Jürgen Wolf

HTML5 und CSS3

Das umfassende Handbuch

- Alle neuen Features von HTML5, CSS3 und JavaScript
- Alle HTML5-APIs im Überblick
- Video, Audio, lokaler Speicher, dynamische 2D- und 3D-Grafiken, Canvas, Geolocation, Responsive Webdesign

Alle Beispielprojekte zum Download

Rheinwerk Computing

Kapitel 5

Tabellen und Hyperlinks

In diesem Kapitel lernen Sie weitere Elemente von HTML kennen. Genauer gesagt erfahren Sie hier, wie Sie Tabellen und Hyperlinks hinzufügen und verwenden können.

Weitere essenzielle HTML-Elemente, die ich bisher noch nicht beschrieben habe, erläutere ich in diesem Kapitel. Sie erfahren mehr zu folgenden Themen:

- **Tabellen**: Sie lernen, wie Sie Tabellen zur Darstellung von Informationen oder Daten in einem Raster verwenden können.
- **Hyperlinks**: Jeder, der das Internet kennt, kennt Hyperlinks, mit denen er sich von einer Website zur anderen bewegen kann. Hier erfahren Sie, wie Sie ein HTML-Dokument mit anderen (HTML-)Dokumenten verlinken.

5.1 Daten in einer Tabelle strukturieren

Tabellen sind hilfreich, wenn Sie zusammenhängende Daten wie z. B. Börsenkurse, Finanzdaten, Reisepläne, Zugfahrpläne, Busfahrpläne, Reiseberichte oder Sportergebnisse in einem Raster aus Zeilen und Spalten darstellen wollen. HTML bietet gute Möglichkeiten an, eine solche Tabelle zu strukturieren.

Formatierung mit CSS

HTML wird seit HTML5 nur noch für eine semantische und strukturelle logische Auszeichnung verwendet, und dies gilt auch für Tabellen in HTML. Tabellen in HTML bieten keinerlei Formatierungsmöglichkeiten an. Alle Attribute zur Formatierung aus altem HTML, abgesehen von einem Rahmen mit `border`, werden in HTML5 nicht mehr unterstützt. Daher gilt auch hier: Tabellen werden mit CSS formatiert.

HTML-Element	Bedeutung
`<table>`	Tabelle
`<tr>`	Tabellenzeile

Tabelle 5.1 Schnellübersicht über die hier behandelten Tabellenelemente

HTML-Element	Bedeutung
`<td>`	Tabellenzelle
`<th>`	Tabellenkopfzelle für Überschrift
`<thead>`	Tabellenkopfbereich
`<tbody>`	Tabellenkörper
`<tfoot>`	Tabellenfußbereich
`<colgroup>`	Gruppe von Tabellenspalten
`<col>`	Tabellenspalte
`<caption>`	Tabellenüberschrift/-legende

Tabelle 5.1 Schnellübersicht über die hier behandelten Tabellenelemente (Forts.)

5.1.1 Eine einfache Tabellenstruktur mit <table>, <tr>, <td> und <th>

Jede Tabelle in HTML wird zwischen den Elementen `<table>` und `</table>` erstellt (`table`, deutsch: Tabelle). Die Inhalte der Tabelle werden Zeile für Zeile hingeschrieben. Den Beginn einer Zeile notieren Sie mit einem öffnenden `<tr>` und das Ende der Zeile mit einem schließenden `</tr>` (`tr` = *table row*, deutsch: Tabellenzeile). Innerhalb einer Tabellenzeile zwischen `<tr>` und `</tr>` notieren Sie die einzelnen Zellen (oder auch Spalten) mit `<td>` und `</td>` (`td` = *table data*, deutsch: Tabellendaten).

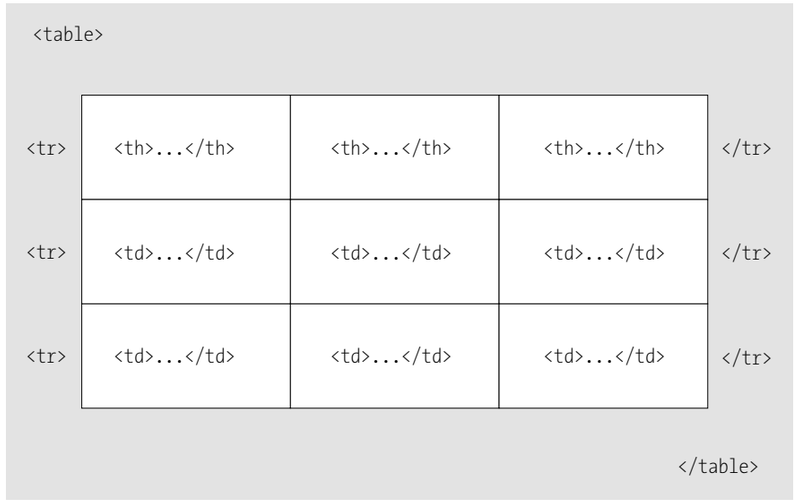

Abbildung 5.1 Eine grundlegende Tabellenstruktur in HTML

Wollen Sie Zellen oder Spalten als Überschrift einer Tabelle verwenden, können Sie die Daten zwischen `<th>` und `</th>` stellen (`th` = *table heading*, deutsch: Tabellenüberschrift). Das `th`-Element können Sie genauso verwenden wie das `td`-Element, nur dass die Webbrowser dieses Element gewöhnlich durch eine in der Spalte zentrierte Fettschrift hervorheben. Wenn es sinnvoll ist, sollten Sie Tabellenüberschriften verwenden, da dies zum einen für die Besucher mit Screenreadern hilfreich ist und zum anderen gegebenenfalls für die Suchmaschinen, die Ihre Website damit besser indizieren können.

Hierzu soll ein einfaches Beispiel einer Tabelle erstellt werden, in der Daten einer Webbrowser-Statistik von einer Website in einem Raster zusammengefasst und übersichtlich dargestellt werden:

```
...
  <table>
    <tr>
      <th>Browser</th>
      <th>Zugriffe</th>
      <th>Prozent</th>
    </tr>
    <tr>
      <td>Chrome</td>
      <td>14478</td>
      <td>59,6 %</td>
    </tr>
    <tr>
      <td>Firefox</td>
      <td>3499</td>
      <td>14,4 %</td>
    </tr>
    <tr>
      <td>Safari</td>
      <td>1619</td>
      <td>6,6 %</td>
    </tr>
  </table>
...
```

Listing 5.1 /Beispiele/Kapitel005/5_1_1/index.html

Wie Sie in Abbildung 5.2 sehen, stellen Webbrowser die Tabelle ohne jede Formatierung dar. Die Höhe und Breite einer Tabelle werden gewöhnlich gemäß dem enthaltenen Inhalt ausgegeben.

Abbildung 5.2 Die strukturierte Darstellung einer grundlegenden Tabelle in HTML

Was darf in eine Tabellenzelle alles rein?

Zwischen einer Zelle in `<td>` und `</td>` können Sie neben einem Text auch weitere HTML-Elemente verwenden. Theoretisch könnten Sie darin eine weitere komplette Tabelle einfügen. Wenn Sie eine leere Zelle ohne Inhalt verwenden wollen, müssen Sie trotzdem ein leeres `<td></td>` bzw. `<th></th>` angeben, um eine leere Zelle zu notieren, da die Tabelle ansonsten nicht richtig dargestellt wird. Bei alten Webbrowsern können Sie zudem zur Sicherheit ein erzwungenes Leerzeichen mit der HTML-Entität ` ` in die Zelle schreiben, weil es dort bei leeren Zellen ohne Inhalt zu Problemen kommen könnte.

5.1.2 Spalten bzw. Zeilen mit »colspan« bzw. »rowspan« zusammenfassen

Wenn Sie Tabelleneinträge über mehrere Zellen zusammenfassen (oder auch überspannen) wollen, können Sie dies mit dem HTML-Attribut `colspan` und `rowspan` machen. Anhand des Zahlenwerts, den Sie diesen Attributen übergeben, wird die Anzahl der Zellen angegeben, die Sie zusammenfassen wollen. Wie Sie anhand des Attributnamens vielleicht erahnen, wird `colspan` für das Zusammenfassen von Spalten und `rowspan` für das Zusammenfassen von Zeilen verwendet.

Hierzu ein einfaches Beispiel, in dem der Tagesplan eines Fotografieseminars in einer Tabelle zusammengefasst wurde:

```
…
  <table>
    <tr>
      <th></th>
      <th scope="col">Vormittag</th>
      <th scope="col">Mittag</th>
      <th scope="col">Nachmittag</th>
    </tr>
    <tr>
```



```
      <th scope="row">Montag</th>
      <td colspan="2">Fotoshooting (outdoor)</td>
      <td>Workshop Bildbearbeitung</td>
    </tr>
    <tr>
      <th scope="row">Dienstag</th>
      <td>Straßenfotografie (Stadt)</td>
      <td colspan="2">Fotoshooting (Porträt)</td>
    </tr>
    <tr>
      <th scope="row">Mittwoch</th>
      <td>Aktfotografie</td>
      <td>Workshop Bildbearbeitung</td>
      <td>Abschlussfeier</td>
    </tr>
  </table>
…
```

Listing 5.2 /Beispiele/Kapitel005/5_1_2/index.html

Wie Sie in Abbildung 5.3 sehen, wurde beim Rahmen der Tabelle mit CSS nachgeholfen, damit das Ergebnis von `colspan` deutlicher sichtbar ist.

Sie sehen hierbei, wie sich am Montag die Zelle `Fotoshooting (outdoor)` dank `colspan="2"` über 2 Spalten vom `Vormittag` und `Mittag` überspannt. Das Gleiche gilt für den Dienstag und die Spalte `Fotoshooting (Porträt)`, in der von `Mittag` bis `Nachmittag` zusammenfasst wurde.

Bei der Verwendung von `colspan` müssen Sie beachten, dass Sie die Anzahl der Zellen reduzieren müssen, wenn Sie z. B. einen `colspan` über zwei Zellen zusammenfassen. Im Beispiel von `Montag` haben Sie somit nur zwei `td`-Elemente notieren müssen anstelle von drei, da sich das erste `td`-Element bereits über zwei Spalten erstreckt.

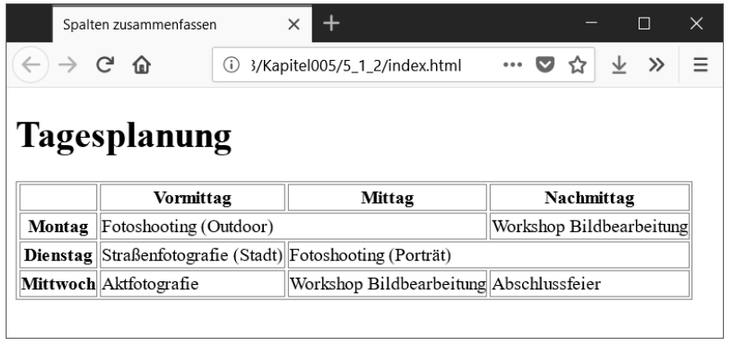

Abbildung 5.3 Zusammenfassen von Spalten mit dem Attribut »colspan«

Es spricht übrigens nichts dagegen, Spalten in mehr als zwei Zellen zusammenzufassen. Hierbei müssen Sie auf die Anzahl der tatsächlich vorhandenen Spalten achten. Folgendermaßen könnten Sie z. B. am `Dienstag` das `Fotoshooting (Porträt)` über drei Spalten zusammenfassen:

```
…
    <tr>
      <th scope="row">Dienstag</th>
      <td colspan="3">Fotoshooting (Porträt)</td>
    </tr>
    <tr>
…
```

Die Zelle `Straßenfotografie (Stadt)` müsste dann allerdings ebenfalls entfernt werden.

Das »scope«-Attribut von <th>

Im Beispiel wurde das `scope`-Attribut beim `th`-Element verwendet. Damit können Sie angeben, ob die Tabellenüberschrift für eine Spalte (`scope="col"`) oder eine Zeile (`scope="row"`) gelten soll.

Alles, was ich eben beschrieben habe, gilt auch, wenn Sie Tabelleneinträge mit `rowspan` über mehrere Zeilen zusammenfassen wollen. Hierzu nochmals das Beispiel, in dem der Tagesplan für das Fotoseminar etwas geändert wurde, denn jetzt wird am `Dienstag` und `Mittwoch` die `Straßenfotografie (Stadt)` am Vormittag durchführt:

```
…
  <table>
    <tr>
      <th></th>
      <th scope="col">Vormittag</th>
      <th scope="col">Mittag</th>
      <th scope="col">Nachmittag</th>
    </tr>
…
      <th scope="row">Dienstag</th>
      <td rowspan="2">Straßenfotografie (Stadt)</td>
      <td colspan="2">Fotoshooting (Porträt)</td>
    </tr>
    <tr>
      <th scope="row">Mittwoch</th>
      <td>Workshop Bildbearbeitung</td>
      <td>Abschlussfeier</td>
```



```
    </tr>
  </table>
…
```

Listing 5.3 /Beispiele/Kapitel005/5_1_2/index2.html

Im letzten `tr`-Element müssen Sie das `td`-Element mit `Aktfotografie` entfernen, weil Sie den Eintrag `Straßenfotografie (Stadt)` darüber mit dem Attribut `rowspan` nach unten ausgedehnt haben, wodurch dieser Eintrag den Platz in der darunter liegenden Zelle einnimmt, wie Sie in Abbildung 5.4 sehen.

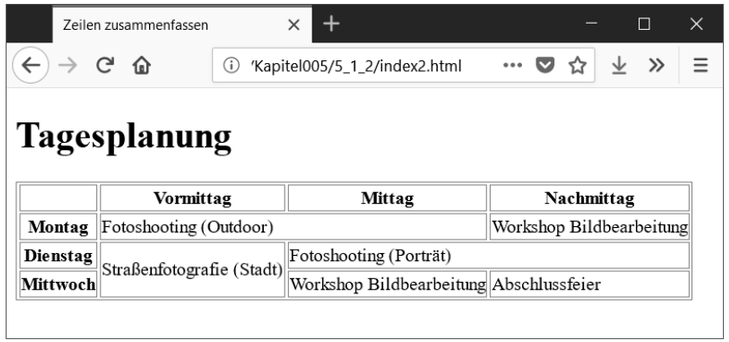

Abbildung 5.4 Zusammenfassen von Zeilen mit dem Attribut »rowspan«

5.1.3 HTML-Attribute für die Tabellenelemente

Mit HTML5 wurden alle Attribute des einleitenden `table`-Elements, die zur Formatierung von Tabellen verwendet wurden, entfernt. Wie am Anfang bereits erwähnt, sollten Sie zur Formatierung nur noch CSS verwenden. Für das `table`-Element unterstützt HTML5 lediglich das `border`-Attribut, bei dem der Wert `"1"` oder `""` sein darf, um einen Rahmen anzuzeigen. Hier wird CSS als bessere Alternative empfohlen. Um z. B. `border="1"` nachzubilden, fügen Sie einfach folgendes CSS-Konstrukt im Kopf des HTML-Dokuments hinzu:

```
…
<style>
  table, td, th { border: 1px solid gray }
</style>
…
```

Für die Tabellenzeile mit `<tr>` gibt es gar keine Attribute mehr, die von HTML5 unterstützt werden. Die Attribute von `<td>` und `<th>` mit `colspan`, `rowspan` und `scope` haben Sie bereits kennengelernt. Die restlichen Attribute der beiden Elemente wurden ebenfalls von HTML5 als missbilligt erklärt bzw. gestrichen.

Layout mit Tabellen?

Sie sollten Tabellen nicht mehr verwenden, um das Layout einer Website zu erstellen. Dies wurde im vorherigen Jahrtausend gemacht. Ich erwähne es nur, weil Sie sich vielleicht schon den einen oder anderen Quelltext einer älteren Website angesehen haben und sich wohl noch ansehen werden und noch zahlreiche Websites aus dieser Zeit vorhanden sind, die eine Tabelle zum Layouten bzw. Ausrichten des Dokumentinhalts verwenden. Meistens handelt es sich um Sites, die nicht weitergepflegt werden, oder sie stammen von Entwicklern, die nicht mehr auf dem Laufenden sind. Heute greifen Sie für das Layout einer Website auf CSS zurück.

5.1.4 Tabellen mit <thead>, <tbody> und <tfoot> strukturieren

Optional zu den grundlegenden Tabellenelementen von HTML können Sie noch eine Tabelle mit den Elementen `<thead>`, `<tbody>` und `<tfoot>` in einen Kopf-, Daten- und Fußbereich einteilen.

Den Tabellenkopf schließen Sie zwischen `<thead>` und `</thead>` ein (`thead` = *table head*, deutsch: Tabellenkopf). Sinnvollerweise sollten Sie dafür das `th`-Element für die einzelnen Zellen verwenden. Die eigentlichen Daten für die Tabelle markieren Sie zwischen `<tbody>` und `</tbody>` (`tbody` = *table body*, deutsch: Tabellenkörper). Wollen Sie einen Bereich als Tabellenfuß notieren, fassen Sie ihn zwischen `<tfoot>` und `</tfoot>` (`tfoot` = *table foot*, deutsch: Tabellenfuß) zusammen.

Hierzu ein Beispiel, das diese drei Elemente in einer Tabelle verwendet:

```
…
  <table>
  <thead>
    <tr>
      <th>Monat</th>
      <th>Besucher</th>
      <th>Bytes</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <th>Gesamt</th>
      <th>23423</th>
      <th>3234 MB</th>
    </tr>
  </tfoot>
  <tbody>
```



```
    <tr>
      <td>Januar</td>
      <td>3234</td>
      <td>132 MB</td>
    </tr>
…
…
    <tr>
      <td>Dezember</td>
      <td>7193</td>
      <td>894 MB</td>
    </tr>
  </tbody>
  </table>
…
```

Listing 5.4 /Beispiele/Kapitel005/5_1_4/index.html

Wenn Sie den HTML-Quelltext und die dazugehörende Darstellung in Abbildung 5.5 betrachten, werden Sie feststellen, dass der Webbrowser in der Lage ist, die Reihenfolge der Tabelle selbstständig richtig wiederzugeben. Obwohl im Quelltext der Fußbereich oben angegeben wurde, wird er vom Webbrowser passend unten angezeigt.

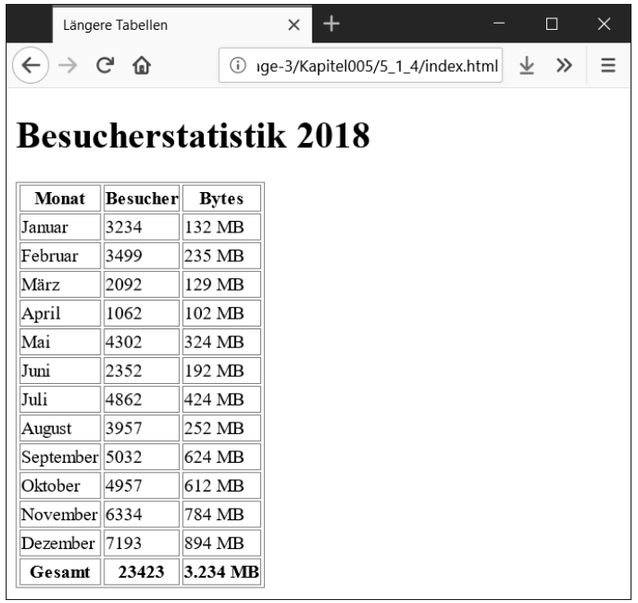

Abbildung 5.5 Eine längere Tabelle mit den Elementen <thead>, <tbody> und <tfoot> im Einsatz

Die Aufteilung einer Tabelle in drei verschiedene Bereiche ist optional und beeinflusst in der Regel nicht die Darstellung im Webbrowser. Es handelt sich um eine rein semantische Darstellung. Allerdings werden diese Elemente häufig verwendet, um das Erscheinungsbild dieser Bereiche mit CSS zu formatieren.

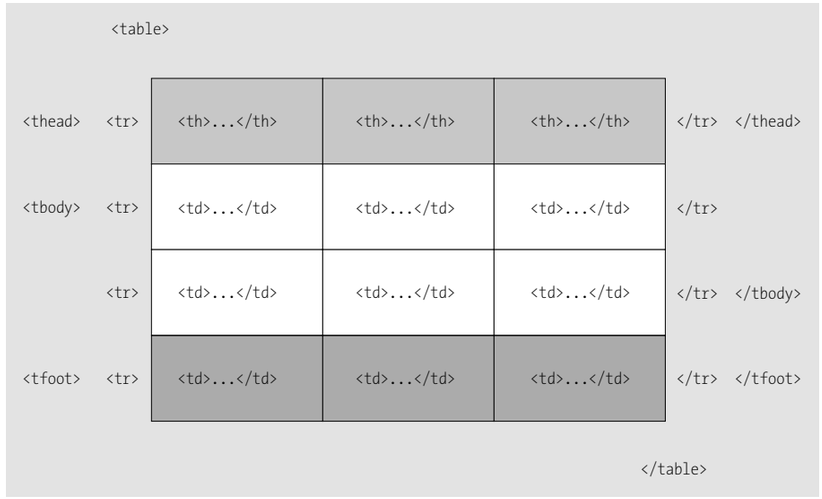

Abbildung 5.6 Die Aufteilung einer Tabelle in drei Bereiche ist zunächst rein semantischer Natur. Erst mit CSS können Sie diese Bereiche gesondert visualisieren.

Auch beim Ausdruck langer Tabellen über mehrere Seiten könnte der Webbrowser diese Aufteilung verwenden, um auf jeder Seite den Kopf- und Fußbereich der Tabelle mit auszudrucken, womit besser erkannt werden kann, in welcher Spalte die einzelnen Daten stehen bzw. was die Daten bedeuten. Eine weitere Möglichkeit wäre, bei langen Tabellen nur den Körperbereich zwischen `<tbody>` und `</tbody>` zu scrollen, während die Kopf- und Fußzeile fest stehen bleiben. Leider unterstützt noch kein Webbrowser diese Funktionen, aber das können Sie u. U. selbst mit CSS und gegebenenfalls JavaScript realisieren.

5.1.5 Spalten einer Tabelle gruppieren mit <colgroup> und <col>

So, wie Sie eben die Tabellenzeilen mit `<thead>`, `<tbody>` und `<tfoot>` in drei Bereiche aufteilen konnten, können Sie mit den Elementen `<colgroup>` und `<col>` auch die einzelnen Spalten in semantische und logische Bereiche aufteilen, sofern dies sinnvoll erscheint. Eine Gruppierung von Spalten ist z. B. sinnvoll, um eine bestimmte Spalte oder eine bestimmte Gruppe von Spalten mit einer bestimmten CSS-Formatierung zu versehen, anstatt den Style für jede Zelle der Spalte zu wiederholen.

Die Elemente `<colgroup>` und `<col>` müssen Sie hinter dem öffnenden `table`-Element und vor allen anderen Elementen wie `tr`, `thead`, `tfoot` oder `tbody` notieren. Eine Spaltengruppe

öffnen Sie mit `<colgroup>` und schließen sie wieder mit `</colgroup>` (`colgroup` = *column group*, deutsch: Spaltengruppe). Um eine Spalte zu gruppieren, müssen Sie für jede Spalte (die sich über die komplette Spalte erstrecken soll) das allein stehende Tag `<col>` verwenden. Wollen Sie mehrere Spalten in einem `col`-Element zusammenfassen, können Sie dies mit dem Attribut `span` und der Anzahl der Spalten als Attributwert machen.

Hierzu ein einfaches Beispiel, das das eben Beschriebene in der Praxis erläutert:


```
...
<table>
  <colgroup>
    <col span="2" style="background-color:lightgrey;">
    <col style="background-color:snow;">
  </colgroup>
    <tr>
      <th>Browser</th>
      <th>Zugriffe</th>
      <th>Prozent</th>
    </tr>
    <tr>
      <td>Chrome</td>
      <td>14478</td>
      <td>59,6 %</td>
    </tr>
    ...
    ...
  </table>
...
```

Listing 5.5 /Beispiele/Kapitel005/5_1_5/index.html

Abbildung 5.7 Hier wurden die ersten zwei Spalten mit »span="2"« zu einer Gruppe zusammengefasst und zur Demonstration farblich mit CSS hervorgehoben. Die letzte Spalte ist eine eigene Spaltengruppe.

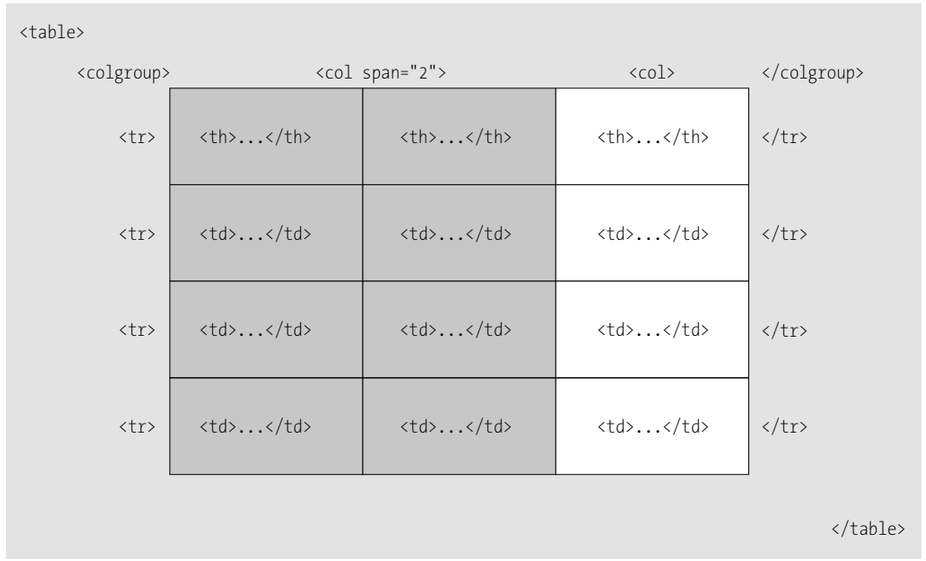

Abbildung 5.8 Die semantische Aufteilung von Spalten in Gruppen. In der Abbildung sehen Sie eine Gruppe mit zwei Spalten und mit einer Spalte.

Wollen Sie hingegen für jede Spalte eine eigene Gruppe verwenden, können Sie dies wie folgt realisieren:

```
  <table>
  <colgroup>
    <col style="background-color: lightgrey;">
    <col style="background-color: snow;">
    <col style="background-color: lightgrey;">
  </colgroup>
    <tr>
      <th>Browser</th>
      <th>Zugriffe</th>
      <th>Prozent</th>
    </tr>
…
  </table>
…
```

Jetzt wurde jede Spalte in einer eigenen `col`-Gruppe zusammengefasst. Der Vorteil wird erst ersichtlich, wenn Sie Spalten mit CSS stylen wollen. Die semantische Aufteilung in drei Spalten finden Sie in Abbildung 5.9 dargestellt.

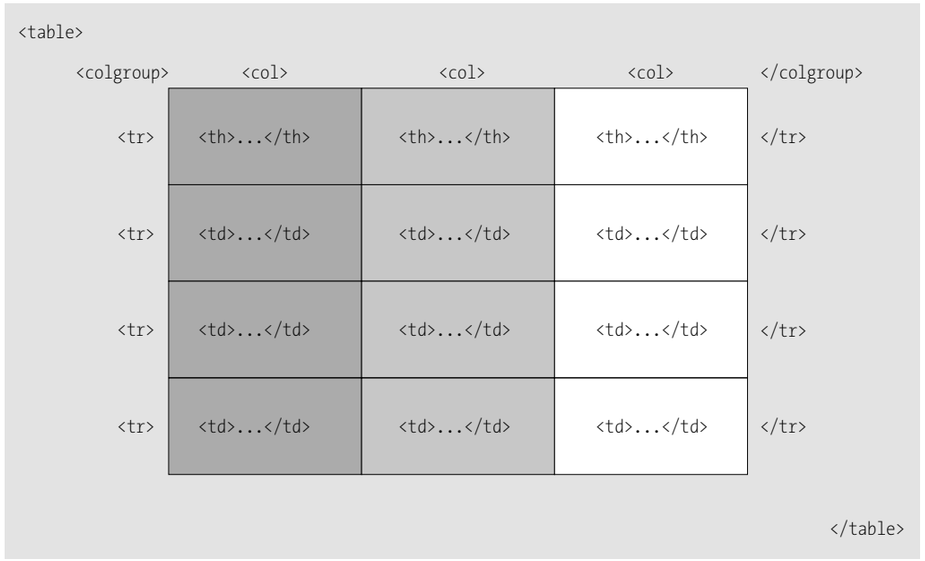

Abbildung 5.9 Semantische Aufteilung in drei Spalten

Wollen Sie `<col>` XHTML-konform verwenden, müssen Sie es mit `<col />` schreiben.

5.1.6 Tabellen beschriften mit <caption> bzw. <figcaption>

Zur Beschriftung einer Tabelle mit einem Titel können Sie entweder das `caption`-Element verwenden, das unmittelbar nach dem öffnenden `<table>`-Tag verwendet werden muss, oder Sie benutzen die neuen `figure`- und `figcaption`-Elemente.

Tabelle beschriften mit <caption>

Wie bereits erwähnt, muss das `caption`-Element gleich nach dem öffnenden `<table>`-Tag folgen. Außerdem kann nur eine Beschriftung pro Tabelle verwendet werden. Hierzu ein einfaches Beispiel:

```
…
  <table>
    <caption>Browserstatistik 11/2018</caption>
  <tr>
      <th>Browser</th>
      <th>Zugriffe</th>
      <th>Prozent</th>
```

```
    </tr>
    <tr>
      <td>Chrome</td>
      <td>14478</td>
      <td>59,6 %</td>
    </tr>
    …
    …
  </table>
…
```

Listing 5.6 /Beispiele/Kapitel005/5_1_6/index.html

Abbildung 5.10 Die Überschrift wird standardmäßig zentriert über der Tabelle angezeigt.

