondert darauf hin.

1.5 Aufbau des Buchs

Zunächst eine Anmerkung in eigener Sache: Für die Hilfe bei der Erstellung des Buchs bedanke ich mich beim Team vom Rheinwerk Verlag, besonders bei Patricia Schiewald und Anne Scheibe.

Beispielprogramme mit Erläuterung

Die Themen in diesem Buch stelle ich jeweils mit einer kurzen Beschreibung der Theorie, einem aussagefähigen Screenshot, einem vollständigen, lauffähigen Beispielprogramm und einer ausführlichen praktischen Erläuterung vor. Die Screenshots sind im Browser Google Chrome entstanden, entweder auf einem PC mit Windows 10 oder auf einem Android-Smartphone.

Downloadpaket

Auf diese Weise haben Sie einen raschen Einstieg in jedes Thema. Sie sind nicht gezwungen, vereinzelte Codezeilen zunächst in einen passenden Kontext zu setzen, um ihre Wirkung zu betrachten. Sie finden alle Beispielprogramme im Downloadpaket zum Buch, das Sie über *https://www.rheinwerk-verlag.de/5363* erreichen.

Übungsaufgaben

Im Buch finden Sie Hinweise auf Übungsaufgaben im zugehörigen Bonuskapitel. Diese geben Ihnen die Möglichkeit, Ihre Kenntnisse zu testen. Eine Lösung zu jeder Übungsaufgabe finden Sie ebenfalls im Downloadpaket zum Buch.

Schrittweiser Aufbau

Die Inhalte des Buchs bauen normalerweise in kleinen, übersichtlichen Schritten aufeinander auf. Dies hat den *Vorteil*, dass die Voraussetzungen zu jedem Thema vorher geklärt sind. Es hat allerdings den *Nachteil*, dass Sie das Buch tatsächlich von vorn nach hinten lesen sollten. Schlagen Sie es dagegen einfach an einer beliebigen Stelle auf, können Sie nicht davon ausge-

hen, dass an dieser Stelle alle Einzelheiten erklärt werden. Dies ist eventuell in einem früheren Abschnitt geschehen.

Grundlagen und Objekte

Nach der Einleitung in diesem Kapitel 1 folgen die Grundlagen der Programmierung in Kapitel 2. Hier zeigen sich Ähnlichkeiten mit vielen anderen Programmiersprachen. Objekte spielen in JavaScript eine große Rolle. In Kapitel 3 erschaffen Sie eigene Objekte und lernen auf diese Weise ihren Aufbau kennen. In Kapitel 6 erläutere ich Ihnen viele vordefinierte Objekte von JavaScript.

DOM

Zur Interaktion mit dem Benutzer wird mit Ereignissen und ihren Folgen gearbeitet, insbesondere im Zusammenhang mit Formularen, siehe Kapitel 4. Die Kenntnis des Aufbaus einer Internetseite nach dem *Document Object Model* (*DOM*, siehe Kapitel 5) ermöglicht Ihnen, auf beliebige Stellen im Dokument zuzugreifen und sie zu verändern.

Ajax und CSS

Die *Ajax*-Technologie (siehe Kapitel 7) ermöglicht Ihnen u. a. den Austausch einzelner Teile eines Dokuments, ohne eine Seite vollständig neu laden zu müssen. *CSS* (*Cascading Style Sheets*) bieten vielfältige Möglichkeiten der Formatierung und Positionierung von Elementen eines HTML-Dokuments. Diese werden mithilfe von JavaScript dynamisch erweitert, bis hin zur Animation, siehe Kapitel 8.

SVG und Three.js

Mithilfe des Standards *SVG* (*Scalable Vector Graphics*) und JavaScript lassen sich dynamische, zweidimensionale Vektorgrafiken erstellen, siehe Kapitel 9. Die JavaScript-Bibliothek *Three.js* (siehe Kapitel 10) bietet die Möglichkeit, dreidimensionale Grafiken und Animationen zu entwickeln.

jQuery, Onsen UI und MathML

Die weitverbreitete Bibliothek *jQuery* (siehe Kapitel 11) ermöglicht einen browserunabhängigen, komfortablen Zugriff auf viele Elemente von JavaScript. Die Bibliothek *Onsen UI* (siehe Kapitel 12) dient speziell zur Programmierung mobiler Endgeräte. Mathematische Ausdrücke lassen sich mithilfe von *MathML* und der JavaScript-Bibliothek *MathJax* (siehe Kapitel 13) in Ihren Dokumenten darstellen.

Cookies, Zeichnungen und Sensoren

Cookies (siehe Kapitel 14) bieten einen Zugriff auf Daten des Benutzers, allerdings nur in geringem Umfang und in einem eingeschränkten Bereich. In Kapitel 15 verweise ich auf eine Reihe von größeren, ausführlich kommentierten Beispielprojekten, bei denen das Zusammenspiel vieler Elemente gezeigt wird. Den Zugriff auf Medien und Sensoren sowie die Erstellung von Zeichnungen erläutere ich in Kapitel 16.

1.6 Erstes Beispiel mit HTML und CSS

Erstes Beispiel

Zum Verständnis der Beispiele dieses Buchs werden nur wenige Kenntnisse in HTML und CSS vorausgesetzt. Die wichtigsten Elemente werden anhand eines ersten Beispielprogramms erläutert.

1.6.1 Ausgabe des Programms

Ausgabe

In Abbildung 1.1 und Abbildung 1.2 sehen Sie das Ergebnis des Programms im Browser.

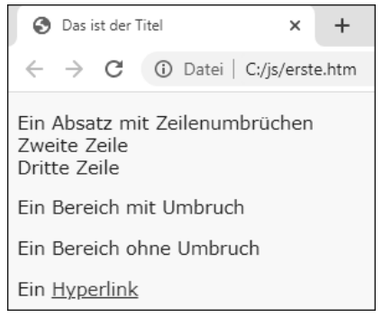


Abbildung 1.1 Erstes HTML-Dokument im Browser, oberer Teil

Abbildung 1.2 Erstes HTML-Dokument im Browser, unterer Teil

1.6.2 HTML-Datei

HTML-Code

Das Beispielprogramm enthält einige Grundelemente eines HTML-Dokuments. Es folgt der HTML-Code:

```
<!DOCTYPE html><html lang="de">
<head>
   <meta charset="utf-8">
   <title>Das ist der Titel</title>
   <link rel="stylesheet" href="js4.css">
</head>
<body>
   <p>
      Ein Absatz mit Zeilenumbrüchen<br>
      Zweite Zeile<br>Dritte Zeile
   </p>

   <div>Ein Bereich mit Umbruch</div>
   <p>Ein Bereich <span>ohne</span> Umbruch</p>

   <p>Ein <a href="einbetten.htm">Hyperlink</a></p>
   <p>Ein Bild:<br><img src="im_paradies.jpg" alt="Paradies"></p>

   <p>Eine Liste:</p>
   <ul>
      <li>Erster Eintrag</li>
      <li>Zweiter Eintrag</li>
   </ul>

   <p>Eine Tabelle:</p>
   <table>
      <tr>
         <td>Zelle A</td>
         <td>Zelle B</td>
      </tr>
      <tr>
         <td>Zelle C</td>
         <td>Zelle D</td>
      </tr>
   </table>
</body>
</html>
```

Listing 1.1 Datei »erste.htm«

Dokumente validieren

Mithilfe von `<!DOCTYPE html>` wird festgelegt, dass es sich um ein HTML-Dokument handelt. Je mehr Sie sich an die einheitlichen Definitionen für HTML-Dokumente halten, desto höher ist die Wahrscheinlichkeit, dass die Seite in allen Browsern fehlerfrei dargestellt wird. Sie können Ihre Seiten über *http://validator.w3.org* validieren lassen, also auf Übereinstimmung mit der Definition prüfen lassen.

Markierungen und Attribute

Ein HTML-Dokument besteht aus Markierungen (auch *Tags* genannt) und Text. Die meisten Markierungen bilden einen *Container* (= Behälter), der eine Start-Markierung und eine End-Markierung besitzt. Start-Markierungen können Attribute mit Werten haben. Letztgenannte stehen dabei standardmäßig in doppelten Hochkommata.

html, head, body

Das gesamte Dokument steht im `html`-Container, von der Start-Markierung `<html>` bis zur End-Markierung `</html>`. Die Start-Markierung `html` besitzt das Attribut `lang` mit dem Wert `de`. Damit geben Sie an, dass der Text des Dokuments in deutscher Sprache verfasst ist. Im `html`-Container liegen nacheinander ein `head`-Container mit Informationen über das Dokument und ein `body`-Container mit dem eigentlichen Dokumentinhalt.

Zeichensatz UTF-8

Im `head`-Container finden Sie zunächst einen `title`-Container, der den Inhalt für die Titelleiste des Browsers bereitstellt. Außerdem stehen hier Metadaten über das Dokument. Im vorliegenden Beispiel sehen Sie, dass es sich um ein HTML-Dokument handelt, das den weit verbreiteten Zeichensatz *UTF-8* nutzt, siehe Abschnitt 1.6.3. Er enthält viele Sonderzeichen, z. B. auch die deutschen Umlaute.

Externe CSS-Datei

Mithilfe der Markierung `link` und den Attributen `rel` und `href` können Sie eine externe CSS-Datei zur Formatierung des Dokuments einbinden. Im vorliegenden Beispiel handelt es sich um die Datei *js4.css*, die im selben Verzeichnis wie die Datei *erste.htm* liegt. Sie wird in Abschnitt 1.6.4, »Responsives Webdesign«, erläutert.

Absätze und Bereiche

Absätze stehen in `p`-Containern. Ein einzelner Zeilenumbruch innerhalb eines Absatzes wird mithilfe der Markierung `<br>` gebildet. Bestimmte Bereiche, die eine andere Formatierung erhalten sollen, können Sie sowohl in einen `div`-Container als auch in einen `span`-Container setzen. Vor und nach einem `div`-Container wird zudem ein Umbruch erzeugt.

Hyperlinks und Bilder

Ein anklickbarer Hyperlink zu einem anderen Dokument steht in einem `a`-Container mit dem Attribut `href`. Ein Bild kann mithilfe der `img`-Markierung und dem Attribut `src` eingebunden werden. Das Attribut `alt` ist für die Va-

lidierung erforderlich. Es enthält einen erläuternden Text für den Fall, dass die Bilddatei nicht geladen werden kann.

Liste Eine nicht nummerierte Liste steht in einem `ul`-Container, die einzelnen Listeneinträge stehen in `li`-Containern.

Tabelle Eine Tabelle wird mithilfe eines `table`-Containers erstellt. Innerhalb der Tabelle gibt es einzelne Zeilen; diese werden jeweils mithilfe eines `tr`-Containers erstellt. Innerhalb einer Zeile wiederum gibt es einzelne Zellen, die jeweils durch einen `td`-Container gebildet werden.

Datei im Browser Die Datei *erste.htm* wird mithilfe des Editors Notepad++ erstellt und (bei mir) im Verzeichnis *C:/js* gespeichert. Zur Darstellung einer *htm*-Datei (oder einer *html*-Datei) in Ihrem Standardbrowser öffnen Sie den Windows-Explorer, und führen Sie einen Doppelklick auf die *htm*-Datei aus.

Werden weitere HTML-Markierungen genutzt, werden sie an der passenden Stelle erläutert.

1.6.3 Codierung UTF-8

Unicode-Zeichen In allen HTML-Dokumenten dieses Buchs wird die Codierung UTF-8 verwendet. *UTF-8* steht abkürzend für das *8-Bit UCS Transformation Format*. *UCS* steht abkürzend für *Universal Character Set*. UTF-8 ist diejenige Codierung für Unicode-Zeichen mit der weitesten Verbreitung.

Codierung muss übereinstimmen Es ist wichtig, dass die Codierung, die im `head`-Container angegeben ist, mit der Codierung der Datei übereinstimmt. Sie können, falls noch nicht geschehen, die Codierung einer Datei im Editor Notepad++ wie folgt auf UTF-8 umstellen: Menü Kodierung • Konvertiere zu UTF-8. Anschließend ist in diesem Menü auch die Codierung UTF-8 markiert.

Standardwert Sie können die Codierung im Editor Notepad++ wie folgt auch automatisch für alle Dateien wählen, die Sie neu erstellen: Menü Einstellungen • Einstellungen • Neues Dokument • Kodierung • UTF-8, Schaltfläche Schliessen.

1.6.4 Responsives Webdesign

In diesem ersten Beispiel wird die externe CSS-Datei *js4.css* zur Formatierung des Dokuments eingebunden. Darin wird mithilfe eines *Media Query* auf vereinfachte Weise ein responsives Webdesign erzeugt. Es werden folgende Ziele erreicht:

- Die Dokumente sind einheitlich formatiert. **Einheitlich**
- Bei Bedarf kann die Formatierung für alle Dokumente schnell und einheitlich geändert werden.
- Die Beispiele können nicht nur auf einem PC oder einem Laptop genutzt werden, sondern auch auf einem Mobilgerät. **PC oder Mobilgerät**

Es folgt der Code in der CSS-Datei:

```
body {font-family:Verdana; font-size:11pt; color:#202020;
      background-color:#f8f8f8;}
td   {font-size:11pt; background-color:#e0e0e0; padding:5px;}

@media only screen and (max-width: 992px)
{
   @media only screen and (orientation:landscape)
   {
      body                { font-size:20pt; }
      td                  { font-size:20pt; }
      img                 { width:240px; height:180px; }
      input               { font-size:20pt; }
      select              { font-size:20pt; }
      input[type=radio]   { width:30px; height:30px; }
      input[type=checkbox] { width:30px; height:30px; }
      input[type=color]   { width:250px; height:30px; }
      input[type=range]   { width:250px; height:30px; }
      textarea            { font-size:20pt; height:80px; }
   }
   @media only screen and (orientation:portrait)
   {
      body                { font-size:32pt; }
      td                  { font-size:36pt; }
      img                 { width:320px; height:240px; }
      input               { font-size:32pt; }
      select              { font-size:32pt; }
      input[type=radio]   { width:45px; height:45px; }
      input[type=checkbox] { width:45px; height:45px; }
      input[type=color]   { width:250px; height:45px; }
      input[type=range]   { width:250px; height:45px; }
      textarea            { font-size:32pt; height:140px; }
   }
}
```

Listing 1.2 Datei »js4.css«

An dieser Stelle folgt nur eine kurze Erläuterung zu CSS. In Kapitel 8 sehen Sie mehr zu diesem Thema.

Formatierung für HTML-Markierung

Eine CSS-Angabe kann bezüglich einer HTML-Markierung gelten, auf die sich die Formatierung bezieht. Innerhalb von geschweiften Klammern stehen eine oder mehrere Formatierungen. Diese bestehen wiederum aus einer CSS-Eigenschaft und einem Wert für diese Eigenschaft, getrennt durch einen Doppelpunkt und abgeschlossen durch ein Semikolon.

Dokument

Im vorliegenden Beispiel wird für die Markierung `body`, also den Inhalt des Dokuments, eine Reihe von Formatierungen vorgenommen:

- Schriftart Verdana, mit `font-family`
- Schriftgröße 11 Punkt, mit `font-size`
- Schriftfarbe `#202020`, mit `color`
- Hintergrundfarbe `#f8f8f8`, mit `background-color`

Farbe

Farben können mithilfe von RGB-Werten angegeben werden. Nach dem Zeichen # folgen jeweils zwei hexadezimale Ziffern für die Farbanteile Rot, Grün und Blau. Die Farbe `#202020` entspricht einem dunklen Grau, die Farbe `#f8f8f8` einem sehr hellen Grau.

Tabellenzelle

Die Schriftgröße innerhalb von Tabellenzellen wird noch einmal gesondert auf 11 Punkt eingestellt. Die Hintergrundfarbe von Tabellenzellen (Markierung `td`) wird mit `#e0e0e0` auf ein helles Grau gesetzt. Damit sind sie etwas dunkler als der Hintergrund des Dokuments. Mithilfe der Angabe `padding` kann der innere Abstand eines Elements zu seinem umgebenden Element eingestellt werden. Hier wird rund um den Textinhalt einer Tabellenzelle ein Abstand von 5 Pixeln zum Rand der Tabellenzelle gesetzt.

An dieser Stelle enden die CSS-Angaben für die Nutzung auf einem PC oder einem Laptop.