<caption> mit CSS formatieren

Die Webbrowser stellen die Beschriftung gewöhnlich zentriert über der Tabelle dar. Mit CSS ist es kein Aufwand, mithilfe der CSS-Eigenschaften `text-align` und `caption-side` die Tabellenbeschriftung anders auszurichten und woanders zu platzieren.

Wollen Sie einer Tabellenbeschriftung noch weitere Hinweise hinzufügen, setzen Sie die neuen HTML5-Elemente `details` und `summary` zwischen `<caption>` und `</caption>`. Zur Drucklegung konnten die meisten Browser mit den neuen Elementen umgehen. Für andere Webbrowser müssen Sie hier ein Workaround wie *html5shiv.js* verwenden.

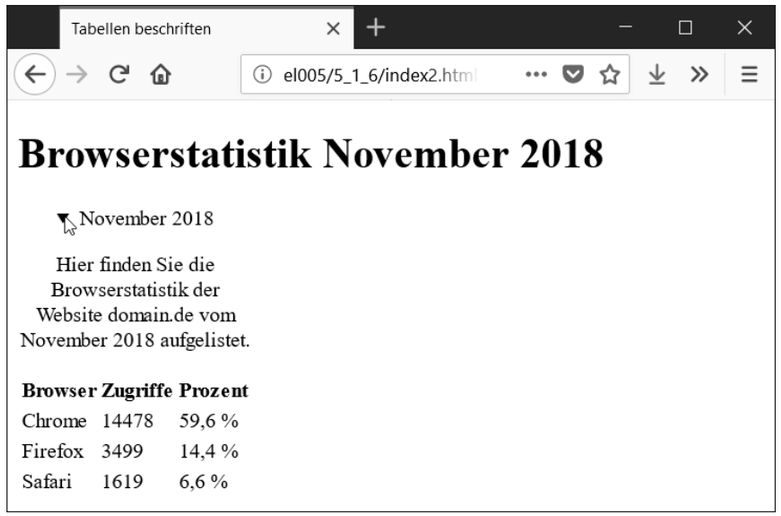

Abbildung 5.11 Informationen zum Auf- und Zuklappen dank der neuen HTML5-Elemente <details> und <summary>. Das Beispiel dazu finden Sie unter »/Beispiele/Kapitel005/5_1_6/index2.html«.

Beschriften einer Tabelle mit <figcaption>

Auf das `figcaption`- und `figure`-Element bin ich bereits in Abschnitt 4.2.9, »Gesonderte Beschriftung von Inhalten mit <figure> und <figcaption>«, eingegangen. Es bietet sich für Tabellen an, diese zwischen `<figure>` und `</figure>` zu verpacken und eine Beschriftung dieser Tabelle am Anfang nach dem öffnenden `<figure>` oder am Ende vor dem schließenden `</figure>` einzufügen. Hierzu ein Beispiel, wie Sie eine Tabelle mit den neuen `figure`- und `figcaption`-Elementen beschriften:

```
…
  <article>
  <h1>Browserstatistik November 2018</h1>
  <figure>
  <table>
  <tr>
      <th>Browser</th>
      <th>Zugriffe</th>
      <th>Prozent</th>
    </tr>
    <tr>
      <td>Chrome</td>
      <td>14478</td>
      <td>59,6 %</td>
```

```
    </tr>
…
  </table>
  <figcaption>Tabelle 1: Browserstatistik 11/2018</figcaption>
  </figure>
  </article>
…
```

Listing 5.7 /Beispiele/Kapitel005/5_1_6/index3.html

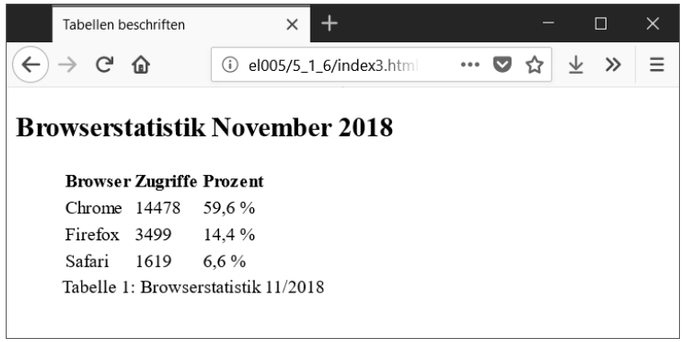

Abbildung 5.12 Tabellen beschriften mit <figure> und <figcaption>

5.2 »Elektronische« Verweise aka Hyperlinks mit <a>

Die Hyperlinks dürften wohl zu den wichtigsten Elementen von HTML gehören, weil es damit erst möglich wird, sich von einer Website zur anderen zu bewegen. Hyperlinks, oft nur Links oder Verweise genannt, werden Sie benötigen, um Ihr Projekt zu strukturieren und zu verlinken. Ausgehend von Ihrer Hauptseite, benötigen Sie oft Verweise zu weiteren Unterseiten und eventuell wieder Verweise zurück zur Hauptseite. Erst durch die Verlinkung mehrerer Dateien wird eine Website zu einer sinnvoll bedienbaren Website. Neben der Verlinkung eigener Inhalte können Sie Links zu anderen Websites oder anderen Dokumenten erstellen, die sich woanders im Internet befinden.

Einen Link erstellen Sie in HTML mit dem `a`-Element (`a` = *anchor*, deutsch: Anker). Der Text, den Sie zwischen `<a>` und `</a>` schreiben, heißt Linktext oder Verweistext und wird aktiviert, indem Sie im öffnenden `<a>`-Tag das `href`-Attribut verwenden. Als Linktext können Sie notieren, was Sie wollen, aber nicht immer ist es hilfreich, einfach `Bitte hier klicken` hinzuschreiben. Mit einem sinnvollen Linktext helfen Sie Ihren Besuchern, schneller dorthin zu gelangen, wo sie hinwollen, und auch Besuchern mit Screenreadern. Zwischen `<a>` und `</a>` können auch andere Elemente als ein Text stehen. Häufig finden Sie hier z. B. eine Grafik als Link wieder.

**Erlaubtes zwischen <a> und **

Wie bereits erwähnt, können Sie neben Text auch andere HTML-Elemente wie Grafiken zwischen `<a>` und `</a>` verwenden. Mit HTML5 dürfen Sie sogar, im Gegensatz zu HTML 4.01, gruppierende Elemente wie Absätze, Listen, Artikel und Blocksätze verwenden. Praktisch können Sie fast alles zwischen `<a>` und `</a>` einsetzen, abgesehen von interaktiven Elementen wie Links, Formularelementen, `audio`, `video`. Trotzdem empfehle ich Ihnen nicht, zu viel Inhalt in einen einzelnen Link zwischen `<a>` und `</a>` zu stecken. Screenreader würden den Text mehrmals vorlesen, und Besucher könnten damit überfordert sein, da sie daran gewöhnt sind, einzelne Links im traditionellen Linkstil zu aktivieren. Natürlich hängt dies vom Inhalt der Webseite ab. Ich werde hier nicht mehr näher darauf eingehen, aber Sie wissen jetzt, dass Ihnen in HTML5 »mehr« HTML-Elemente für Links zur Verfügung stehen. Wenn Sie extrem viel zwischen `<a>` und `</a>` gesteckt haben und sich nicht mehr sicher sind, ob es noch gültig ist, können Sie den Quelltext validieren.

Das wichtigste Attribut, mit dem das `a`-Element verwendet wird, ist das `href`-Attribut. Mit dem `href`-Attribut geben Sie den Verweis an, zu dem der Benutzer gelangt, wenn er auf den Linktext klickt.

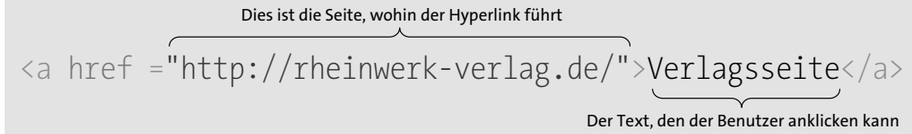

Abbildung 5.13 Der klassische Aufbau eines Hyperlinks

Ein Linktext wird gewöhnlich vom Webbrowser (meistens in Blau) unterstrichen. Dies können Sie mit CSS jederzeit ändern.

Zum Nachlesen

Auf die Begriffe Verzeichnisnamen, Verzeichnisstrukturen, vollständige, absolute und relative Pfadangaben bin ich bereits in Abschnitt 3.3, »Exkurs: Namenskonvention und Referenzierung«, eingegangen. Lesen Sie gegebenenfalls dort nach, wenn Sie in den folgenden Abschnitten Probleme mit den dort verwendeten Begriffen haben.

5.2.1 Links zu anderen HTML-Dokumenten der eigenen Website einfügen

Wenn Sie Ihre Website erstellen, dürften diese Verweise wohl die ersten Links sein, die Sie verwenden, um die losen Sammlungen von HTML-Dokumenten zu einer zusammenhängenden Website zu strukturieren – genauer: die Navigation der Website zu erstellen. Wenn Sie einen Link zu einer anderen Seite derselben Website angeben wollen, müssen Sie in der

Regel nicht den kompletten Domain-Namen mitangeben, sondern verwenden gewöhnlich eine *relative URL*. Die in Abbildung 5.14 abgedruckte Verzeichnisstruktur sei als Beispiel gegeben.

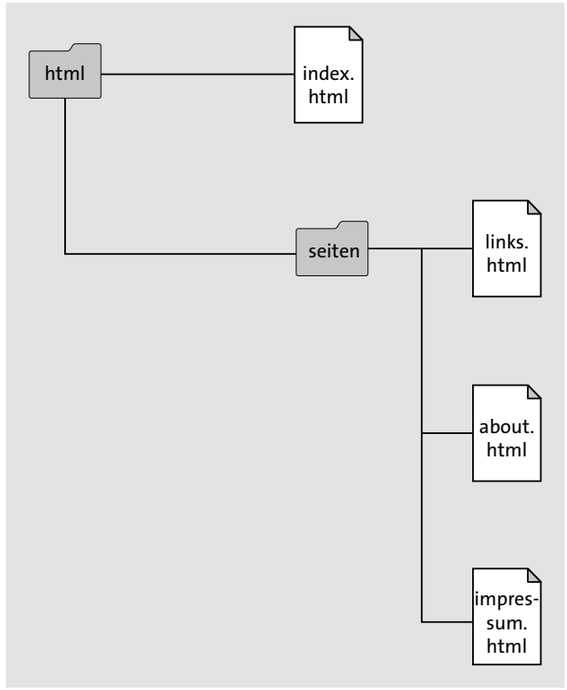

Abbildung 5.14 Verzeichnisstruktur für ein Beispiel von Links zu anderen Seiten derselben Website

Die Verlinkung für die Startseite, hier *index.html*, zu den anderen Seiten *links.html*, *about.html* und *impressum.html* sieht demnach in der Praxis wie folgt aus:

```
...
  <nav>
    Blog |
    <a href="seiten/links.html">Links</a> |
    <a href="seiten/about.html">Über mich</a> |
    <a href="seiten/impressum.html">Impressum</a>
  </nav>
  <h1>Mein Blog</h1>
  <p>Neueste Berichte zu HTML</p>
...
```

Listing 5.8 /Beispiele/Kapitel005/5_2_1/index.html

Abbildung 5.15 Dank der Verlinkung über eine relative URL kann innerhalb der Seiten derselben Website jede Seite besucht und betrachtet werden.

Natürlich müssen Sie auch die Links zu den anderen Seiten – wie hier im Beispiel *links.html*, *about.html* und *impressum.html* – anpassen. Hierbei müssen Sie bei der Angabe der relativen URL beachten (siehe Abbildung 5.14), dass sich die Seiten (in diesem Beispiel) in einem Unterordner namens *seiten* befinden. Bezogen auf die Seite *links.html*, würden die Angaben für das Attribut `href` wie folgt aussehen:

```
...
  <nav>
    <a href="../index.html">Blog</a> |
    Links |
    <a href="about.html">Über mich</a> |
    <a href="impressum.html">Impressum</a>
  </nav>
...
```

Listing 5.9 /Beispiele/Kapitel005/5_2_1/seiten/links.html

Abbildung 5.16 Das HTML-Dokument »links.html«

Hier sehen Sie, wie Sie aus dem Unterordner *seiten* mit *../* (hier *../index.html*) zum übergeordneten Ordner navigieren, wo sich *index.html* befindet. Die anderen beiden Dateien, *about.html* und *impressum.html*, befinden sich im selben Ordner wie *links.html*, daher reicht es aus, nur den Dateinamen anzugeben. Ähnlich müssen Sie auch die Dateien *about.html* und *impressum.html* verlinken.

5.2.2 Links zu anderen Websites einfügen

Links zu anderen Websites werden genauso notiert wie die Links zu den Seiten derselben Website, nur mit dem Unterschied, dass Sie im Attribut `href` die komplette Adresse, also die *absolute URL*, zu dieser Seite angeben müssen. Hierzu ein einfaches Beispiel, in dem Links auf externe Seiten enthalten sind:

```
…
  <article>
    <header>
      <h2>Empfehlung zu HTML5.2</h2>
    </header>
    <p>Wie bereits berichtet, hat das
       <a href="http://www.w3.org/">World Wide Web Consortium
       </a> eine
       <a href=" https://www.w3.org/TR/html52/">
       neue Empfehlung</a> für HTML vorgelegt, welcher in
       Version 5.2 weiterentwickelt wird ...
    </p>
    <aside>
    <h3>Weiterführende Links</h3>
      <nav>
      <ul>
       <li>
       <a href="https://www.w3.org/TR/html52/">
        HTML 5.2 Recommendation</a></li>
       <li><a href="http://www.w3.org/">W3C</a></li>
       <li><a href="http://www.whatwg.org/">WHATWG</a></li>
      </ul>
      </nav>
    </aside>
  </article>
…
```

Listing 5.10 /Beispiele/Kapitel005/5_2_2/index.html

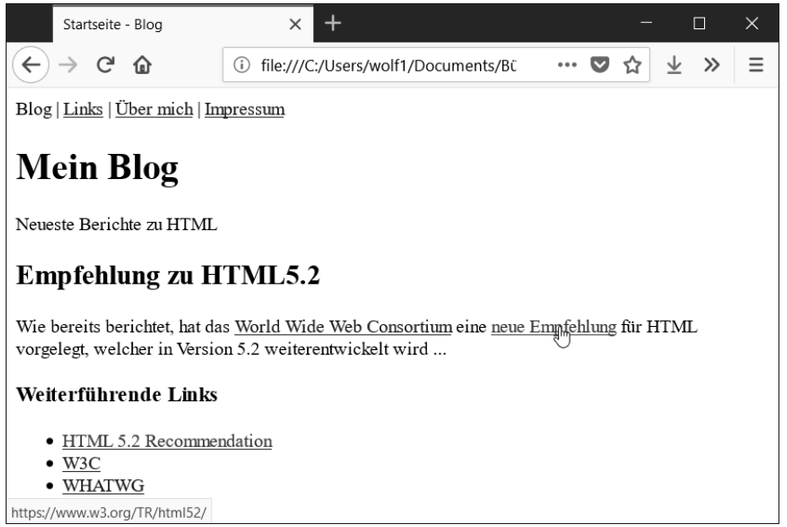

Abbildung 5.17 Viele Webbrowser zeigen die Zieladresse des Links unten in der Statusleiste an, wenn Sie mit der Maus darüber stehen. Aktivieren Sie den Link, …

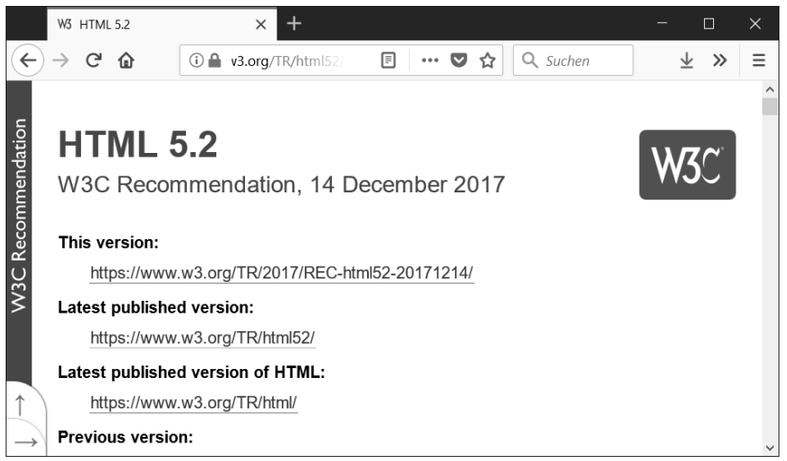

Abbildung 5.18 … wird die Zieladresse in den Webbrowser geladen und dargestellt.

5.2.3 Links mit dem »target«-Attribut in einem neuen Fenster öffnen

Mit dem HTML-Attribut `target` des `a`-Elements können Sie dafür sorgen, dass ein Verweisziel in einem neuen Fenster oder Tab geöffnet wird. Hierbei müssen Sie `target` nur den Attributwert `_blank` übergeben. Ein Beispiel hierzu:

```
<p>Wie bereits berichtet, hat das
   <a target="_blank" href="http://www.w3.org/">W3C</a> einen
   neuen Entwurf für HTML vorgelegt, welcher in Version 5.2
   weiterentwickelt wird ...
</p>
```

Würden Sie in diesem Beispiel den Linktext `W3C` aktivieren, würde die Zieladresse (hier: *www.w3.org*) hier tatsächlich in einem neuen Fenster oder Tab geöffnet und geladen. Ziel der Verwendung von `target="_blank"` ist natürlich vorrangig, den Besucher der Seite nicht zu »verlieren«, sondern die Seite offen zu lassen, damit er dort zurückkehrt, wenn er die Seite im neu geöffneten Fenster oder Tab gelesen hat.

Neben dem am meisten verwendeten Attributwert `_blank` können Sie hier `_self` (= aktuelles Fenster), `_parent` (= Eltern-Fenster), `_top` (= oberste Fenster-Ebene) und Namen von Fenstern verwenden, die mit JavaScript verarbeitet werden können.

Das Attribut »target="_blank"« verwenden oder nicht?

Auch wenn viele Websites dieses Attribut recht häufig und gerne verwenden, sollten Sie nicht auf Teufel komm raus für jeden Link ein neues Fenster öffnen. In der Praxis sollten Sie es dem Nutzer überlassen, ob er für einen Link eine neue Seite öffnen will oder nicht. Auch wenn Sie es vielleicht gewohnt sind, unzählige Tabs und mehrere Websites auf einmal geöffnet zu haben, so sollten Sie an die etwas unerfahreneren Besucher denken, die eben nicht so im World Wide Web unterwegs sind oder eben nicht so unterwegs sein wollen. Setzen Sie das Attribut `target="_blank"` sparsam ein, und weisen Sie, wenn möglich, den Besucher darauf hin, dass ein neues Fenster oder Tab geöffnet wird, wenn er den Link aktiviert.

5.2.4 E-Mail-Links mit »href=mailto:...«

Sicherlich kennen Sie auch die Sorte von Links, bei denen, wenn Sie sie aktivieren, sich die E-Mail-Anwendung mit einer bestimmten E-Mail-Adresse öffnet. Auch diese Links werden mit dem `a`-Element und dem `href`-Attribut erzeugt. Solche E-Mail-Verweise beginnen bei `href` mit `mailto:`, gefolgt von der gewünschten E-Mail-Adresse, z. B.:

```
...
  <footer>
    <a href="mailto:1@woafu.de">E-Mail senden</a>
  </footer>
...
```

Listing 5.11 /Beispiele/Kapitel005/5_2_4/index.html

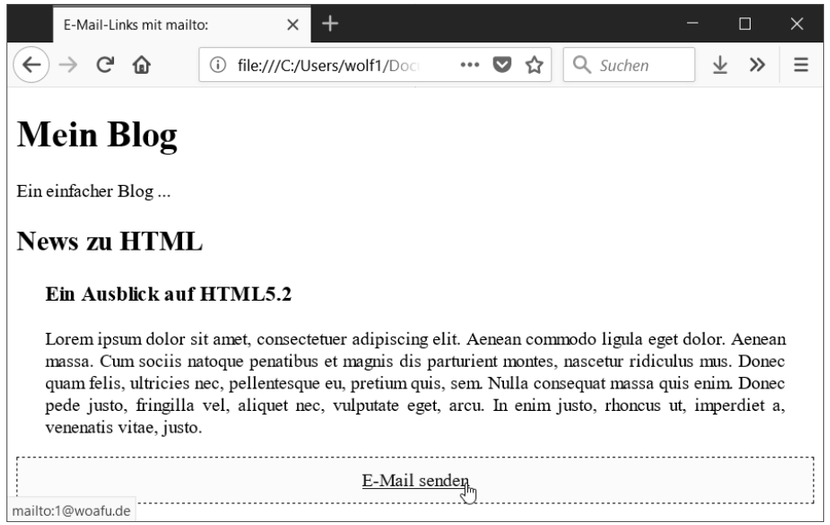

Abbildung 5.19 Wenn Sie mit der Maus über dem Link stehen bleiben, sehen Sie gewöhnlich in der Statusleiste die E-Mail-Adresse, die mit diesem Link verknüpft ist. Aktivieren Sie den Link, ...

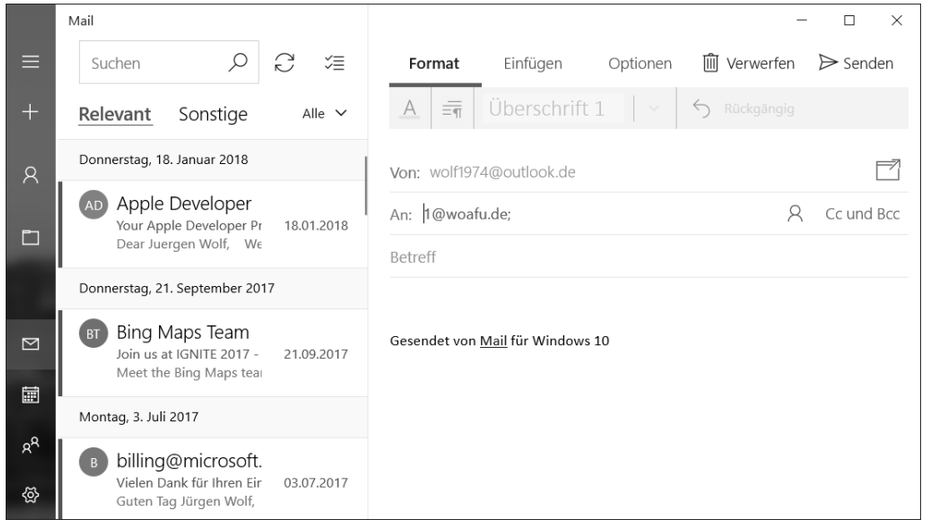

Abbildung 5.20 ... wird häufig die E-Mail-Anwendung geöffnet, automatisch eine E-Mail erstellt und darin die E-Mail-Adresse als Empfänger eingetragen.

Die Funktionalität eines »mailto«-Verweises ist nicht zuverlässig

Es gibt aber keine Garantie, dass eine solche `mailto`-Verknüpfung funktioniert. Dafür muss entweder der Webbrowser das Erstellen und Versenden von E-Mail unterstützen, oder es muss bei einem `mailto`-Verweis eine lokale E-Mail-Anwendung gestartet werden. Wenn der Besucher keine lokale E-Mail-Anwendung verwendet bzw. eingerichtet hat, sondern lediglich die klassische Webmail im Webbrowser nutzt, funktioniert der `mailto`-Verweis nur, wenn der Benutzer den Webbrowser entsprechend konfiguriert hat. Außerdem gibt es Webbrowser, die man dafür gar nicht konfigurieren kann. Es ist daher sinnvoll und empfehlenswert, die E-Mail-Adresse zusätzlich in lesbarer Form anzugeben, sodass Besucher, die den E-Mail-Verweis nicht ausführen können, Ihnen trotzdem eine E-Mail senden können.

Achtung vor Spam!

Leider müssen Sie aufgrund solcher E-Mail-Adressen auf einer Website früher oder später mit unerwünschten Werbe-E-Mails (Spam) rechnen, weil es Webcrawler gibt, die Websites nach E-Mail-Adressen durchsuchen. Sie haben sogar die Pflicht, die E-Mail-Adresse im Impressum zu nennen (§ 5 Allgemeine Informationspflicht; *http://www.gesetze-im-internet.de/tmg/__5.html*). Der einzige Schutz wäre, die E-Mail-Adresse nicht im Quelltext zu nennen. Die erste Möglichkeit, dies zu vermeiden, wäre die Einbindung als Grafik. Allerdings wäre dies diskriminierend gegenüber Personen, die auf Screenreader angewiesen sind, und auch so ist eine »Grafik-E-Mail-Adresse« rechtlich bedenklich. Häufig sind noch Versionen im Einsatz, in denen das @-Zeichen durch *(at)* ausgetauscht wird (z. B. *webmaster (at) dieter-baer.de*). Ebenso wird gerne noch der Punkt mit *(dot)* beschrieben (z. B. *webmaster (at) dieter-baer (dot) de*). Natürlich bedeutet das dann, dass der Besucher die E-Mail-Adresse von Hand eingeben muss.

Alternativ könnten Sie die E-Mail-Adresse im Unicode-Format mit numerischen Entitäten angeben. Die Adresse *webmaster@dieter-baer.de* sähe mit einer numerischen Entität wie folgt aus:

```
#109;&#97;&#105;&#108;&#116;&#111;&#58;&#119;&#101;&#98;&#109;&#97;&#115;&#116;
&#101;&#114;&#64;&#100;&#105;&#101;&#116;&#101;&#114;-&#98;&#97;&#101;&#114;.
&#100;&#101;
```

Hiermit würde nach wie vor die E-Mail-Adresse korrekt angezeigt, nur lässt sie sich nicht mehr im Quelltext so einfach erkennen. Ein richtiger Schutz ist das trotzdem nicht. Auch die Softwareentwickler von Spam-Crawlern wissen, wie man solche Informationen verwerten kann. Eine interessante Website, wie Sie es besser machen können, Ihre E-Mail-Adresse z. B. mit JavaScript zu verstecken, finden Sie hier: *http://alistapart.com/article/gracefulemailobfuscation*.

5.2.5 Links zu anderen Inhaltstypen setzen

Wenn Sie Links zu anderen, nicht im Web gebräuchlichen Dokumenttypen wie z. B. Word-, Excel-, PDF-Dokumenten setzen, hängt es vom Webbrowser ab, wie er diesen Dokumenttyp

weiterbehandelt. Darauf haben Sie als Webentwickler keinen Einfluss. Hier lautet zunächst die allgemeine Empfehlung, weitverbreitete Formate zu verwenden. So ist die Wahrscheinlichkeit höher, dass bei einem Link auf ein PDF-Dokument der Webbrowser einen entsprechenden PDF-Reader aufruft und das Dokument darin zum Lesen öffnet, als wenn der Link zum Inhaltstyp ein plattformabhängiges oder herstellerspezifisches Dokument ist (wie z. B. ein Word-Dokument). Hierzu ein einfaches Beispiel:


```
…
<h1>Verweis auf andere Inhaltstypen</h1>
<p>Ein PDF-Dokument öffnen: <a href="dokument.pdf">PDF</a></p>
<p>Einen MOV-Film öffnen: <a href="ganges.mov">MOV</a></p>
<p>Ein Word-Dokument öffnen: <a href="worddokument.doc"
   type="application/msword">DOC</a></p>
…
```

Listing 5.12 /Beispiele/Kapitel005/5_2_5/index.html

Abbildung 5.21 Hier haben Sie drei Links auf verschiedene Inhaltstypen.

Was bei diesen drei im Beispiel verwendeten Links passiert, kann nicht 100%ig vorhergesagt werden und hängt vom Webbrowser ab. Beim PDF-Dokument dürfte der Webbrowser wissen, wie er damit umzugehen hat. Schwieriger dürfte es mit dem Film im MOV-Format sein, weil dafür gewöhnlich ein QuickTime-Plug-in von Apple benötigt wird. Einige Webbrowser bieten an, das entsprechende Plug-in herunterzuladen und zu installieren. Andere wiederum nicht.

Dasselbe gilt für das Word-Dokument. Ist Word auf Ihrem Rechner installiert, bietet der Webbrowser häufig einen Dialog an, das Dokument mit Microsoft Word zu öffnen, oder zumindest die Möglichkeit, eine entsprechende Anwendung auszuwählen, mit der Sie dieses Dokument öffnen wollen. Häufig wird zusätzlich die Möglichkeit zum Herunterladen des Dokuments angeboten.

In Abbildung 5.22 kennt der Webbrowser die Standardverknüpfung der Anwendung mit dem Word-Dokument auf dem System. Die Anwendung zum Öffnen des Dokuments kann hier auch geändert werden. Auch ein Herunterladen mit DATEI SPEICHERN wird angeboten.

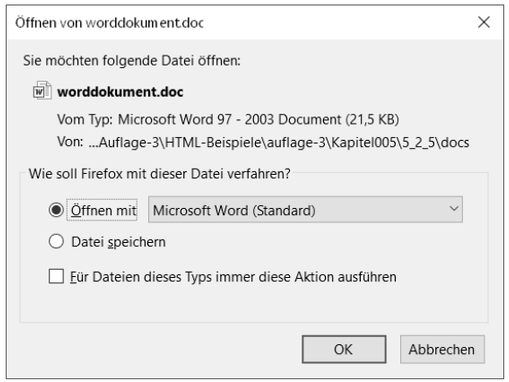

Abbildung 5.22 Öffnen eines Word-Dokuments im Webbrowser

Plug-ins nachrüsten

Bei vielen Webbrowsern gibt es die Möglichkeit, verschiedene Dateiformate über Plug-ins oder Add-ons nachzurüsten, um eine bestimmte Datei anzuzeigen oder wiederzugeben. Wenn Sie wirklich einen solchen Inhaltstyp auf Ihrer Webseite anbieten, der ein bestimmtes Plug-in oder Add-on benötigt, sollten Sie den Besucher vorher darauf hinweisen. Ob dieser das Plug-in oder Add-on installiert, nur um den einen Inhalt betrachten zu können, bleibt fraglich und hängt wohl auch vom Inhalt ab (lohnt sich der Aufwand?). Mit HTML haben Sie keinen Einfluss darauf.

Inhaltstyp mitangeben

Bei besonderen Inhaltstypen können Sie dem Webbrowser den Internet-MIME-Typ mit dem `type`-Attribut im öffnenden `<a>`-Tag mitteilen, wie ich dies im Beispiel mit `application/msword` für ein Word-Dokument gemacht habe. Die Informationen sind für den Webbrowser und auch andere Webclients sehr nützlich. Sinnvoll ist eine solche Angabe des Dateiformats fast immer, wenn das Linkziel kein HTML-Dokument ist. Eine Liste mit bekannten MIME-Typen finden Sie in Abschnitt A.19, »HTML-Zeichenreferenz gängiger benannter Zeichen«.

Informieren Sie den Besucher darüber, was sich hinter einem Link verbirgt

Wenn Sie Nicht-HTML-Dokumente anbieten, sollten Sie den Besucher auf jeden Fall darüber informieren, was sich hinter dem Link versteckt und eventuell auch, wie groß diese Datei ist. Sie können hierzu das globale `title`-Attribut im öffnenden `a`-Element verwenden, aber es ist

empfehlenswert, genauere Angaben direkt beim Linktext zu erwähnen. Ein Negativbeispiel, wie Sie es nicht machen sollten, sieht so aus:

```
<a href="jahresumsatz.pdf">Jahresumsatz 2018</a>
```

Der Besucher wird hier nur den Linktext `Jahresumsatz 2018` zu sehen bekommen und vielleicht verdutzt reagieren, wenn dieser Link ein PDF-Dokument ist, das vielleicht etwas länger zum Laden benötigt. Besser ist daher, Folgendes zu schreiben:

```
<a title="Öffnet die PDF-Datei mit dem Jahresumsatz von 2018"
   href="jahresumsatz.pdf">
    Jahresumsatz 2014 (PDF, 3,9 MB)
</a>
```


5.2.6 Downloadlinks mit dem »download«-Attribut hinzufügen

Seit HTML5 gibt es die Möglichkeit, Links als Downloadverweis hinzuzufügen – und dies unabhängig vom Inhaltstyp (= MIME-Typ) des Linkziels. Für diesen Zweck wird das Attribut `download` im öffnenden `<a>`-Tag verwendet. Hier nochmals derselbe HTML-Code vom Beispiel */Beispiele/Kapitel005/5_2_5/index.html* zuvor, nur werden jetzt alle drei Dateien mit dem `download`-Attribut zum Download angeboten:

```
...
<h1>Verweis auf andere Inhaltstypen</h1>
<p>Ein PDF-Dokument herunterladen:
   <a href="dokument.pdf" download>PDF</a></p>
<p>Einen MOV-Film herunterladen:
   <a href="ganges.mov" download="film.mov">MOV</a></p>
<p>Ein Word-Dokument herunterladen: <a href="worddokument.doc"
   download="worddokument.doc">DOC</a></p>
<p>Ein HTML-Dokument herunterladen: <a href="website.html"
   download="website.html">HTML</a></p>
...
```

Listing 5.13 /Beispiele/Kapitel005/5_2_6/index.html

Mit dem Attribut `download` weisen Sie einen Webbrowser an, diese Datei zum Download anzubieten, auch wenn er die Datei selbst anzeigen könnte oder das entsprechende Plug-in bzw. Add-on dazu kennt, das er für gewöhnlich für einen solchen Inhaltstyp verwenden würde.

Das Attribut `download` können Sie als allein stehendes Attribut verwenden, wie im ersten Beispiel mit dem PDF-Dokument zu sehen ist. Der Name der Datei, die heruntergeladen wird, entspricht der Angabe in `href` (hier: *dokument.pdf*). Enthält der Link in `href` keinen sinnvol-

len Namen, können Sie dem `download`-Attribut auch einen anderen Namen zuweisen, wie es im Beispiel mit dem MOV-Film der Fall ist, wo der eigentliche Dokumentname *ganges.mov* lautet, der Downloadname der Datei aber *film.mov* heißt. Bei XHTML müssen Sie bei `download` einen Dateinamen verwenden, weshalb im dritten Beispiel mit dem Word-Dokument für `href` und `download` derselbe Dokumentname benutzt wird. Das letzte Beispiel mit dem HTML-Dokument soll nur demonstrieren, dass selbst webbrowsertypische Inhaltstypen wie hier ein HTML-Dokument mit dem Attribut `download` wirklich nur noch als Download angeboten werden.

Den Besucher darüber informieren, was hier heruntergeladen wird?

Wie schon beim Verlinken von Nicht-HTML-Dokumenten sollten Sie den Leser darauf hinweisen, was er herunterlädt und womit er das Dokument betrachten oder weiterverwenden kann. Wenn Sie z. B. Excel-Tabellen mit einem Jahresumsatzbericht zum Download anbieten, sollten Sie den Leser darüber informieren, welche Software er benötigt, um diese Tabelle zu anzuschauen.

Dasselbe gilt für ZIP-verpackte Archive. Auch hier sollten Sie einen zusätzlichen Hinweis, wie ein solches Archiv entpackt werden kann, oder einen Link zu einer entsprechenden Software hinzufügen. Bedenken Sie, dass viele Besucher nichts mit Dateiendungen wie **.odt*, **.xls*, **.zip*, **.tar.bz* usw. anfangen können. Halten Sie es nicht für selbstverständlich, bloß weil Sie täglich mit unzähligen Datenformaten zu tun haben, dass Ihre Besucher dies auch tun. Es empfiehlt sich, beim Download die Dateigröße mitanzugeben. Den Download eines umfangreichen ZIP-Archivs könnten Sie somit wie folgt notieren:

```
…
<a title="Jahresumsatz im Excel-Format als ZIP-Archiv verpackt"
   href="archiv.zip" download="jahresumsatz2018.zip">
      Jahresumsatz 2018 (ZIP-Archiv; 2,5 MB)</a>
   <small>(Um das ZIP-Archiv zu entpacken, benötigen Sie ein
           Packprogramm wie z. B. 7-Zip. Die Jahresumsätze sind
           im Excel-Format gehalten und benötigen somit eine
           Software, die Excel-Tabellen betrachten kann.)
   </small>
…
```

Hier habe ich neben dem `title`-Attribut das Dateiformat (hier: ZIP-Archiv) und die Dateigröße angegeben. Zusätzlich habe ich ein paar klein gedruckte Informationen zwischen `<small>` und `</small>` notiert.

Alte Webbrowser, die das »download«-Attribut nicht kennen?!

Bei Webbrowsern (der Internet Explorer 11 und der Safari-iOS-Browser), die das neue `download`-Attribut nicht kennen, wird vorgegangen wie bisher, als wäre das `download`-Attribut nicht vorhanden. Inhaltstypen, die der Webbrowser nicht kennt, werden wie gehabt entweder zum lokalen Speichern angeboten, oder Sie können aus einer Liste von Anwendungen auswählen, mit welcher Sie dieses Dokument öffnen wollen. Eine beliebte Methode ist, die Dateien in das ZIP-Format zu packen und anzubieten.

5.2.7 Links zu bestimmten Teilen einer Webseite setzen

Nichts kann für den Besucher lästiger sein, als eine lange wissenschaftliche Abhandlung eines speziellen Themas auf einer Webseite zu lesen und dabei lange hoch- und herunterscrollen zu müssen, um zu einem speziellen Abschnitt zu gelangen. Für solche Fälle können Sie sogenannte *Anker* mit dem globalen Attribut `id` setzen, den Sie mit einem gewöhnlichen Link mit dem `a`-Element anspringen können. Vorbildlich werden solche Zielanker z. B. bei Wikipedia für das Inhaltsverzeichnis eines Themas verwendet. Für eine Verlinkung zu einem bestimmten Bereich einer Webseite benötigen Sie nur:

- einen Anker (Sprungmarke), den Sie mit dem Attribut `id="ankername"` erstellen. Zum Beispiel:

  ```
  <h1 id="ankername">Überschrift xyz</h1>
  ```

- einen Link, der auf den Anker mit `href="#ankername"` verweist. Hierzu wird das Doppelkreuz-Zeichen `#` vor den Ankernamen geschrieben. Zum Beispiel:

  ```
  <a href="#ankername">Zur Überschrift xyz springen</a>
  ```

Hierzu ein einfaches Beispiel, wie Sie solche Sprungmarken in der Praxis setzen und verwenden können:

```
…
  <h1 id="top">Inhaltsverzeichnis</h1>
  <ul>
    <li><a href="#intro">Einführung in HTML</a></li>
    <li><a href="#syntax">Die Syntax von HTML</a></li>
    <li><a href="#versionen">Versionen von HTML</a></li>
    <li><a href="#techniken">Techniken rund um HTML</a></li>
    <li><a href="#praxis">Einstieg in die Praxis</a></li>
  </ul>
  <h1 id="intro">Einführung in HTML</h1>
  <p>Lorem ipsum dolor sit amet … <p>
  <p><a href="#top">Zum Inhaltsverzeichnis</a></p>
  <h2 id="syntax">Die Syntax von HTML</h2>
  <p>Lorem ipsum dolor sit amet … <p>
  <p><a href="#top">Zum Inhaltsverzeichnis</a></p>
```

```
  <h2 id="versionen">Versionen von HTML</h2>
  <p>Lorem ipsum dolor sit amet … <p>
  <p><a href="#top">Zum Inhaltsverzeichnis</a></p>
  <h2 id="techniken">Techniken rund um HTML</h2>
  <p>Lorem ipsum dolor sit amet … <p>
  <p><a href="#top">Zum Inhaltsverzeichnis</a></p>
  <h2 id="praxis">Einstieg in die Praxis</h2>
  <p>Lorem ipsum dolor sit amet … <p>
  <p><a href="#top">Zum Inhaltsverzeichnis</a></p>
…
```

Listing 5.14 /Beispiele/Kapitel005/5_2_7/index.html

In Abbildung 5.23 sehen Sie das Beispiel bei der Ausführung, wo Sie dank Sprungmarken schneller zum gewünschten Abschnitt gelangen.

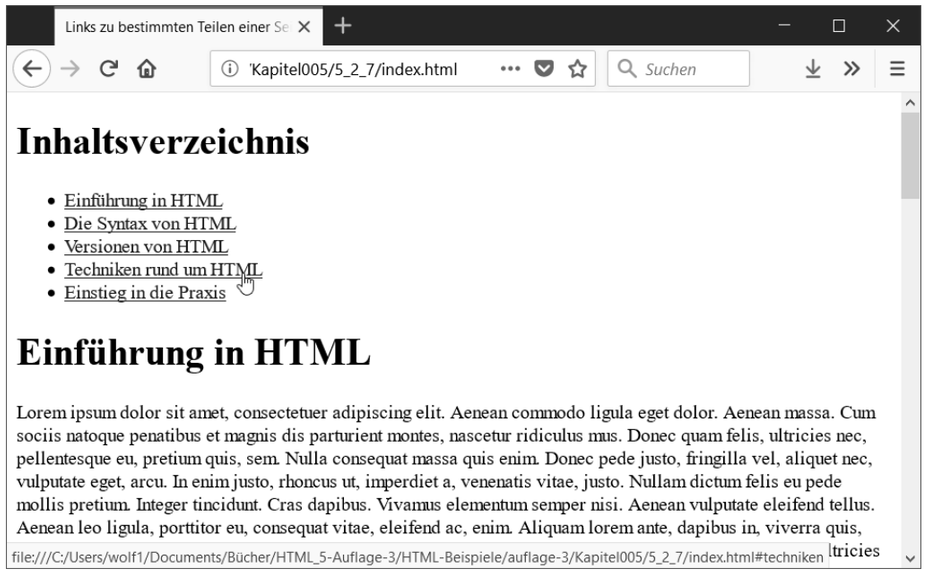

Abbildung 5.23 Ein etwas längeres Dokument, in dem Sie dank Sprungmarken …

Aktivieren Sie z. B. den Link TECHNIKEN RUND UM HTML, wird direkt zum entsprechenden Abschnitt mit der Sprungmarke gesprungen, wie Sie in Abbildung 5.24 sehen. Unterhalb von jedem Abschnitt wurde außerdem ein weiterer Link zur Sprungmarke zurück nach oben zum Inhaltsverzeichnis eingefügt.

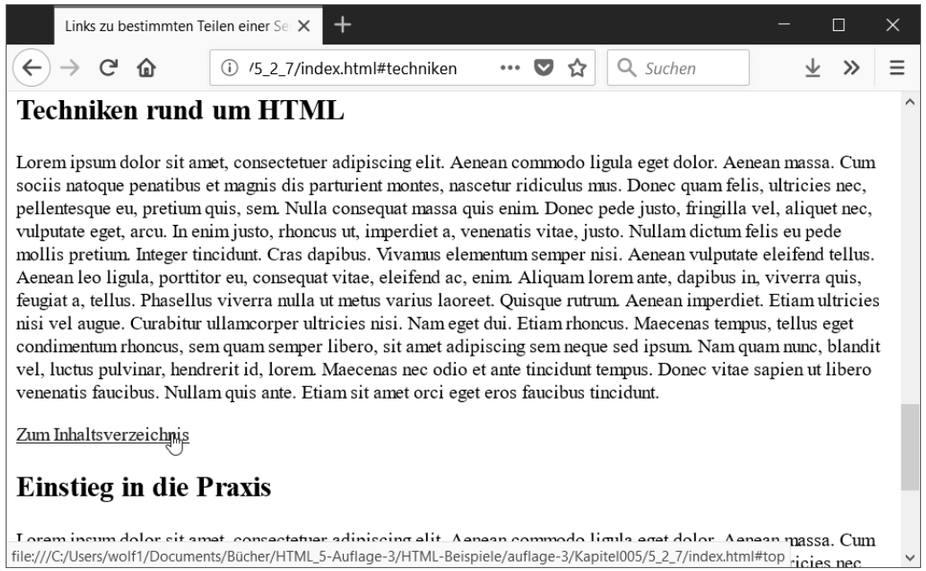

Abbildung 5.24 … schneller zum gewünschten Abschnitt gelangen.

Anker setzen mit dem »id«-Attribut (»id="ankername"«)

Bevor Sie einen Link zu einem bestimmten Teil innerhalb einer Webseite erstellen können, müssen Sie die Sprungmarke (oder auch einen Anker) mit dem globalen Attribut `id` innerhalb eines öffnenden HTML-Tags festlegen. Im Beispiel wurde dies bei den Hauptüberschriften `<h1>` und `<h2>` gemacht (z. B. `<h2 id="techniken">`). Der Attributwert von `id` muss mit einem Buchstaben oder einem Unterstrich anfangen (auf keinen Fall mit einer Zahl) und darf keine Leerzeichen enthalten. Es ist außerdem ratsam, einen aussagekräftigen Namen, um nicht den Überblick zu verlieren, und einen semantisch sinnvollen Namen für das HTML-Dokument zu verwenden. Auf nichtssagende Bezeichnungen wie `anker1`, `anker2` usw. sollten Sie verzichten. Außerdem wird zwischen Groß- und Kleinschreibung unterschieden.

> **»name«-Attribut aus HTML 4.01 oder »id«-Attribut aus HTML5**
>
> Die Verwendung des `id`-Attributs für das Setzen eines Ankers mit z. B. `<h2 id="techniken">` wurde neu mit HTML5 eingeführt. Mit HTML 4.01 wurde dies noch mit dem Attribut `name` realisiert (z. B.: `<h2 name="techniken">`).

Auf einen Anker verweisen mit »#ankername«

Um einen Link zu den Ankern zu verwenden, geben Sie im öffnenden `<a>`-Tag den Attributwert zum Anker in `href` an. Lautet der Anker z. B. `<h2 id="techniken">`, schreiben Sie vor die-

sen Ankernamen (hier mit: `"techniken"`) noch das Doppelkreuz-Zeichen `#`. Bezogen auf unser Beispiel, müssten Sie dies wie folgt notieren:

```
<li><a href="#techniken">Techniken rund um HTML</a></li>
```

Wenn Sie diesen Link aktivieren, wird im HTML-Dokument zu dem Element gesprungen, wo der Wert des Attributs `id` gleich `"techniken"` lautet. In diesem Beispiel wäre dies das `h2`-Element mit der Überschrift *Techniken rund um HTML*.

Links zu einem bestimmten Bereich einer anderen Website erstellen

Genauso ist es möglich, einen Link zu einem Bereich eines anderen HTML-Dokuments zu erstellen. Voraussetzung hierfür ist, dass das andere HTML-Dokument einen entsprechenden Anker enthält. Wenn sich der Anker in einem anderen Dokument befindet, können Sie wie folgt einen Verweis dorthin erstellen:

```
<a href="tech.html#techniken">Techniken rund um HTML</a>
```

Hiermit würden Sie in einem anderen HTML-Dokument, das sich im selben Verzeichnis befindet und dessen Dateiname *tech.html* lautet, zum Bereich mit dem Anker `#techniken` springen.

Befindet sich die Datei mit dem Anker gar auf einer anderen Website, müssen Sie die komplette URL dorthin angeben:

```
<a href="http://www.domain.de/pfad/tech.html#techniken">…</a>
```

Selbstverständlich ist es möglich, einen Link auf Bereiche von fremden Websites zu verwenden. Allerdings sollte klar sein, dass Sie hier keinen Anker setzen, sondern nur bereits vorhandene Anker verlinken können. Hier z. B. ein Link zu einem verankerten Bereich einer Wikipedia-Seite:

```
<a href="http://de.wikipedia.org/wiki/Html#Versionen">…</a>
```

Hier würden Sie direkt zur Wikipedia-Seite mit dem Eintrag *HTML* zum Anker `#Versionen` springen. Dies setzt voraus, dass der Anker existiert – was zur Drucklegung des Buches zwar noch der Fall war, sich aber jederzeit ändern könnte. Wenn der Anker nicht mehr existiert oder falsch ist, wird die Website aufgerufen und der Anker ignoriert, wie dies ohne eine Angabe von `#ankername` bei der Verlinkung mit dem `a`-Element der Fall gewesen wäre.

5.2.8 Die HTML-Attribute für das HTML-Element <a>

Zum Schluss sollen hier noch die HTML-Attribute für die Links erläutert werden, die u. a. für die Suchmaschinen recht nützlich sein können. In Tabelle 5.2 finden Sie eine Übersicht über alle vorhandenen Attribute für das `a`-Element. Einige davon haben Sie bereits kennengelernt.

Attribut	Beschreibung	HTML
`download`	Damit geben Sie an, dass Sie das Verweisziel zum Download anbieten, auch wenn der Webbrowser den Inhaltstyp des Ziels selbst darstellen könnte.	5
`href`	Damit geben Sie die URL der Seite an, zu der der Link führt, wenn er aktiviert wird.	
`hreflang`	Hier können Sie die Sprache des verlinkten Dokuments angeben. Als Angaben sind die üblichen Sprachenkürzel erlaubt (z. B. `de` für Deutschland).	
`media`	Damit können Sie Angaben zu den Medien machen, für die das Verweisziel optimiert wurde. Sie können entweder Medientypen, durch Kommata getrennt, aufzählen oder `all` für alle Medientypen angeben.	5
`rel`	Das Attribut kennen Sie bereits vom `link`-Element aus Abschnitt 3.5.1, »Die HTML-Attribute für das allein stehende HTML-Element <link>«, wohin Sie zurückblättern können, wenn Sie mehr Informationen benötigen. Hiermit bestimmen Sie den Typ der Verlinkung. Speziell für das `a`-Element sind hier noch die `rel`-Attributwerte `bookmark`, `external`, `nofollow` und `noreferrer` von besonderer Bedeutung, da diese nur im `a`-Element verwendet werden können. ▸ `rel="bookmark"`: Hier legen Sie fest, dass das Verweisziel ein übergeordneter Abschnitt (bzw. Dokument) des aktuellen Dokuments ist. Dies stellt praktisch eine Verlinkung zurück zu einem umfangreichen HTML-Dokument dar, wie es bei wissenschaftlichen oder technischen Dokumenten der Fall ist. In der Praxis wird dieser Linktyp auch für Permalinks verwendet, damit Besucher eine ältere Version des aktuellen Dokuments ansehen können. ▸ `rel="external"`: Damit geben Sie an, dass der Link zu einem externen Webangebot gehört. Häufig wird dieser mit CSS noch gesondert formatiert. ▸ `rel="nofollow"`: Damit weisen Sie die Webcrawler an, dem Link nicht zu folgen. ▸ `rel="noreferrer"`: Hiermit weisen Sie den Webbrowser des Besuchers an, beim Anklicken des Links keine Referrer-Adresse zu verwenden, wodurch vermieden werden sollte, dass der Webserver der Zieladresse Informationen erhält, von woher der Besucher gekommen ist. ▸ Nicht verwenden hingegen können Sie die Attributwerte `icon`, `pingback`, `prefetch` und `stylesheet` für `a`-Elemente.	

Tabelle 5.2 Attribute für Links mit <a>-Element

Attribut	Beschreibung	HTML
target	Hier tragen Sie ein, wo das Verweisziel geöffnet werden soll. Mögliche Werte dafür sind: ▶ _blank: neues Fenster/Tab ▶ _parent: Eltern-Fenster ▶ _self: aktuelles Fenster ▶ _top: oberste Fenster-Ebene ▶ framename: Name des Fensters, das mit JavaScript geöffnet und auch darin vergeben wurde	
type	Damit können Sie dem Webbrowser den MIME-Typ (Dateiformat) nennen, zu dem die verlinkte Datei gehört. Eine Liste bekannter MIME-Typen finden Sie in Abschnitt A.19, »HTML-Zeichenreferenz gängiger benannter Zeichen«. Diese Angabe ist sinnvoll, wenn das Ziel kein HTML-Dokument ist.	5

Tabelle 5.2 Attribute für Links mit <a>-Element (Forts.)

Veraltete Attribute

Die ehemaligen Attribute charset, coord, name, rev und shape werden nicht mehr von HTML5 unterstützt und sind daher nicht in dieser Tabelle aufgelistet.

5.3 Zusammenfassung

In diesem Kapitel haben Sie einige essenzielle HTML-Elemente kennengelernt. Die wichtigsten Elemente, die Sie vermutlich auf fast jeder Website vorfinden und verwenden werden, sind:

- a-Element, mit dem Sie Hyperlinks erzeugen
- Tabellen, mit denen Sie zusammenhängende Daten und Informationen in einem Raster aus Zeilen und Spalten präsentieren

Kapitel 13

Responsive Layouts mit CSS erstellen

Wenn es um das Thema Layouts von Websites geht, dürfte die Wahl meistens auf die responsiven Varianten fallen, weil Sie sich hierbei keine Gedanken mehr um die verschiedenen Bildschirmgrößen machen müssen. In diesem Kapitel finden Sie eine Einführung, wie Sie selbst responsive Websites erstellen können.

Es ist nicht in Stein gemeißelt, wie Sie Ihre Layouts erstellen sollen, noch gibt es hier ein richtig oder falsch. In der ersten Auflage des Buches wurden hier zur Erstellung von Layouts auch das Positionierungs-Modell mit der Eigenschaft `position` und das Float-Modell mit `float` beschrieben, ohne die Bildschirmbreite zu berücksichtigen. In dieser Auflage habe ich mich entschieden, hier nur noch responsive Layouts zu demonstrieren. Sollten Sie trotzdem noch an der Erstellung von Layouts mit den Eigenschaften `position` und `float` interessiert sein, finden Sie in Anhang E Informationen dazu.

Zuvor noch eine Übersicht, was Sie in diesem Kapitel lernen:

- Sie lernen den grundlegenden Umgang mit Medienabfragen (*Media Queries*) kennen.
- Auch lernen Sie grundlegend, worauf es beim responsiven Webdesign ankommt.
- Sie erfahren, wie Sie ein einfaches responsives Layout erstellen können.

Um hier die Erwartungen etwas zu dämpfen, sollte noch erwähnt werden, dass Sie hier nur grundlegend erfahren, was responsive Layouts sind und wie Sie sie in der Praxis verwenden können. Die Beispiele in diesem Kapitel sind relativ einfach gehalten. Das Thema »responsives Webdesign« ist ein umfangreiches, sich stets weiterentwickelndes Thema, und nicht umsonst widmen sich ihm ganze Bücher. Dennoch wissen Sie am Ende des Kapitels, worum es hier geht wie Sie selbst responsive Layouts erstellen können.

13.1 Theoretisches Grundlagenwissen zum responsiven Webdesign

Die Art und Weise, wie heute auf das Internet zugegriffen wird, ist sehr vielseitig geworden. Wurde vor ein paar Jahren noch lediglich mit einem Desktop-PC oder Laptop eine Website betrachtet, so sind heute viele weitere Geräte wie Tablets, Smartphones, E-Book-Reader, Spielekonsolen oder Fernsehgeräte hinzugekommen. Die Herausforderung dabei ist, auf die

Bildschirmgröße und die Bildschirmauflösung der einzelnen Geräte mit einem passenden Layout zu reagieren.

Das Erscheinungsbild einer Website war vor der Zeit des responsiven Webdesigns ziemlich stark von dem Endgerät abhängig, mit dem sie betrachtet wurde. Hatte man in der Vergangenheit noch die Websites für die Bildschirmauflösung eines Desktop-PCs optimiert, müssen Sie heute aufgrund des stark steigenden Anteils an Smartphones und Tablets etwas umdenken. Standardauflösungen von Smartphones beginnen bei 320 bis 480 Pixel, für Tablets gelten häufig 768 bis 1.024 Pixel, und gängige Desktop-PCs beginnen ab 1.024 Pixel. Verschiedene Mobil- und Desktopversionen einer Website anzubieten, wäre eine Möglichkeit, das Problem zu lösen. Der Aufwand ist allerdings nicht unbeachtlich, und wenn ein neues Tablet- oder Smartphone-Format der nächsten Generation ansteht, wird eine weitere Version des Layouts nötig.

Meistens waren die rein mobilen Versionen lediglich abgespeckte Versionen mit einer reduzierten Funktionalität der Desktopvariante, die mit einem einspaltigen Layout auf einer Subdomain mit *m* oder *mobil* (z. B.: *mobil.mydomain.de*) ausgelagert wurden. Spätestens mit den mobilen Geräten wie Smartphones und Tablets waren diese abgespeckten mobilen Varianten einer Website nicht unbedingt befriedigend. Zusätzlich waren und sind die Webbrowser auf den mobilen Geräten technisch auf einem sehr hohen Niveau und den Desktopvarianten mindestens ebenbürtig. Es wäre daher schade, das Potenzial mit einer abgespeckten mobilen Version zu verschenken.

Mobile dominiert

Neueste Statistiken bestätigen den Trend, dass mobile Endgeräte mittlerweile die am meisten verwendeten Geräte sind, wenn Besucher im Web unterwegs sind.

Anstatt unzählige Layoutversionen für dieselbe Website zu erstellen und einzupflegen, kommt das *responsive Webdesign* oder auch *responsive Layout* zum Einsatz. Bei dieser Technik werden die Eigenschaften des Endgeräts berücksichtigt, um die Website so anzupassen, dass eine für das Endgerät optimale und benutzerfreundliche Darstellung erreicht wird. Das Hauptkriterium für ein solches angepasstes Layout ist die Bildschirmgröße (meistens Breite) des Geräts und eventuell die vorhandenen Eingabemethoden (Maus oder Touchscreen).

Wörtlich übersetzt bedeutet *responsive* auf Deutsch so viel wie *reagierend* oder *reaktionsfähig* – also *reaktionsfähiges Webdesign*. Das hört sich seltsam an und wird eher selten so verwendet, aber es trifft die Sache schon recht gut, weil mit dieser Technik der strukturelle Aufbau und Inhalt einer Website auf die Bildschirmauflösung des Endgeräts des Benutzers reagieren und das Layout entsprechend ausgegeben wird. Wenn also die Rede von responsiven Layouts ist, dann passt sich die Website dem Bildschirm des Nutzers an.

13.1.1 Die Verwendung von Medientypen mit CSS 2

Die Idee, auf bestimmte Medientypen zu reagieren, gab es bereits in CSS Level 2, was ich kurz in Abschnitt 8.3.7, »Stilanweisungen aus einer externen CSS-Datei mit ›@import‹ einbin-

den«, behandelt habe. Dabei haben wir verschiedene separate Stylesheets für die unterschiedlichen Ausgabemedien wie z. B. den Bildschirm (`media="screen"`) oder den Drucker (`media="print"`) zur Verfügung gestellt:

```
…
<link href="css/screen.css" rel="stylesheet" media="screen">
<link href="css/print.css" rel="stylesheet" media="print">
…
```

Listing 13.1 /Beispiele/Kapitel013/13_1_1/index.html

Mithilfe des `link`-Elements stellen Sie eine Version für den Bildschirm (`media="screen"`) und eine spezielle Version für den Drucker zur Verfügung. Die Version für den Bildschirm mit *screen.css* sehen Sie in Abbildung 13.1.