Media Query

Mithilfe eines Media Query kann auf die Eigenschaften der Geräte reagiert werden, mit denen die Dokumente betrachtet werden. Die CSS-Angabe `@media only screen and (max-width: 992px)` bewirkt, dass sich die CSS-Angaben innerhalb der nachfolgenden geschweiften Klammern:

- nur auf eine Ausgabe auf einem Bildschirm (englisch: *screen*) bezieht, im Unterschied zu einer Ausgabe auf einem Drucker
- nur auf Geräte bezieht, die eine maximale Ausgabebreite von 992 Pixeln besitzen, wie z. B. Mobilgeräte wie Tablets und Smartphones

Quer- oder Hochformat

Anschließend wird mithilfe der Angabe `orientation` unterschieden, ob sich das Tablet bzw. das Smartphone aktuell im Querformat (Landschaftsmodus) oder im Hochformat (Porträtmodus) befindet.

Schriftgröße

Die Schriftgröße im Dokument wird im Querformat auf 20 Punkt und im Hochformat auf 32 Punkt vergrößert. In Tabellenzellen findet eine Vergrößerung auf 20 Punkt bzw. 36 Punkt statt.

Größe von Bildern

Viele Bilder in meinen Beispielprogrammen haben zur Vereinfachung eine einheitliche Größe von 160 × 120 Pixeln. Im Querformat werden sie auf 240 × 180 Pixel, im Hochformat auf 320 × 240 Pixel vergrößert.

Formulare

Die weiteren Elemente begegnen Ihnen erst innerhalb der Formulare ab Kapitel 4. Auch ihre Größe wird medienabhängig eingestellt:

- Eingabe allgemein, einzeiliges Textfeld (`input`)
- Auswahl mithilfe eines Menüs (`select`)
- Auswahl mithilfe von Radiobuttons (`input type=radio`)
- Kontrollkästchen (`input type=checkbox`)
- Auswahl einer Farbe (`input type=color`)
- Einstellung einer Zahl in einem Bereich (`input type=range`)
- Mehrzeiliges Textfeld (`textarea`)

1.7 Einige Sonderzeichen

Entity

Das folgende Programm dient zur Ausgabe einiger Sonderzeichen. Einige davon können Sie auch unmittelbar über die Tastatur eingeben. Alle Sonderzeichen können mithilfe von sogenannten *Entities* ausgegeben werden. Es folgt das Programm:

```
...
<body>
   <p>Einige Sonderzeichen der Tastatur: > & € @<br>
      dazu auch die Entity: &gt; & &euro; &#64;<br>
      Einige weitere Sonderzeichen: &copy; &reg; &permil;
         &frac14; &frac12; &frac34; &sup2; &sup3; &micro; &pi;<br>
      Das Zeichen &lt; wird nur als Entity validiert</p>
</body></html>
```

Listing 1.3 Datei »sonderzeichen.htm«

Hinweis

In diesem Beispiel und in vielen folgenden Beispielen wird aus Platzgründen der Beginn des Dokuments weggelassen. Er wird nur dann abgedruckt, wenn er zusätzliche Angaben enthält. Zudem wird das Ende des Dokuments kompakter dargestellt.

Über Tastatur

Die Sonderzeichen in der ersten Zeile können Sie unmittelbar über die Tastatur in Ihr Dokument einfügen: > (größer), & (Kaufmanns-Und), € (Euro) und @ (at).

&entity;

Zu vielen Sonderzeichen gibt es sogenannte *Entities*. Eine Entity beginnt mit dem Zeichen & und endet mit einem Semikolon:

- In der zweiten Zeile sehen Sie die Entities für >, &, € und @: `>`, `&`, `&euro;` und `&#64;`.
- In der dritten Zeile folgen die Entities für © (Copyright), ® (Registered Trademark = eingetragenes Warenzeichen), ‰ (Promille), ¼ (ein Viertel), ½ (ein halb), ¾ (drei Viertel), ² (hoch 2), ³ (hoch 3), µ (mikro) und π (Pi): `&copy;`, `&reg;`, `&permil;`, `&frac14;`, `&frac12;`, `&frac34;`, `&sup2;`, `&sup3;`, `&micro;` und `&pi;`.

Zeichen »<«

- Das Zeichen < (kleiner) in der vierten Zeile sollten Sie mithilfe der Entity `<` ausgeben, ansonsten wird das zugehörige Dokument nicht als gültiges HTML-Dokument validiert.

In Abbildung 1.3 sehen Sie das Dokument im Browser.

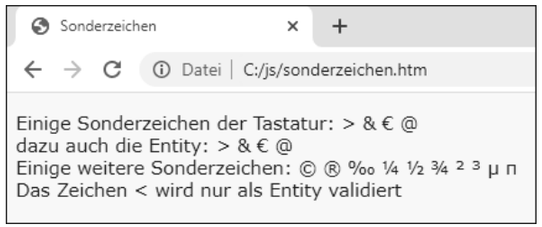


Abbildung 1.3 Einige Sonderzeichen

1.8 JavaScript im Dokument

Erstes Programm

Nun geht es aber endlich los mit dem ersten JavaScript-Programm. Betrachten Sie zunächst den folgenden Code:


```
...
<body>
   <p>Teil 1 in HTML</p>
   <script>

      document.write("<p>Teil 2 in JavaScript</p>");

   </script>
   <p>Teil 3 in HTML</p>
   <script>

      document.write("<p>Teil 4 in JavaScript<br>");
      document.write("Hier stehen <span style='font-weight:bold;'>"
         + " einfache</span> Hochkommata</p>");

   </script>
</body></html>
```

Listing 1.4 Datei »einbetten.htm«

script

Sie können JavaScript an beliebig vielen Stellen im `head` oder im `body` eines HTML-Dokuments einbetten. Es wird jeweils ein `script`-Container benötigt. Dieser beginnt mit `<script>` und endet mit `</script>`.

Semikolon

Innerhalb des Containers stehen JavaScript-Anweisungen, die nacheinander ausgeführt werden. Jede Anweisung sollte mit einem Semikolon abgeschlossen werden.

document.write()

Bei `document.write()` handelt es sich um eine Methode des `document`-Objekts, mit der Sie Zeichenketten ausgeben können. Diese stehen standardmäßig in doppelten Hochkommata. In den Zeichenketten können sowohl Texte als auch HTML-Markierungen stehen. Den Begriff *Objekt* erläutere ich in Kapitel 3 in aller Ausführlichkeit.

Doppelte und einfache Hochkommata

Die Werte von Attributen werden in HTML standardmäßig ebenfalls in doppelten Hochkommata notiert. Innerhalb der Zeichenkette von `document.write()` müssen für HTML-Attribute einfache Hochkommata verwendet werden. Im Beispiel betrifft das den Wert des Attributs `style`. Ich habe die CSS-Eigenschaft `font-weight` mit dem Wert `bold` genutzt, um einen Teil des Textes in Fettschrift zu setzen.

Einige Hinweise

- Beachten Sie beim Programmieren die richtige Schreibweise der Anweisungen. Die Browser verzeihen, anders als in HTML, in JavaScript keine Fehler.
- JavaScript unterscheidet zwischen Groß- und Kleinschreibung. Mit der Anweisung `document.Write(...)` werden Sie keinen Erfolg haben, da es die Methode `Write()` mit großem Anfangsbuchstaben `W` nicht gibt.
- Sie können auch mehrere Anweisungen in eine Zeile schreiben. Hauptsache, es steht ein Semikolon am Ende jeder Anweisung.

In Abbildung 1.4 sehen Sie verschiedene Teile des Dokuments, die zum Teil aus HTML und zum Teil aus JavaScript stammen.

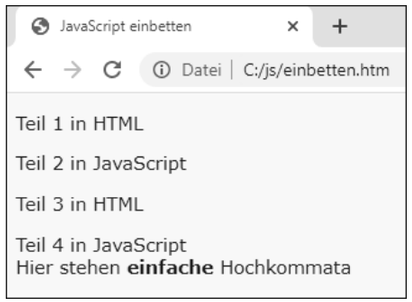


Abbildung 1.4 JavaScript innerhalb einer Datei

1.9 JavaScript aus externer Datei

Datei einbinden

Sie können Programmteile, die Sie in mehreren JavaScript-Programmen nutzen möchten, in einer externen Datei speichern. Auf den Code einer solchen externen Datei können Sie leicht zugreifen, indem Sie die Datei in Ihr Programm einbinden. Es folgt ein Beispiel:

```
...
<body>
   <script src="externe_datei.js"></script>
   <script>

      document.write("<p>Das kommt aus extern.htm</p>");
   </script>
</body></html>
```

Listing 1.5 Datei »extern.htm«

Attribut »src«

Der erste `script`-Container ist leer. Allerdings wird das Attribut `src` mit dem Wert `externe_datei.js` notiert. Damit wird der Code aus der betreffenden Datei in die Datei *extern.htm* eingebunden. In der Datei *externe_datei.js* steht lediglich der folgende Code:

```
document.write("<p>Das kommt aus externe_datei.js</p>");
```

Listing 1.6 Datei »externe_datei.js«

In Abbildung 1.5 sehen Sie die beiden Absätze, die jeweils mithilfe der Methode `document.write()` aus dem zusammengefügten Programm erzeugt werden.

Abbildung 1.5 Zusätzliches JavaScript aus externer Datei

Endung »js«

Beachten Sie, dass in der externen Datei kein `script`-Container steht. Der Name dieser Datei kann eine beliebige Endung haben. Als Konvention hat sich die Endung *js* eingebürgert.

JavaScript-Bibliotheken

Auf die genannte Weise werden die Bibliothek jQuery (siehe Kapitel 11) und andere große JavaScript-Bibliotheken mit ihren vielen nützlichen Funktionen in Anwendungen eingebunden.

1.10 Kommentare

Zur Beschreibung

Kommentare dienen zur Beschreibung der einzelnen Teile Ihrer Programme. Sie erleichtern Ihnen und anderen das Verständnis eines Programms. Betrachten wir ein Beispiel:

```
...
<body>
   <!-- Das ist ein Kommentar
        im HTML-Bereich -->
   <p>Ein Absatz aus dem HTML-Bereich</p>
   <script>

      /* Das ist ein Kommentar über mehrere Zeilen
         im JavaScript-Bereich */
      document.write("<p>Ein Absatz aus dem JS-Bereich</p>");
      // Ein kurzer Kommentar, nur bis zum Zeilenende

   </script>
</body></html>
```

Listing 1.7 Datei »kommentar.htm«

Im Beispiel sehen Sie drei verschiedene Arten von Kommentaren:

<!-- ... -->

- Ein Kommentar im HTML-Bereich kann sich über eine oder über mehrere Zeilen erstrecken. Er steht zwischen den Zeichenfolgen `<!--` und `-->`.

/* ... */

- Im JavaScript-Bereich wird ein Kommentar, der über eine oder mehrere Zeilen geht, zwischen den Zeichenfolgen `/*` und `*/` notiert.

// ...

- Möchten Sie einen kurzen Kommentar im JavaScript-Bereich notieren, beispielsweise hinter einer Anweisung, so eignet sich die Zeichenfolge `//`. Ein solcher einzeiliger Kommentar geht nur bis zum Ende der jeweiligen Zeile.

Im Quelltext sichtbar

Der Inhalt der Kommentare wird nicht im Browser dargestellt, siehe Abbildung 1.6. Allerdings kann jeder Benutzer bei Bedarf den Quelltext einer Seite in seinem Browser ansehen und damit auch die Kommentare.

Hinweis

Häufig möchten Sie ein Programm, das von Ihnen oder von jemand anderem stammt, nach längerer Zeit noch einmal ansehen oder erweitern. Dann werden Sie für jede Zeile Kommentar dankbar sein, die Sie darin vorfinden. Aus den gleichen Gründen ist es auch sehr zu empfehlen, übersichtliche, leicht lesbare Programme zu schreiben.

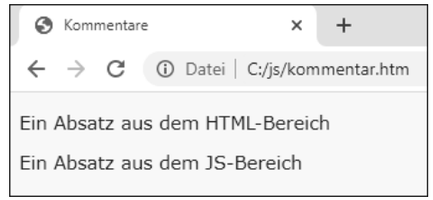


Abbildung 1.6 Kommentare sind nicht sichtbar.

1.11 Kein JavaScript möglich

Einschalten nicht möglich

Wie bereits in Abschnitt 1.2, »Was kann JavaScript nicht?«, erwähnt: Es wird immer einzelne Benutzer geben, die JavaScript in ihrem Browser ausgeschaltet haben. Was können wir dann machen? Es ist nicht möglich, mithilfe eines Programms JavaScript beim Benutzer einzuschalten.

Erkennung möglich

Wir können aber erkennen, ob es eingeschaltet ist oder nicht. Ist es nicht eingeschaltet, können wir entweder eine einfache Version der Seite in reinem HTML anbieten oder einen Hinweis geben, dass die Nutzung der betreffenden Seite das Einschalten von JavaScript voraussetzt.

Ein Beispiel:

```
...
<body>
   <script>

      document.write("<p>Hier läuft JavaScript</p>");

   </script>
   <noscript>
      <p>Hier läuft JavaScript nicht<br>
         Bitte schalten Sie es ein</p>
   </noscript>
</body></html>
```

Listing 1.8 Datei »kein_script.htm«

noscript

Innerhalb des `noscript`-Containers können Sie Text und HTML-Markierungen für diejenigen Benutzer notieren, bei denen JavaScript ausgeschaltet ist.

Bei eingeschaltetem JavaScript werden nur die Anweisungen aus dem `script`-Container ausgeführt. Die Seite sieht dann aus wie in Abbildung 1.7.

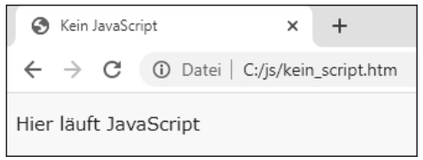


Abbildung 1.7 JavaScript ist eingeschaltet.

JavaScript ausschalten

Sie können JavaScript einmal zu Testzwecken ausschalten. Die notwendige Vorgehensweise wird dazu nachfolgend am Beispiel des Browsers Google Chrome erläutert. Rufen Sie über die drei Punkte oben rechts das Menü auf. Wählen Sie den Menüpunkt EINSTELLUNGEN. Geben Sie im Suchfenster den Begriff »JavaScript« ein. In den Suchergebnissen finden Sie die WEBSITE-EINSTELLUNGEN. Dort finden Sie den Eintrag JAVASCRIPT, standardmäßig mit dem Eintrag ZUGELASSEN. Wählen Sie den Eintrag über den Pfeil rechts aus, und stellen Sie den Schalter auf BLOCKIERT.

Allgemein oder spezifisch einschalten

Anschließend sehen Sie beim Aufruf einer Datei, die JavaScript beinhaltet, ganz rechts in der Adresszeile des Browsers ein Symbol mit der Information, dass JavaScript auf dieser Internetseite blockiert wurde. Klicken Sie auf das Symbol, öffnet sich ein Dialogfeld. Hier haben Sie die Möglichkeit, JavaScript *für diese Seite* einzuschalten. Über die Schaltfläche VERWALTEN gelangen Sie auch unmittelbar zu dem oben genannten Schalter zum Ein- und Ausschalten von JavaScript, und zwar allgemein oder für einzelne Seiten.

Bei ausgeschaltetem JavaScript sieht die Seite *kein_script.htm* aus wie in Abbildung 1.8.

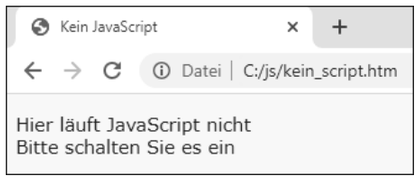


Abbildung 1.8 JavaScript ist ausgeschaltet.

Kapitel 3
Eigene Objekte

Sie erschaffen eigene Objekte und verstehen ganz nebenbei besser den Aufbau der vorhandenen Objekte.

Vordefinierte Objekte

In JavaScript programmieren Sie objektorientiert. Es gibt eine ganze Reihe vordefinierter Objekte:

- Sie kennen bereits die Objekte `window`, `document` und `Math`.
- In Abschnitt 4.8, »Wechsel des Dokuments«, nutzen Sie das Objekt `location`.
- In Kapitel 6, »Standardobjekte nutzen«, werden weitere vordefinierte Objekte erläutert.

Eigene Objekte

Sie können aber auch eigene Objekte erzeugen, so wie es in diesem Kapitel beschrieben wird. Damit haben Sie die Möglichkeit, thematisch zusammengehörige Daten zu bündeln und auf diese zuzugreifen. Das erleichtert Ihnen das Verständnis und den Umgang mit den zahlreichen vordefinierten Objekten. Benötigen Sie spezielle Möglichkeiten in Ihren Anwendungen, können Sie sowohl eigene als auch vordefinierte Objekte mithilfe der Vererbung erweitern.

Prototypen oder Klassen

Klassisch geschieht die Erzeugung eigener Objekte in JavaScript mithilfe von Prototypen und Konstruktorfunktionen. Seit der Einführung des Standards ECMAScript 2015 gibt es eine zusätzliche Schreibweise, in der mit Klassen, Konstruktoren, Eigenschaften und Methoden gearbeitet wird, wie Sie es eventuell bereits aus anderen objektorientierten Sprachen kennen. Diese Schreibweise wird im vorliegenden Kapitel eingesetzt. Alle modernen Browser setzen diese Schreibweise um. In Abschnitt 3.6, »Prototypen und Konstruktorfunktionen«, wird zusätzlich auf eine Reihe von Programmen in klassischer Schreibweise verwiesen.

JSON

Am Ende des Kapitels wird auf die kompakte *JavaScript Object Notation* (JSON) eingegangen. Sie vereinfacht den Transport von Daten zwischen verschiedenen Anwendungen.

3.1 Objekte und Eigenschaften

Klasse und Objekt

In diesem Abschnitt sehen Sie, wie Sie eine Klasse mit dem Namen `Fahrzeug` und den Eigenschaften `farbe` und `geschwindigkeit` definieren. Anschließend werden zwei Objekte dieser Klasse erzeugt und ausgegeben. Verschiedene Objekte derselben Klasse sind miteinander verwandt. Sie haben dieselben Eigenschaften, allerdings mit unterschiedlichen Werten.

Es folgt das Programm:

```
... <head> ...
   <script>

      class Fahrzeug
      {
         constructor(f, g)
         {
            this.farbe = f;
            this.geschwindigkeit = g;
         }
      }

   </script>
</head>
<body>
   <p><script>

      const dacia = new Fahrzeug("Rot", 50);
      document.write("Farbe: " + dacia.farbe
        + ", Geschwindigkeit: " + dacia.geschwindigkeit + "<br>");

      dacia.geschwindigkeit = 75;
      document.write("Farbe: " + dacia.farbe
         + ", Geschwindigkeit: " + dacia.geschwindigkeit + "<br>");

      let renault = new Fahrzeug("Gelb", 65);
      renault = new Fahrzeug("Blau", 85);
      document.write("Farbe: " + renault.farbe
         + ", Geschwindigkeit: " + renault.geschwindigkeit);
```

```
   </script></p>
</body></html>
```

Listing 3.1 Datei »obj_eigenschaft.htm«

class Das Schlüsselwort `class` leitet die Definition der Klasse ein. Beachten Sie, dass nach dem Namen der Klasse keine runden Klammern folgen, wie dies bei einer Funktion der Fall wäre.

constructor Die Konstruktormethode dient zum Erzeugen eines Objekts der Klasse. Sie besitzt den festgelegten Namen `constructor`. Ihre Parameter entsprechen den Eigenschaften eines Objekts der Klasse.

this Innerhalb der Konstruktormethode werden zwei Eigenschaften festgelegt, jeweils mit dem Schlüsselwort `this`, dem Operator `.` (Punkt) und einem Namen. Den beiden Eigenschaften werden die Werte zugewiesen, die beim Erzeugen eines Objekts der Klasse `Fahrzeug()` als Parameter übergeben werden. Das Schlüsselwort `this` verweist auf *dieses Objekt*, also auf das aktuelle Objekt. Damit wird erreicht, dass auf eine Eigenschaft des aktuellen Objekts zugegriffen wird oder eine Methode für das aktuelle Objekt ausgeführt wird.

new Im Programm wird die unveränderliche Variable `dacia` deklariert. Ihr wird mithilfe des Schlüsselworts `new` ein Verweis auf ein neues Objekt der Klasse `Fahrzeug()` zugewiesen. Dabei werden zwei Werte als Anfangswerte der beiden Eigenschaften übergeben. Sie können anschließend mithilfe der Punktschreibweise auf die Eigenschaften des Objekts zugreifen. Die Werte können ausgegeben oder verändert werden.

Verweis veränderlich Anschließend wird die veränderliche Variable `renault` deklariert. Dieser Variablen wird zweimal nacheinander ein Objekt der Klasse `Fahrzeug` zugewiesen. Diese Objekte verfügen ebenfalls über die Eigenschaften `farbe` und `geschwindigkeit`. Die Eigenschaftswerte der Objekte unterscheiden sich allerdings.

Verweis unveränderlich Bei der Deklaration von Variablen, die auf Objekte verweisen, sieht es bei Objekten aus wie bei Feldern: Verweist eine unveränderliche Variable auf ein Objekt, ist nur der Verweis unveränderlich. Sie können die einzelnen Elemente des Objekts verändern, aber der Variablen kein anderes Objekt zuweisen.

In Abbildung 3.1 sehen Sie die Ausgabe des Programms.

Abbildung 3.1 Objekte und Eigenschaften

3.2 Methoden

Funktion für Objekte Bei einer Methode handelt es sich um eine Funktion, die nur für Objekte derjenigen Klasse aufgerufen werden kann, in der sie definiert wurde. Ähnlich verhält es sich mit der Methode `write()`, die das definierte Verhalten nur zeigt, falls sie für das `document`-Objekt aufgerufen wird. Häufig dienen Methoden zum Verändern der Eigenschaften eines Objekts der betreffenden Klasse. Die Eigenschaften und die Methoden einer Klasse werden zusammen auch als Member dieser Klasse bezeichnet.

toString() Im folgenden Beispiel wird die Klasse `Fahrzeug` aus dem vorherigen Abschnitt erweitert. Es werden die beiden Methoden `lackieren()` und `beschleunigen()` definiert, die zur Änderung der Eigenschaften `farbe` und `geschwindigkeit` dienen. Außerdem wird eine besondere Methode mit dem festgelegten Namen `toString()` definiert. Sie ermöglicht eine einfache Ausgabe eines Objekts, ähnlich der Ausgabe einer Variablen.

Das Programm:

```
... <head> ...
   <script>

      class Fahrzeug
      {
         constructor(f, g)
         {
            this.farbe = f;
            this.geschwindigkeit = g;
         }

         beschleunigen(wert)
         {
```

```
            this.geschwindigkeit += wert;
         }

         lackieren(f)
         {
            this.farbe = f;
         }

         toString()
         {
            return "Farbe: " + this.farbe
               + ", Geschwindigkeit: " + this.geschwindigkeit;
         }
      }

   </script>
</head>
<body><p>
   <script>

      const dacia = new Fahrzeug("Rot", 50);
      document.write("Farbe: " + dacia.farbe
         + ", Geschwindigkeit: " + dacia.geschwindigkeit + "<br>");

      dacia.beschleunigen(35);
      dacia.lackieren("Blau");
      document.write(dacia);

   </script></p>
</body></html>
```

Listing 3.2 Datei »obj_methode.htm«

Eigene Methoden

Die Methode `beschleunigen()` erwartet einen Zahlenwert. Dieser dient zum Ändern des Werts der Eigenschaft `geschwindigkeit`. Die Methode `lackieren()` erwartet eine Zeichenkette, die den neuen Wert der Eigenschaft `farbe` darstellt.

toString()

Die Methode `toString()` liefert eine Zeichenkette, die die Werte aller Eigenschaften des Objekts enthält, und dient zur Ausgabe eines Objekts, wie Sie in Abbildung 3.2 sehen.

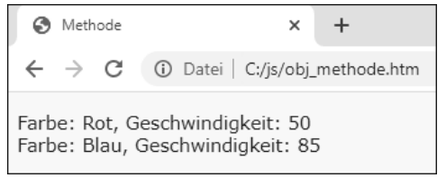


Abbildung 3.2 Nutzung von Methoden

Zu diesem Thema finden Sie die Übungsaufgabe »u_klasse« in den Zusatzmaterialien zum Download.

3.3 Objekt in Objekt

Eigenschaft ist Objekt

Die Eigenschaft eines Objekts einer Klasse kann wiederum ein Objekt einer anderen Klasse sein. Zur Verdeutlichung dieses Zusammenhangs wird die Klasse `Fahrzeug` um die Eigenschaft `antrieb` der Klasse `Motor` erweitert. Das Beispielprogramm:

```
... <head> ...
   <script>

      class Motor
      {
         constructor(l, z, k)
         {
            this.leistung = l;
            this.zylinder = z;
            this.kraftstoff = k;
         }

         tunen(x)
         {
            this.leistung += x;
         }

         toString()
         {
            return "Leistung: " + this.leistung + ", Zylinder: "
               + this.zylinder + ", Kraftstoff: " + this.kraftstoff;
         }
```

```
      }

      class Fahrzeug
      {
         constructor(f, g, a)
         {
            this.farbe = f;
            this.geschwindigkeit = g;
            this.antrieb = a;
         }

         toString()
         {
            return "Farbe: " + this.farbe + ", Geschwindigkeit: "
               + this.geschwindigkeit + ", Antrieb: " + this.antrieb;
         }
      }

   </script>
</head>
<body><p>
   <script>

      const dacia = new Fahrzeug("Rot", 50,
         new Motor(60, 4, "Diesel"));
      dacia.antrieb.tunen(10);
      document.write(dacia + "<br>");

      dacia.antrieb.leistung = 80;
      dacia.antrieb.zylinder = 6;
      dacia.antrieb.kraftstoff = "Benzin";
      document.write(dacia);

   </script></p>
</body></html>
```

Listing 3.3 Datei »obj_in_objekt.htm«

Ein Objekt der Klasse `Motor` hat die Eigenschaften `leistung`, `zylinder` und `kraftstoff`. Außerdem gibt es die Methode `tunen()` zur Veränderung der

Leistung und die Methode `toString()` zur Ausgabe der drei Eigenschaftswerte.

In der Klasse `Fahrzeug` ist die Eigenschaft `antrieb` hinzugekommen. Mithilfe der Methode `toString()` wird auch der Wert dieser Eigenschaft ausgegeben.

Parameter ist Objekt

Im Programm wird ein Objekt der Klasse `Fahrzeug` mithilfe von `new` erzeugt. Der dritte Parameter ist ein Objekt der Klasse `Motor`, das ebenfalls mit `new` erzeugt wird.

Mit dem ersten Punkt nach dem Namen des Objekts `dacia` wird die Eigenschaft `antrieb` des `Fahrzeug`-Objekts angesprochen, der zweite Punkt führt zur Untereigenschaft des `Motor`-Objekts.

Aufruf von »toString()«

Zur Ausgabe eines `Fahrzeug`-Objekts wird die Methode `toString()` der Klasse `Fahrzeug` aufgerufen. Diese ruft intern, mithilfe von `this.antrieb`, die gleichnamige Methode der Klasse `Motor` auf. Auf diese Weise wird die gesamte Zeichenkette zusammengesetzt, siehe Abbildung 3.3.

Abbildung 3.3 »Motor«-Objekt in »Fahrzeug«-Objekt

3.4 Vererbung

Klasse ableiten

Benötigen Sie eine Klasse, die Eigenschaften besitzt, die bereits in einer anderen Klasse definiert werden, können Sie das Prinzip der Vererbung benutzen. Sie erschaffen eine abgeleitete Klasse auf der Grundlage einer Basisklasse und fügen weitere Eigenschaften hinzu.

Basisklasse

Im folgenden Beispiel wird die abgeleitete Klasse `Lastwagen` erschaffen, auf der Grundlage der Basisklasse `Fahrzeug`. Ein Objekt der Klasse `Lastwagen` soll die zusätzliche Eigenschaft `nutzlast` und die zusätzliche Methode `beladen()` haben. Außerdem verfügen beide Klassen jeweils über eine eigene Definition der Methode `toString()`.

Es folgt das Programm:

```
... <head> ...
   <script>
```

```
      class Fahrzeug
      {
         constructor(f, g)
         {
            this.farbe = f;
            this.geschwindigkeit = g;
         }

         beschleunigen(wert)
         {
            this.geschwindigkeit += wert;
         }

         toString()
         {
            return "Farbe: " + this.farbe
               + ", Geschwindigkeit: " + this.geschwindigkeit;
         }
      }

      class Lastwagen extends Fahrzeug
      {
         constructor(f, g, n)
         {
            super(f, g);
            this.nutzlast = n;
         }

         beladen(wert)
         {
            this.nutzlast += wert;
         }

         toString()
         {
            return super.toString() + ", Nutzlast: " + this.nutzlast;
         }
      }

   </script>
</head>
```



```
<body><p>
   <script>

      const iveco = new Lastwagen("Orange", 30, 15);
      document.write(iveco + "<br>");

      iveco.beschleunigen(50);
      iveco.beladen(25);
      document.write(iveco);

   </script></p>
</body></html>
```

Listing 3.4 Datei »obj_vererbung.htm«

Konstruktor

Der Konstruktor der Basisklasse `Fahrzeug` erwartet zwei Parameter für die Anfangswerte der Eigenschaften `farbe` und `geschwindigkeit`. Der Konstruktor der abgeleiteten Klasse `Lastwagen` erwartet drei Parameter. Dabei handelt es sich um die Anfangswerte für die Eigenschaften `farbe` und `geschwindigkeit` der Basisklasse `Fahrzeug` und für die Eigenschaft `nutzlast` der abgeleiteten Klasse `Lastwagen`.

super()

Die Methode `super()` ruft innerhalb eines Konstruktors den jeweiligen Konstruktor der Basisklasse auf. Auf diese Weise werden hier die ersten beiden Parameter an den Konstruktor der Basisklasse weitergereicht. Der dritte Parameter wird der Eigenschaft `nutzlast` zugewiesen. Auf diese Weise erhalten alle Eigenschaften des Objekts der Klasse `Lastwagen` einen Anfangswert.

super

Zudem gibt es den Verweis `super` (ohne Klammern), der für den Zugriff auf die Eigenschaften und Methoden der Basisklasse dient. In der Methode `toString()` der abgeleiteten Klasse `Lastwagen` wird damit die gleichnamige Methode der Basisklasse `Fahrzeug` aufgerufen: `super.toString()`.

Im Programm wird ein Objekt der Klasse `Lastwagen` mit drei Anfangswerten erzeugt und ausgegeben. Es wird durch den Aufruf der Methode `beschleunigen()` der Basisklasse und durch den Aufruf der Methode `beladen()` der abgeleiteten Klasse verändert. Anschließend wird es erneut ausgegeben, siehe Abbildung 3.4.

Zu diesem Thema finden Sie die Übungsaufgabe »u_vererbung« in den Zusatzmaterialien zum Download.

Abbildung 3.4 Vererbung

3.5 Operationen mit Objekten

Operationen und Prüfungen

In diesem Abschnitt sehen Sie ein Beispielprogramm, in dem eine Reihe von nützlichen Operationen und Prüfungen bezüglich einer Klasse, ihrer Eigenschaften und Methoden und einiger Objekte dieser Klasse durchgeführt wird. Die Klasse wird im `head` des Dokuments definiert, und zwar wie folgt:

```
... <head> ...
   <script>

      class Fahrzeug
      {
         constructor(f, g)
         {
            this.farbe = f;
            this.geschwindigkeit = g;
         }

         beschleunigen(wert)
         {
            this.geschwindigkeit += wert;
         }

         lackieren(f)
         {
            this.farbe = f;
         }

         toString()
         {
            return "Farbe: " + this.farbe
                 + ", Geschwindigkeit: " + this.geschwindigkeit;
         }
      }

   </script>
</head>
```

Listing 3.5 Datei »obj_operation.htm«, Teil 1 von 8

3.5.1 Zugriffsoperatoren

».« oder »[]«

Sie haben zwei Möglichkeiten, auf die Eigenschaften eines Objekts zuzugreifen: zum einen mit dem Operator . (Punkt), zum anderen mithilfe der rechteckigen Klammern []. Dies sehen Sie in folgendem Programmteil:

```
<body><p>
   <script>

      const dacia = new Fahrzeug("Rot", 50);
      document.write("Schreibweise mit Punkt: " + dacia.farbe + " "
         + dacia.geschwindigkeit + "<br>");
      document.write("Schreibweise mit [ und ]: " + dacia["farbe"]
         + " " + dacia["geschwindigkeit"] + "<br>");
      document.write("Schreibweise mit toString(): "
         + dacia + "<br><br>");
```

Listing 3.6 Datei »obj_operation.htm«, Teil 2 von 8

Rechteckige Klammern

Nach Erzeugung eines Objekts werden seine Eigenschaften mehrmals ausgegeben. Innerhalb der rechteckigen Klammern wird der Name der Eigenschaft wie eine Zeichenkette innerhalb von doppelten Hochkommata notiert. Sie können diese Zeichenkette aus einzelnen Teilen zusammensetzen, auch mithilfe von Variablen. Diese Schreibweise macht die Erstellung von Programmen noch ein Stück flexibler. Die Ausgabe sehen Sie, neben anderen, in Abbildung 3.5.