Die zweite Version mit *print.css*, die ebenfalls mit dem `link`-Element zur Verfügung gestellt wurde, ist für den Drucker (`media="print"`) als Ausgabemedium bestimmt. Ohne hier auf den Inhalt der CSS-Datei einzugehen, wurde bei dieser Version auf Farbe verzichtet, und es wurden die Rahmen entfernt. Außerdem wurden die Elemente `<header>`, `<nav>`, `<aside>` und `<footer>` mit `display: none` versteckt, um nur den eigentlichen Inhalt der `article`-Elemente auszudrucken. Die Version für den Drucker mit *print.css* sehen Sie in Abbildung 13.2.

Wird der Medientyp hingegen nicht definiert, dann gelten die CSS-Anweisungen automatisch für alle Ausgabetypen; dies entspricht somit `media="all"`.

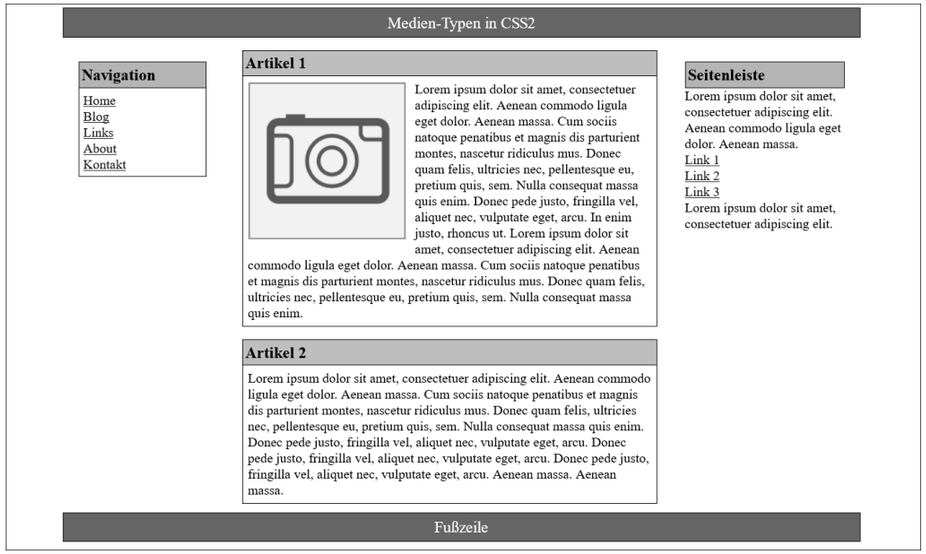

Abbildung 13.1 Die Webseite wurde mit der CSS-Version für den Bildschirm (»media="screen"«) gestylt.

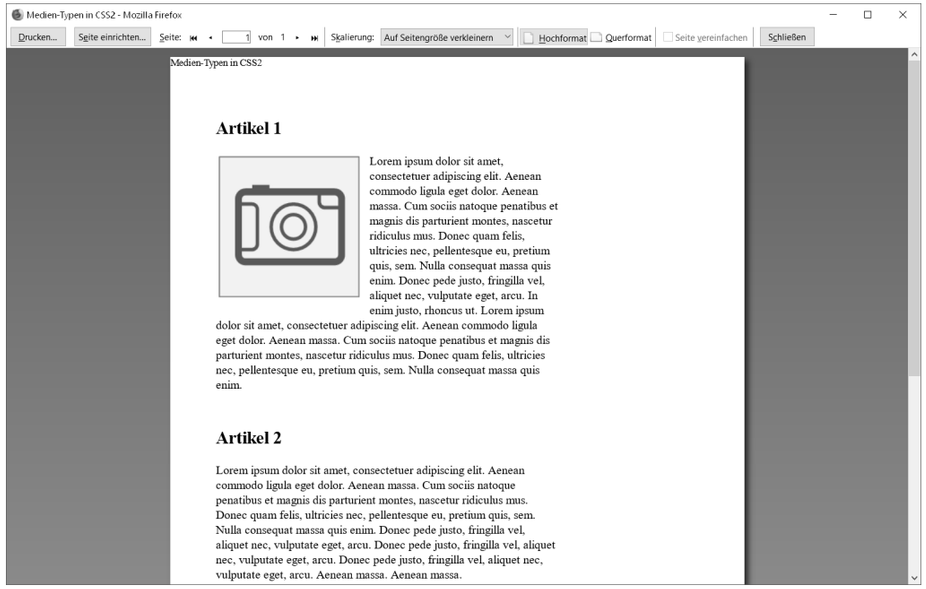

Abbildung 13.2 Die Version »print.css«, die für den Drucker (»media="print"«) verwendet wird, bei der Ausführung

Die beiden CSS-Dateien *print.css* und *screen.css* zu diesem Beispiel finden Sie unter */Beispiele/Kapitel013/13_1_1/css/*.

Medienspezifische Abschnitte mit der CSS-Regel »@media« definieren

Innerhalb einer CSS-Datei können Sie mithilfe von `@media [medientyp]` die CSS-Eigenschaften für unterschiedliche Medientypen in geschweiften Klammern definieren:

```
@media screen {
  /* CSS-Eigenschaften für den Bildschirm */
}
@media print {
  /* CSS-Eigenschaften für den Drucker */
}
```

Sie können auch im HTML-Dokument innerhalb eines `style`-Elements die Formatierung in CSS mit einer `@media`-Regel zusammenfassen:

```
…
    <style>
      @media screen {
        /* CSS-Eigenschaften für den Bildschirm */
        …
      }
      @media print {
        /* CSS-Eigenschaften für den Drucker */
        …
      }
    </style>
…
```


»media="handheld"« funktioniert nicht bei Tablets oder Smartphones

Der Wert `handheld` des Attributs `media` war schon für Tablets und Smartphones gedacht, also Geräte mit kleinen Bildschirmen. Aber keines der Geräte beachtete wirklich `media="handheld"`, sondern ignorierte es gewöhnlich. Dies hat damit zu tun, dass mit der Einführung des iPhones von Apple nur wenige Websites für den Medientyp `handheld` erstellt wurden, was bedeutet hätte, dass sich vieles nicht hätte damit darstellen lassen. Daher reagierte schon das erste iPhone von Apple nur auf `screen` als Medientyp. Daran haben sich die anderen Tablet- und Smartphone-Hersteller gehalten. Somit gilt, dass Smartphones und Tablets sich nur mit dem Medientyp `screen` kleiden.

13.1.2 Die mächtigen Medienabfragen für Medieneigenschaften

Mit CSS3 wurden die Medientypen um Medienabfragen (*Media Queries*) von Medieneigenschaften (*Media Features*) erweitert. Solche Abfragen stellen gleichzeitig das Herzstück beim responsive Webdesign dar, auf dem diese Technik aufbaut. Damit können Sie Medienabfragen machen – etwa bezüglich der Größe des Geräts, Bildschirmauflösung, Orientierung (Hoch- oder Querformat) oder Eingabemöglichkeiten (Maus, Touch, Tastatur, Sprache) – und entsprechend mit einem passenden Design reagieren.

Welcher Webbrowser kann diese Medieneigenschaften abfragen?

Jeder moderne Webbrowser kann mit den Medieneigenschaften umgehen. Auch der Internet Explorer versteht diese Medienabfrage bereits seit Version 9. Älteren Webbrowsern können Sie diese Medieneigenschaften u. U. auch mit einem JavaScript wie *css3-mediaqueries-js* (*https://code.google.com/p/css3-mediaqueries-js*) beibringen. Aber auch wenn Sie kein spezielles JavaScript für alte Webbrowser verwenden, werden diese CSS-Regeln mit den Medienabfragen einfach ignoriert, und es wird die Basisversion verwendet, die Sie als Allererstes definiert haben.

13.1.3 Einbinden und Anwenden von Medienabfragen für Medieneigenschaften

Die Medieneigenschaften können auf verschiedene Arten eingebunden und verwendet werden. Die Verwendung einer solchen Medienabfrage in HTML können Sie z. B. wie folgt notieren:

```
…
<head>
  <link rel="stylesheet" href="css/basis.css">
  <link rel="stylesheet" media="screen and (max-width: 480px)"
       href="css/mobile.css">
</head>
…
```

Hier wird *mobile.css* nur dann verwendet, wenn die maximale Bildschirmbreite von 480 Pixeln nicht überschritten wird. Bei Geräten mit einer höheren Auflösung wird *basis.css* benutzt. Ältere Webbrowser, die diese Medienabfrage (hier: `media="screen and (max-width: 480px)"`) nicht kennen, ignorieren diese Abfrage und verwenden immer *basis.css* – auch wenn der Bildschirm unter 480 Pixel breit ist.

Es ist ebenfalls wichtig zu wissen, dass bei der Verwendung von Medienabfragen, wie hier mit dem `link`-Element, alle vorhandenen Stylesheets heruntergeladen werden, auch wenn diese bei der Abfrage gar nicht zutreffen – sie werden allerdings nicht ausgeführt. Im Beispiel oben werden also immer *basis.css* und *mobile.css* geladen. Der Grund dafür ist, dass eine eventuelle Verzögerung durch Nachladen verhindert werden soll, wenn die Größe des Webbrowserfensters geändert wird oder sich die Orientierung des Smartphones oder Tablets ändert.

Das Einbinden von Abfragen im öffnenden `<style>`-Tag ist wie folgt möglich:

```
…
<style type="text/css" media="screen and (max-width: 480px)">
   /* CSS-Anweisungen für Bildschirm bis max. 480 Pixel */
</style>
…
```

Neben dem `link`- und `style`-Element können Sie Medienabfragen von Eigenschaften auch als `@media`-Regel innerhalb eines Stylesheets notieren:

```
…
.mainarticle {
  background-color: yellow;
}
@media screen and (max-width: 480px) {
  .mainarticle {
    background-color: orange;
  }
}
…
```


Hier wird die `@media`-Regel, den Hintergrund von `.mainarticle` orange zu färben, nur verwendet, wenn die maximale Bildschirmgröße nicht die 480 Pixel überschritten hat. Ansonsten wird der Hintergrund von `#mainarticle` mit gelber Farbe eingefärbt.

Zu guter Letzt können die Abfragen der Medieneigenschaften mit der `@import`-Regel wie folgt verwendet werden:

```
@import url('css/mobile_480.css') screen and (max-width: 480px);
```

Somit können Sie die Abfragen von Medieneigenschaften in *HTML* mit dem `link`-Element oder im `style`-Element und in *CSS* mit der `@media`- oder `@import`-Regel verwenden.

13.1.4 Der grundlegende Aufbau einer Abfrage von Medieneigenschaften

Nun, da Sie wissen, wie Sie Medienabfragen in HTML oder in CSS verwenden können, sollten wir den Aufbau einer solchen Abfrage etwas genauer betrachten. Hierzu zerlegen wir die bereits des Öfteren abgedruckte Abfrage `screen and (max-width: 480px)` in ihre einzelnen Bestandteile.

In Abbildung 13.3 sehen Sie eine Media Query und ihre einzelnen Bestandteile. Eine solche Abfrage besteht aus einem *Medientyp* (bzw. Ausgabegerät), gefolgt von einer *Verknüpfung* wie hier mit `and`. Innerhalb des *Ausdrucks* werden ein *Medienmerkmal* (oder auch Eigenschaft) und ein entsprechender *Wert* zwischen zwei runden Klammern notiert.

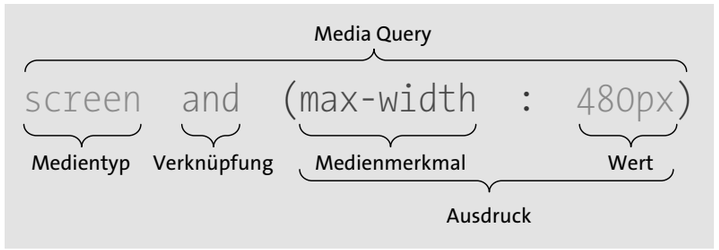

Abbildung 13.3 Die einzelnen Bestandteile einer Media Query

Die Medientypen bzw. die Ausgabegeräte einer Media Query

Die Medientypen bzw. die Ausgabegeräte habe ich bereits in Abschnitt 8.3.7, »Stilanweisungen aus einer externen CSS-Datei mit ›@import‹ einbinden«, behandelt und aufgelistet. Dazu muss noch hinzugefügt werden, dass CSS zwar die Schlüsselwörter wie `screen`, `print`, `handheld`, `tv` und noch mehr definiert, aber es kann nicht immer genau gesagt werden, welche Geräte tatsächlich welchem Medientyp zugeordnet sind. Es handelt sich dabei lediglich um eine Richtlinie. Bestes Beispiel sind z. B. Smartphones wie das iPhone oder Android-Systeme, bei denen Sie vielleicht den Medientyp `handheld` vermuten würden. Wie bereits erwähnt, ordnen sich die Webbrowser auf diesen mobilen Geräten dem Medientyp `screen` zu.

Die Medieneigenschaften verknüpfen

Die Medieneigenschaft wird mit dem Schlüsselwort `and` verknüpft. Dabei können durchaus mehrere `and`-Merkmale verknüpft und verarbeitet werden. Die Verknüpfung kann mit und ohne Medientyp erfolgen. Ein theoretisches Beispiel für mehrere Verknüpfungen kann demnach wie folgt aussehen:

```
@media screen and (min-width: 960px) {
  /* CSS-Anweisungen für Desktop */
}
@media screen and (min-width: 768px) and (max-width: 960px) {
  /* CSS-Anweisungen für Tablets und Netbooks */
}
@media screen and (max-width: 480px) {
  /* CSS-Anweisungen für Smartphones */
}
```

Bei der zweiten Medienabfrage wird das Stylesheet dazwischen nur dann verwendet, wenn alle mit `and` verbundenen Ausdrücke und Kriterien erfüllt werden. Im Beispiel muss der Medientyp ein Bildschirm sein, *und* die Bildschirmbreite muss mindestens 768 Pixel *und* darf maximal 960 Pixel betragen.

Wenn Sie einen Medientyp verwenden, können Sie vor den Medientyp noch eine Angabe mit `only` setzen. Mit `only` sorgen Sie nur dafür, dass ältere Webbrowser quasi gar nichts mit der Media Query anfangen können. Hört sich recht sinnlos an, ist es aber nicht. Zunächst ein Beispiel ohne `only`:

```
…
@media screen and (max-width: 480px) {
  /* CSS-Anweisungen für Smartphones */
}
…
```

Ein alter Webbrowser kann vielleicht nichts mit dem in CSS3 eingeführten `and (max-width: 480px)` anfangen, aber kennt das in CSS 2 eingeführte `@media screen`. Um sicherzugehen, dass der Webbrowser die Angabe von `and (max-width: 480px)` ignoriert und somit die CSS-Anweisungen für Smartphones auch auf einem Desktop verwendet, können Sie das Schlüsselwort `only` vor `screen` setzen, weil ältere Webbrowser dann mit der Abfrage gar nichts mehr anzufangen wissen:

```
…
@media only screen and (max-width: 480px) {
  /* CSS-Anweisungen für Smartphones */
}
…
```


Sie können auch, wenn Sie einen Medientyp verwendet haben, `not` voranstellen, womit Sie eine Abfrage negieren (verneinen).

13.1.5 Welche Medieneigenschaften können abgefragt werden?

Die vielen verschiedenen Ausgabegeräte verfügen über viele verschiedene Eigenschaften. Zweifelsohne ist das häufigste verwendete Merkmal, das abgefragt wird, die Minimal- und Maximalbreite der Anzeigefläche. In Tabelle 13.1 finden Sie eine Übersicht über die wichtigsten Medieneigenschaften, die abgefragt werden können. Eine Übersicht über alle Medienmerkmale finden Sie auf *https://www.w3.org/TR/mediaqueries-4/*. Die Medieneigenschaften die in der Tabelle mit dem Präfix `device-` beginnen, sind mittlerweile veraltet und werden nur aufgelistet für den Fall, dass Sie mal darauf stoßen sollten.

> **Die Präfixe »min-« und »max-«**
>
> Gerade bei den Medienmerkmalen für den Anzeigebereich ist es meistens sinnvoller, die Versionen mit den Präfixen `min-` bzw. `max-` zu verwenden, da man selten die genaue Anzeigebreite des Anwenders kennt. Also anstatt z. B. eine Media Query zu verwenden, bei der die exakte Breite mit `width` abgefragt wird, sollten Sie die Version `min-width` und/oder `max-width` bevorzugen, die bereits reagiert, wenn die Anzeigebreite mindestens oder höchstens dem übergebenen Wert entspricht.

Medieneigenschaft	Bedeutung	Werte

`width`
`min-width`
`max-width`
	Breite der Anzeigefläche (Viewport) des Webbrowsers. Mögliche Werte sind positive Längenangaben. Beispiel: `(min-width: 480px)`	`px`, `%`, `em`

`height`
`min-height`
`max-height`
	Höhe der Anzeigefläche (Viewport) des Webbrowsers. Mögliche Werte sind positive Längenangaben. Beispiel: `(max-height: 720px)`	`px`, `%`, `em`

`device-width`
`min-device-width`
`max-device-width`
	Deprecated! Breite des Geräts. Mögliche Werte sind positive Längenangaben. Auch wenn das Gerät diese Abmessung besitzt, können Sie sich nicht darauf verlassen, dass dieser verfügbare Bereich verwendet werden kann. Beispiel: `(min-device-width: 480px)`	`px`, `%`, `em`

Tabelle 13.1 Einige gängige Medienmerkmale, die mit Media Queries abgefragt werden können

Medieneigenschaft	Bedeutung	Werte

`device-height`
`min-device-height`
`max-device-height`
	Deprecated! Höhe des Geräts. Mögliche Werte sind positive Längenangaben. Auch wenn das Gerät diese Abmessung besitzt, können Sie sich nicht darauf verlassen, dass dieser verfügbare Bereich auch verwendet werden kann. Beispiel: `(max-device-height: 480px)`	`px`, `%`, `em`

`orientation`	Damit fragen Sie die Ausrichtung des Geräts ab. Die Orientierung kann Hochformat (`portrait`) oder Querformat (`landscape`) sein. Im Hochformat ist der Wert von `height` höher als der von `width`. Beim Querformat ist es umgekehrt. Beispiel: `(orientation: landscape)`	`portrait`, `landscape`

`aspect-ratio`
`min-aspect-ratio`
`max-aspect-ratio`
	Gibt das Seitenverhältnis von `width` und `height` zueinander an. Ein Wert von 1.280 × 720 entspricht einem Seitenverhältnis von 16:9, das Sie mit `(aspect-ratio: 16/9)` ansprechen können. Diese Angabe entspricht aber auch der Angabe `(aspect-ratio: 1280/720)`.	Breite/Höhe, z. B. `16/9`, `1280/720`

`device-aspect-ratio`
`min-device-aspect-ratio`
`max-device-aspect-ratio`
	Deprecated! Wie eben bei `aspect-ratio` (und den `min-` und `max`-Versionen), nur gilt hier das Seitenverhältnis des Geräts von `device-width` und `device-height` zueinander.	Breite/Höhe, z. B. `16/9`, `1280/720`

`color` `min-color`/`max-color`	Abfrage der Farbtiefe des Geräts. Bei Schwarzweißgeräten ist der Wert für `color:0`.	Ganzzahlwert (integer)
`color-index`, `min-color-index`, `max-color-index`	Prüft die Verwendung von indizierten Farben des Ausgabegerätes.	Ganzzahlwert (integer)

Tabelle 13.1 Einige gängige Medienmerkmale, die mit Media Queries abgefragt werden können (Forts.)

Medieneigenschaft	Bedeutung	Werte
`monochrome`, `min-monochrome`, `max-monochrome`	Prüft die Farbtiefe des Gerätes, ob es sich um ein monochromes Ausgabegerät handelt. Der Wert `monochrome:0` wäre kein monochromes Gerät.	Ganzzahlwert (integer)

`resolution`
`min-resolution`
`max-resolution`
	Abfrage der Pixeldichte des Geräts, z. B.: `(resolution: 72dpi)`	`dpi`, `dcm`

`pointer`, `any-pointer`	Testet, ob das Ausgabegerät eine Maus als Eingabegerät (oder überhaupt ein Eingabegerät) bereitstellt.	`none`, `coarse`, `fine`
`hover`, `any-hover`	Prüft, ob das Ausgabegerät Hovereffekte beim primären Eingabegerät zur Verfügung stellt.	`none`, `hover`

Tabelle 13.1 Einige gängige Medienmerkmale, die mit Media Queries abgefragt werden können (Forts.)

Die Medieneigenschaften `pointer`, `any-pointer`, `hover` und `any-hover` für Interaktionen wurden zur Drucklegung noch überhaupt nicht von Firefox unterstützt (*https://caniuse.com/#search=css-media-interaction*).

13.1.6 Von enormer Bedeutung: der Viewport für mobile Geräte

Gerade in Bezug auf die Abfrage von Medieneigenschaften von mobilen Geräten spielt der Viewport eine essenzielle Rolle, damit das responsive Webdesign auch dort so funktioniert, wie es vorgesehen ist. Hier sorgen häufig der Viewport bei Desktoprechnern und der Viewport auf mobilen Geräten für ein wenig Verwirrung. Dass hier noch hochauflösende Displays dazukommen, macht die Sache eher noch komplizierter, weil ein Pixel auf einmal kein Pixel mehr ist. Ein Blick auf die Website *http://screensiz.es/* zeigt Ihnen eine Sammlung der vielen verschiedenen Größen von Displays auf unterschiedlichen Geräten. In der Sammlung finden Sie auch unterschiedliche Breitenangaben für »width« und »device-width«, was auf die eben erwähnten hochauflösenden Displays zurückzuführen ist. Ich werde es an dieser Stelle möglichst einfach halten und Sie nicht mit den verschiedenen Viewport-Begriffen konfrontieren. Vielmehr zeige ich Ihnen, wie Sie das Problem mit einer einzigen Zeile lösen können.

Bezogen auf Desktoprechner, ist der Viewport der innere Bereich des Browserfensters ohne die Ränder. Wenn Sie das Browserfenster verkleinern oder vergrößern, wird auch der Viewport verkleinert bzw. vergrößert. Diesen visuellen Viewport können Sie mit den Medieneigenschaften `width` und `height` ansprechen. Bei mobilen Geräten wie einem Smartphone sind die Bildschirme zwar deutlich kleiner als bei einem Desktoprechner, aber der Viewport dort ist häufig trotzdem größer als bei Desktopbildschirmen. Ohne spezielle Anpassungen des Viewports für mobile Geräte würde daher die Website auf diesen Geräten in der Breite eines typischen Desktopbildschirmes angezeigt. Der Viewport auf mobilen Geräten wird häufig auch als *Layout-Viewport* bezeichnet.

In Abbildung 13.4 sehen Sie die Website *www.tagesschau.de* auf einem Desktopbildschirm und in Abbildung 13.5 dieselbe Website im Chrome-Browser eines Smartphones, wo der Viewport nicht für mobile Geräte angepasst wurde. Beim Mobilgerät wurde daher die Ansicht automatisch skaliert, sodass sie auf den gesamten Bildschirm passt wie etwa bei einem Desktopbildschirm. Hierbei wird quasi der Layout-Viewport des Mobilgeräts auf den visuellen Viewport skaliert. Als Ergebnis erhalten Sie einen unlesbaren Text und alles in einer viel zu kleinen Miniaturdarstellung. Um den Text und die Berichte auf dem Mobilgerät zu lesen, müssten Sie hineinzoomen und dann nach oben und unten bzw. nach rechts und links scrollen.

Abbildung 13.4 Die Website »tagesschau.de« auf einem gewöhnlichen Desktopbildschirm

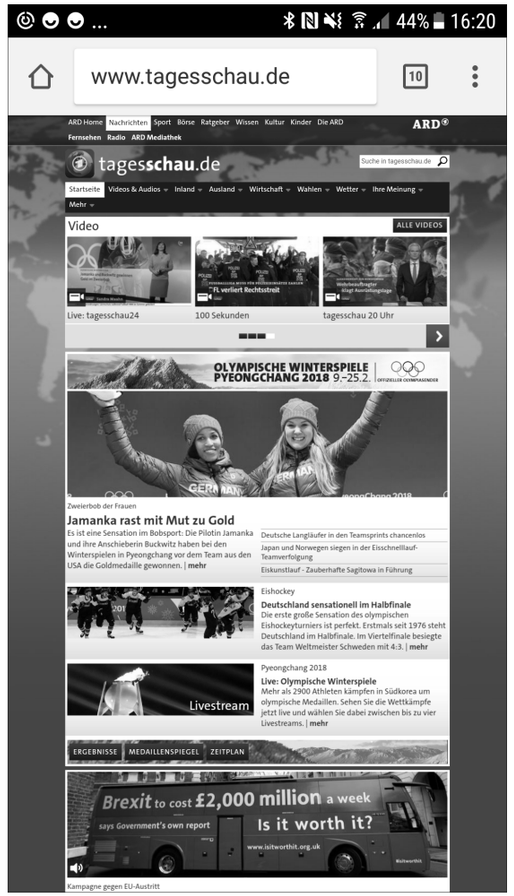

Abbildung 13.5 Die Website »tagesschau.de« ohne angepassten Viewport auf einem Smartphone

Das Problem mit den Mobilgeräten lässt sich ganz einfach mit dem Metatag `viewport` oder der CSS-Regel `@viewport` beheben. Hierzu fügen Sie im `head`-Bereich des HTML-Dokumentes Folgendes hinzu:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Mit `width=device-width` setzen Sie die Breite des Layout-Viewportes auf die Breite des visuellen Viewports. Mit dieser Zeile werden alle unterschiedlichen Layout-Viewports von verschiedenen Geräten normalisiert und an die aktuelle Displaygröße angepasst. Sie müssen

sich dann nicht mehr um die verschiedenen Displaygrößen kümmern und können sich in Ruhe dem responsiven Webdesign widmen. Neben `width` gibt es hier das Gegenstück mit `height=device-height` für die Höhe, das Sie in der Praxis allerdings selten benötigen. Für die Viewport-Einstellung `width` können Sie auch Pixelwerte verwenden.

Ebenfalls eine wichtige Viewport-Eigenschaft ist `initial-scale:1.0`, mit der Sie den anfänglichen Zoomwert auf 100 % oder 1:1 stellen. Hierbei können Sie auch kleinere oder größere Werte als Anfangszoomstufe definieren.

Neben `initial-scale` gibt es mit `minimum-scale` oder `maximum-scale` weitere Viewport-Eigenschaften, mit denen Sie die minimale und maximale Zoomstufe festlegen. Mit `user-scalable=no` unterbinden Sie das Zoomen sogar komplett. Das mag für Web-Apps vielleicht praktisch sein, aber bei Websites ist von Einschränkungen der Zoomstufe abzuraten. Bedenken Sie, dass es Website-Besucher gibt, die auf eine hohe Zoomstufe angewiesen sind. Solche Besucher schließen Sie damit von der Website aus.

»@viewport«-Regel

In Zukunft dürfte wohl die CSS-Regel `@viewport` das Metatag ablösen. Derzeit ist die Browserunterstützung allerdings noch sehr gering (*https://caniuse.com/#search=%40viewport*). Der Vorteil der CSS-Regel wäre, dass hiermit theoretisch mehrere Möglichkeiten als beim Viewport-Metatag verwendet und somit in verschiedenen Medienabfragen mit unterschiedlichen Angaben deklariert werden können. Das gleichwertige Gegenstück zum Viewport-Metatag oben sieht wie folgt aus:

```
@viewport {
  width:device-width;
  zoom:1;
}
```

Diese Angabe entspricht demselben wie:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Voraussetzung dafür, dass Sie den Viewport auf mobilen Geräten anpassen, ist natürlich, dass Sie auch eine per Media Queries optimierte Darstellung der Website anbieten. Die Einstellung des Viewports allein sorgt lediglich dafür, dass Sie die automatische Skalierung des Webbrowsers abschalten und von nun an alles selbst in die Hand nehmen. Wie Sie das responsive Layout dafür erstellen, erfahren Sie in Kürze. Bezogen auf die Website *tagesschau.de* und Abbildung 13.5 oben sieht die Website mit Anpassung des Viewports für mobile Geräte aus wie in Abbildung 13.6. Natürlich haben die Webentwickler von *tagesschau*.de auch eine Extraversion für mobile Geräte via Media Queries zur Verfügung gestellt, die ich in Abbildung 13.5 zur Demonstration deaktiviert habe.

Abbildung 13.6 Die Website der »tagesschau.de« mit angepassten Viewport für mobile Geräte

13.1.7 Verwenden Sie »em« anstatt Pixel für einen Layoutumbruch in Media Queries

Ebenfalls recht hilfreich ist es, die Medienabfragen mit der Einheit `em` durchzuführen. Das mag auf dem ersten Blick suspekt erscheinen, weil ja eigentlich der Bildschirm in Pixel gemessen wird. Aber der Vorteil hierbei ist, dass die Medienabfrage dann auch bei einer Schriftgrößenänderung über das Betriebssystem oder die Funktion Nur Text zoomen in Firefox korrekt funktioniert. Damit stellen Sie sicher, dass bei einer größeren Textdarstellung der Schriften auch wirklich die nächste Layoutstufe angesteuert wird und das Layout nicht zusammenbricht. Bei der Website in Abbildung 13.7 wurde beispielsweise der Text mit der Funktion Nur Text zoomen in Firefox maximal gezoomt. In Abbildung 13.8 sehen Sie, was passiert, wenn hier Pixel anstelle von `em` verwendet werden. In Abbildung 13.9 hingegen wurde `em` als Einheit für den Layoutumbruch bei den Media Queries verwendet, und jetzt reagiert auch das Layout auf die vergrößerte Textdarstellung und wählt die nächste Layoutstufe aus. Auch hier natürlich vorausgesetzt, es wurde ein entsprechendes Layout dafür zur Verfügung gestellt.