3.5.2 Verweise auf Objekte erzeugen und vergleichen

Zweiter Verweis

Weisen Sie eine Variable, die auf ein Objekt verweist, einer anderen Variablen zu, so erschaffen Sie kein neues Objekt und auch keine Kopie des ursprünglichen Objekts, sondern nur einen zweiten Verweis auf dasselbe Ob-

jekt. Sie können anschließend über beide Variablen auf dasselbe Objekt zugreifen, siehe auch Abbildung 3.5.

Verweise vergleichen

Sie können mithilfe der beiden Vergleichsoperatoren == bzw. != prüfen, ob zwei Variablen auf dasselbe Objekt verweisen.

Im folgenden Programmteil werden Variablen miteinander verglichen:

- zum einen zwei Variablen, die auf dasselbe Objekte verweisen
- zum anderen zwei Variablen, die auf zwei verschiedene Objekte mit denselben Eigenschaften und Eigenschaftswerten verweisen

```
const renault = dacia;
document.write("Zweiter Verweis: " + renault + "<br>");
if(renault == dacia)
   document.write("Dasselbe Objekt<br>");
const simca = new Fahrzeug("Rot", 50);
if(simca != dacia)
   document.write("Nicht dasselbe Objekt<br><br>");
```

Listing 3.7 Datei »obj_operation.htm«, Teil 3 von 8

Diese Ausgabe sehen Sie ebenfalls in Abbildung 3.5.

Abbildung 3.5 Zugriffsmöglichkeiten und Vergleiche

3.5.3 Instanzen prüfen

instanceof

Objekte einer Klasse werden als *Instanzen* einer Klasse bezeichnet. Der Vorgang des Erzeugens eines Objekts wird *Instantiieren* genannt. Der Operator `instanceof` prüft, ob ein Objekt die Instanz einer bestimmten Klasse (oder deren Basisklasse) darstellt:


```
if(dacia instanceof Fahrzeug)
   document.write("instanceof:<br>Objekt "
      + " ist Instanz der Klasse<br><br>");
```

Listing 3.8 Datei »obj_operation.htm«, Teil 4 von 8

Die Ausgabe sehen Sie in Abbildung 3.6.

3.5.4 Typ ermitteln

»function« und »object«

Der Operator `typeof` liefert den Typ einer Variablen, siehe auch Abschnitt 2.3.5, »Wert und Typ prüfen«. Die Methoden einer Klasse haben den Typ *function*, die Objekte einer Klasse den Typ *object*. Die Klassen in JavaScript basieren klassisch auf Prototypen und Konstruktorfunktionen. Daher haben auch die Klassen selbst den Typ *function*, wie nachfolgend gezeigt wird:

```
document.write("typeof:<br>"
   + "Fahrzeug: " + typeof Fahrzeug + "<br>"
   + "dacia: " + typeof dacia + "<br>"
   + "farbe: " + typeof dacia.farbe + "<br>"
   + "geschwindigkeit: "
      + typeof dacia.geschwindigkeit + "<br>"
   + "beschleunigen: " + typeof dacia.beschleunigen + "<br>"
   + "lackieren: " + typeof dacia.lackieren + "<br><br>");
```

Listing 3.9 Datei »obj_operation.htm«, Teil 5 von 8

In Abbildung 3.6 sehen Sie, dass die Klasse `Fahrzeug` und die beiden Methoden `beschleunigen()` und `lackieren()` den Typ *function* haben. Das Objekt `dacia` hat den Typ *object*. Die beiden Eigenschaften sind einfache Variablen des Typs *string* bzw. *number*.

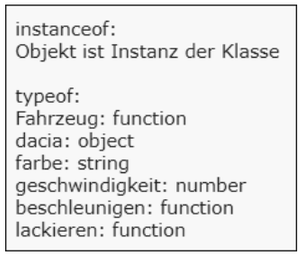


Abbildung 3.6 Operatoren »instanceof« und »typeof«

3.5.5 Member prüfen

Operator »in«

Mithilfe des Operators `in` prüfen Sie, ob eine Klasse ein bestimmtes Member, also eine bestimmte Eigenschaft oder Methode, besitzt:

```
document.write("in:<br>");
if("farbe" in dacia)
   document.write("farbe ist Member<br>");
if("beschleunigen" in dacia)
   document.write("beschleunigen ist Member<br>");
if("lackieren" in dacia)
   document.write("lackieren ist Member<br>");
if(!("leistung" in dacia))
   document.write("leistung ist kein Member<br><br>");
```

Listing 3.10 Datei »obj_operation.htm«, Teil 6 von 8

Die Prüfung ergibt, dass die Zeichenketten `farbe`, `beschleunigen` und `lackieren` Member der Klasse des Objekts `dacia` sind, `leistung` jedoch nicht, siehe Abbildung 3.7.

3.5.6 Objekte und Funktionen

Parameter

Sie können eine Variable, die einen Verweis auf ein Objekt darstellt, als Parameter an eine Funktion übergeben. Innerhalb der Funktion können Sie das Objekt ändern bzw. Methoden des Objekts aufrufen.

Rückgabewert

Sie können auch eine Variable, die einen Verweis auf ein Objekt darstellt, als Rückgabewert aus einer Funktion zurückliefern und einer weiteren Variablen zuweisen:

```
function aendern(x)
{
   x.beschleunigen(10);
   return x;
}
const lada = aendern(dacia);
document.write(lada + "<br><br>");
```

Listing 3.11 Datei »obj_operation.htm«, Teil 7 von 8

Die Variable `dacia`, der Parameter `x` und die Variable `lada` verweisen auf dasselbe Objekt. Die Ausgabe sehen Sie ebenfalls in Abbildung 3.7.


```
in:
farbe ist Member
beschleunigen ist Member
lackieren ist Member
leistung ist kein Member

Farbe: Rot, Geschwindigkeit: 60
```

Abbildung 3.7 Operator »in«, Objekte und Funktionen

3.5.7 Eigenschaften löschen

Operator »delete«

Sie können einzelne Eigenschaften eines Objekts mithilfe des Operators `delete` löschen. Die Eigenschaften anderer Objekte derselben Klasse sind davon nicht betroffen. Das hat allerdings den Nebeneffekt, dass sich die Objekte nicht mehr ähnlich sind.

Es folgt der letzte Teil des Programms:

```
   delete dacia.geschwindigkeit;
   document.write("Werte der Member:<br>"
      + "farbe: " + dacia.farbe + "<br>"
      + "geschwindigkeit: " + dacia.geschwindigkeit + "<br>"
      + "lackieren: " + dacia.lackieren + "<br>"
      + "leistung: " + dacia.leistung);


  </script></p>
</body></html>
```

Listing 3.12 Datei »obj_operation.htm«, Teil 8 von 8

Die Eigenschaft `geschwindigkeit` wird gelöscht. Anschließend werden die Werte der Eigenschaften `farbe`, `geschwindigkeit`, `lackieren` und `leistung` ausgegeben. Die Eigenschaft `geschwindigkeit` existiert nicht mehr, die Eigenschaft `leistung` hat es nie gegeben. Die resultierende Ausgabe sehen Sie in Abbildung 3.8.

```
Werte der Member:
farbe: Rot
geschwindigkeit: undefined
lackieren: lackieren(f) { this.farbe = f; }
leistung: undefined
```

Abbildung 3.8 Eigenschaftswerte nach Löschvorgang

3.6 Prototypen und Konstruktorfunktionen

Klassische Schreibweise

Die klassische Schreibweise für die objektorientierte Programmierung in JavaScript arbeitet mit Prototypen und Konstruktorfunktionen. Im Downloadpaket zum Buch finden Sie vergleichbare Programme zu den bisherigen Programmen dieses Kapitels in klassischer Schreibweise. Sie können auch weiterhin in allen modernen Browsern genutzt werden.

Es handelt sich um die folgenden Dateien:

- Objekte und Eigenschaften: *obj_eigenschaft_prot.htm*
- Methoden: *obj_methode_prot.htm*
- Objekt in Objekt: *obj_in_objekt_prot.htm*
- Vererbung: *obj_vererbung_prot.htm*
- Operationen mit Objekten: *obj_operation_prot.htm*

3.7 Objekte in JSON

Objekt kompakt speichern

JSON steht für *JavaScript Object Notation*. Dies ist der Name für eine kompakte Schreibweise, die auf Objekte und auf Felder angewendet werden kann. Die JSON-Schreibweise ermöglicht Ihnen die Speicherung der Eigenschaften eines Objekts mitsamt deren Werten innerhalb einer einzigen Zeichenkette. Auf diese Weise wird der Transport von Daten zwischen verschiedenen Anwendungen vereinfacht.

JSON-Objekt

Moderne Browser kennen das `JSON`-Objekt. Es besitzt Methoden, mit deren Hilfe Sie ein Objekt in eine transportable Zeichenkette umwandeln können. Dies gilt unabhängig davon, ob das Objekt mithilfe von JSON oder mithilfe einer Konstruktorfunktion erzeugt wird. Umgekehrt können Objekte aus einer solchen Zeichenkette erschaffen werden.

Im folgenden Beispiel wird ein Objekt im JSON-Format angelegt. Anschließend wird es mithilfe des `JSON`-Objekts umgewandelt.

```
...
<body><p>
   <script>

      const dacia = { "farbe":"Rot", "geschwindigkeit":50.2 };
      document.write(dacia.farbe + " "
         + dacia.geschwindigkeit + "<br>");

      const zkette = JSON.stringify(dacia);
      document.write(zkette + "<br>");

      const renault = JSON.parse(zkette);
      document.write(renault.farbe + " "
         + renault.geschwindigkeit + "<br>");
      renault.geschwindigkeit += 2.5;
      document.write("Schneller: " + renault.geschwindigkeit);

   </script></p>
</body></html>
```

Listing 3.13 Datei »obj_json.htm«

Aufbau

Bei Erzeugung eines Objekts in der kompakten Schreibweise steht das gesamte Objekt in geschweiften Klammern. Die einzelnen Eigenschaft-Wert-Paare sind durch Kommata getrennt. Die Eigenschaft und der Wert eines Paars werden durch einen Doppelpunkt getrennt. Der Name der Eigenschaft wird in doppelte Hochkommata gesetzt. Handelt es sich bei dem Wert um eine Zeichenkette, geschieht dies ebenfalls. Ein Zahlenwert wird dagegen ohne Hochkommata notiert. Nachkommastellen werden mithilfe des Dezimalpunkts abgetrennt.

stringify(), parse()

Ein Objekt wird mithilfe der Methode `stringify()` in eine Zeichenkette umgewandelt. Es folgt der umgekehrte Vorgang. Eine Zeichenkette wird mithilfe der Methode `parse()` in ein Objekt umgewandelt. Ein Zahlenwert kann auch in eine Berechnung einbezogen werden.

Die Ausgabe des Programms sehen Sie in Abbildung 3.9.

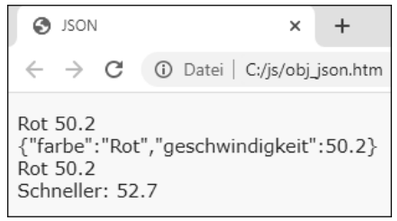


Abbildung 3.9 JSON-Format und JSON-Objekt

In Abschnitt 6.1.11 zeige ich Ihnen die Anwendung von JSON auf Felder und Objekte. In Abschnitt 7.4 werden JSON-Dateien gelesen.

Kapitel 5
Das Document Object Model (DOM)

Das DOM ermöglicht den lesenden und schreibenden Zugriff auf alle Elemente eines HTML-Dokuments.

Das *Document Object Model* (DOM) stellt ein Modell für den Zugriff auf die Elemente eines Dokuments dar, z. B. auf ein XML- oder ein HTML-Dokument.

Baum mit Knoten

Das Modell ist aufgebaut wie ein Baum, ähnlich einem Verzeichnisbaum. Es besteht aus einzelnen Knoten (englisch: *nodes*). Es gibt einen Wurzelknoten, der Kindknoten besitzt. Die Kindknoten können wiederum Kindknoten haben usw. Jeder Knoten kann außerdem Attributknoten besitzen.

Knotentypen

In JavaScript wird jeder Knoten durch ein `node`-Objekt repräsentiert. Es kann sich dabei um einen HTML-Elementknoten, einen HTML-Attributknoten oder einen Textknoten handeln. Sie können gelesen, verändert, hinzugefügt und gelöscht werden.

Objekt »document«

Der Inhalt des Dokuments ist in JavaScript über das `document`-Objekt erreichbar. Es stellt Methoden zur Verfügung, mit denen Sie Objekte des Dokuments neu erzeugen und auf sie zugreifen. Sie haben bereits häufig die Methode `getElementById()` verwendet.

In diesem Kapitel sehen Sie, wie Sie auf die Elemente eines HTML-Dokuments zugreifen und diese verändern. Das ist Voraussetzung für viele Techniken, die in den nachfolgenden Kapiteln behandelt werden.

5.1 Baum und Knoten

In diesem Abschnitt soll der DOM-Baum eines einfachen HTML-Dokuments gezeigt werden. Zunächst der HTML-Code:

```
<!DOCTYPE html><html lang="de">
<head>
   <meta charset="utf-8">
   <title>DOM, erstes Beispiel</title>
   <link rel="stylesheet" href="js4.css">
</head>
<body>
   <p>Erster Absatz</p>
   <p>Zweiter Absatz mit <span style="font-weight:bold;">
      Fettschrift</span></p>
   <p>Dritter Absatz mit <a href="einbetten.htm">einem Link</a></p>
</body>
</html>
```

Listing 5.1 Datei »dom_beispiel.htm«

Drei Absätze

Das Dokument besteht aus drei p-Containern, die teilweise weitere Container enthalten. Im Browser sieht es aus wie in Abbildung 5.1.

Abbildung 5.1 Einfaches Beispiel für DOM-Baum

Moderne Browser bieten Ihnen die Möglichkeit, sich den Aufbau des Dokuments gemäß DOM zu verdeutlichen. Im Browser Google Chrome geht dies über die Entwicklertools, die Sie bereits in Abschnitt 2.5.5, »Programm debuggen«, kennengelernt haben.

Entwicklertools

Es wird davon ausgegangen, dass der Browser mit dem oben angegebenen Programm geöffnet ist. Öffnen Sie das Menü des Browsers über die drei Punkte oben rechts. Wählen Sie darin den Menüpunkt Weitere Tools • Entwicklertools. Anschließend werden die Entwicklertools eingeblendet, die viele verschiedene Möglichkeiten bieten.

Elemente ein- und ausblenden

Wählen Sie im oberen Bereich die Registerkarte Elements. Wie bei einem Verzeichnisbaum können Sie nun hier die einzelnen Ebenen ein- oder ausblenden, siehe Abbildung 5.2.

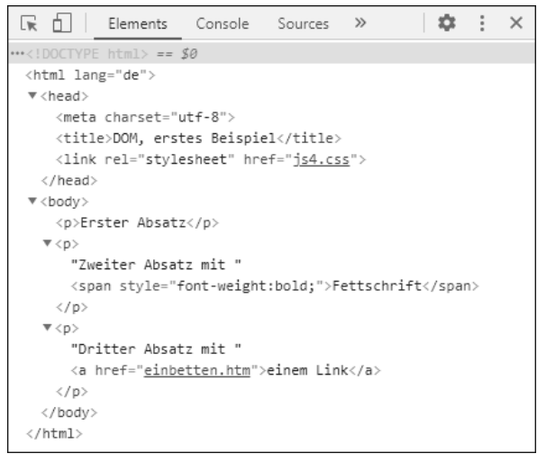


Abbildung 5.2 Entwicklerwerkzeuge, Elements

Hinweis

In den folgenden Beispielen betrachten oder verändern wir einzelne Knoten auf verschiedenen Ebenen. Zur Verdeutlichung sollten Sie sich stets den Dokumentaufbau gemäß DOM vor Augen halten.

5.2 Knoten abrufen

getElementsByTagName()

Neben der Methode `getElementById()` wird auch die Methode `getElementsByTagName()` eingesetzt. Parameter der Methode ist eine Zeichenkette, die den Namen der Markierung ohne spitze Klammern enthält. Die Methode liefert ein Feld mit Verweisen auf alle HTML-Elemente mit der gewünschten Markierung (englisch: *tag*).

Feld, Index, Länge

Die Elemente des Felds sind mit einem Index nummeriert, beginnend bei 0. Ein einzelnes Feldelement erreichen Sie über seinen Index in rechteckigen Klammern. Die Eigenschaft `length` eines Felds liefert die Anzahl der Elemente. Mehr zu Feldern sehen Sie in Abschnitt 6.1.

getAttribute()

Die Attributknoten eines HTML-Elements erhalten Sie über die Methode `getAttribute()` des betreffenden `node`-Objekts. Parameter der Funktion ist eine Zeichenkette, die den Namen des gesuchten Attributs enthält.

Es folgt ein Beispielprogramm:

```
... <head> ...
   <script>

      function abrufen()
      {
         let ausgabe = "";
         const absatzFeld = document.getElementsByTagName("p");
         for (let i=0; i<absatzFeld.length; i++)
            ausgabe += "Inhalt: "
               + absatzFeld[i].firstChild.nodeValue + "\n"
               + "Attribut Style: "
               + absatzFeld[i].getAttribute("style") + "\n";
         alert(ausgabe);
      }

   </script>
</head>
<body>
   <p style="font-family:Tahoma;">Erster Absatz</p>
   <p>Zweiter Absatz</p>
   <p style="font-weight:bold;">Dritter Absatz</p>
   <form>
      <input type="button" id="idAbrufen" value="Abrufen">
   </form>
   <script>

      document.getElementById("idAbrufen")
         .addEventListener("click", abrufen);

   </script>
</body>
</html>
```

Listing 5.2 Datei »dom_abrufen.htm«

Das Dokument enthält im HTML-Teil drei `p`-Container. Daher besitzt das Feld `absatzFeld` drei Elemente.

Kindknoten

Auf den ersten Kindknoten eines Knotens können Sie über die Eigenschaft `firstChild` eines `node`-Objekts zugreifen. Die drei Absätze besitzen jeweils nur einen Textknoten als Kindknoten.

Wert eines Knotens

Die Eigenschaft `nodeValue` eines `node`-Objekts liefert zu einem Textknoten den zugehörigen Text als Wert. Der Aufruf der Methode `getAttribute("style")` gibt den Wert des Attributs `style` eines HTML-Elementknotens zurück. Der zweite Absatz besitzt dieses Attribut nicht, daher wird `null` geliefert.

Die Ausgabe des Programms sehen Sie in Abbildung 5.3. Nach dem Betätigen der Schaltfläche erscheint ein Dialogfeld mit einer Ausgabe wie in Abbildung 5.4.

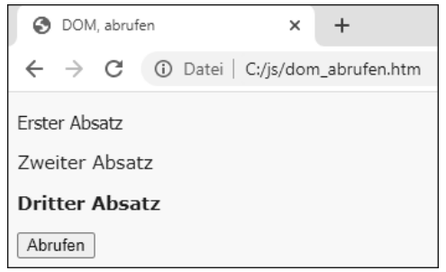


Abbildung 5.3 Aufbau des Dokuments mit drei Absätzen

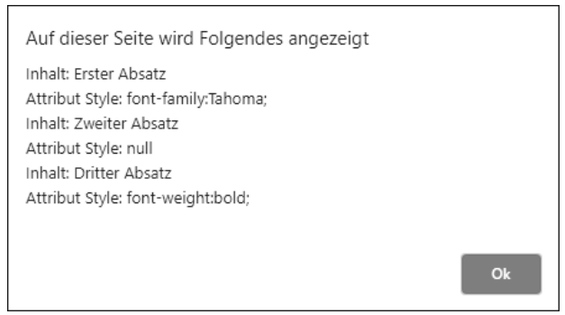


Abbildung 5.4 Abrufen von Knoten und Knotenwerten

5.3 Kindknoten

Objekt »node«

Sowohl ein Knoten als auch seine Kindknoten werden mithilfe eines `node`-Objekts repräsentiert. Die Methode `hasChildNodes()` liefert die Information, ob ein Knoten Kindknoten hat oder nicht. Die Eigenschaft `childNodes` eines Knotens enthält ein Feld mit den Kindknoten eines Knotens.

Typ und Wert eines Knotens

Die Eigenschaft `nodeType` enthält den Typ des Knotens. Der Wert 1 steht für einen Elementknoten, hier für einen HTML-Elementknoten, der Wert 3 für einen Textknoten. Der Textinhalt eines Textknotens steht in der Eigenschaft `nodeValue`. Der Name der Markierung eines HTML-Elementknotens steht in der Eigenschaft `nodeName`.

Rekursive Funktion

Im nachfolgenden Beispielprogramm wird mithilfe der genannten Eigenschaften und Methoden sowie einer rekursiven Funktion ein Teil des DOM-Baums des Dokuments durchlaufen:

```
... <head> ...
   <script>

      function knoten(verweis, ebene)
      {
         for(let i=0; i<verweis.childNodes.length; i++)
         {
            const ch = verweis.childNodes[i];
            if(ch.nodeType == 3)
               document.write("Ebene:" + ebene + ", Textwert: "
                  + ch.nodeValue + "<br>");
            else if(ch.nodeType == 1)
               document.write("Ebene:" + ebene + ", Elementname: "
                  + ch.nodeName + "<br>");
            if(ch.hasChildNodes())
            {
               ebene++;
               knoten(ch, ebene);
               ebene--;
            }
         }
      }

   </script>
</head>
```



```
<body>
   <p id="idAbsatz">
      Das <span style="font-style:italic;">ist</span> ein Absatz
      <span style="font-weight:bold;">mit Markierungen
      <span style="font-style:italic;">auf mehreren
      </span></span>Ebenen</p>
   <script>

      const ebene = 0;
      knoten(document.getElementById("idAbsatz"), ebene);

   </script>
</body></html>
```

Listing 5.3 Datei »dom_kinder.htm«

Erster Aufruf

Es wird derjenige Teil des DOM-Baums betrachtet, der den ersten Absatz beinhaltet. Die rekursive Funktion `knoten()` wird erstmals mit einem Verweis auf diesen Absatz aufgerufen.

Baum-Ebene

Die Variable `ebene` steht für die aktuelle Ebene innerhalb des Baums. Die Kindknoten des Absatzes stehen auf Ebene 0, deren Kindknoten auf Ebene 1 und wiederum deren Kindknoten auf Ebene 2.

Alle Kindknoten

In der Funktion `knoten()` wird das Feld mit den Kindknoten des aktuellen Elements mithilfe einer `for`-Schleife durchlaufen. Innerhalb der Schleife wird zur Verkürzung ein Verweis auf das aktuelle Feldelement eingerichtet. Handelt es sich bei dem Feldelement um einen Verweis auf einen Textknoten, wird dessen Wert ausgegeben. Handelt es sich dagegen um einen Verweis auf einen HTML-Elementknoten, wird dessen Name ausgegeben.

Erneuter Aufruf

Hat das aktuelle Feldelement selbst Kindknoten, wird die Nummer der Ebene erhöht, und es wird wiederum die Funktion `knoten()` aufgerufen.

Alle Zweige

Auf diese Weise können sämtliche Zweige des DOM-Baums durchlaufen werden. Irgendwann wird für jeden dieser Zweige festgestellt, dass keine Kindknoten vorhanden sind, und die Rekursion kehrt zurück. Die Nummer der Ebene wird wieder vermindert. Am Ende gelangt man wieder zur Ebene 0.

Die Ausgabe des Programms sehen Sie in Abbildung 5.5.

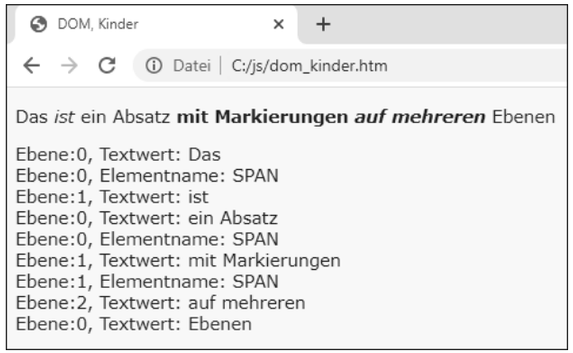


Abbildung 5.5 Kindknoten des ersten Absatzes

5.4 Knoten hinzufügen

In diesem Abschnitt werden Textknoten, HTML-Elementknoten und Attributknoten erzeugt und einem Dokument hinzugefügt. Folgende Methoden des `document`-Objekts werden genutzt:

createTextNode()
- Die Methode `createTextNode()` erzeugt einen Textknoten. Parameter der Methode ist eine Zeichenkette, die den Text enthält.

createElement()
- Die Methode `createElement()` erstellt einen HTML-Elementknoten. Parameter der Methode ist eine Zeichenkette mit der Markierung ohne spitze Klammern.

Außerdem werden drei Methoden für ein `node`-Objekt eingesetzt:

appendChild()
- `appendChild()` fügt einen Kindknoten zu einem Knoten hinzu, am Ende des Felds der Kindknoten. Parameter der Methode ist ein Verweis auf den Knoten, der hinzugefügt wird.

setAttribute()
- `setAttribute()` setzt den Wert eines Attributknotens. Existiert der Attributknoten noch nicht, wird er zuvor erzeugt. Parameter der Methode sind zwei Zeichenketten, die den Namen und den neuen Wert des Attributs enthalten.

insertBefore()
- `insertBefore()` fügt einen Kindknoten zu einem Knoten hinzu, vor einem anderen Kindknoten. Parameter der Methode sind zwei Verweise: ein Verweis auf den Knoten, der hinzugefügt wird, und ein Verweis auf den Knoten, vor dem eingefügt wird.

Es folgt das Programm:


```
... <head> ...
   <script>

      function anhaengen()
      {
         const text = document.createTextNode("Angehängt");
         const absatz = document.createElement("p");
         absatz.appendChild(text);
         absatz.setAttribute("style", "font-weight:bold;");
         document.getElementById("idBody").appendChild(absatz);
      }

      function einschieben()
      {
         const text = document.createTextNode("Eingeschoben");
         const absatz = document.createElement("p");
         absatz.appendChild(text);
         absatz.setAttribute("style", "font-style:italic;");
         document.getElementById("idBody").insertBefore(absatz,
            document.getElementById("idAbsatz3"));
      }

   </script>
</head>
<body id="idBody">
   <p id="idAbsatz1">Erster Absatz</p>
   <p id="idAbsatz2">Zweiter Absatz</p>
   <p id="idAbsatz3">Dritter Absatz</p>
   <p><input id="idAnhaengen" type="button" value="Anhängen">
      <input id="idEinschieben" type="button" value="Einschieben"></p>

   <script>

      document.getElementById("idAnhaengen")
         .addEventListener("click", anhaengen);
      document.getElementById("idEinschieben")
         .addEventListener("click", einschieben);

   </script>
</body></html>
```

Listing 5.4 Datei »dom_hinzufuegen.htm«

In Abbildung 5.6 sehen Sie das ursprüngliche Dokument.

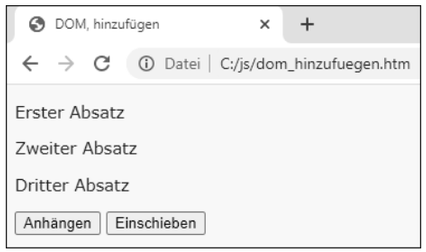


Abbildung 5.6 Vor dem Hinzufügen

Die beiden Schaltflächen ANHÄNGEN und EINSCHIEBEN dienen zum Anhängen bzw. zum Einschieben eines neuen HTML-Elementknotens.

Anhängen In der JavaScript-Funktion `anhaengen()` wird ein neuer Textknoten mit dem Text `Angehängt` erzeugt. Anschließend wird ein HTML-Elementknoten für einen Absatz erzeugt. Dem HTML-Elementknoten werden der Textknoten und ein Attributknoten hinzugefügt, anschließend wird er dem `body`-Knoten des Dokuments am Ende angehängt.

Einschieben In der Funktion `einschieben()` wird ein neu erzeugter HTML-Elementknoten mitsamt Textknoten und Attributknoten vor dem angegebenen Kindknoten des `body`-Knotens eingeschoben.

In Abbildung 5.7 sehen Sie das Dokument, nachdem beide Schaltflächen jeweils zweimal betätigt wurden.

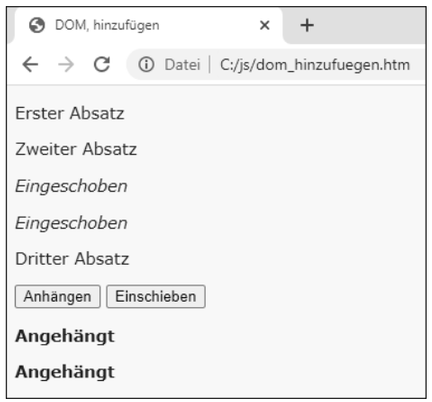


Abbildung 5.7 Nach dem Hinzufügen

5.5 Knoten ändern

In diesem Abschnitt zeige ich Ihnen, wie Sie Textknoten und HTML-Elementknoten ändern können. Es werden zwei weitere Methoden des `node`-Objekts eingeführt:

- `cloneNode()` erzeugt die Kopie eines Knotens, wahlweise mit oder ohne alle Kindknoten, deren Kindknoten usw. Parameter der Methode ist ein Wahrheitswert. Hat dieser den Wert `true`, wird der Knoten mitsamt seiner gesamten Kindknotenstruktur kopiert. **cloneNode()**
- `replaceChild()` ersetzt den Kindknoten eines Knotens durch einen anderen Kindknoten. Parameter der Methode sind zwei Verweise: auf den Knoten, der hinzugefügt wird, und auf den Knoten, der ersetzt wird. **replaceChild()**

Recht nützlich erweist sich auch die Eigenschaft `innerHTML` des `element`-Objekts. Sie dient zum Abrufen bzw. Zuweisen einer HTML-Markierung inklusive Text. **innerHTML**

Es folgt das Programm:

```
... <head> ...
   <script>

      function textAendern(id)
      {
         const absatz = document.getElementById(id);
         if(absatz.hasChildNodes())
            absatz.firstChild.nodeValue = "Text geändert";
         else
         {
            const text = document.createTextNode("Text erzeugt");
            absatz.appendChild(text);
         }
      }

      function htmlAendern()
      {
         const absatz = document.getElementById("idAbsatz4");
         absatz.innerHTML = "Änderung <i>in viertem</i> Absatz";
      }

      function umgeben()
```

```
      {
         const absatz = document.getElementById("idAbsatz3");
         const kursiv = document.createElement("span");
         kursiv.setAttribute("style", "font-style:italic;");
         const ersetzt = absatz.replaceChild(kursiv,
            absatz.firstChild);
         kursiv.appendChild(ersetzt);
      }

      function klonen()
      {
         const absatz = document.getElementById("idAbsatz3");
         const kopie = absatz.cloneNode(true);
         document.getElementById("idBody").appendChild(kopie);
      }

   </script>
</head>
<body id="idBody">
   <p id="idAbsatz1">Erster Absatz</p>
   <p id="idAbsatz2"></p>
   <p id="idAbsatz3"><span style="font-weight:bold;">Dritter</span>
      Absatz</p>
   <p id="idAbsatz4">Vierter Absatz</p>
   <p><input id="idTextAendern" type="button" value="Text ändern">
      <input id="idTextErzeugen" type="button" value="Text erzeugen">
      <input id="idHtml" type="button"
         value="Text und HTML ändern"></p>
   <p><input id="idUmgeben" type="button"
         value="Knoten mit HTML umgeben">
      <input id="idKlonen" type="button"
         value="Knoten klonen"></p>

   <script>

      document.getElementById("idTextAendern").addEventListener(
         "click", function() {textAendern("idAbsatz1");});
      document.getElementById("idTextErzeugen").addEventListener(
         "click", function() {textAendern("idAbsatz2");});
```



```
      document.getElementById("idHtml")
         .addEventListener("click", htmlAendern);
      document.getElementById("idUmgeben")
         .addEventListener("click", umgeben);
      document.getElementById("idKlonen")
         .addEventListener("click", klonen);

   </script>
</body></html>
```

Listing 5.5 Datei »dom_aendern.htm«

In Abbildung 5.8 sehen Sie das ursprüngliche Dokument.

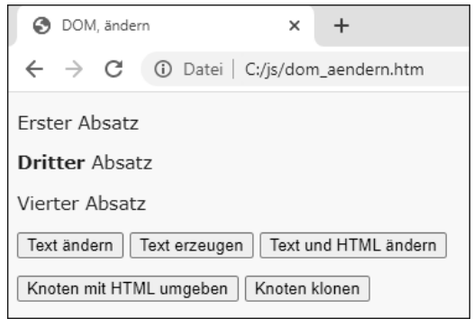


Abbildung 5.8 Vor dem Ändern

Der zweite Absatz ist nicht zu sehen. Er wird zwar mit einem p-Container erzeugt, besitzt aber keinen Inhalt, also keinen Kindknoten. Die beiden Schaltflächen TEXT ÄNDERN und TEXT ERZEUGEN dienen zum Ändern eines Textknotens. Existiert er noch nicht, wird er zunächst erzeugt.