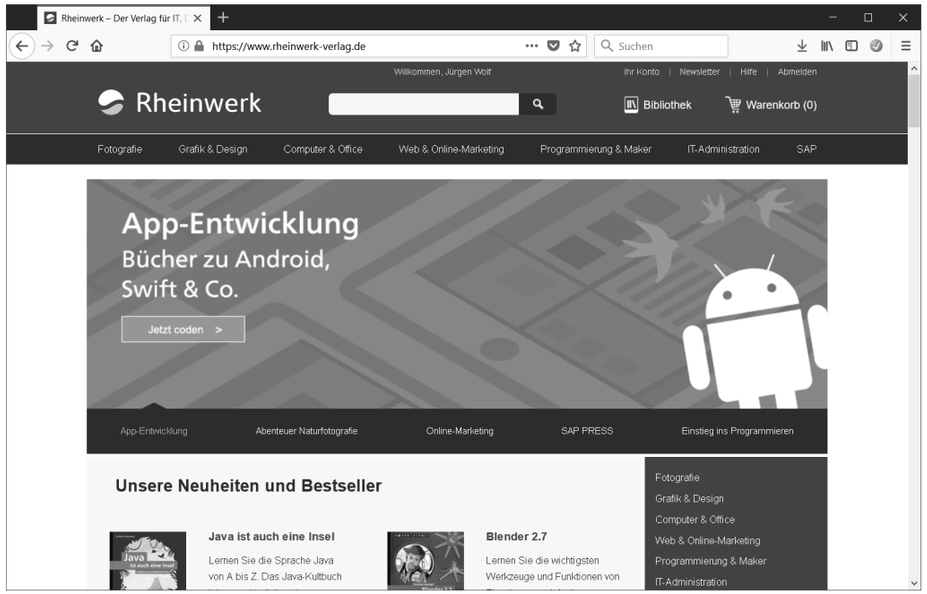

Abbildung 13.7 Die Website nach dem Laden im Firefox-Webbrowser

Abbildung 13.8 Hier wurde die Funktion »Nur Text zoomen« verwendet, aber es wurden Pixel für den Layoutumbruch bei den Media Queries benutzt. Das Layout ist hinüber.

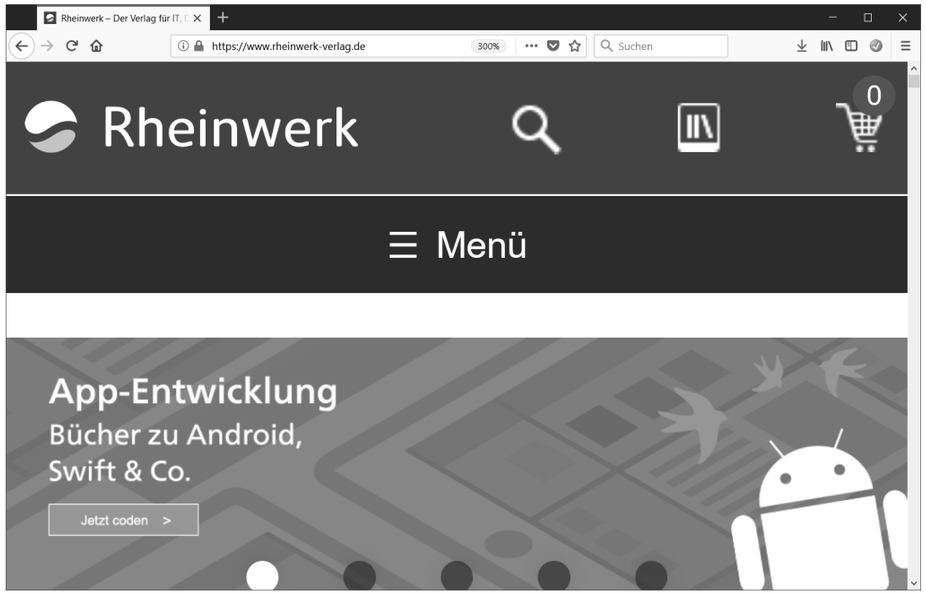

Abbildung 13.9 So sollte es aussehen, wenn die Funktion »Nur Text zoomen« ausgeführt und die Einheit em beim Layoutumbruch der Media Queries verwendet wird. Hier wird jetzt das mobile Layout ausgeführt.

Betrachten Sie hierzu folgende Media Query:

```
@media screen and (min-width: 640px) {
   /* CSS-Anweisungen für Bildschirme ab 640 Pixel Breite */
}
```

Hier haben Sie einen Layoutumbruch (auch *Breakpoint* genannt) für Bildschirme ab 640 Pixel eingerichtet. Alle CSS-Anweisungen zwischen den geschweiften Klammern werden somit nur ab einer Bildschirmbreite von 640 Pixeln ausgeführt. Bezogen auf die Empfehlung, für solche Layoutumbrüche die Einheit `em` zu verwenden, müssen Sie die Bildschirmbreite nur durch 16 teilen. 16 (Pixel) ist gewöhnlich die Standardbrowser-Basisschriftgröße und wird somit als Bezugsgröße verwendet. 640 Pixel geteilt durch 16 Pixel/em sind somit 40 `em`. Somit können Sie für den Layoutumbruch von 640 Pixeln folgende `em`-Angabe verwenden:

```
/* 640px / 16px/em = 40em */
@media screen and (min-width: 40em) {
   /* CSS-Anweisungen für Bildschirme ab 640 Pixel Breite */
}
```

13.1.8 Layoutumbrüche (Breakpoints)

Im Abschnitt zuvor bin ich kurz auf einen Layoutumbruch (Breakpoint) eingegangen. Solche Layoutumbrüche sind essenziell für die Flexibilität einer Website. Bei diesen Umbrüchen wird das Layout verändert. In der Praxis stellen Sie hierbei unterschiedliche Layouts für verschiedene Auflösungen zur Verfügung, die Sie mit Media Queries steuern können. Mehrere solche Layoutumbrüche mithilfe von Media Queries lassen sich beispielsweise folgendermaßen definieren:

```
/* CSS-Anweisungen für Bildschirme bis 640 Pixel Breite */

/* 640px / 16px/em = 40em */
@media screen and (min-width: 40em) {
   /* CSS-Anweisungen für Bildschirme ab 640 Pixel Breite */
}
/* 1024px / 16px/em = 64em */
@media screen and (min-width: 64em) {
   /* CSS-Anweisungen für Bildschirme ab 1024 Pixel Breite */
}
/* 1280px / 16px/em = 80em */
@media screen and (min-width: 80em) {
   /* CSS-Anweisungen für Bildschirme ab 1280 Pixel Breite */
}
```

In diesem Beispiel habe ich drei gängige Layoutumbrüche mit Media Queries definiert. Die Anweisungen vor dem ersten Layoutumbruch werden auf jeden Fall ausgeführt. Hierbei könnten Sie neben den grundlegenden CSS-Eigenschaften auch gleich das mobile Layout für die Smartphones definieren. Anschließend finden Sie angepasste Layouts für die Bildschirmbreiten 640 Pixel (Tablets), 1.024 Pixel (Desktop) und 1.280 Pixel (extra großer Desktop) wie empfohlen mit der Einheit `em`. Entsprechend der Fensterbreite werden die Anweisungen zwischen den geschweiften Klammern ausgeführt.

Das Beispiel soll allerdings nicht den Eindruck erwecken, dass Sie so viele Layoutumbrüche definieren müssen. So ist es durchaus häufig anzutreffen, dass nur ein Layout für eine mobile Version und ein weiteres Layout für alle anderen Bildschirme definiert werden.

13.1.9 Keine Rechenspiele mehr dank »box-sizing: border-box;«

Um beim Erstellen des Layouts später nicht in Verlegenheit zu kommen, nachrechnen zu müssen, sollten Sie gleich das neuere Box-Modell mit `border-box` verwenden. Damit verzichten Sie auf Rechnereien mit `width`, `padding` und `border` wie ich es in Abschnitt 11.1.6, »Gesamtbreite und Gesamthöhe einer Box ermitteln«, gezeigt habe, sondern schließen diese Angaben gleich mit ein. Gerade beim responsiven Webdesign ist dies eine erhebliche Erleichterung. Das neue Box-Modell habe ich bereits ausführlich in Abschnitt 11.2, »Das neue alternative Box-Modell von CSS3«, beschrieben.

Aus dem Grund ist es empfehlenswert, folgende CSS-Anweisungen gleich zu Beginn festzulegen:

```
html {
  box-sizing: border-box;
}
*, *::before, *::after {
  box-sizing: inherit;
}
```

13.1.10 Und was passiert mit Webbrowsern, die Media Queries nicht verstehen?

Media Queries werden mittlerweile von allen Mainstream-Webbrowsern verstanden. Wenn Sie trotzdem auf alte Webbrowser oder andere Clients stoßen, die nicht mit den Media Queries umgehen können, dann verwendet dieser Webbrowser die Basisversion Ihrer Website, die Sie vor dem ersten Layoutumbruch mit einer Media Query definiert haben. Wenn Sie beispielsweise als Basisversion eine mobile Version erstellt haben, dann bekommen die Webbrowser, die keine Media Queries können, diese mobile Version. Daher ist es immer empfehlenswert, vor dem ersten Layoutumbruch mit einer Media Query eine Basisversion zu erstellen.

13.2 Wir erstellen ein einfaches responsives Layout

Nachdem Sie die nötigen Grundlagen zu den Medienabfragen (Media Queries) kennen und wissen, wie Sie damit die Medieneigenschaften abfragen, wird es Zeit, ein kleines responsives Layout zu erstellen. Ich werde hierbei nicht auf alle Details eingehen, die im responsiven Webdesign ebenfalls von Bedeutung sind. Hier geht es rein um die Anordnung der Inhalte einer Website für bestimmte Bildschirmgrößen.

13.2.1 Wir erstellen das Grundgerüst mit HTML

Bevor Sie überhaupt anfangen können, das responsive Layout mit CSS zu erstellen, müssen Sie zunächst das Grundgerüst mit HTML schreiben. Hierzu finden Sie mit */Beispiele/Kapitel013/13_2_1/index.html* ein einfaches Beispiel, das grundlegende HTML-Elemente wie `<header>`, `<nav>`, `<main>`, `<aside>` und `<footer>` enthält. Diese fünf Grundelemente werden anschließend im responsiven Layout entsprechend platziert. Zunächst das HTML-Grundgerüst:

```
…
   <body>
      <header> Responsive Webdesign - Logo </header>
```

```
<nav>
   <ul>
      <li><a href="#">Startseite</a></li>
      <li><a href="#">Portfolio</a></li>
      <li>Blog</li>
      <li><a href="#">Kontakt</a></li>
      <li><a href="#">Impressum</a></li>
   </ul>
</nav>
<main>
   <article>
      <h2>LR Classic und PS</h2>
      <p>Holen Sie das Optimum aus … </p>
   </article>
   <article>
      <h2>Capture One 11</h2>
      <p>Sie wollen Ihren Bildbestand … </p>
   </article>
   <article>
      <h2>macOS High Sierra</h2>
      <p>In diesem umfassenden Ratgeber … </p>
   </article>
   <article>
      <h2>PSE 2018</h2>
      <p>Ob Foto-Optimierung, Bildretusche … </p>
   </article>
</main>
<aside>
   <h3>Über den Autor</h3>
   <p>Jürgen Wolf ist … </p>
   <p> … </p>
</aside>
<footer> Fußzeile -  &copy; 2018 </footer>
</body>
…
```

Listing 13.2 /Beispiele/Kapitel013/13_2_1/index.html

Das nackte HTML-Grundgerüst werden wir nun in den folgenden Abschnitten mit einem responsiven Layout versehen. Das Grundgerüst ohne CSS sehen Sie in Abbildung 13.10 bei der Ausführung.

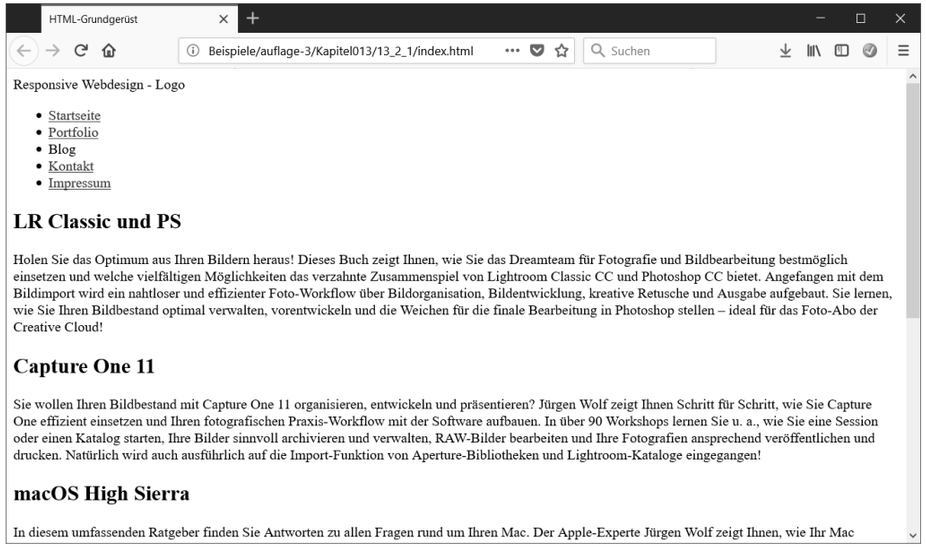

Abbildung 13.10 Das HTML-Grundgerüst für unser responsives Layout

13.2.2 Allgemeine CSS-Eigenschaften setzen

Bevor Sie anfangen, sich um die Layouts für bestimmte Bildschirmbreiten zu befassen, sollten Sie zunächst die allgemeinen CSS-Eigenschaften wie das Box-Modell, die Schriftart- und -größe, Farbe usw. schreiben. Hierbei sollte es sich um CSS-Eigenschaften handeln, die sich bei einem Layoutumbruch nicht ändern. Solche allgemeinen Eigenschaften können Sie entweder in einer Extra-CSS-Datei notieren und im Kopf des HTML-Dokumentes hinzufügen oder alles in eine einzige CSS-Datei schreiben, wie ich es in diesem Beispiel gemacht habe. Wie bereits erwähnt, werde ich dieses Beispiel sehr einfach halten. Hier habe ich zunächst lediglich das neue Box-Modell via `box-sizing` aktiviert und die Schriftart in `body` gesetzt.

CSS zurücksetzen/normalisieren

Ebenfalls in der Praxis wird am Anfang der Layouterstellung häufig das CSS zurückgesetzt/normalisiert, um die unterschiedlichen Browser-Grundeinstellungen der verschiedenen Hersteller auf eine gemeinsame Basis zu bringen. Dies stellt sicher, dass keine Unterschiede zwischen den verschiedenen Browsern existieren. Hat man zu Beginn der Webentwicklung noch einen kompletten Reset der CSS-Eigenschaften durchgeführt, wird nun gewöhnlich die Datei *normalize.css* (von *https://necolas.github.io/normalize.css/*) verwendet, um eine sinnvolle Grundlage für die weitere Verwendung von CSS zu haben. Zwar habe ich hierauf in diesem Beispiel verzichtet, aber es sollte zumindest schon mal erwähnt werden.

13.2.3 Was nehme ich als Basisversion ohne Media Queries: Mobile First!?

Wenn Sie anfangen, das Layout zu erstellen, müssen Sie sich zunächst überlegen, womit Sie beginnen wollen. Wollen Sie mit der Desktopversion anfangen und sich dann über Layoutumbrüche via Media Queries auf die kleineren Versionen für Tablets und dann Smartphones herunterarbeiten? Oder wollen Sie mit der mobilen Version für Smartphones anfangen und sich dann hoch bis zur Desktopversion arbeiten? Das ist Ihre eigene Entscheidung und hängt wohl auch von der Art und dem Inhalt Ihres Projektes ab.

Persönlich bevorzuge ich es mittlerweile, die mobile Version zuerst zu erstellen, weil diese Geräte zum einen das am häufigsten verwendete Medium sind, mit die Nutzer im Internet unterwegs sind, zum anderen zwingt es den Entwickler, sich zunächst auf das Wesentliche zu beschränken. Und das Wesentliche im Web ist der Inhalt. Man könnte auch ganz frech sagen: Mobile First ist Content First. Auf der anderen Seite profitiert davon auch die Desktopversion, weil Sie auf diese Weise die unkomplizierte Darstellung auch auf diese Layoutbreite erweitern können und der Fokus weiterhin auf den Inhalt bleibt.

In Abbildung 13.11 finden Sie den ersten Entwurf der Planung für das mobile Design. Wenn Sie hierbei HTML-Grundgerüst mit */Beispiele/Kapitel013/13_2_1/index.html* und Abbildung 13.10 hernehmen, dann dürften Sie feststellen, dass Sie mit dem HTML-Grundgerüst praktisch schon das mobile Design haben. Es fehlt hier eigentlich nur noch das Styling mit CSS.

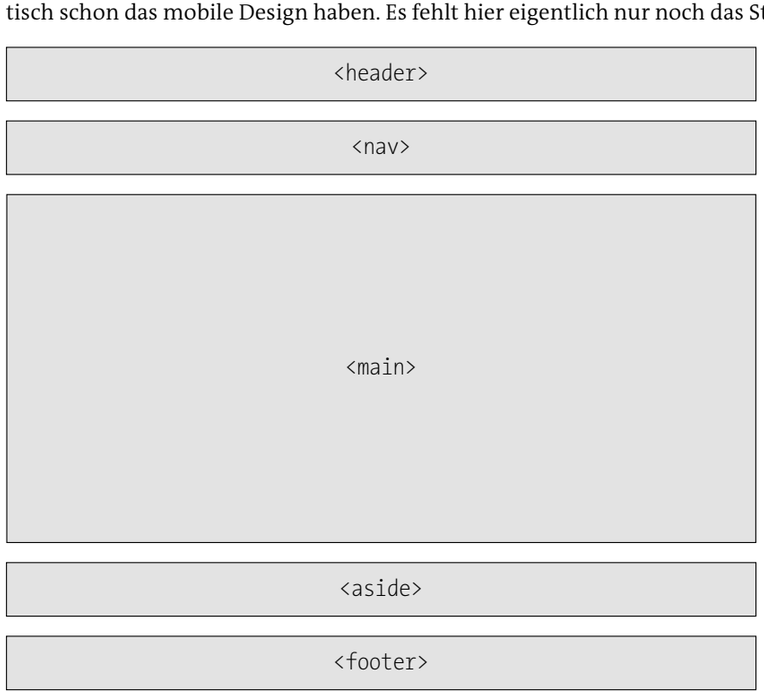

Abbildung 13.11 Das Design für die mobile Version

Die folgende CSS-Datei */Beispiele/Kapitel013/13_2_3/css/layout.css* dürfte daher keine besonderen Überraschungen mehr bereithalten. Nachdem Sie `box-sizing` mit `border-box` auf das neue Box-Modell gesetzt haben, finden Sie das Styling der einzelnen Bereiche für die mobile Version vor.

```
@charset "UTF-8";
/* Allgemeine Grundeinstellungen */
html {
  box-sizing: border-box;
}
*, *::before, *::after {
  box-sizing: inherit;
}
body {
   color: #1d2731;            /* Ivory Black*/
   background-color: #efefef; /* Neutral */
   font-family: Georgia;
}
ul{
  padding: 0;
}
.wrapper {
   background-color: #ff3b3f; /* Watermelon */
}
.header {
   text-align: center;
   padding: 1em;
   background-color: #07889b; /* Teal */
   color: #efefef;            /* Neutral */
   border-bottom: 1px solid #efefef;
}
.aside {
  border-top: 1px solid #a9a9a9;
  padding-top: 0.5em;
}
.footer {
   background-color: #a9a9a9; /* Carbon */
   color: #efefef;            /* Neutral */
   padding: 1em;
   text-align: center;
   border-top: 1px solid #efefef;
}
```

```
.nav-ul {
   background-color: #ff383f; /* Watermelon */
   margin:0;
}
.nav-li {
   list-style: none;
   margin-left: 0;
   border-bottom: 1px solid #efefef;
}
.nav-li-a {
    padding: 0.6em 2rem;
    display: block;
}
.nav-ul a:link {
    text-decoration: none;
}
.nav-ul a:link, .nav-ul a:visited {
   color: #fff; /* white*/
}
.nav-ul a:hover, .nav-ul a:focus, .nav-ul a:active {
   background-color: #000; /* Black */
   color: #efefef; /* Neutral */
}
.nav-active {
   color: #000; /* Black */
   background-color: #fff; /* White */
}
.container {
   background-color: #fff; /* weiss */
   padding: 2em 2rem;
}
```

Listing 13.3 /Beispiele/Kapitel013/13_2_3/css/layout.css

Als Nächstes können Sie im HTML-Grundgerüst die einzelnen Klassen eintragen. Zuvor sollten Sie allerdings noch das Viewport-Metatag setzen und natürlich die CSS-Datei für das Layout hinzulinken. Das HTML-Grundgerüst ist somit schon fertig und ändert sich nicht mehr auf den weiteren Seiten im Buch. Jetzt arbeiten wir nur noch mit der CSS-Datei am Layout. Hierzu das fertige HTML-Grundgerüst:


```
…
  <head>
    …
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="css/layout.css">
  </head>
  <body>
    <div class="wrapper">
      <header class="header">Responsive Webdesign - Logo</header>
      <nav>
        <ul class="nav-ul">
          <li class="nav-li"><a href="#" class="nav-li-a">Startseite</a></li>
          <li class="nav-li"><a href="#" class="nav-li-a">Portfolio</a></li>
          <li class="nav-li nav-active"><strong class="nav-li-a">Blog</strong></li>
          <li class="nav-li"><a href="#" class="nav-li-a">Kontakt</a></li>
          <li class="nav-li"><a href="#" class="nav-li-a">Impressum</a></li>
        </ul>
      </nav>
      <div class="container">
        <main class="content">
          <article class="article">
            <h2>LR Classic und PS</h2>
            <p>Holen Sie das Optimum … </p>
          </article>
          <article class="article">
            <h2>Capture One 11</h2>
            <p>Sie wollen Ihren …</p>
          </article>
          <article class="article clear">
            <h2>macOS High Sierra</h2>
            <p>In diesem umfassenden … </p>
          </article>
          <article class="article">
            <h2>PSE 2018</h2>
            <p>Ob Foto-Optimierung, Bildretusche … </p>
          </article>
        </main>
        <aside class="aside">
          <h3>Über den Autor</h3>
          <p>Jürgen Wolf ist … </p>
          <p> … </p>
        </aside>
```

```
      </div>
      <footer class="footer"> Fußzeile - &copy; 2018</footer>
    </div>
  </body>
</html>
```

Listing 13.4 /Beispiele/Kapitel013/13_2_3/index.html

Ohne viel Aufwand haben Sie bereits das Layout für die mobile Version erstellt. Das Layout erstreckt sich jetzt auf allen Geräten auf 100 % des Layout-Viewports. Dieses mobile Layout würde auch für (alte) Webbrowser verwendet, die nichts mit Layoutumbrüchen der Media Queries anfangen können. In Abbildung 13.13 sehen Sie das mobile Layout auf einem Smartphone. Dasselbe Layout bekommen Sie zum jetzigen Zeitpunkt auch auf dem Tablet oder Desktop zu sehen, so wie in Abbildung 13.12. Auf dem Desktop sieht dieses einspaltige Layout zwar nicht unbedingt spektakulär aus, aber es ist übersichtlich, der Inhalt wird ordentlich wiedergegeben, und die Website funktioniert.

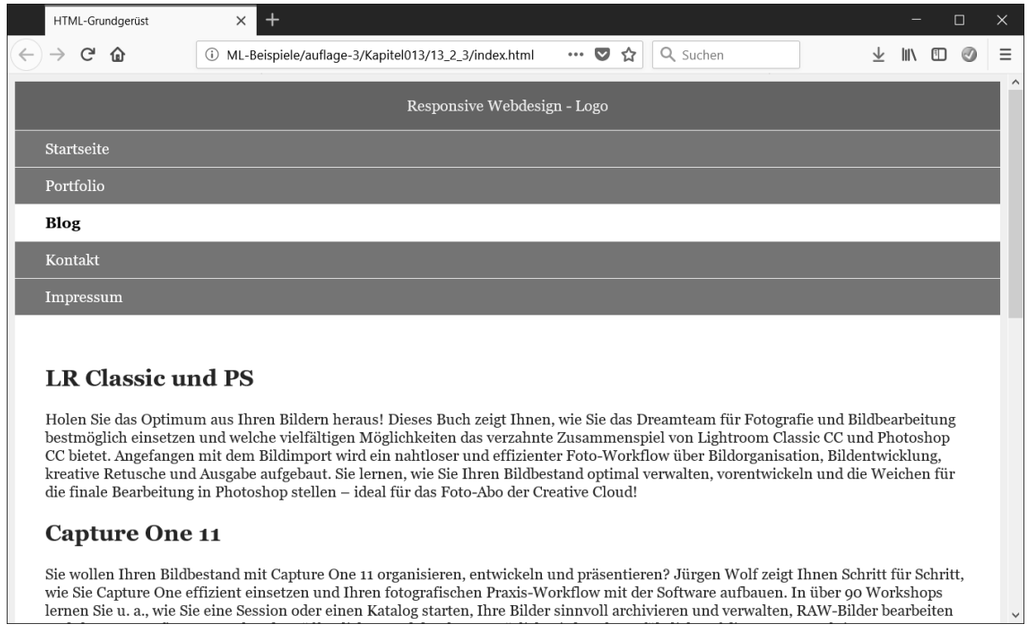

Abbildung 13.12 Die Basisversion ohne Media Queries auf einem Desktopbildschirm

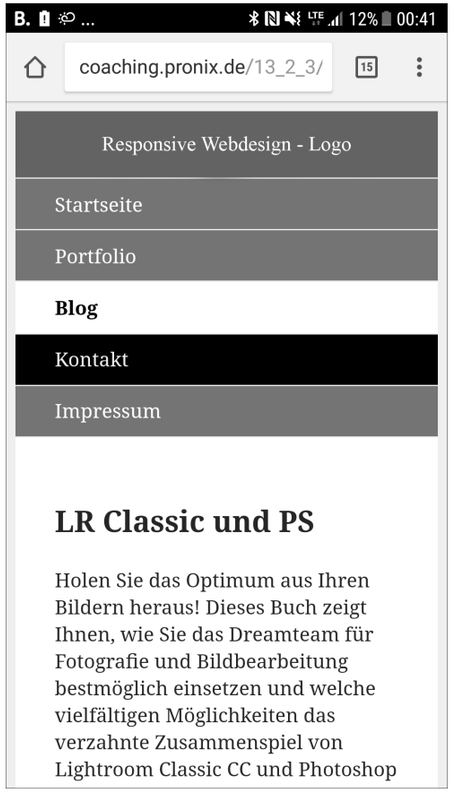

Abbildung 13.13 Die Basisversion auf einem Smartphone, wofür sie auch erstellt wurde

13.2.4 Den Layoutumbruch (Breakpoint) setzen

Auch wenn die Basisversion den Besucher mit allen Grundfunktionen einer Website versorgt und der Inhalt ordentlich angezeigt wird, so ist es doch eher ungewöhnlich, dieses einspaltige Layout auch für die Tablet- und Desktopversion zu verwenden. Daher soll jetzt eine weitere Ansicht des Layouts für Tablets angeboten werden. Bezogen auf unser Beispiel soll das Layout zweispaltig werden. Die Kopfzeile, Navigation und die Fußzeile dürfen bleiben, wo Sie sind. Den Hauptinhalt und die Seitenleiste hingegen stellen wir jetzt nebeneinander. Abbildung 13.14 zeigt das für ein Tablet angedachte Layout.

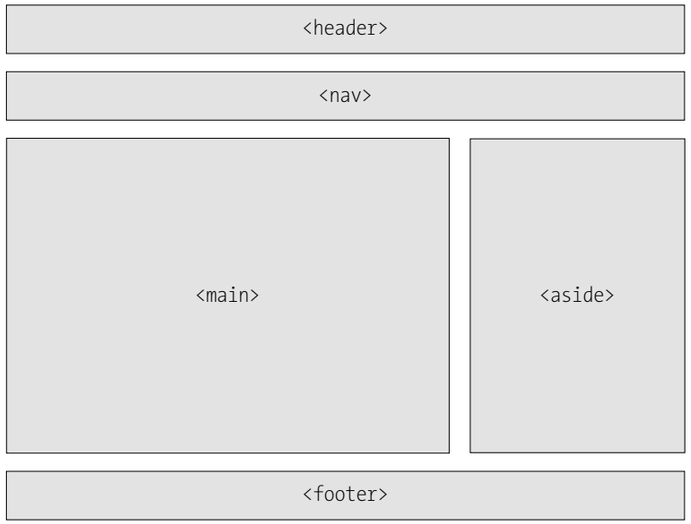

Abbildung 13.14 Dieses Layout ist für Tablets gedacht.