Die Schaltfläche TEXT UND HTML ÄNDERN ändert den Inhalt eines Absatzes. Die Schaltfläche KNOTEN MIT HTML UMGEBEN dient zum Umgeben eines Knotens, der gegebenenfalls Kindknoten hat, mit einem HTML-Knoten.

Als Letztes folgt die Schaltfläche KNOTEN KLONEN. Sie dient zum Kopieren eines Absatzes inklusive aller Kindknoten. Er wird am Ende des Dokuments eingefügt.

Die JavaScript-Funktion `textAendern()` wird zweimal aufgerufen, jeweils mit der eindeutigen ID eines Absatzes als Parameter:

Inhalt ersetzen

- Beim Aufruf über die Schaltfläche TEXT ÄNDERN geht es um den ersten Absatz. Dieser hat einen Textknoten als Inhalt. Die Methode `hasChildNodes()` liefert daher `true`. Der Inhalt des vorhandenen Textknotens wird durch einen anderen Inhalt ersetzt. Dies passiert auch beim zweiten Betätigen der Schaltfläche TEXT ERZEUGEN, da der betreffende Knoten mittlerweile einen Kindknoten besitzt.

Knoten erzeugen

- Beim Aufruf über die Schaltfläche TEXT ERZEUGEN geht es um den zweiten Absatz. Dieser hat keinen Inhalt. Die Methode `hasChildNodes()` liefert daher `false`. Es wird ein neuer Textknoten erzeugt und dem Absatz als neuer Kindknoten hinzugefügt.

Text und HTML ändern

In der Funktion `htmlAendern()` wird mithilfe der Eigenschaft `innerHTML` der Inhalt des vierten Absatzes neu gesetzt, und zwar die HTML-Markierung inklusive Text.

Mit HTML umgeben

In der Funktion `umgeben()` wird ein neuer HTML-Elementknoten erzeugt. Er ersetzt den ersten Kindknoten des dritten Absatzes. Anschließend wird dieser (frühere) erste Kindknoten unter den neuen Knoten gesetzt. Damit wird dieser Kindknoten mit HTML-Code umgeben.

Klonen

In der Funktion `klonen()` wird ein neuer Knoten erzeugt, als Kopie des dritten Absatzes, inklusive aller Kindknoten. Diese Kopie wird an das Ende des Dokuments angehängt.

In Abbildung 5.9 sehen Sie das Dokument, nachdem jede Schaltfläche einmal geklickt wurde.

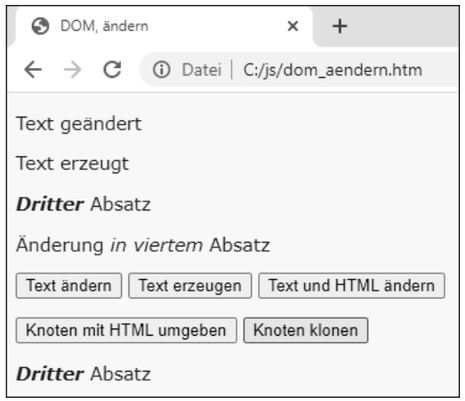


Abbildung 5.9 Nach dem Ändern

5.6 Knoten löschen

Sie können auch Knoten aus dem DOM-Baum löschen. Dazu dienen die folgenden Methoden für ein `node`-Objekt:

removeChild()

- `removeChild()` löscht den Kindknoten eines Knotens. Parameter der Methode ist ein Verweis auf den Kindknoten, der gelöscht wird.

removeAttribute()

- `removeAttribute()` löscht den Attributknoten eines Knotens. Parameter der Methode ist eine Zeichenkette mit dem Namen des Attributs, das gelöscht wird.

Es folgt das Programm:

```
... <head> ...
   <script>

      function knotenLoeschen()
      {
         const absatz = document.getElementById("idAbsatz2");
         document.getElementById("idBody").removeChild(absatz);
      }

      function attributLoeschen()
      {
         const absatz = document.getElementById("idAbsatz3");
         absatz.removeAttribute("style");
      }

   </script>
</head>
<body id="idBody">
   <p id="idAbsatz1">Erster Absatz</p>
   <p id="idAbsatz2">Zweiter Absatz</p>
   <p id="idAbsatz3" style="font-weight:bold;">Dritter Absatz</p>
   <p><input id="idKnoten" type="button" value="Knoten löschen">
      <input id="idAttribut" type="button"
         value="Attribut löschen"></p>

   <script>

      document.getElementById("idKnoten")
         .addEventListener("click", knotenLoeschen);
```

```
      document.getElementById("idAttribut")
         .addEventListener("click", attributLoeschen);

   </script>
</body></html>
```

Listing 5.6 Datei »dom_loeschen.htm«

In Abbildung 5.10 sehen Sie das ursprüngliche Dokument.

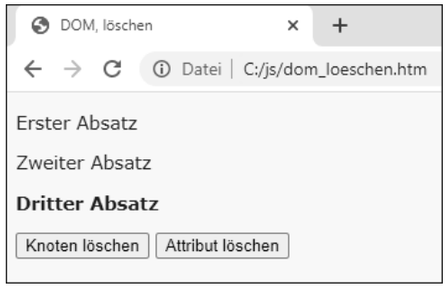


Abbildung 5.10 Vor dem Löschen

Löschen

Die Schaltfläche KNOTEN LÖSCHEN dient zum Löschen des zweiten Absatzes. Die Schaltfläche ATTRIBUT LÖSCHEN löscht das Attribut `style` des dritten Absatzes.

In Abbildung 5.11 sehen Sie das Dokument nach dem Löschen.

Abbildung 5.11 Nach dem Löschen

5.7 Eine Tabelle erzeugen

In einem letzten größeren Beispiel wird mithilfe einer doppelten `for`-Schleife eine Tabelle vollständig neu erzeugt und im Dokument eingebettet. Dabei kommen die Methoden `createTextNode()` und `createElement()` des `document`-Objekts sowie die Methode `appendChild()` für ein `node`-Objekt zum Einsatz. Man arbeitet sich im DOM-Baum zunächst von außen nach innen, anschließend wieder von innen nach außen.

Die Tabelle wird wie folgt erzeugt:

- Zuerst wird der HTML-Elementknoten für die Tabelle erstellt. **Tabelle**
- Innerhalb der äußeren Schleife folgen die HTML-Elementknoten für die Zeilen. **Zeilen**
- Innerhalb der inneren Schleife werden nacheinander: **Zellen mit Inhalt einbetten**
  - die Textknoten für den Zelleninhalt angelegt,
  - die HTML-Elementknoten für die Zellen erzeugt,
  - die Textknoten in die HTML-Elementknoten für die Zellen eingebettet und
  - die Zellen in die HTML-Elementknoten für die Zeilen eingebettet.
- Innerhalb der äußeren Schleife werden dann die HTML-Elementknoten für die Zeilen in den HTML-Elementknoten für die Tabelle eingefügt. **Zeilen einbetten**
- Als Letztes wird der HTML-Elementknoten für die Tabelle in das Dokument eingebettet. **Tabelle einbetten**

Der Programmcode:

```
... <head> ...
   <script>

      function tabelleErzeugen()
      {
         const tabelle = document.createElement("table");

         for(let z=1; z<=3; z++)
         {
            const zeile = document.createElement("tr");
            for(let s=1; s<=5; s++)
            {
               const text = document.createTextNode(
                  "Zelle " + z + "/" + s);
               const zelle = document.createElement("td");
               zelle.appendChild(text);
               zeile.appendChild(zelle);
```

```
            }
            tabelle.appendChild(zeile);
         }

         document.getElementById("idBody").appendChild(tabelle);
      }

   </script>
</head>
<body id="idBody">
   <p><input id="idTabelle" type="button" value="Tabelle"></p>
   <script>

      document.getElementById("idTabelle")
         .addEventListener("click", tabelleErzeugen);

   </script>
</body></html>
```

Listing 5.7 Datei »dom_tabelle.htm«

Nach der einmaligen Betätigung der Schaltfläche TABELLE sieht das Dokument aus wie in Abbildung 5.12.

Abbildung 5.12 Nach Erzeugung der Tabelle

Kapitel 11
jQuery

Die browserunabhängigen, einheitlichen Methoden der Bibliothek jQuery sind von vielen Websites nicht mehr wegzudenken.

Browserunabhängige Methoden

Bei *jQuery* handelt es sich um eine JavaScript-Bibliothek mit einer großen Verbreitung. Sie bietet komfortable, browserunabhängige Methoden, u. a. mithilfe von CSS, Ajax und Animationen. Für die Beispiele dieses Kapitels habe ich die zurzeit (im April 2021) aktuelle Version 3.6.0 verwendet.

Website

Die Datei *jquery-3.6.0.min.js* mit der gesamten Bibliothek ist nur ca. 88 KB groß. Sie finden sie zusammen mit den Beispieldateien im Downloadpaket zum Buch.

Etwaige aktuellere Versionen können Sie über *http://www.jquery.com* herunterladen. Die hier genutzte Datei finden Sie auf der Seite Download über den Begriff *compressed production*. In den nachfolgenden Beispielen wird davon ausgegangen, dass sich die Datei im selben Verzeichnis wie die Beispieldateien befindet.

11.1 Aufbau

Anhand eines ersten Beispiels erläutere ich verschiedene Möglichkeiten, jQuery zu nutzen. Abbildung 11.1 stellt eine Datei dar, die einen Absatz mit einem Text und drei Absätze mit einem Hyperlink beinhaltet.

Nach Betätigen der verschiedenen Hyperlinks ändert sich durch den Einsatz von jQuery jeweils der Inhalt des ersten Absatzes, wie Sie in Abbildung 11.2, Abbildung 11.3 und Abbildung 11.4 sehen.

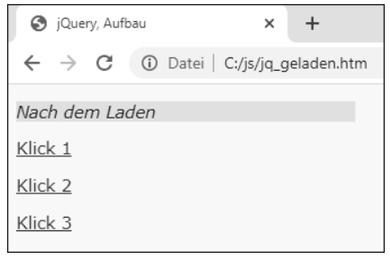


Abbildung 11.1 Erste jQuery-Datei

Abbildung 11.2 Erste Änderung

Abbildung 11.3 Zweite Änderung

Abbildung 11.4 Dritte Änderung

Es folgt der Code:

```
... <head> ...
   <script src="jquery-3.6.0.min.js"></script>
   <script>

      $(document).ready(function()
      {
         $("#idAbsatz").html("<i>Nach dem Laden</i>");
         $("#idLink1").click(function()
            { $("#idAbsatz").html("<b>Nach Klick 1</b>"); });
      });

   </script>
</head>
<body>
   <p id="idAbsatz" style="background-color:#e0e0e0;
      width:300px;">Hallo</p>
   <p><a id="idLink1" href="#">Klick 1</a></p>
   <p><a id="idLink2" href="#">Klick 2</a></p>
   <p><a id="idLink3" href="#">Klick 3</a></p>

   <script>

      $("#idLink2").click(function(){
         $("#idAbsatz").html("Nach Klick 2"); });
      jQuery("#idLink3").click(function(){
         jQuery("#idAbsatz").html("<b><i>Nach Klick 3</i></b>"); });

   </script>
</body></html>
```

Listing 11.1 Datei »jq_geladen.htm«

Das Dokument beinhaltet einen Absatz mit dem Beispieltext `Hallo`. Darunter stehen drei Absätze, die jeweils einen Hyperlink beinhalten.

ready()

Im oberen JavaScript-Bereich wird die Methode `ready()` aufgerufen. Sie besitzt als Parameter einen Verweis auf eine Callback-Funktion. Intern sorgt die Methode `ready()` dafür, dass die Callback-Funktion erst aufgerufen wird, nachdem die Datei mit allen Elementen im Browser des Benutzers geladen

wurde. Ansonsten könnte es vorkommen, dass auf ein Element zugegriffen wird, das noch nicht existiert.

In jQuery werden häufig anonyme Callback-Funktionen verwendet:

- Mithilfe der ersten Anweisung in der Funktion wird die Methode `html()` für das Element mit der ID `idAbsatz` aufgerufen. Sie ändert den Text eines Elements inklusive der HTML-Markierungen. Daher erscheint nach dem vollständigen Laden des Dokuments im ersten Absatz ein Text in Kursivdruck. **html()**
- Durch die zweite Anweisung wird die Methode `click()` für das Element mit der ID `idLink1` aufgerufen. Sie besitzt als Parameter ebenfalls einen Verweis auf eine anonyme Callback-Funktion. Intern sorgt die Methode `click()` dafür, dass die Callback-Funktion nach einem Klick auf den Hyperlink aufgerufen wird. Anschließend erscheint im ersten Absatz ein Text in Fettdruck. **click()**

Am Ende des Dokuments wird nach Betätigung des zweiten bzw. dritten Hyperlinks wiederum jeweils die Methode `click()` aufgerufen und damit der Inhalt des ersten Absatzes geändert.

Der Aufruf der Methode `ready()` ist hier nicht mehr notwendig, da alle Elemente des Dokuments bereits geladen sind.

Eine jQuery-Anweisung kann sowohl mithilfe der `$`-Funktion als auch mit der jQuery-Funktion aufgerufen werden. Beide Aufrufe führen zum selben Ergebnis. **jQuery oder $**

Hinweis

Eine jQuery-Anweisung wird für das im Selektor genannte Element durchgeführt. Häufig handelt es sich dabei um CSS-Selektoren.

Aufgrund der vielen Klammerebenen in jQuery-Anweisungen empfehle ich die hier verwendete kompakte Schreibweise. Beachten Sie bei Ihren eigenen Programmen auch die häufig notwendigen Hochkommata.

11.2 Selektoren und Methoden

Selektoren dienen der Auswahl des Elements, auf das sich der jQuery-Code bezieht. In diesem Abschnitt werden einige Selektoren vorgestellt. Zudem wird mit verschiedenen Methoden gearbeitet:

css()
- `css()` zur Änderung der CSS-Eigenschaften
- `html()` zur Änderung des Textes mit HTML-Code

text()
- `text()` zur Änderung des Textes ohne HTML-Code

In Abbildung 11.5 sehen Sie vier verschiedene `div`-Elemente. Sie werden mithilfe der Hyperlinks verändert.

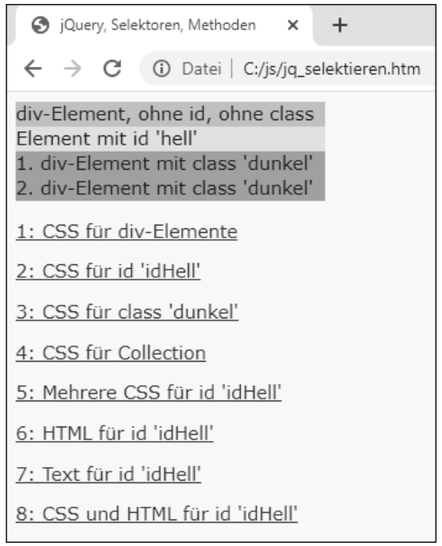


Abbildung 11.5 Selektoren und Methoden

Es folgt der Code, zunächst ohne den Inhalt der Methode `ready()`:

```
... <head> ...
   <script src="jquery-3.6.0.min.js"></script>
   <script>

      $(document).ready(function() { ... });

   </script>
   <style>

      div     {width:250px; height:200px; background-color:#c0c0c0;}
      #idHell {background-color:#e0e0e0;}
      .dunkel {background-color:#a0a0a0;}
```



```
   </style>
</head>
<body>
   <div>div-Element, ohne id, ohne class</div>
   <div id="idHell">Element mit id 'hell'</div>
   <div class="dunkel">1. div-Element mit class 'dunkel'</div>
   <div class="dunkel">2. div-Element mit class 'dunkel'</div>

   <p><a id="idLink1" href="#"> 1: CSS für div-Elemente</a></p>
   <p><a id="idLink2" href="#"> 2: CSS für id 'idHell'</a></p>
   <p><a id="idLink3" href="#"> 3: CSS für class 'dunkel'</a></p>
   <p><a id="idLink4" href="#"> 4: CSS für Collection</a></p>
   <p><a id="idLink5" href="#">
      5: Mehrere CSS für id 'idHell'</a></p>
   <p><a id="idLink6" href="#"> 6: HTML für id 'idHell'</a></p>
   <p><a id="idLink7" href="#"> 7: Text für id 'idHell'</a></p>
   <p><a id="idLink8" href="#">
      8: CSS und HTML für id 'idHell'</a></p>
</body></html>
```

Listing 11.2 Datei »jq_selektieren.htm«, ohne Methode »ready()«

Der Inhalt der Methode `ready()` wird weiter unten erläutert. Es folgen allgemeine Einstellungen für alle `div`-Elemente.