Zunächst müssen Sie sich entscheiden, ab welcher Bildschirmbreite Sie den ersten Layoutumbruch (Breakpoint) setzen. Im folgenden Beispiel wird der Umbruch bei 640 Pixeln (= 40 em) vollzogen:

```
…
/* Tabletversion ab 640 Pixel - 640px / 16px/em = 40em */
@media screen and (min-width: 40em) {
  .header {
    padding: 1.5em;
    text-align: left;
  }
  .container {
    padding: 3rem 0;
    display: block;
    overflow: auto;
  }
  .content {
    display: block;
    float: left;
    width: 65%;
    padding: 0 1rem 0 2rem;
  }
```



```
  .aside {
    display: block;
    margin: 0 0 0 65%;
    width: 35%;
    padding: 0 2rem 0 2rem;
    border-top: none;
  }
  .nav-ul {
    padding: 0 2rem;
    overflow: hidden;
  }
  .nav-li {
      float: left;
      display: inline-block;
      border: none;
      width: auto;
   }
   .nav-li-a {
     padding: 0.7em 1.2rem;
     display: inline-block;
    }
}
    }
}
```

Listing 13.5 /Beispiele/Kapitel013/13_2_4/css/layout.css

Um hier den Hauptinhalt und die Seitenleiste nebeneinander zu bekommen, geben wir dem Hauptinhalt `.content` 65 % Breite und lassen für die restlichen 35 % die Seitenleiste `.aside` dank `float:left;` in `.content` daneben fließen. Damit die anderen folgenden Elemente nicht »mitfließen«, umgeben wir diese beiden Elemente mit einem Container `.container` und lösen das Umfließen der Elemente außerhalb mit `overflow:auto;` wieder auf. Hierzu der entsprechende HTML-Teil:

```
…
<div class="container">
  <main class="content">
    <article class="article">
      …
    </article>
    …
  </main>
  <aside class="aside">
    …
```

```
  </aside>
</div>
...
```

Listing 13.6 /Beispiele/Kapitel013/13_2_4/index.html

Ab einem Viewport von 640 Pixeln wird dieser Layoutumbruch ausgeführt. In Abbildung 13.15 sehen Sie auf der linken Seite noch das Layout auf einem Viewport mit weniger als 640 Pixeln, und rechts wurde der Layoutumbruch ausführt, weil die Bildschirmbreite mehr als 640 Pixel betrug. Dieser Layoutumbruch wird auch bei Smartphones ausgeführt, wenn Sie hier in das Querformat wechseln und die Breite dabei mehr als 640 Pixel beträgt.

Abbildung 13.15 Jetzt wird die Basisversion ab einer Bildschirmbreite von 640 Pixeln in die nächste Layoutversion umgeschaltet.

13.2.5 Weitere Layoutumbrüche hinzufügen

In unserem Beispiel könnten wir jetzt schon recht gut mit diesen beiden Versionen leben – eine mobile Version für Bildschirme kleiner als 640 Pixel und eine zweite Version für den Rest ab 640 Pixeln. Dennoch sollen hier noch zwei weitere Umbrüche für einen Desktop und einen extra breiten Desktop hinzugefügt werden, weil bei breiteren Bildschirmen die Textzeilen des Hauptinhaltes sonst viel zu lang werden. Um den Platz für einen größeren Viewport zu füllen, soll jetzt noch die Navigation als dritte Spalte hinzugefügt werden, sodass Sie links vom Hauptinhalt die Navigation und rechts die Seitenleiste vorfinden. Den Kopf- und Fußbereich belassen wir wie gehabt. Den Entwurf für das Layout von Desktopbildschirmen finden Sie in Abbildung 13.16.

Abbildung 13.16 Dies soll für das Layout von Desktopbildschirmen verwenden werden.

Den nächsten Layoutumbruch definieren wir für Bildschirme mit einem breiteren Viewport als 1.024 Pixel (64 em). Hier der entsprechende CSS-Code:

```
...
/* Bildschirme ab 1024 Pixel - 1024px / 16px/em = 64em */
@media screen and (min-width: 64em) {
  .container {
    width: 85%;
    padding: 0;
    margin-left: 15%;
  }
  .content {
    width: 70%;
    padding: 1em 1.5em;
  }
  .aside {
    width: 30%;
    padding: 1em 1.5em;
    margin: 0 0 0 70%;
  }
  .nav-ul {
    width: 15%;
    float: left;
    margin: 1em 0;
    padding:0;
  }
```

```
  .nav-li {
    width: 100%;
    float: none;
    text-align: center;
  }
  .nav-li-a {
    padding: 0.5em 3rem;
    display: block;
  }
}
```

Listing 13.7 /Beispiele/Kapitel013/13_2_5/css/layout.css

Für den `.container` mit dem Hauptinhalt `.content` und der Seitenleiste `.aside` wurden jetzt 85 % der Fensterbreite reserviert. Von diesen 85 % innerhalb von `.container` teilen sich der Hauptinhalt `.content` mit 70 % und die Seitenleiste `.aside` mit 30 % diesen Platz. Die restlichen 15 % Platz bekommt die Navigation `.nav`. Hierfür wurde beim Container `.container` mit `margin-left` entsprechend Platz freigelassen. Dank `float:left;` bei `.nav` wird der Container `.container` mit seinem Hauptinhalt und der Seitenleiste rechts davon platziert. Auch hierzu ist in Listing 13.8 das HTML-Dokument mit entsprechenden Stellen abgedruckt, um das eben Beschriebene zu illustrieren:

```
…
<nav>
  <ul class="nav-ul">
    <li class="nav-li"><a href="#" class="nav-li-a">Startseite</a></li>
    …
  </ul>
</nav>
<div class="container">
  <main class="content">
    <article class="article">
      …
    </article>
    …
  </main>
  <aside class="aside">
    …
  </aside>
</div>
…
```

Listing 13.8 /Beispiele/Kapitel013/13_2_5/index.html

In Abbildung 13.17 sehen Sie das Layout für den Desktop ab einer Viewport-Breite von 1.024 Pixeln bei der Ausführung.

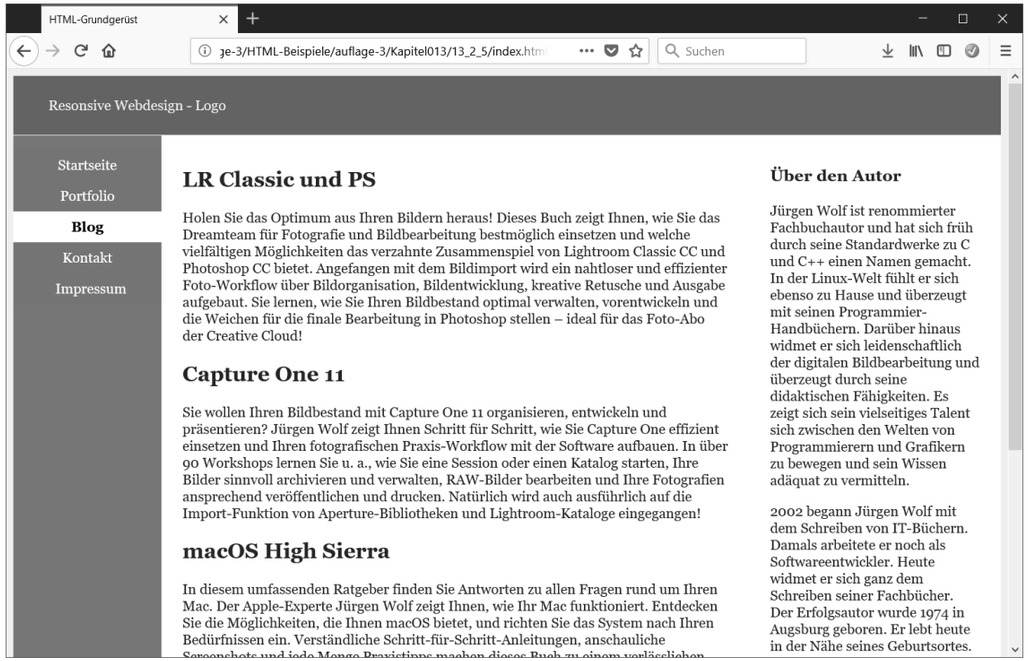

Abbildung 13.17 Das Layout für die Desktopversion ab einem Viewport von 1.024 Pixel Breite

Einen weiteren Breakpoint sollten Sie hier jetzt noch für extra große Bildschirme definieren, weil sonst dort die Zeilen wiederum zu lang werden. Hierbei können Sie sich entweder entscheiden, ob sich das komplette Layout ab einer bestimmten Bildschirmbreite nicht weiter ausdehnen darf, oder Sie teilen die Artikel des Hauptinhaltes auf. Im Beispiel habe ich einen letzten Breakpoint für einen Bildschirm ab 1.280 Pixeln (80 em) Breite erstellt, der beide eben erwähnten Lösungen anbietet. Es wird hierbei mit `.wrapper` ab einer Breite von 1.280 Pixeln die Eigenschaft `max-width` auf 1.280 Pixel gesetzt, und somit kann sich die Website nicht mehr weiter ausdehnen. Des Weiteren habe ich die Artikel `.article` auf 50 % reduziert und mit `float:left;` entsprechend nebeneinander angeordnet. Sie können bei diesem Beispiel allerdings auch die Klasse `.wrapper` oder `.article` allein verwenden. Hier der entsprechende CSS-Code zum letzten Breakpoint:

```
…
/* Große Bildschirme (>1280 Pixel) - 1280px / 16px/em = 80em */
@media screen and (min-width: 80em) {
```

```
  .wrapper {
    margin: 0 auto;
    max-width: 80em;
  }
  .article {
    display: block;
    width: 50%;
    float:left;
    padding: 0 1rem 0 1rem;
  }
  .clear { clear: both; }
}
```

Listing 13.9 /Beispiele/Kapitel013/13_2_5/css/layout.css

Das Ergebnis bei einem extra breiten Viewport ab 1.280 Pixeln sehen Sie in Abbildung 13.18.

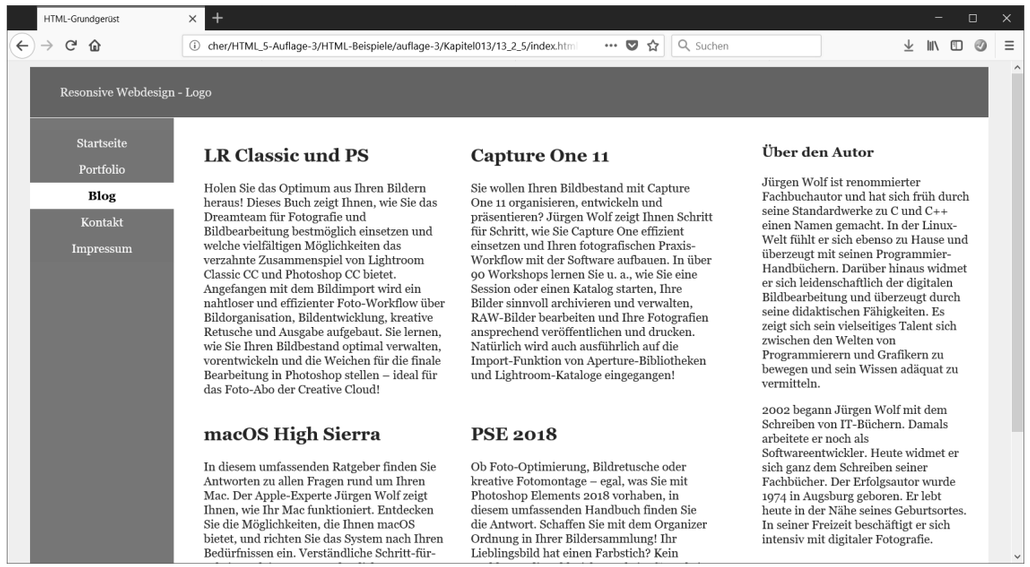

Abbildung 13.18 Dieses Layout ist für extra große Bildschirme ab 1.280 Pixel Breite.

Das so erstellte Beispiel passt sich jetzt dank den Layoutumbrüchen mit den Media Queries flexibel an das Gerät des Besuchers an. Hiermit haben Sie auf eine sehr einfache Weise eine simple Version für Smartphones, eine für Tablets und eine weitere Version für Desktops erstellt.

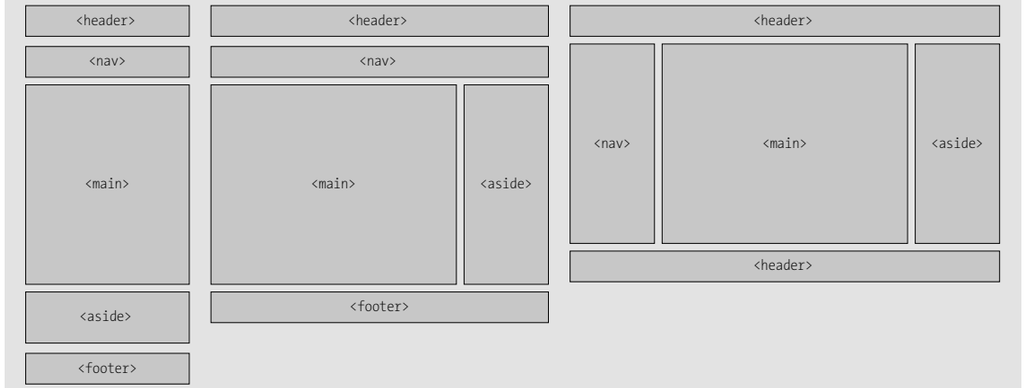

Abbildung 13.19 Mehrere Layoutumbrüche für verschiedene Bildschirmauflösungen dank Media Queries

13.2.6 Den Hauptinhalt anpassen

Haben Sie das responsive Design erstellt, können Sie sich an die Arbeit machen und die Details bearbeiten. Im Beispiel soll noch ein Auge auf den Hauptinhalt mit den Artikeln bei extra großer Desktopbreite geworfen werden. Ist hier beispielsweise ein Artikel auf der linken Seite länger als üblich und Sie haben keine Vorkehrung getroffen, kann der übernächste Artikel nicht nach links rutschen und bleibt an dem überlangen Artikel hängen. Abbildung 13.20 zeigt einen solchen Fall.

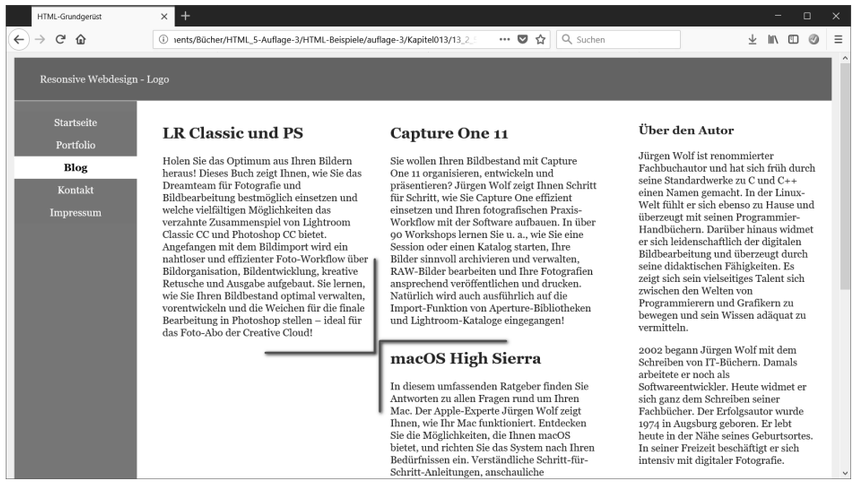

Abbildung 13.20 Zur Verdeutlichung habe ich die beiden kollidierenden Artikel an den entsprechenden Ecken hervorgehoben.

Im Beispiel */Beispiele/Kapitel013/13_2_5/css/layout.css* habe ich hierfür am Ende eine Klasse `.clear` mit `clear:both;` geschrieben. Diese Klasse müssen Sie nur noch beim entsprechenden Artikel wie folgt hinzufügen:

```
…
<article class="article clear">
   …
</article>
…
```

Listing 13.10 /Beispiele/Kapitel013/13_2_5/index.html

Dies löst zwar das Problem aus Abbildung 13.20, und der »hängende« Artikel kann wieder nach unten »wegrutschen«, aber dieses Vorgehen ist doch recht mühsam, wenn Sie die Website ändern und neue Artikel hinzufügen. Stattdessen bietet es sich an, eine Klasse für eine Zeile mit Artikeln zu schreiben, die Sie mit `<div class="row">` verwenden können und die automatisch ein `clear:both;` am Zeilenende enthält. In der Praxis könnten Sie dies wie folgt realisieren:

```
…
<main class="content">
  <div class="row">
    <article class="article">
       …
    </article>
    <article class="article">
       …
    </article>
  </div>
  <div class="row">
    <article class="article">
     …
    </article>
    <article class="article">
     …
    </article>
  </div>
</main>
…
```

Listing 13.11 /Beispiele/Kapitel013/13_2_6/index.html

Die Klasse `.row` dazu haben Sie praktisch schon beim Layoutumbruch von 1.280 Pixeln mit der Bezeichnung `.clear` definiert. Sinnesgemäß sollte sie hier jetzt `.row` heißen. Hier der CSS-Code dazu:

```
…
@media screen and (min-width: 80em) {
  .wrapper {
    margin: 0 auto;
    max-width: 80em;
  }
  .article {
    display: block;
    width: 50%;
    float:left;
    padding: 0 1rem 0 1rem;
  }
  .row { clear: both; }
}
```

Listing 13.12 /Beispiele/Kapitel013/13_2_6/css/layout.css

Dank dieser Zeile mit `.row` wird auch das Layout des Hauptinhaltes etwas flexibler. So kann durchaus nur ein Artikel in der Zeile stehen, wie es in Abbildung 13.21 zu sehen ist, ohne dass weitere vorhandene Artikel dafür nach oben rutschen.

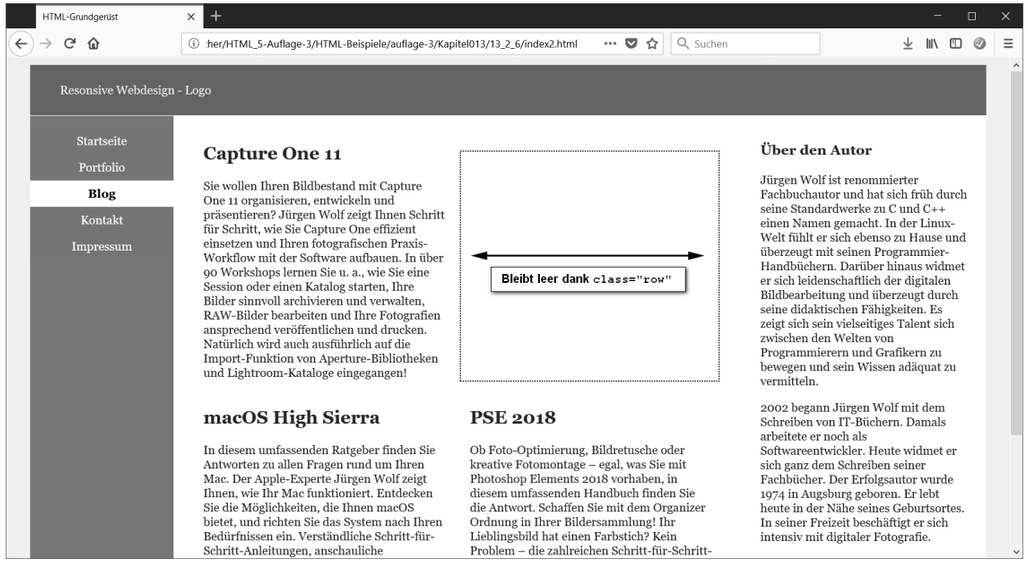

Abbildung 13.21 Dank der »geclearten« Zeile kann auch mal etwas leer bleiben.

Wollen Sie hingegen einen leeren Bereich auf der linken Seite mit den Artikeln im `main`-Bereich einfügen, so können Sie dies ganz einfach mit einem leeren `<article>` wie folgt machen:

```
…
<div class="row">
  <article class="article"> </article>
  <article class="article">
    <h2>Capture One 11</h2>
    <p>Sie wollen … </p>
  </article>
</div>
…
```

Listing 13.13 /Beispiele/Kapitel013/13_2_6/index2.html

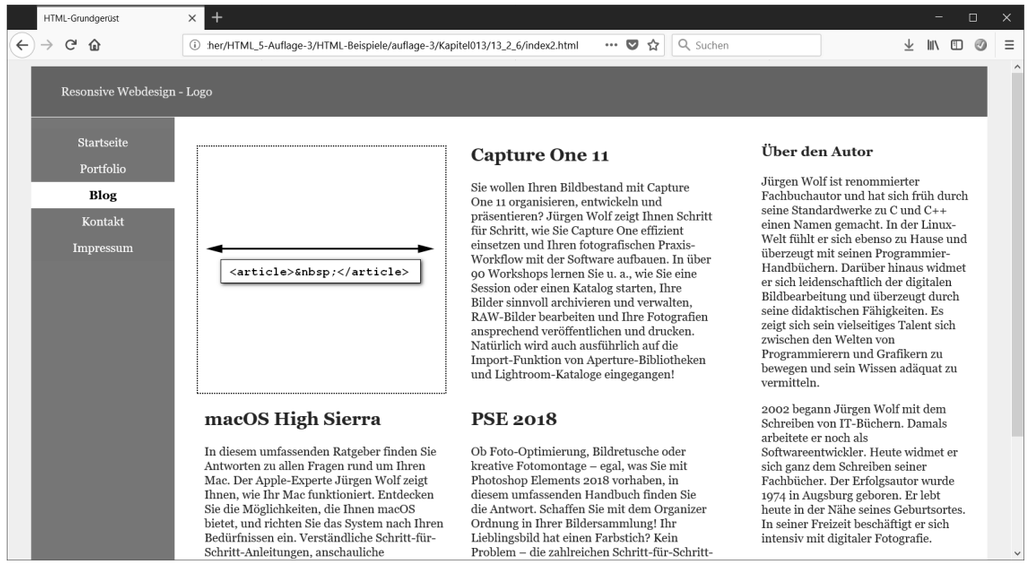

Abbildung 13.22 Ein leeres <article>-Element wurde eingefügt.

13.3 Noch mehr flexible Elemente

Bezogen auf das fluide responsive Layout, das Sie auf den Seiten zuvor erstellt haben, ist die Entwicklung einer responsiven Website noch lange nicht abgeschlossen. Wichtige Themen, die dabei noch nicht berücksichtig wurden, sind die Typografie, flexible Bildelemente oder

eine mobile Navigation. Darauf soll hier noch kurz eingegangen werden, damit Sie zumindest wissen, worauf es ankommt und worauf Sie noch alles achten sollten.

13.3.1 Verwenden Sie relative Schriftgrößen anstelle von Pixeln

Die Textgestaltung ist ein sehr wichtiges Thema im Webdesign überhaupt, und wird daher auch gesondert im Buch in Abschnitt 14.1, »Textgestaltung mit CSS«, behandelt. Trotzdem an dieser Stelle ein paar Anmerkungen dazu. Wie auch schon bei den Layoutumbrüchen der Media Queries sollten Sie bei den Schriftgrößen stets relative Angaben anstelle von Pixeln verwenden. Beim Zoomen einer ganzen Seite (Seitenzoom) haben die meisten Webbrowser keine Probleme, wenn die Angaben in Pixel oder relativen Einheiten gemacht wurden. Wenn aber nur der Text gezoomt wird, dann gibt es beim älteren Internet Explorer bis zur Version 10 ein Problem, weil dieser bei Pixelangaben nicht skaliert und somit Nutzer mit Sehbehinderung, die auf das Zoomen angewiesen sind, ausschließt.

Auch bei Bildschirmen mit einer höheren Pixeldichte werden die Schriften, die mit Pixeln angegeben sind, relativ klein dargestellt. Zwar könnte der Anwender hier wiederum nachträglich zoomen, allerdings sollte eine Website gleich nach dem Laden gut leserlich wiedergegeben werden. Daher sollten Sie auf Schriftangaben in Pixel verzichten und relative Angaben wie `em`, `rem` oder Prozente verwenden.

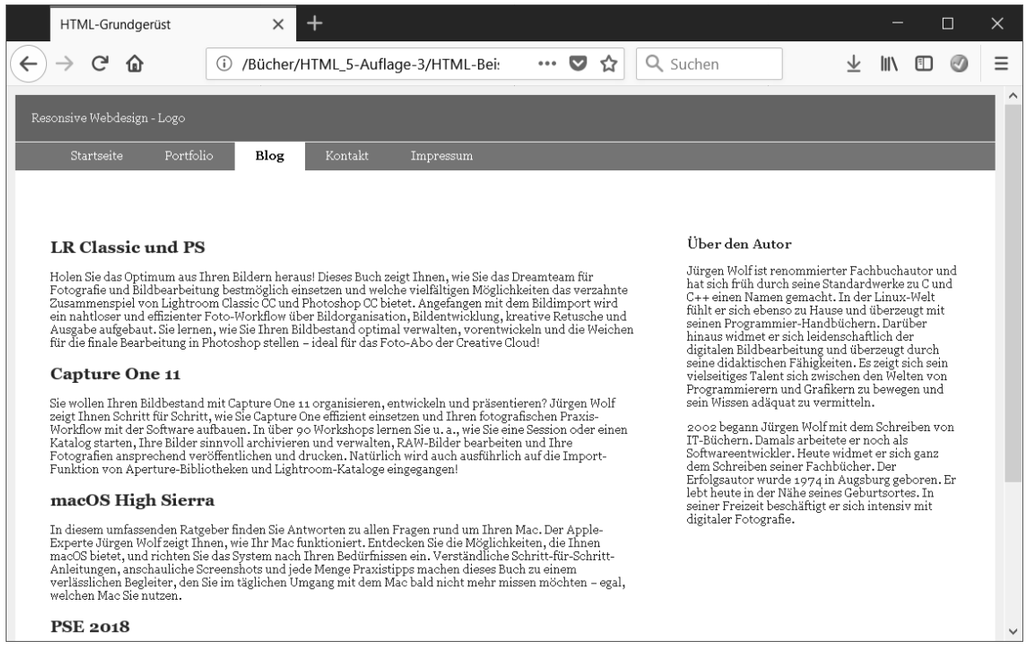

Abbildung 13.23 Bei falscher Schriftgröße bringt das beste responsive Layout nichts.

13.3.2 Bilder reaktionsfähig machen

Wenn Sie Bilder zum responsiven Layout hinzufügen wollen, dann werden Sie sie auch reaktionsfähig machen und nicht starr in der Breite lassen wollen. In Abbildung 13.24 sehen Sie, was bei starren Bildern passiert, wenn der Bildschirm schmaler wird. Hier bleiben möglicherweise ein paar Wörter neben dem Bild hängen oder rutschen nach unten weg – ein normales Verhalten von `float`, aber nicht sehr sehenswert.

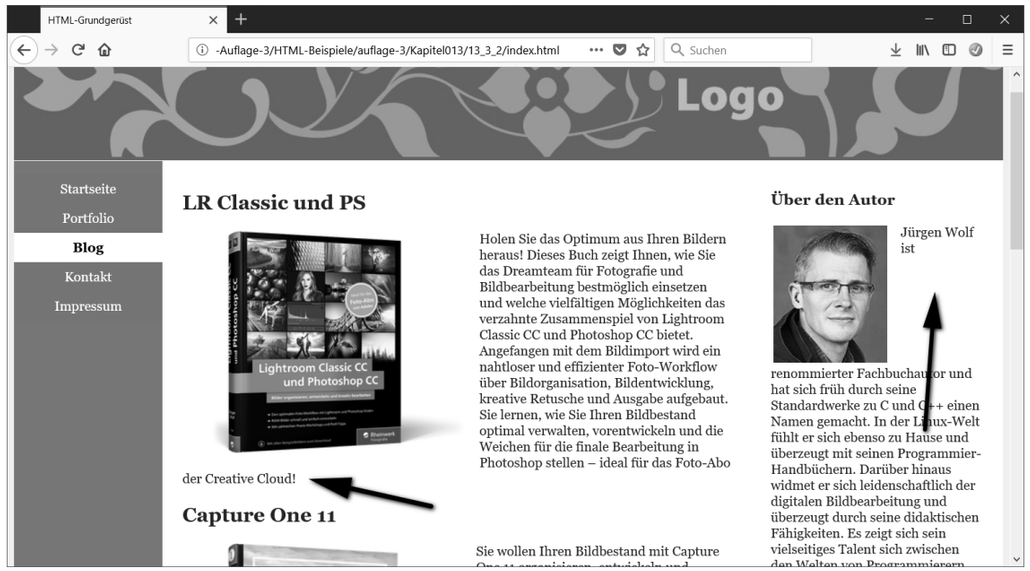

Abbildung 13.24 Durch das Umfließen des Textes mit starrer Bildgröße kann der Text nach unten wegrutschen und/oder es können einzelne Wörter oben stehen bleiben, wenn der Platz nicht mehr ausreicht.

Diesen Textumbruch können Sie vermeiden, indem Sie die maximale Breite des Bildes mit der CSS-Eigenschaft `max-width` entsprechend prozentual setzen. Mit `max-width` bestimmen Sie die maximale Breite eines Elements. Im Beispiel soll das Bild 40 % im `article`-Element beanspruchen dürfen. Dasselbe soll für das Autorenbild im `aside`-Element gelten. Dafür habe ich die Basisversion um die Klassenselektoren `.img-art` und `.img-side` erweitert:

```
…
.img-art {
  float: left;
  margin: 0 1em 0.2em 0;
  max-width: 40%;
  height: auto;
}
```



```
.img-side {
  float: left;
  margin: 0.1em 1em 0.2em 0.2em;
  max-width: 40%;
  height: auto;
}
…
```

Listing 13.14 /Beispiele/Kapitel013/13_3_2/css/layout.css

In Abbildung 13.25 passen sich die Bilder im Artikel und in der Seitenleiste jetzt um 40 % zur Artikelbreite bzw. Seitenleiste an. Da diese Selektoren in der Basisversion geschrieben wurden, gelten diese Eigenschaften für alle Layoutbreiten. In Abbildung 13.26 sehen Sie das Layout für extra breite Desktops und in Abbildung 13.27 die Smartphone-Version. Es spricht natürlich nicht dagegen, für jeden Layoutumbruch eine eigene Breite mit `max-width` für die Bilder festzulegen.

Neben der Breite mit `max-width` können Sie die Höhe mit `height` angeben. Hier habe ich `height` auf `auto` gesetzt, damit der Webbrowser automatisch die Höhe proportional zur Breite anpasst.

Die Klassen können Sie gewohnt einfach zu den `<img>`-Tags im HTML-Dokument dort hinzufügen, wo Sie wollen, dass diese Bilder reaktionsfähig sind:

```
…
<div class="container">
  <main class="content">
    <article class="article">
      <h2>LR Classic und PS</h2>
      <p><img src="grafik/cover-3.png" alt="Cover" class="img-art">… </p>
    </article>
      …
  </main>
  <aside class="aside">
    <h3>Über den Autor</h3>
    <p><img src="grafik/me.jpg" alt="Cover" class="img-side"> … </p>
  </aside>
</div>
…
```

Listing 13.15 /Beispiele/Kapitel013/13_3_2/index.html

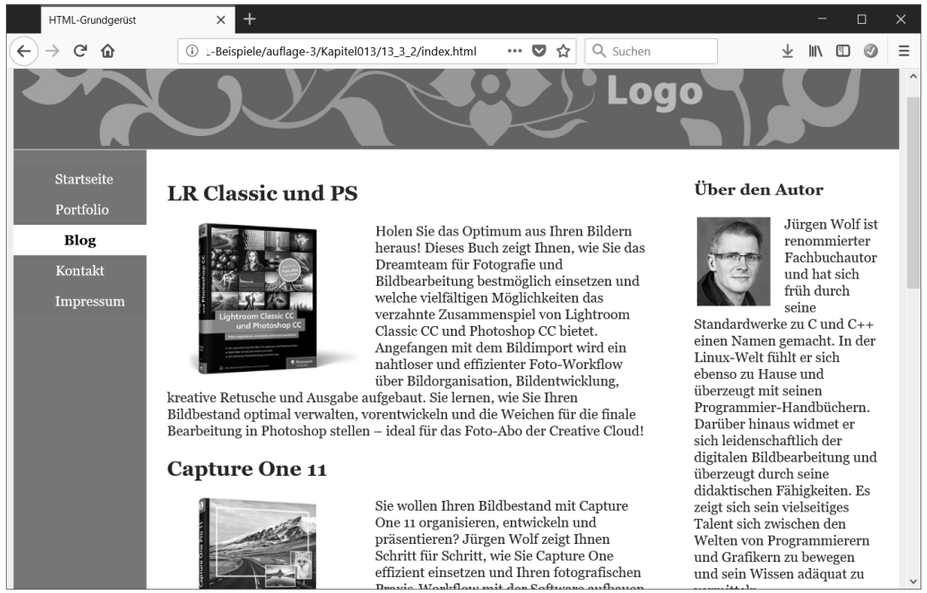

Abbildung 13.25 Die Bildgröße passt sich jetzt auch der Bildschirmbreite an und wird relativ zum <article>- bzw. <aside>-Element in der entsprechenden Größe (hier: 40 %) angezeigt.

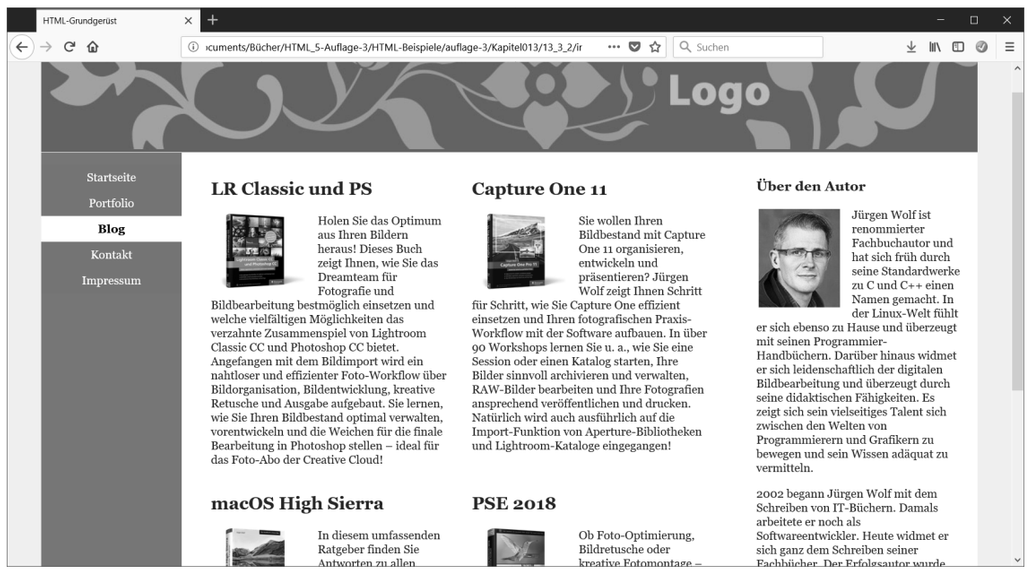

Abbildung 13.26 Die 40 % Bildbreite bei einem extra breiten Desktop

Abbildung 13.27 Auch auf einem Smartphone machen sich reaktionsfähige Bilder bezahlt.

13.3.3 Flexible Bilder in maximal möglicher Breite

Wollen Sie Bilder unabhängig vom Gerät immer über die volle Breite ausdehnen und sie trotzdem reaktionsfähig lassen, können Sie `max-width` auf 100 % setzen. Hierbei hängt es davon ab, wo Sie diese Bilder platzieren, um sie reaktionsfähig zu machen. Wenn Sie `max-width` auf 100 % setzen, dann bedeutet dies nämlich auch, dass ein Bild erst dann reagiert, wenn die Spalte, wo es definiert ist, kleiner ist als das Bild. Das heißt, dass ein Bild mit einer Breite von 300 Pixeln erst reaktionsfähig wird, wenn die Breite, wo es verwendet wird, weniger als 300 Pixel beträgt. Ein Bild mit `max-width` auf 100 % zu setzen, hängt also vom Kontext ab, in dem das Bild verwendet wird. Hierzu ein Beispiel, in dem Sie eine Grafik mit einer Größe von 1.280 × 150 Pixeln mit `<img>` in das `header`-Element des HTML-Dokumentes eingefügt haben:

```
...
<header class="header">
  <img src="grafik/logo.jpg" alt="Logo">
</header>
...
```

Ohne weitere Vorkehrungen würde das Bild in voller Größe angezeigt, und bei kleineren Bildschirmen als 1.280 Pixel wäre das Logo auf der rechten Seite abgeschnitten, und im Browser würde ein horizontaler Scrollbalken angezeigt, wie es in Abbildung 13.28 der Fall ist.

Abbildung 13.28 Bei zu kleiner Browserbreite wird das Bild abgeschnitten, und es tritt ein horizontaler Scrollbalken auf.

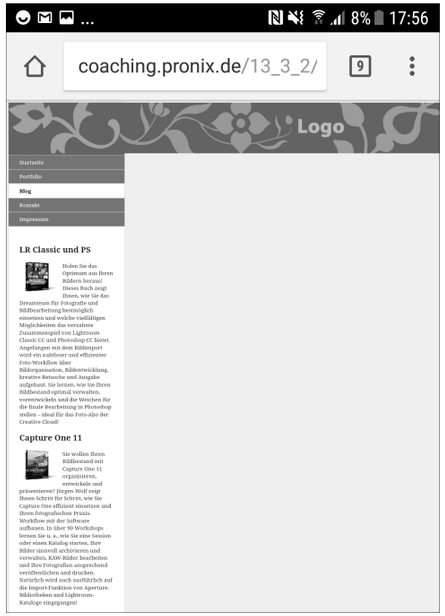

Abbildung 13.29 In der mobilen Version sieht es nicht viel besser aus.

In diesem Beispiel können Sie mit CSS und `max-width: 100%` wie folgt reagieren:

```
…
.img-logo {
  max-width: 100%;
  height: auto;
}
…
```

Listing 13.16 /Beispiele/Kapitel013/13_3_3/css/layout.css

Die Klasse müssen Sie nur noch zum `img`-Element hinzufügen:

```
<header class="header">
  <img src="grafik/logo.jpg" alt="Logo" class="img-logo">
</header>
```

Listing 13.17 /Beispiele/Kapitel013/13_3_3/index.html

In Abbildung 13.30 und Abbildung 13.31 sehen Sie, wie das Bild im `<header>` auf die entsprechende Bildschirmbreite reagiert und flexibel gemacht wird.

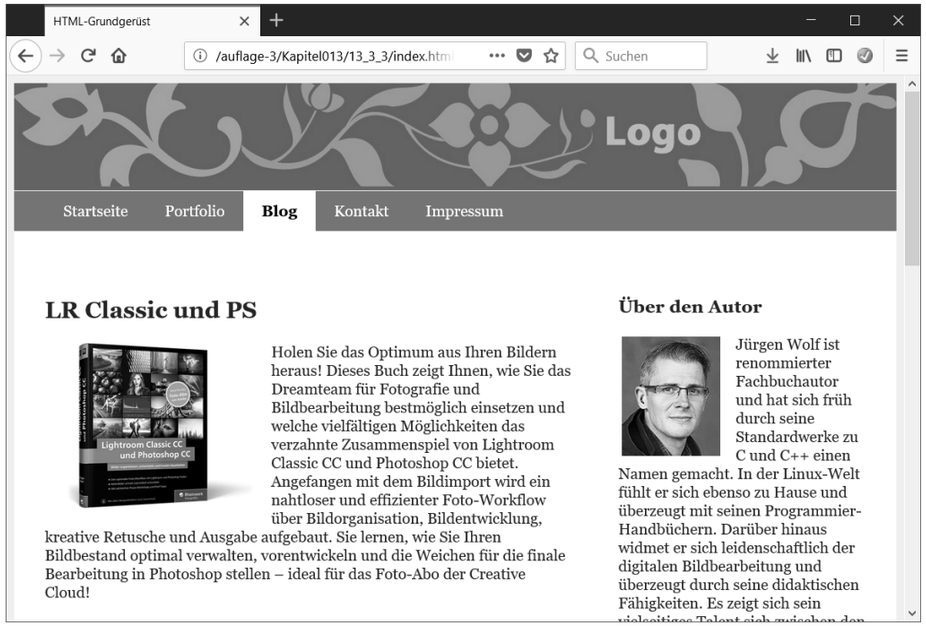

Abbildung 13.30 Jetzt passt sich die Breite für das Bild im `<header>` sowohl für Tablets …

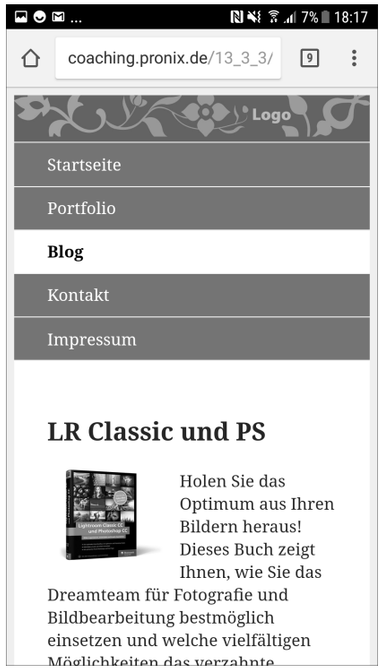

Abbildung 13.31 ... als auch für Smartphones an.

Reaktionsfähige Videos – <video>

Alles, was Sie hier mit Bildern machen können, können Sie auch mit denselben CSS-Anweisungen mit über `<video>` eingebundenen Videos machen.

13.3.4 Bilder ganz ausblenden

Wenn Sie Abbildung 13.31 mit dem Logo im Header betrachten, dann wirkt es schon relativ klein und verloren darin. Ich würde in diesem Beispiel bei der Smartphone-Version komplett auf das Logo im Header verzichten. Um es aus der Smartphone-Version, also Basisversion, zu entfernen, setzen Sie `display` auf `none`:

```
…
.img-logo {
  display: none;
}
…
```

Listing 13.18 /Beispiele/Kapitel013/13_3_4/css/layout.css

Entsprechend müssen Sie bei den anderen Layoutumbrüchen den Header mit `display:block` wieder sichtbar machen und natürlich auch `max-width` wieder auf 100 % setzen:

```
…
.img-logo {
  display: block;
  max-width: 100%;
  height: auto;
}
…
```

Listing 13.19 /Beispiele/Kapitel013/13_3_4/css/layout.css

13.3.5 Das passende Bild zur Bildschirmbreite laden – <picture>

Sie wissen nun, wie Sie Bilder mit `max-width` reaktionsfähig machen können. Der Nachteil an dieser Methode ist, dass bei kleinen Displays häufig zu große Dateien geladen werden, was unnötig die Performance bremst und den Umfang der Datenübertragung erhöht. Darüber hinaus müssen auf kleinen Displays die Bilder wieder herunterskaliert werden, was die Bildqualität negativ beeinflusst. Und hochauflösende Displays werden überhaupt nicht berücksichtigt. Wenn Sie das Buch von Anfang an gelesen haben, kennen Sie bereits eine alternative Lösung. In Abschnitt 6.3, »Das passende Bild mit <picture> laden«, haben Sie bereits erfahren, wie Sie mithilfe der HTML-Elemente `<picture>` und `<source>` und der Media Queries mit dem `media`-Attribut alternative Bildquellen für verschiedene Viewports angeben.

Mit dem `picture`-Element haben Sie ein HTML-Element, das als Container-Element mehreren Bildquellen dient. Die einzelnen Bildquellen geben Sie mit dem `source`-Element an. Im folgenden Beispiel soll beim ersten Artikel eine zum Display passende Bildquelle für das Buchcover geladen werden. Zur Unterscheidung, wann welches Bild geladen wird, habe ich die Grafiken mit einem entsprechenden Text versehen. Hierbei werden gleich hochauflösende Displays berücksichtigt.

```
…
<header class="header">
<picture class="img-logo">
   <source media="(min-width: 1023px)"
           srcset="grafik/logo-desktop.jpg 1x,grafik/logo-desktop-HD.jpg 2x">
   <source media="(min-width: 639px)"
           srcset="grafik/logo-tablet.jpg 1x,grafik/logo-tablet-HD.jpg 2x">
   <source srcset="grafik/logo-smartphone.jpg 1x,grafik/logo-smartphone-HD.jpg 2x">
   <!-- Fallback für Browser die kein <picture> können -->
   <img src="grafik/logo.jpg" alt="Logo" class="img-logo">
```

```
</picture>
</header>
...
```

Listing 13.20 /Beispiele/Kapitel013/13_3_5/index.html

Zwischen dem HTML-Container `<picture>` und `</picture>` geben Sie mit dem `source`-Element die einzelnen Bildquellen an. Abhängig vom Attribut `media` werden entsprechende Bilder geladen. Im Beispiel wird für Displays ab 1.023 Pixel Breite (`min-width: 1023px`) das Bild *logo-desktop.jpg* bei gewöhnlichen Displays mit einer einfacher Pixeldichte (`1x`) geladen. Bei hochauflösenden Displays mit doppelter Pixeldichte (`2x`) hingegen wird *logo-desktop-HD.jpg* geladen. (siehe Abbildung 13.32).

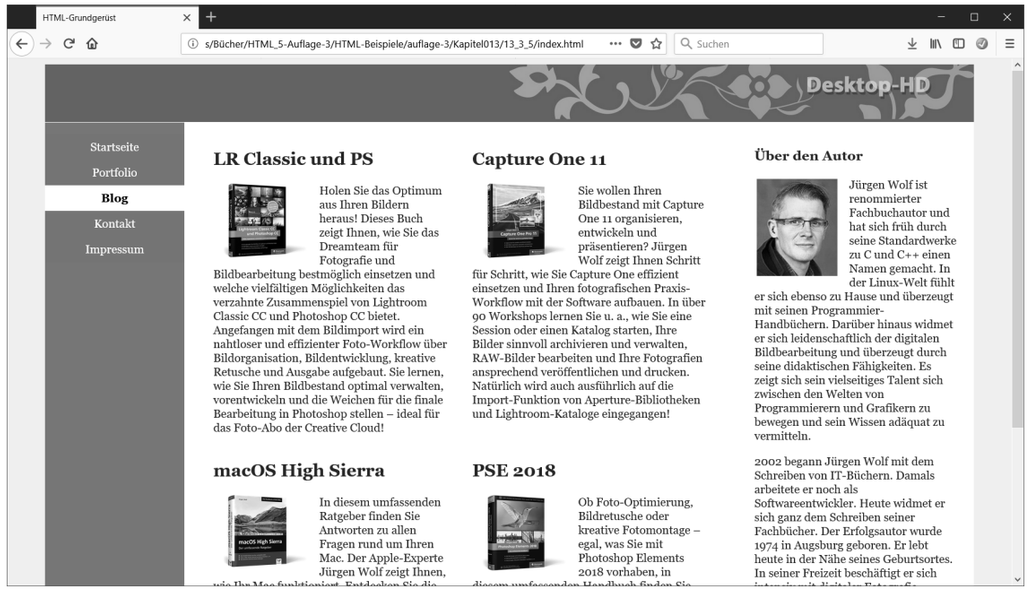

Abbildung 13.32 Hier wurde das Logo für die hochauflösende Desktopversion ab 1.023 Pixeln geladen.

Für eine Bildschirmbreite mit weniger als 1.023 Pixeln, also ab 1.022 Pixel bis 639 Pixel (`min-width:639px`), wird als Bildquelle entweder *logo-tablet.jpg* bei normaler oder *logo-tablet-HD.jpg* bei doppelter Pixeldichte geladen. Diese Bildquellen sind für Tablets oder kleinere Bildschirme gedacht. Bei einer kleineren Displaybreite ab 638 Pixeln wird für Smartphones die Bildquelle *logo-smartphone.jpg* bzw. *logo-smartphone-HD.jpg* (abhängig von der Pixeldichte) geladen. Für Webbrowser, die das `picture`-Element nicht verstehen, wird eine alternative Bildquelle im `img`-Element angegeben (hier: *logo.jpg*), das stattdessen verwendet wird.

Abbildung 13.33 Ab einer Displaybreite von 1.022 bis 639 Pixeln wird ein kleineres Bild (Tablet) für das Logo verwendet und unterhalb von 639 Pixeln die kleinste Version (Smartphone).

13.3.6 Flächendeckende Bilder verwenden

Wenn Sie Hintergrundbilder mit `background-image` einfügen, können Sie die Höhe und Breite mit der CSS3-Eigenschaft `background-size` anpassen. Verwenden Sie z. B.:

```
...
.img-background {
  background-size: 100% 100%;
  background-image: url("../grafik/background.jpg");
  background-repeat: no-repeat;
}
...
```

Damit füllt das Hintergrundbild immer das entsprechende HTML-Element aus, wo Sie die Klasse verwenden. Sie können so auch ein Bild als Hintergrundbild im `body`-Element für das gesamte HTML-Dokument definieren. Der erste Wert steht für die Breite und der zweite Wert für die Höhe. Abhängig vom HTML-Element, wo das Bild als Hintergrund verwendet wird, wird dieses häufig verzerrt, wie Sie im Vergleich zwischen der Desktopversion in Abbildung 13.34 und der Tablet-Version in Abbildung 13.35 mit einem quadratisches 500 × 500 Pixel großen Hintergrundbild in einem `article`-Element sehen.

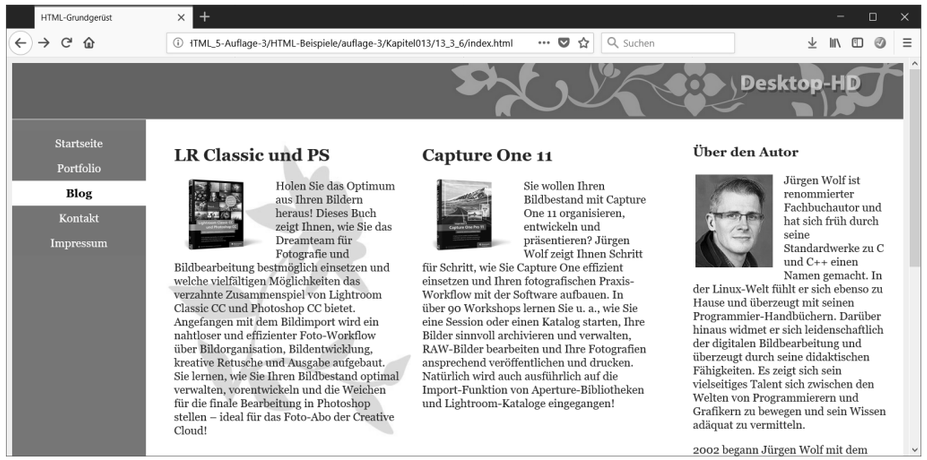

Abbildung 13.34 Zwar ist die Verzerrung hier auf einem Desktopbildschirm mit »background-size: 100% 100%;« noch erträglich, ...

Abbildung 13.35 ... aber bei einer kleineren Bildschirmbreite wird das Hintergrundbild des ersten Artikels schon recht stark verzerrt, und es sieht nicht mehr schön aus.

Wollen Sie, dass das Bild in der Breite immer um 100 % gestreckt und in der Höhe proportional dazu angepasst wird, können Sie den zweiten Wert von `background-size` auf `auto` setzen, damit die Höhe an das Seitenverhältnis angepasst und das Bild nicht mehr unproportioniert gestreckt wird:

```
...
background-size: 100% auto;
...
```

Jetzt wird das Bild nur noch in der Breite gestreckt und die Höhe daran automatisch proportional angepasst. Wenn das HTML-Element schmaler wird und der Wert der Höhe größer ist als die Breite und Sie `background-repeat` auf `no-repeat` gesetzt haben, wird ein übergangsloser Rand unten sichtbar.

In Abbildung 13.36 beim ersten Artikel sehen Sie diesen unschönen Effekt: Das Hintergrundbild wird zwar proportional zur Breite angepasst, aber im Hochformat wird entweder das Bild wiederholt oder, wenn wie im Beispiel die Wiederholung deaktiviert wurde, die Farbe des Hintergrundes (was hier ein weißer Rand ist) kommt zum Vorschein. Zwar ist es in diesem Beispiel nicht so dramatisch, weil ein Teil des Hintergrundbildes ohnehin weiß war, aber trotzdem wirkt es nicht so richtig passend. Zwar sieht hiermit die Tablet-Version wieder besser aus, aber spätestens bei kleineren Smartphones erhalten Sie ein ähnliches Ergebnis wie in Abbildung 13.36.

Abbildung 13.36 Hier bleibt ein weißer Rand um unteren Rand des ersten <article>-Elements stehen.

Die Rechnerei und das Anpassen mit relativen und absoluten Werten mit `background-size` führen selten zu einer optimalen reaktionsfähigen Lösung. Abhilfe schaffen hier die zwei Schlüsselwörter `contain` und `cover`, die Sie der CSS-Eigenschaft `background-size` zuweisen können.

Mit `background-size: contain;` wird das Hintergrundbild immer komplett angezeigt. Dieses »Mitwachsen« und »Schrumpfen« geschieht dabei proportional. Da das Bild immer zu sehen ist, wird je nach Verhältnis von der Breite und Höhe das Hintergrundbild niemals die gesamte Fläche des Bildschirms ausfüllen.

Anders hingegen ist der Wert `cover`. Damit wird das Bild immer auf dem kompletten Bildschirm so weit wie möglich angezeigt. Wenn das Seitenverhältnis unterschiedlich ist, wird das Bild »abgeschnitten« und nicht komplett angezeigt. In unserem Beispiel scheint somit die Lösung mit

```
…
background-size: cover;
…
```

zum besten Ergebnis zu führen, wie Sie in Abbildung 13.37 sehen, wo mit dem Wert `cover` versucht wird, möglichst das komplette Hintergrundbild anzuzeigen. Wenn es nicht hineinpasst, wird es nicht verzerrt oder unschön wiederholt oder abgebrochen, sondern »beschnitten«.

Abbildung 13.37 Mit »background-size: cover« wird immer versucht, möglichst das komplette Hintergrundbild anzuzeigen.

13.4 Das neue Grid-Layout von CSS3

Beim reaktionsfähigen Beispiel, wie Sie es in den Abschnitten vorher erstellt haben, ist es recht aufwendig, ein komplexeres Layout zu erstellen oder es umzugestalten. Natürlich ist es auch damit möglich, aber Sie können häufig nicht mal eben ohne größeren Aufwand schnell ein Element nach oben, unten, rechts oder links verschieben.

Für solche Zwecke bieten sich *Rasterlayouts* an. Wurde hier in der Vorgängerauflage des Buches noch ein eigenes Rasterlayout erstellt, bietet CSS mit CSS-Grid-Layouts nun auch echte Gestaltungsraster an, die Sie auf den folgenden Seiten etwas näher kennen lernen werden. Im Gegensatz zum ebenfalls beliebten Flexbox-Layoutmodell, das sich eher für eine lineare Strukturierung anbietet, eignen sich die CSS-Grids für komplexere Layouts und dürften in Zukunft wohl eine Alternative schlechthin zur Layouterstellung werden. Um nicht ein komplett neues Projekt erstellen zu müssen, werden wir hier das bereits bekannte Beispiel aus den vorherigen Abschnitten für das Grid-Layout umschreiben. Das komplette Beispiel finden Sie mit */Beispiele/Kapitel013/13_4/css/layout.css* und das HTML-Dokument dazu mit */Beispiele/Kapitel013/13_4/index.html*.

13.4.1 Ein Grid für den Inhalt erstellen

Das Prinzip eines CSS-Grids beruht darauf, dass Sie, wie Sie es schon vom reaktionsfähigen Layout her kennen, ein Raster in einem Eltern-Element erstellen und darin die Kind-Elemente positionieren. Hierfür müssen Sie im Eltern-Element der CSS-Eigenschaft `display` den Wert `grid` zuweisen und dann mit den Eigenschaften `grid-template-columns` und `grid-template-rows` die einzelnen Rasterlinien definieren. Betrachten Sie hierzu folgendes einfache Beispiel:

```
.grid {
  display: grid;
  grid-template-rows: 150px auto auto 100px;
  grid-template-columns: 20% 20% 20% 20% 20%;
}
```

Damit würden Sie ein CSS-Grid mit vier Zeilen und fünf Spalten erzeugen. Die erste Zeile ist 150 Pixel und die letzte 100 Pixel hoch. Die mittleren zwei Zeilen werden mittels `auto` entsprechend dem Inhalt noch angepasst. Alle fünf Spalten sind zudem 20 % breit. Neben den Einheiten in Prozent oder Pixeln können Sie auch `em` oder `fr` (beispielsweise `1fr` oder `2fr`) verwenden. Die Einheit `fr` steht für ein flexibles Fragment (Bruchteil), was Sie sich wie die Prozentangabe für einen noch vorhandenen Platz vorstellen können. Abbildung 13.38 zeigt die Angabe des CSS-Grid-Layouts.

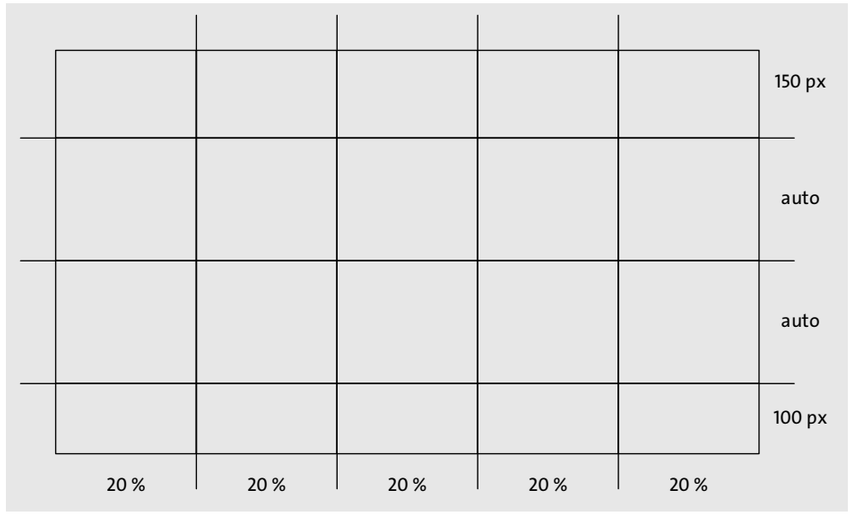

Abbildung 13.38 Ein Grid-Layout mit »display:grid;« ist schnell erstellt.

Da hier das reaktionsfähige Layout aus Abschnitt 13.2, »Wir erstellen ein einfaches responsives Layout«, überarbeitet wurde und für das Grid-Layout verwendet werden soll, ist das hier erstellte Raster in Abbildung 13.38 etwas eng bemessen. Da hier fünf Spalten mit 20 % definiert wurden, können Sie den Inhalt der HTML-Elemente somit in Bereichen von 20 %, 40 %, 60 %, 80 % und 100 % aufteilen. Für das Beispiel wollen wir den Raster noch etwas feiner in 10 %-Schritten aufteilen und verwenden daher folgendes Grid-Layout:

```
…
.grid {
  display: grid;
  grid-template-rows: 100px auto auto auto 100px;
  grid-template-columns: repeat(10, 10%);
}
…
```

Listing 13.21 /Beispiele/Kapitel013/13_4/css/layout.css

Das Grid wird hier noch vor den ersten Layoutumbrüchen in der Basisversion definiert. Hiermit definieren Sie ein Rasterlayout mit fünf Zeilen und zehn Spalten. Für die Kopf- und Fußzeile definieren wir eine Höhe von 100 Pixeln. Die drei Zeilen dazwischen passen sich mit `auto` dem Inhalt entsprechend an. Um hier nicht zehnmal `10%` schreiben zu müssen, wurde hier die `repeat()`-Funktion verwendet. Der erste Wert der Funktion steht für die Anzahl der

Wiederholungen des zweiten Wertes – hier also: zehnmal `10%`. Anstelle der `repeat()`-Funktion hätten Sie auch Folgendes notieren können:

```
…
.grid {
  display: grid;
  grid-template-rows: 100px auto auto auto 100px;
  grid-template-columns: 10% 10% 10% 10% 10% 10% 10% 10% 10% 10%;
}
…
```

Hiermit haben Sie ein Rasterlayout festgelegt, wie in Abbildung 13.39 dargestellt.