- Sie besitzen eine Größe von 250 × 20 Pixeln und sind mittelgrau.
- Das Element mit der ID `idHell` ist hellgrau.
- Alle Elemente der CSS-Klasse `dunkel` sind dunkelgrau.

Es folgen die vier `div`-Elemente:

ID oder Klasse

- Das erste Element besitzt keine ID. Es ist zudem keiner CSS-Klasse zugeordnet.
- Das zweite Element hat die ID `idHell`.
- Dem dritten und dem vierten Element wird jeweils die CSS-Klasse `dunkel` zugeordnet.

Anschließend folgen acht Absätze, die jeweils einen Link enthalten. Die Links haben die IDs `idLink1` bis `idLink8`.

Es folgt der Inhalt der Methode `ready()`:

```
$("#idLink1").click(function(){
   $("div").css({"width":"300px"}); });
$("#idLink2").click(function(){
   $("#idHell").css({"width":"350px"}); });
$("#idLink3").click(function(){
   $(".dunkel").css({"width":"400px"}); });
$("#idLink4").click(function(){
   $("#idHell, .dunkel").css({"width":"450px"}); });
$("#idLink5").click(function(){
   $("#idHell").css({"background-color":"#f0f0f0",
                     "width":"500px"}); });
$("#idLink6").click(function(){
   $("#idHell").html("<b>HTML neu</b>"); });
$("#idLink7").click(function(){
   $("#idHell").text("Text neu"); });
$("#idLink8").click(function(){
   $("#idHell").css({"width":"+=20px"}).html
      ("CSS und HTML neu");});
```

Listing 11.3 Datei »jq_selektieren.htm«, Inhalt der Methode »ready()«

CSS-Eigenschaften ändern

Nach Betätigung des ersten Hyperlinks wird für alle `div`-Elemente die Methode `css()` ausgeführt, die CSS-Eigenschaften ändert. Hier wird die Breite der Elemente auf den Wert 300 Pixel gesetzt. Ein Eigenschaft-Wert-Paar geben Sie im JSON-Format an. Der zweite Hyperlink ändert das Element mit der ID `idHell`. Der dritte Hyperlink ändert alle Elemente mit der CSS-Klasse `dunkel`.

Mehrere Selektoren

Sie können mehrere Selektoren in einer Collection zusammenfassen: Der vierte Hyperlink ändert das Element mit der ID `idHell` und alle Elemente mit der CSS-Klasse `dunkel`.

Mehrere Eigenschaften

Mehrere Eigenschaft-Wert-Paare werden im JSON-Format durch Kommata voneinander getrennt. Der fünfte Hyperlink ändert die Hintergrundfarbe und die Breite für das Element mit der ID `idHell`.

Text und HTML ändern

Der sechste Hyperlink ruft die Methode `html()` auf, die zum Ändern des Textes inklusive des HTML-Codes dient.

Nur Text ändern

Der siebte Hyperlink ruft die Methode `text()` auf, die den Text ohne Berücksichtigung des HTML-Codes ändert. Dabei würden versehentlich enthaltene HTML-Elemente ebenfalls als Text ausgegeben werden.

Verkettung

Durch Verkettung können Sie mehrere Methoden für einen Selektor ausführen. Zudem können Sie einen CSS-Wert mithilfe der beiden Operatoren `+=` und `-=` in Bezug auf den ursprünglichen Wert verändern. Der achte Hyperlink führt nacheinander die Methoden `css()` und `html()` für das Element mit der ID `idHell` aus. In der Methode `css()` wird die Breite pro Klick um 20 Pixel erhöht.

11.3 Ereignisse

Neben dem Klick-Ereignis können weitere Ereignisse zum Starten von jQuery-Code genutzt werden.

In Abbildung 11.6 sehen Sie ein `div`-Element, darunter eine Reihe von Hyperlinks. Das Auslösen eines Ereignisses auf einem Link führt zu einer animierten Verbreiterung des Elements. Dabei wird die Methode `animate()` zur Erstellung einer Animation eingesetzt. Mehr zum Thema »Animationen« erfahren Sie in Abschnitt 11.4.

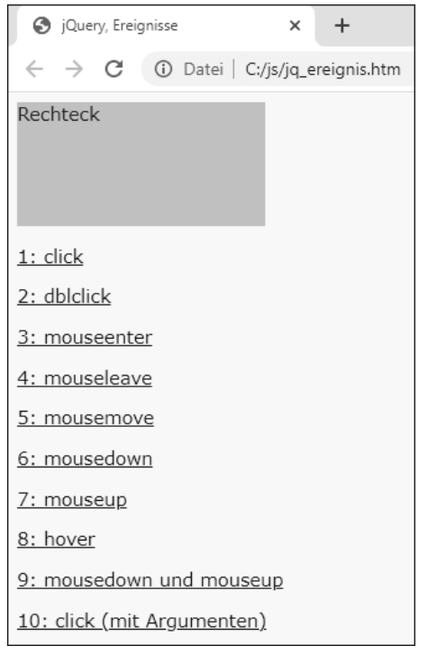


Abbildung 11.6 Verschiedene Ereignisse

Es folgt der Code, zunächst ohne den Inhalt der Methode `ready()`:

```
... <head> ...
   <script src="jquery-3.6.0.min.js"></script>
   <script>

      $(document).ready(function() { ... });

   </script>
</head>
<body>
   <div id="idRect" style="width:200px; height:100px;
        background-color:#c0c0c0;">Rechteck</div>
   <p><a id="idLink1" href="#"> 1: click</a></p>
   <p><a id="idLink2" href="#"> 2: dblclick</a></p>
   <p><a id="idLink3" href="#"> 3: mouseenter</a></p>
   <p><a id="idLink4" href="#"> 4: mouseleave</a></p>
   <p><a id="idLink5" href="#"> 5: mousemove</a></p>
   <p><a id="idLink6" href="#"> 6: mousedown</a></p>
   <p><a id="idLink7" href="#"> 7: mouseup</a></p>
   <p><a id="idLink8" href="#"> 8: hover</a></p>
   <p><a id="idLink9" href="#"> 9: mousedown und mouseup</a></p>
   <p><a id="idLink10" href="#">10: click (mit Argumenten)</a></p>
</body></html>
```

Listing 11.4 Datei »jq_ereignis.htm«, ohne Methode »ready()«

Das `div`-Element hat die ID `idRect`, eine Größe von 200 × 100 Pixeln und eine graue Farbe.

Es folgt der Inhalt der Methode `ready()`:

```
$("#idLink1").click(function(){
   $("#idRect").animate({"width":"+=20px"}); });
$("#idLink2").dblclick(function(){
   $("#idRect").animate({"width":"+=20px"}); });
$("#idLink3").mouseenter(function(){
   $("#idRect").animate({"width":"+=20px"}); });
$("#idLink4").mouseleave(function(){
   $("#idRect").animate({"width":"+=20px"}); });
$("#idLink5").mousemove(function(){
   $("#idRect").animate({"width":"+=20px"}); });
$("#idLink6").mousedown(function(){
   $("#idRect").animate({"width":"+=20px"}); });
$("#idLink7").mouseup(function(){
   $("#idRect").animate({"width":"+=20px"}); });
$("#idLink8").hover(function(){
  $("#idRect").animate({"width":"+=20px"}); });
$("#idLink9").bind("mousedown mouseup", function(){
   $("#idRect").animate({"width":"+=20px"}); });
$("#idLink10").click(function(e){
   $("#idRect").html("Ereignis: " + e.type
      + "<br>Ort X: " + e.pageX + " , Y: " + e.pageY
      + "<br>Zeit: " + Date(e.timeStamp)); });
```

Listing 11.5 Datei »jq_ereignis.htm«, Inhalt der Methode »ready()«

click(), dblclick()

Auf den ersten Hyperlink wird einfach geklickt. Das ruft die bereits bekannte Methode `click()` auf. Das Element wird 20 Pixel breiter, wie bei fast allen nachfolgenden Ereignissen. Auf dem zweiten Hyperlink wird ein Doppelklick ausgeführt. Dadurch wird die Methode `dblclick()` aufgerufen.

mouseenter(), mousemove(), mouseleave()

Der dritte Hyperlink wird durch das Betreten des Hyperlinks ausgelöst, also das Ereignis `mouseenter`. Das führt zum Aufruf der Methode `mouseenter()`. Das Entsprechende passiert nach dem Verlassen des vierten Hyperlinks (Name des Ereignisses und der Methode: `mouseleave`) und dem Überstreichen des fünften Hyperlinks (Name des Ereignisses und Name der Methode: `mousemove`).

mousedown(), mouseup()

Durch das Herunterdrücken einer Maustaste auf dem sechsten Hyperlink wird das Ereignis `mousedown` ausgelöst, durch das Loslassen der Maustaste auf dem siebten Hyperlink das Ereignis `mouseup`. Es werden die Methoden `mousedown()` und `mouseup()` aufgerufen.

hover()

Die Methode `hover()` vereinigt die beiden Ereignisse `mouseenter` und `mouseleave`. Sie wird also sowohl durch das Betreten als auch durch das Verlassen des achten Hyperlinks mit der Maus ausgelöst.

bind()

Die jQuery-Methode `bind()` dient dazu, Ereignisse an Methoden zu binden. Die anderen Methoden in diesem Abschnitt sind eigentlich Spezialisierungen der Methode `bind()` in abgekürzter Form. Beim neunten Hyperlink erfolgt die Bindung für die Ereignisse `mousedown` und `mouseup`. Die Animation wird also sowohl durch das Herunterdrücken als auch durch das Loslassen einer Maustaste ausgelöst.

Ereignisobjekt

Bei jedem der Ereignisse werden Informationen zum Ereignis in einem Ereignisobjekt bereitgestellt. Durch jQuery wird dieses Ereignisobjekt für alle Browser vereinheitlicht. Sie greifen über einen Verweis darauf zu, den Sie der Methode als Parameter übergeben. Bei einem Klick auf den zehnten Hyperlink werden einige Informationen ausgegeben, hier Art, Ort und Zeitpunkt des Ereignisses, mithilfe der Eigenschaften `type`, `pageX`, `pageY` und `timeStamp`. Der `timeStamp` wird in Millisekunden angegeben. Sie können ihn mithilfe der jQuery-Methode `Date()` umwandeln.

11.4 Animationen

animate()

In diesem Abschnitt erläutere ich verschiedene Möglichkeiten der Animation von Elementen. Dabei kommt die Methode `animate()` zum Einsatz. Sie erzeugt wie in einem Film durch eine Abfolge von Einzelbildern den Eindruck eines gleichmäßigen Ablaufs.

In Abbildung 11.7 sehen Sie ein positioniertes `div`-Element. Über die ersten elf Hyperlinks können unterschiedliche Animationen für dieses Element gestartet werden. Zur Verdeutlichung sollten Sie nach jeder Animation den Ausgangszustand nach dem Laden der Seite wiederherstellen. Das gelingt mit dem zwölften Hyperlink.

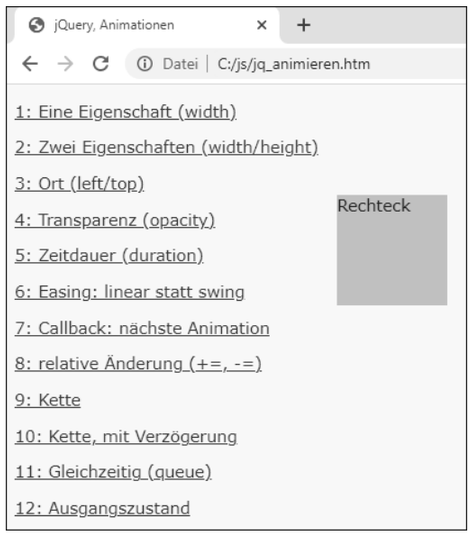


Abbildung 11.7 Animationen

Es folgt der Code, zunächst ohne den Inhalt der Methode `ready()`:

```
... <head> ...
   <script src="jquery-3.6.0.min.js"></script>
   <script>

      $(document).ready(function() { ... });

   </script>
</head>
<body>
   <div id="idRect" style="position:absolute; width:100px;
      height:100px; left:300px; top:100px;
      background-color:#c0c0c0;">Rechteck</div>
   <p><a id="idLink1" href="#"> 1: Eine Eigenschaft (width)</a></p>
   <p><a id="idLink2" href="#">
      2: Zwei Eigenschaften (width/height)</a></p>
   <p><a id="idLink3" href="#"> 3: Ort (left/top)</a></p>
   <p><a id="idLink4" href="#"> 4: Transparenz (opacity)</a></p>
   <p><a id="idLink5" href="#"> 5: Zeitdauer (duration)</a></p>
   <p><a id="idLink6" href="#"> 6: Easing: linear statt swing</a></p>
   <p><a id="idLink7" href="#"> 7: Callback: nächste Animation</a></p>
   <p><a id="idLink8" href="#"> 8: relative Änderung (+=, -=)</a></p>
   <p><a id="idLink9" href="#"> 9: Kette</a></p>
   <p><a id="idLink10" href="#">10: Kette, mit Verzögerung</a></p>
   <p><a id="idLink11" href="#">11: Gleichzeitig (queue)</a></p>
   <p><a id="idLink12" href="#">12: Ausgangszustand</a></p>
</body></html>
```

Listing 11.6 Datei »jq_animieren.htm«, ohne Methode »ready()«

Das `div`-Element hat die ID `idRect`, eine Startgröße von 100 × 100 Pixeln, die Startposition 300 × 100 Pixel und eine graue Farbe. Die Positionierung ist nur notwendig bei einer Animation des Orts.

Es folgt der Inhalt der Methode `ready()`:

```
$("#idLink1").click(function(){
   $("#idRect").animate({"width":"200px"}); });
$("#idLink2").click(function(){
   $("#idRect").animate({"width":"200px",
      "height":"50px"}); });
```

```
$("#idLink3").click(function(){
   $("#idRect").animate({"left":"400px", "top":"200px"}); });
$("#idLink4").click(function(){
   $("#idRect").animate({"opacity":"0.5"}); });
$("#idLink5").click(function(){
   $("#idRect").animate({"width":"200px"},
      {"duration":2000});});
$("#idLink6").click(function(){
   $("#idRect").animate({"left":"400px"},
   {"duration":2000, "easing":"linear"}); });
$("#idLink7").click(function(){
   $("#idRect").animate({"left":"400px"},
      function(){$("#idRect")
       .animate({"left":"300px"}) }); });
$("#idLink8").click(function(){
  $("#idRect").animate({"left":"+=100px",
     "opacity":"-=0.3"});});
$("#idLink9").click(function(){
   $("#idRect").animate({"left":"+=100px"})
      .animate({"left":"-=100px"}); });
$("#idLink10").click(function(){
   $("#idRect")
      .animate({"left":"+=100px"})
      .delay(1000)
      .animate({"left":"-=100px"}); });
$("#idLink11").click(function(){
   $("#idRect").animate({"width":"200px"}, {"duration":1000})
      .animate({"height":"50px"},
         {"duration":2000, "queue":false}); });
$("#idLink12").click(function(){
   $("#idRect").animate({"width":"100px", "height":"100px",
      "left":"300px", "top":"100px", "opacity":1.0}); });
```

Listing 11.7 Datei »jq_animieren.htm«, Inhalt der Methode »ready()«

Größe

Der erste Hyperlink animiert die Breite bis zum Zielwert 200 Pixel. Sie können mehrere Eigenschaften gleichzeitig ändern. Durch den zweiten Hyperlink wird die Breite bis zum Zielwert 200 Pixel und die Höhe bis zum Zielwert 50 Pixel animiert.

Position

Eine animierte Bewegung erreichen Sie über die Änderung der Eigenschaftswerte für `left` und `top`. Der dritte Hyperlink bewegt das Rechteck zum Zielpunkt 400 Pixel / 200 Pixel. Mit dem vierten Hyperlink ändern Sie die Transparenz über die Eigenschaft `opacity` auf den Wert 0.5, siehe auch Abschnitt 8.2.4.

Dauer

Eine Animation dauert ohne weitere Angabe 0.4 Sekunden, also 400 Millisekunden. Sie können die Dauer der Animation im zweiten Parameter der Methode `animate()` mithilfe der Eigenschaft `duration` einstellen. Der Wert wird in Millisekunden notiert, ohne Hochkommata. Der fünfte Hyperlink ändert die Breite auf den Zielwert 200 Pixel innerhalb von zwei Sekunden.

Ablauf der Geschwindigkeit

Die Eigenschaft `easing` kennzeichnet den zeitlichen Ablauf einer Animation. Der Standardwert `swing` bedeutet, dass die Animation zu Beginn beschleunigt, dann mit gleichmäßiger Geschwindigkeit weiterläuft und am Ende abbremst. Dadurch entsteht der Eindruck eines natürlichen Ablaufs.

Easing-Plugins

Geben Sie im zweiten Parameter der Methode `animate()` den Wert linear für die Eigenschaft `easing` an, läuft die Animation mit gleichmäßiger Geschwindigkeit ab, was nicht so natürlich aussieht. Easing-Plugins, die Sie im Internet finden, bieten weitere Möglichkeiten für Easing-Funktionen. Mithilfe des sechsten Hyperlinks wird das Element innerhalb von zwei Sekunden linear bis zum Zielwert verschoben.

Callback-Funktion

Als weiteren Parameter können Sie einen Verweis auf eine Callback-Funktion übergeben. Diese wird nach dem Ende der Animation ausgeführt. Der siebte Hyperlink führt zu einer Verschiebung des Elements zum Zielwert 400 Pixel. Danach wird das Element zum Zielwert 300 Pixel verschoben.

Relative Änderung

Bisher haben wir für die animierte Eigenschaft einen absoluten Zielwert angegeben. Sie können aber auch relative Änderungen durchführen, und zwar mithilfe der Operatoren `+=` und `-=`. Der achte Hyperlink verschiebt das Element bei jeder Betätigung um 100 Pixel nach rechts und ändert die Transparenz um 0.3, ausgehend von den jeweils aktuellen Werten. Ein Wert über 1.0 oder unter 0.0 ist für die Eigenschaft `opacity` nicht sinnvoll, führt aber nicht zu einem Fehler.

Verkettung

Methodenaufrufe können verkettet werden. Mithilfe des neunten Hyperlinks wird das Element um 100 Pixel nach rechts, anschließend wieder um 100 Pixel nach links verschoben.

Verzögerung

Innerhalb einer Animation können Sie mit der Methode `delay()` eine Verzögerung einbauen. Der zehnte Hyperlink führt zur gleichen Bewegung wie

der neunte Hyperlink. Zwischen den beiden Teilanimationen wird allerdings eine Sekunde gewartet.

Gleichzeitigkeit

Bei einer Verkettung laufen die einzelnen Teile einer Animation standardmäßig nacheinander ab. Sie können mithilfe des Parameters `queue` dafür sorgen, dass sie gleichzeitig stattfinden. Der elfte Hyperlink animiert die Breite innerhalb von einer Sekunde zum Zielwert 200 Pixel. Die Höhe wird zum Zielwert 50 Pixel animiert, allerdings innerhalb von zwei Sekunden, und zwar gleichzeitig. Das wird mit dem booleschen Wert `false` (ohne Hochkommata) für die Eigenschaft `queue` erreicht. Der Standardwert ist `true`.

Ausgangszustand

Der zwölfte Hyperlink dient zur Widerherstellung des Ausgangszustands von insgesamt fünf Eigenschaften: `width`, `height`, `left`, `top` und `opacity`.

Die Ereignisse werden gepuffert. Betätigen Sie also einen Hyperlink, bevor eine laufende Animation beendet ist, wird die zugehörige Aktion im Anschluss ausgeführt.

Schreibabkürzungen

Die folgenden Methoden bieten keine zusätzlichen Möglichkeiten, können aber als Schreibabkürzung dienen:

- die Methoden `slideDown()`, `slideUp()` und `slideToggle()` für die Veränderung der Eigenschaft `height`
- die Methoden `fadeIn()`, `fadeOut()`, `fadeToggle()` und `fadeTo()` für die Veränderung der Eigenschaft `opacity`
- die Methoden `show()`, `hide()` und `toggle()` für die gleichzeitige Veränderung der Eigenschaften `width`, `height` und `opacity`

11.5 Beispiel: sinusförmige Bewegung

jQuery und JavaScript

In den einzelnen Funktionen werden bisher nur jQuery-Methoden aufgerufen. Wir sollten aber nicht vergessen, dass uns auch der Rest von JavaScript zur Verfügung steht. Im folgenden Beispiel sehen Sie eine Kombination. Ein Bild mit einer Größe von 16 × 16 Pixeln bewegt sich entlang einer Sinuskurve, sobald der Benutzer daraufklickt. Zusätzlich sind einige Hilfslinien eingezeichnet, siehe Abbildung 11.8.

Linienabschnitte

Die Kurve besteht aus einzelnen Linienabschnitten. Jeder Linienabschnitt wird durch eine jQuery-Animation erzeugt. Eine `for`-Schleife sorgt für die

Aneinanderreihung der Animationen. Je feiner die Kurve zerlegt wird, desto gleichmäßiger ist der Kurvenverlauf. Sie sehen aber bereits bei einer Zerlegung in Abschnitte à 10 Grad einen recht homogenen Verlauf.

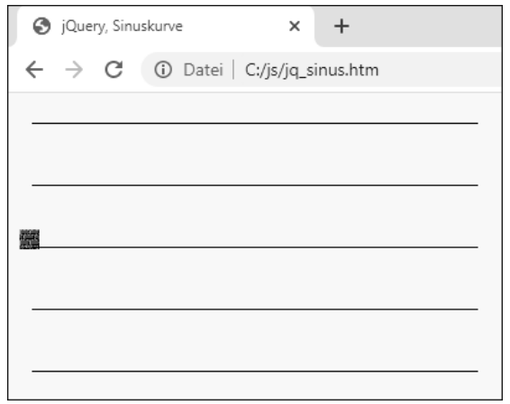


Abbildung 11.8 Animation einer Sinuskurve, Startpunkt

Der Code:

```
... <head> ...
   <script src="jquery-3.6.0.min.js"></script>
   <script>

      function sinus()
      {
         for(let winkel = 10; winkel <= 360; winkel += 10)
         {
            const pLeft = (10 + winkel) + "px";
            const pTop = (110 - Math.sin(winkel /
                  180 * Math.PI) * 100) + "px";
            $("#idBlock").animate({"left":pLeft, "top":pTop},
               {"duration":"100", "easing":"linear"});
         }
      }

      $(document).ready(function() { $("#idLink").click(sinus); });

   </script>
</head>
```

```
<body>
   <div id="idBlock" style="position:absolute; left:10px; top:110px;">
   <a id="idLink" href="#"><img src="block.gif" alt="Block"></a></div>
   <script>

      for(let pTop = 10; pTop < 211; pTop += 50)
         document.write(
         "<div style='position:absolute; left:20px; top:" + pTop
            + "px;'><img src='im_linie.jpg' alt='Linie'></div>");

   </script>
</body></html>
```

Listing 11.8 Datei »jq_sinus.htm«

Hilfslinien

Die Hilfslinien werden mithilfe einer `for`-Schleife erzeugt. Darin nimmt die Eigenschaft `pTop` Werte von 10, 60, 110, 160 und 210 Pixel an.

Bewegung in Einzelschritten

Innerhalb der Funktion `sinus()` nimmt die Variable `winkel` nacheinander die Werte von 10 bis 360 in 10er-Schritten an. Das Bild bewegt sich in x-Richtung gleichmäßig nach rechts. Für die Bewegung in y-Richtung wird die Methode `sin()` des `Math`-Objekts aus JavaScript genutzt. Sie erwartet den Winkel im Bogenmaß, daher muss dieser vorher umgerechnet werden. Die auf diese Weise errechneten Werte für den Zielpunkt werden in den Variablen `pLeft` und `pTop` gespeichert. Diese Variablen können wiederum als Zielwerte für die Eigenschaften `left` und `top` als Parameter der Methode `animate()` eingesetzt werden.

11.6 jQuery und Ajax

Ajax steht für *Asynchronous JavaScript and XML*. Diese Technik ermöglicht Ihnen das Nachladen von Dokumentteilen. Eine ausführliche Beschreibung gab es bereits in Kapitel 7. In jQuery gibt es eine Reihe von Methoden, die intern die Ajax-Technik nutzen. Sie arbeiten browser- und versionsunabhängig, wie Sie es bei jQuery gewohnt sind.

load(), post()

Im vorliegenden Beispiel werden mithilfe der Methoden `load()` und `post()` die Inhalte aus Textdateien, HTML-Dateien, PHP-Programmen und XML-Dateien in das aktuelle Dokument geladen, ohne den Rest der Seite neu aufbauen zu müssen.

Im nachfolgenden Programm sehen Sie einige Beispiele. In Abbildung 11.9 ist der Startzustand der Seite dargestellt. Es ist erforderlich, die Seite über einen Webserver zu laden, siehe auch Kapitel 7.

Abbildung 11.9 jQuery und Ajax

Der Code, zunächst ohne den Inhalt des zweiten JavaScript-Containers:

```
<!DOCTYPE html><html lang="de">
<head>
   <meta charset="utf-8">
   <title>jQuery und Ajax</title>
   <link rel="stylesheet" href="js4.css">
   <script src="jquery-3.6.0.min.js"></script>
   <script> ... </script>
</head>
<body>
   <p><b>Methode load() für Text und HTML:</b></p>
   <p><a id="idLink1" href="#"> 1: Gesamte Text-Datei</a></p>
   <p><a id="idLink2" href="#"> 2: Gesamte HTML-Datei</a></p>
   <p><a id="idLink3" href="#"> 3: Teil 1 aus HTML-Datei</a></p>

   <p><b>Methode post() für XML und PHP:</b></p>
   <p><a id="idLink4" href="#"> 4: HTML-Ergebnis von PHP</a></p>
```

```
   <p><a id="idLink5" href="#">
      5: Senden und HTML-Ergebnis von PHP</a></p>
   <p><a id="idLink6" href="#"> 6: Teil aus XML-Datei</a></p>
   <p><a id="idLink7" href="#">
      7: Teil aus XML-Datei, 2. Version</a></p>

   <p id="idAusgabe">Start-Text</p>
</body></html>
```

Listing 11.9 Datei »jq_ajax.htm«, ohne zweiten JavaScript-Container

Im untersten Absatz mit der ID `idAusgabe` werden die geladenen Inhalte dargestellt.

Es folgt der Inhalt des zweiten JavaScript-Containers:

```
function xmlDaten(ergebnis)
{
   const a = ergebnis.getElementsByTagName
      ("knotenA")[0].firstChild.nodeValue;
   const b = ergebnis.getElementsByTagName
      ("knotenB")[0].getAttribute("attributA");
   $("#idAusgabe").html(a + ", " + b);
}

function xmlAusgabe()
{
   $.post("jq_ajax_test.xml",
      function(ergebnis) { xmlDaten(ergebnis); });
}

$(document).ready(function()
{
   $("#idLink1").click(function() {
      $("#idAusgabe").load("jq_ajax_test.txt"); });
   $("#idLink2").click(function() {
      $("#idAusgabe").load("jq_ajax_test.htm"); });
   $("#idLink3").click(function() {
      $("#idAusgabe").load("jq_ajax_test.htm #t1"); });
   $("#idLink4").click(function() {
      $.post("jq_ajax_test.php", function(ergebnis) {
         $("#idAusgabe").html(ergebnis); }); });
```



```
   $("#idLink5").click(function() {
      $.post("jq_ajax_test_daten.php", {zahl1:12.2,
         zahl2:25.5}, function(ergebnis) {$("#idAusgabe")
         .html(ergebnis);}); });
   $("#idLink6").click(function() {
     $.post("jq_ajax_test.xml", function(ergebnis) {
         $("#idAusgabe").html(ergebnis.getElementsByTagName(
         "knotenA")[0].firstChild.nodeValue + ", " + ergebnis
         .getElementsByTagName("knotenB")[0].getAttribute(
         "attributA")); }); });
   $("#idLink7").click( xmlAusgabe );
});
```

Listing 11.10 Datei »jq_ajax.htm«, Inhalt der Methode »ready()«

Gesamte Textdatei

Der erste Hyperlink lädt mithilfe der Methode `load()` den gesamten Text aus der Textdatei *jq_ajax_test.txt* in den Absatz.

```
Das ist der Text aus der Textdatei
```

Listing 11.11 Datei »jq_ajax_test.txt«

Gesamte HTML-Datei

Mithilfe des zweiten Hyperlinks wird der gesamte Inhalt der HTML-Datei *jq_ajax_test.htm* mit den Markierungen in den Absatz geladen.

Teil 1 aus HTML-Datei

Der dritte Hyperlink lädt nur den Inhalt des Elements mit der ID `t1` aus der HTML-Datei in den Absatz, ebenfalls inklusive der Markierungen. Achten Sie bei dem Parameter der Methode `load()` auf das trennende Leerzeichen zwischen dem Dateinamen und dem Hash-Zeichen der ID.

```
...
<body>
   <p><b>Text in HTML-Datei</b></p>
   <p id="t1"><i>Teil 1 in HTML-Datei</i></p>
   <p id="t2"><i>Teil 2 in HTML-Datei</i></p>
</body></html>
```

Listing 11.12 Datei »jq_ajax_test.htm«

HTML-Ergebnis von PHP

Mithilfe des vierten Hyperlinks und der jQuery-Methode `post()` wird das PHP-Programm in der Datei *jq_ajax_test.php* aufgerufen. Im Parameter der Callback-Funktion (hier: `ergebnis`) steht anschließend die Ausgabe des PHP-Programms. Sie wird mithilfe der Methode `html()` zum Inhalt des Absatzes.

```
<?php
   header("Content-type: text/html; charset=utf-8");
   echo "<b>Daten</b> aus PHP-Datei";
?>
```

Listing 11.13 Datei »jq_ajax_test.php«

Senden und HTML-Ergebnis von PHP

Der fünfte Hyperlink ruft das PHP-Programm *jq_ajax_test_daten.php* mithilfe der jQuery-Methode `post()` auf. Dabei werden Eigenschaft-Wert-Paare gesendet, die durch Doppelpunkte getrennt sind. In diesem Beispiel wird im PHP-Programm die Summe der beiden Werte ermittelt und zurückgeliefert. Diese Rückgabe wird zum Inhalt des Absatzes.

```
<?php
   header("Content-type: text/html; charset=utf-8");
   echo "<b>Summe</b>: "
      . (doubleval($_POST["zahl1"]) + doubleval($_POST["zahl2"]));
?>
```

Listing 11.14 Datei »jq_ajax_test_daten.php«

Teil aus XML-Datei

Mithilfe des sechsten Hyperlinks wird die XML-Datei *jq_ajax_test.xml* aufgerufen, ebenso mithilfe der jQuery-Methode `post()`. Es wird der Wert des Knotens `knotenA` und der Wert des Attributs `attributA` des Knotens `knotenB` ermittelt und zum Inhalt des Absatzes.

```
<?xml version="1.0" encoding="utf-8"?>
<wurzel>
  <knotenA>Wert des ersten Knotens</knotenA>
  <knotenB attributA = "Erstes Attribut des zweiten Knotens"
      attributB = "Zweites Attribut des zweiten Knotens">
      Wert des zweiten Knotens</knotenB>
</wurzel>
```

Listing 11.15 Datei »jq_ajax_test.xml«

Teil aus XML-Datei, zweite Version

Die Verschachtelung von Aufrufen und anonymen Funktionen beim sechsten Hyperlink ist unübersichtlich. Daher habe ich einen siebten Hyperlink hinzugefügt, der zum selben Ergebnis führt. Dabei werden zwei benannte Funktionen sowie Teilergebnisse genutzt. Der gesamte Ablauf lässt sich auf diese Weise besser nachvollziehen.

Auf einen Blick

Inhalt

Materialien zum Buch

Auf der Webseite zu diesem Buch stehen folgende Materialien für Sie zum Download bereit:

- **Alle Beispielprogramme**
- **Alle Übungsaufgaben mit Lösungen**
- **Bonuskapitel**

Gehen Sie auf *https://www.rheinwerk-verlag.de/5363*. Klicken Sie auf den Reiter MATERIALIEN. Sie sehen die herunterladbaren Dateien samt einer Kurzbeschreibung des Dateiinhalts. Klicken Sie auf den Button HERUNTERLADEN, um den Download zu starten. Je nach Größe der Datei (und Ihrer Internetverbindung) kann es einige Zeit dauern, bis der Download abgeschlossen ist.