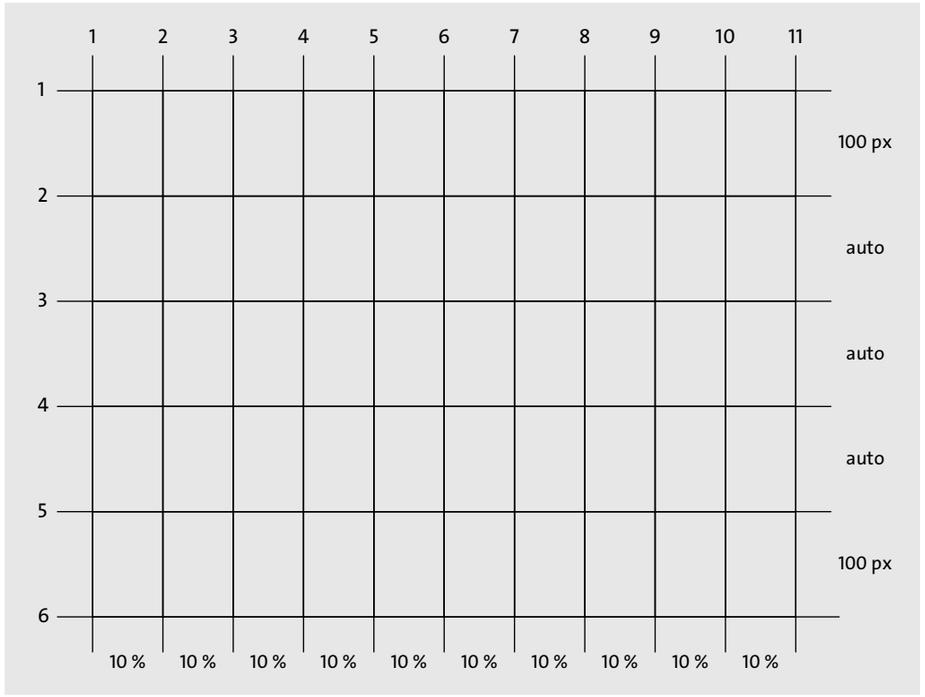

Abbildung 13.39 Das Grid-Layout für unser Beispiel

Um das Grid-Layout mit dem Klassenselektor `.grid` zu benutzen, müssen Sie es im HTML-Dokument im Eltern-Element verwenden, dessen Kind-Elemente in diesem Raster positioniert werden. Im Beispiel machen wir dies gleich nach dem `body`-Element und verwenden ein `div`-Element dafür:

```
…
  <body>
    <div class="grid">
      <header class="header">…</header>
      <nav class="nav">…</nav>
      <main class="content">…</main>
      <aside class="aside">…</aside>
      <footer class="footer">… </footer>
    </div>
  </body>
…
```

Listing 13.22 /Beispiele/Kapitel013/13_4/index.html

Innerhalb des `div`-Eltern-Elementes mit der Klasse `grid` können Sie nun die Kind-Elemente `<header>`, `<nav>`, `<main>`, `<aside>` und `<footer>` in den Rasterzellen Abbildung 13.39 positionieren.

13.4.2 Elemente im Raster platzieren

Wenn Sie das Grid-Layout festgelegt haben, können Sie ganz einfach mit den CSS-Eigenschaften `grid-row-start` und `grid-row-end` bzw. `grid-column-start` und `grid-column-end` angeben, wo Sie die HTML-Elemente im Raster platzieren wollen. Wenn Sie das Rasterlayout in Abbildung 13.39 betrachten, erstrecken Sie die Werte für `grid-row-column-start` und `grid-column-end` von 1 (0 %) bis 11 (100 %), und die Werte für `grid-row-start` und `grid-row-end` lassen sich mit 1 bis 6 definieren. Um also den Header in der ersten Zeile in voller Breite erstrecken zu lassen, müssen Sie Folgendes schreiben:

```
…
.header {
   grid-column-start:1;
   grid-column-end:11;
   grid-row-start:1;
   grid-row-end:2;
   text-align: right;
   background-color: #07889b; /* Teal */
   color: #efefef;            /* Neutral */
   border-bottom: 1px solid #efefef;
}
…
```

In Abbildung 13.40 sehen Sie das Ergebnis dieser Zeilen.

Abbildung 13.40 Das <header>-Element wurde im Rasterlayout hinzugefügt.

Auf diese Weise können Sie nun auch die Start- und Endpunkte der anderen HTML-Elemente im Grid-Layout für die Basisversion definieren:

```
…
.nav {
   grid-column-start:1;
   grid-column-end:11;
   grid-row-start:2;
   grid-row-end:3;
}
.content {
   grid-column-start:1;
   grid-column-end:11;
   grid-row-start:3;
   grid-row-end:4;
}
```

```
.aside {
   grid-column-start:1;
   grid-column-end:11;
   grid-row-start:4;
   grid-row-end:5;
}
.footer {
   grid-column-start:1;
   grid-column-end:11;
   grid-row-start:5;
   grid-row-end:6;
}
…
```

Das Ergebnis unserer Basisversion in Abbildung 13.41 entspricht nun exakt der mobilen Version, wie Sie diese bereits beim reaktionsfähigen Layout in Abschnitt 13.2.3, »Was nehme ich als Basisversion ohne Media Queries: Mobile First!?«, erstellt haben.

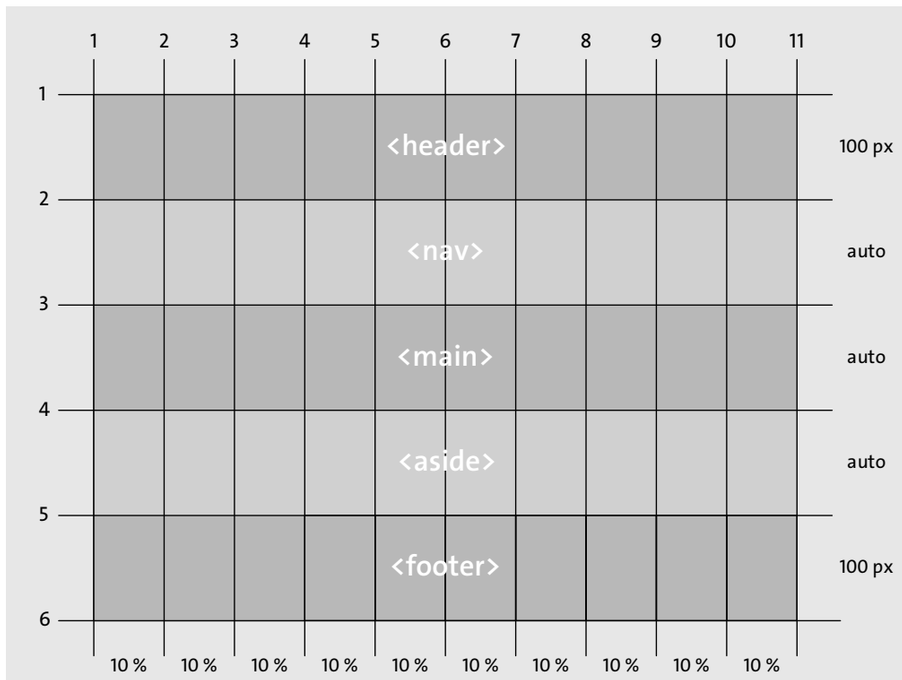

Abbildung 13.41 Die mobile Basisversion für unser Layout mit CSS-Grid

Kürzere Schreibweisen für das Platzieren von Elemente im Grid verwenden

Für die Funktionen `grid-column-start` und `grid-column-end` können Sie die Kurzschreibweise `grid-column` bzw. anstelle von `grid-row-start` und `grid-row-end` die Version `grid-row` verwenden. Bezogen auf das oben gezeigte Beispiel mit dem `nav`-Element könnten Sie daher auch Folgendes schreiben:

```
…
.nav {
   grid-column: 1 / 11;
   grid-row: 2 / 3;
}
…
```

Diese Version entspricht demselben wie:

```
…
.nav {
   grid-column-start:1;
   grid-column-end:11;
   grid-row-start:2;
   grid-row-end:3;
}
…
```

Es geht noch kürzer, wenn Sie mit der Eigenschaft `grid-area` arbeiten. Die Reihenfolge der Start- und Endpunkte sieht hiermit wie folgt aus:

```
grid-area: row-start / column-start / row-end / column-end;
```

Somit könnten Sie für das Platzieren des `nav`-Elements z. B. auch die folgende dritte Schreibweise verwenden:

```
…
.nav {
  grid-area: 2 / 1 / 3 / 11;
}
…
```

Elemente im nächsten Layoutumbruch platzieren

Ausgehend von der Basisversion für das mobile Layout ist nun wenig Arbeit nötig, um beim nächsten Layoutumbruch für die Tablet-Version mit entsprechenden Eigenschaften zu reagieren:

```
…
@media screen and (min-width: 40em) {
  .content {
    grid-column: 1 / 8;
    grid-row: 3 / 4;
  }
  .aside {
    grid-column: 8 / 11;
    grid-row: 3 / 4;
  }
…
}
…
```

Bezogen auf unser Raster haben Sie das in Abbildung 13.42 erstellte Layout für die Tablet-Version erstellt. Das Beispiel bei der Ausführung sehen Sie in Abbildung 13.43.

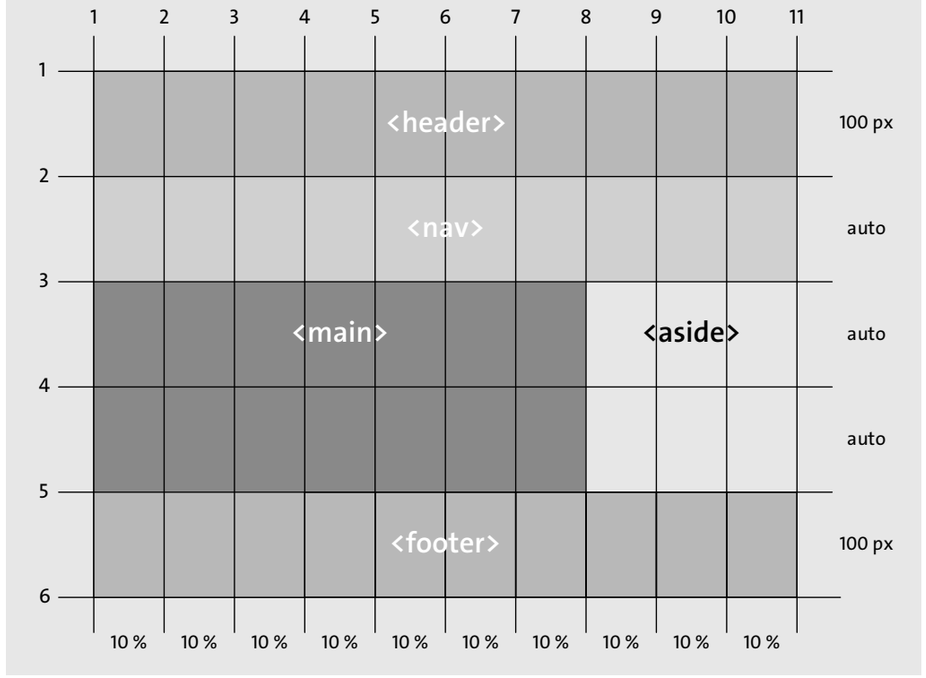

Abbildung 13.42 Das Layout für die Tablet-Version mit CSS-Grid

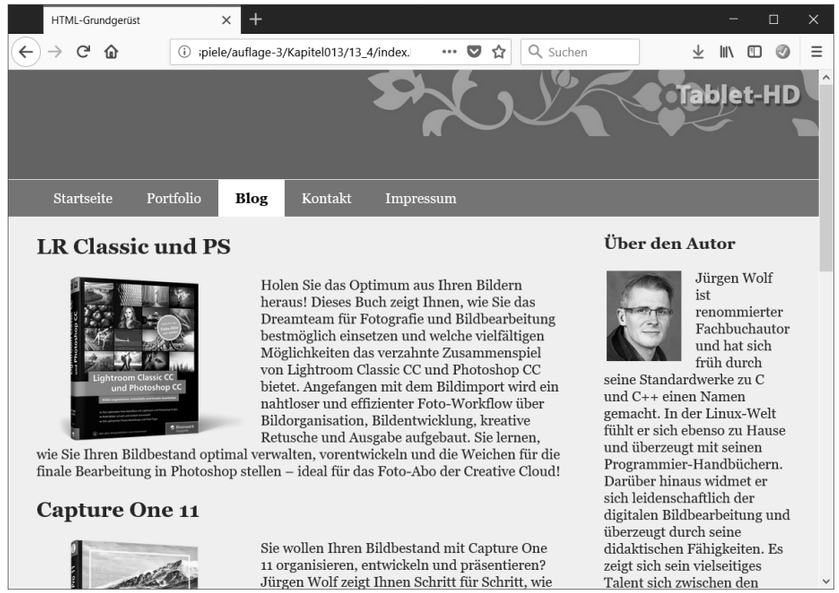

Abbildung 13.43 Die Tablet-Version wurde mithilfe eines CSS-Grids erstellt.

Zum Schluss soll noch eine weitere Version für die Desktopversion erstellt werden:

```
…
@media screen and (min-width: 64em) {
  .content {
    grid-column: 3 / 8;
    grid-row: 2 / 4;
  }
  .aside {
    grid-column: 8 / 11;
    grid-row: 2 / 4;
  }
  .nav {
    grid-column: 1 / 3;
    grid-row: 2 / 4;
  }
…
}
```

Hiermit haben Sie für die Navigation 20 %, den Hauptinhalt 50 % und die Seitenleiste 30 % Platz in der Breite vergeben. Daraus ergibt sich die in Abbildung 13.44 gezeigte Aufteilung der HTML-Elemente im Raster.

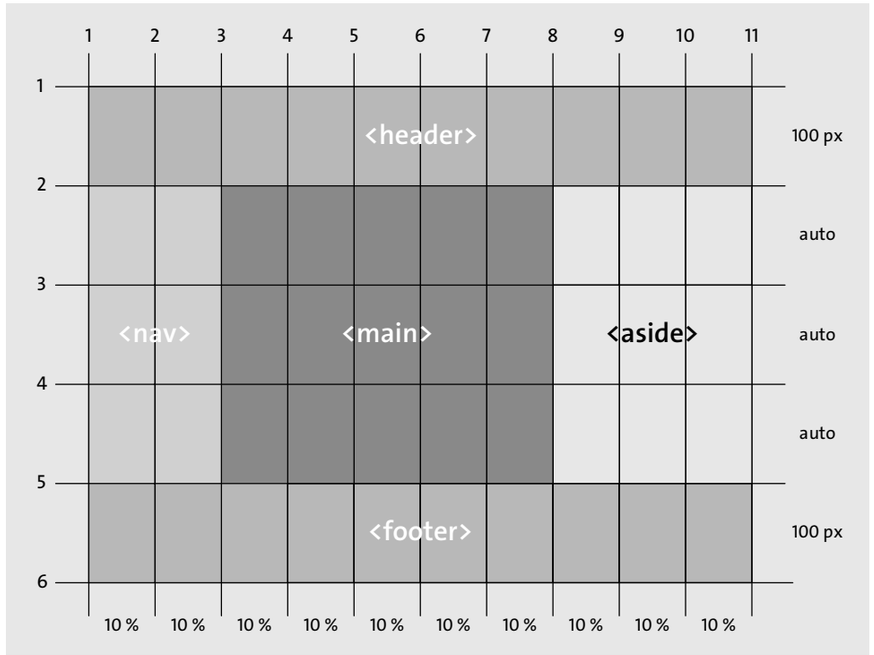

Abbildung 13.44 Die Desktopversion mit dem CSS-Grid

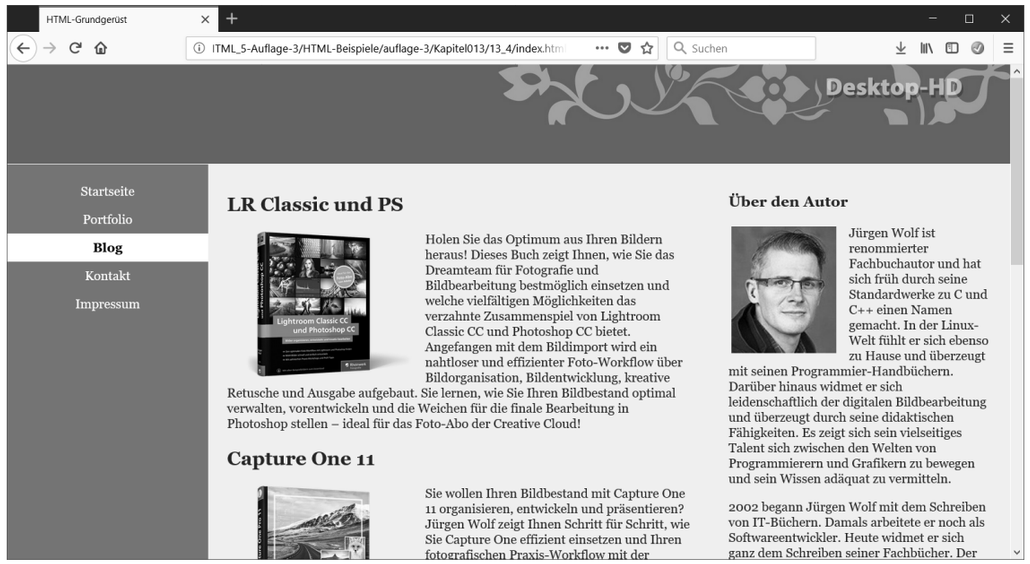

Abbildung 13.45 Die Desktopversion mit dem CSS-Grid bei der Ausführung

13.4.3 Layoutänderung leichtgemacht

Dank der Einfachheit, die Elemente völlig frei im Raster zu positionieren, wird es nun zum Kinderspiel, das Layout umzugestalten. Hierzu müssen Sie nur die Positionen der Zeilen und Spalten im Grid für die HTML-Elemente anpassen. Wollen Sie beispielsweise die Desktopversion ändern, damit die Seitenleiste links und die Navigation rechts ist, so ändern Sie in unserem Beispiel einfach die Werte für `grid-column` wie folgt:

```
@media screen and (min-width: 64em) {
  .content {
    grid-column: 3 / 8;
    grid-row: 2 / 4;
  }
  .aside {
    grid-column: 8 / 11;
    grid-row: 2 / 4;
  }
  .nav {
    grid-column: 1 / 3;
    grid-row: 2 / 4;
  }
…
}
```

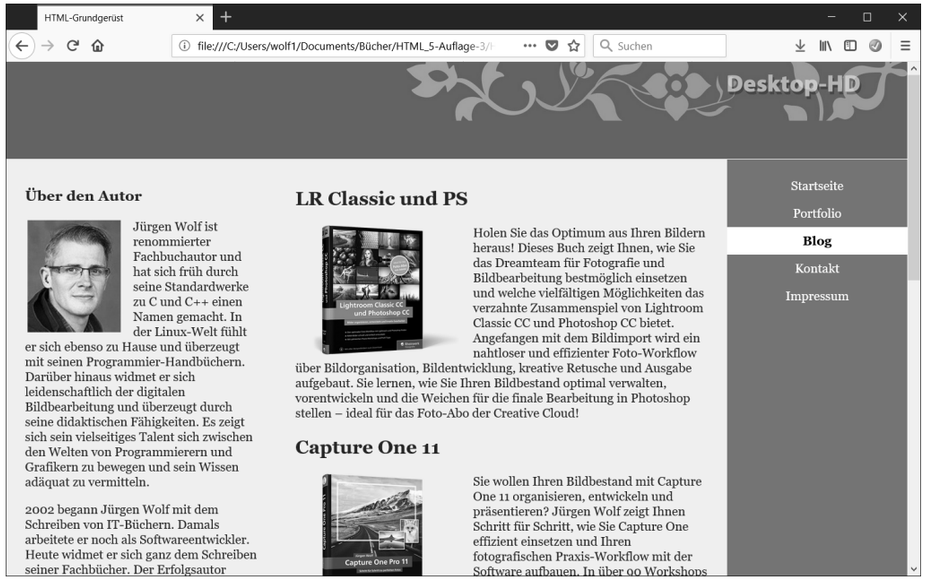

Abbildung 13.46 Eine Layoutänderung mit einem CSS-Grid ist in ein paar Sekunden erledigt.

13.4.4 Abstände zwischen den Rasterzeilen

Wollen Sie Abstände zwischen den Spalten oder Zeilen eines Rasters hinzufügen, können Sie dies im Eltern-Element mit den Befehlen `grid-column-gap` oder `grid-row-gap` bzw. der Kurzschreibweise `grid-gap` machen. Die Abstände werden hierbei nur zwischen den Spalten erzeugt. Am Anfang und Ende der Spalte oder Zeile wird kein Zwischenraum hinzugefügt. Zum Beispiel:

```
.grid {
  display: grid;
  grid-template-rows: 125px auto auto auto 100px;
  grid-template-columns: repeat(10, 10%);
  grid-row-gap: 15px;
  grid-column-gap: 10px;
  /* oder als Kurzschreibweise: grid-gap: 15px 10px; */
}
```

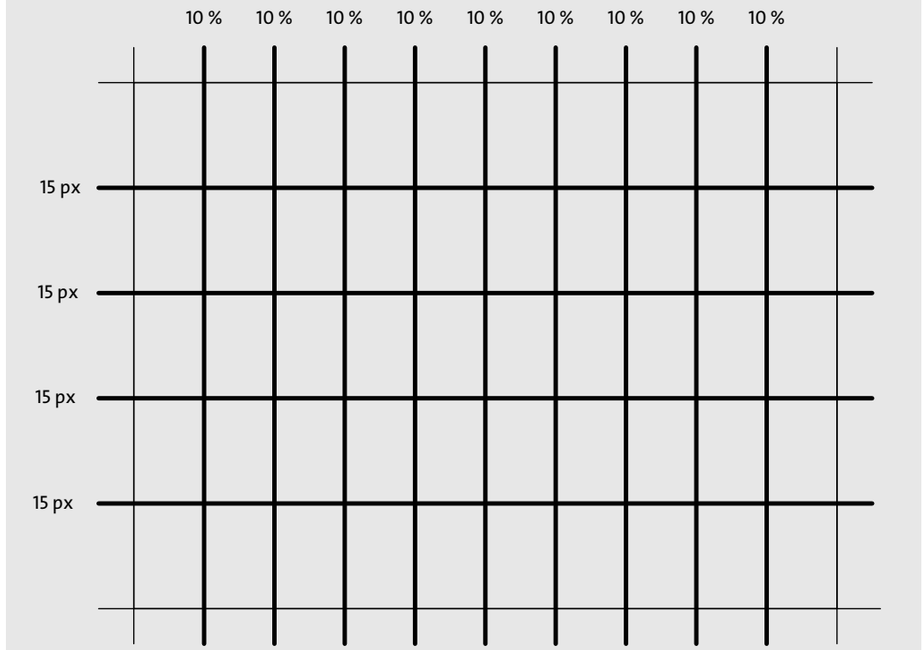

Abbildung 13.47 Hinzufügen von Abständen zwischen den Spalten eines CSS-Grids

Elemente im CSS-Grid ausrichten

Auch die horizontale und vertikale Ausrichtung der Elemente im Eltern-Element können Sie mit den CSS-Eigenschaften `align-items` für vertikales und mit `justify-items` für horizontales Verhalten festlegen. Der Standardwert hierfür ist `stretch`. Außerdem stehen die Werte `start`, `end` und `center` zur Verfügung. Für eine individuelle Ausrichtung einer einzelnen Rasterzelle hingegen steht `align-self` und `justify-self` zur Verfügung. Auch hier ist der Standardwert `stretch`, und darüber hinaus existieren die Werte `start`, `end` und `stretch`.

13.4.5 Browsersupport

Ein Blick auf *https://caniuse.com/#search=grid* zeigt, dass abgesehen vom Internet Explorer 11 alle wichtigen Webbrowser das CSS-Grid vollständig beherrschen. Der Internet Explorer bietet leider nur eine partielle Unterstützung mit dem Präfix `-ms` und zudem eine ältere Version der Spezifikation. Für fehlenden Browsersupport bietet sich daher ein Polyfill wie beispielsweise *https://github.com/FremyCompany/css-grid-polyfill* an.

13.5 Verhalten von HTML-Elementen mit »display« ändern

Sie haben in den Beispielen des Öfteren die CSS-Eigenschaft `display` verwendet, weshalb ihr noch ein paar Abschnitte für eine kurze Beschreibung gewidmet werden sollen.

Wie Sie mittlerweile mehrmals feststellen konnten, können Sie mit der CSS-Eigenschaft `display` das Verhalten eines HTML-Elements bei der Darstellung im Webbrowser ändern. Für jedes HTML-Element ist eine Box festgelegt, die das Verhalten des Elements beschreibt. Selbst einfache HTML-Elemente innerhalb einer Textzeile wie `<em>` oder `<a>` sind Boxen, und deren (Standard-)Verhalten können Sie mit `display` ändern.

So können Sie das Verhalten eines HTML-Elements wie `<p>` mit `display: inline;` ändern, womit kein Zeilenumbruch bzw. Absatzwechsel mehr ausgeführt wird. Umgekehrt können Sie das Verhalten eines Elements wie `<a>` mit `display: block;` so umändern, dass es einen Zeilenumbruch bzw. Absatzwechsel durchführt. Neben der häufigen Verwendung von `display: block;` und `display: inline;` wird noch `display: none;` eingesetzt, womit Sie ein Element ausblenden, sodass es keinen Platz mehr im HTML-Dokument beansprucht.

13.5.1 »display: block«, »display: inline« und »display: inline-block«

Mit `display: block;` wird ein Element als Block dargestellt, das einen Zeilenumbruch enthält. Diese Eigenschaft wird gerne im Zusammenspiel mit `display: inline;` verwendet, womit zum Beispiel Elemente für die Navigation entweder mit `display: block;` mit einem Zeilen-

umbruch von oben nach unten untereinander oder mit `display: inline;` in derselben Zeile ohne Zeilenumbruch von links nach rechts dargestellt werden.

Zur Demonstration sollen hier folgende CSS-Eigenschaften auf mehrere `p`-Elemente im HTML-Dokument angewendet werden:

```
p {
  display: block; /* Nicht nötig hier, da <p> ohnehin display:block ist */
  width: 150px;
  border: 1px solid black;
  background-color: white;
  padding: 1em;
}
```

Listing 13.23 /Beispiele/Kapitel013/13_5_1/index.html

Auf `display: block;` könnten Sie in diesem Fall verzichten, weil das `p`-Element ohnehin ein Block-Element ist. Diese Zeilen CSS sollen nun auf vier Absatztexte mit dem `p`-Element angewendet werden. Das Beispiel dazu sehen Sie in Abbildung 13.48.

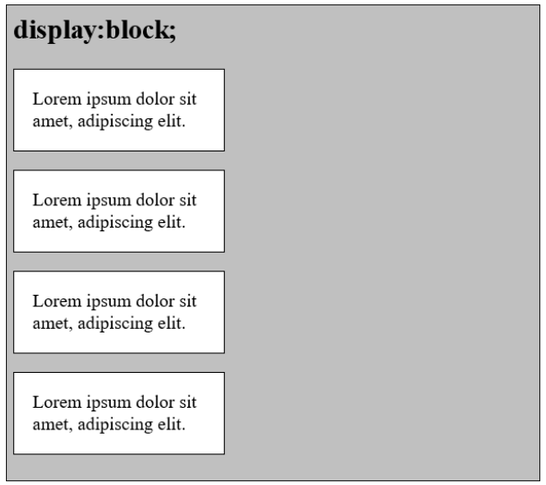

Abbildung 13.48 Das Verhalten, das Sie vom <p>-Element her kennen.

Als nächstes Beispiel soll nun `display: inline;` anstelle von `display: block;` verwendet werden:

```
p {
  display: inline;
  width: 150px;
  border: 1px solid black;
```



```
  background-color: white;
  padding: 1em;
}
```

Listing 13.24 /Beispiele/Kapitel013/13_5_1/index2.html

In Abbildung 13.49 erkennen Sie schon, dass mit `display: inline;` die Angabe von `width` ignoriert wurde und hier wirkungslos ist. Zwar können Sie hier Innen- und Außenabstände und Rahmen wie gewöhnlich festlegen, allerdings haben auch diese Angaben keinen Effekt in der Zeilenhöhe. Somit nimmt eine `inline`-Box nur die Breite ein, die der Inhalt benötigt. In Abbildung 13.50 sehen Sie, dass eine `inline`-Box sich auch über mehrere Zeilen erstrecken kann, was in diesem Fall nicht mehr schön anzusehen ist.

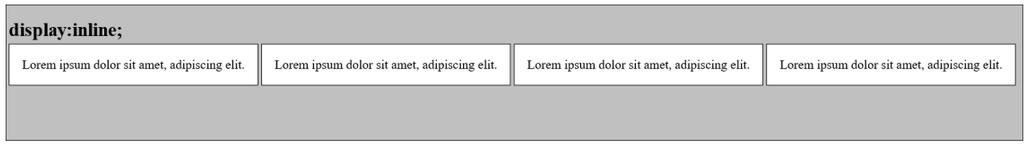

Abbildung 13.49 Das Verhalten der <p>-Elemente wurde auf »display: inline;« gesetzt.

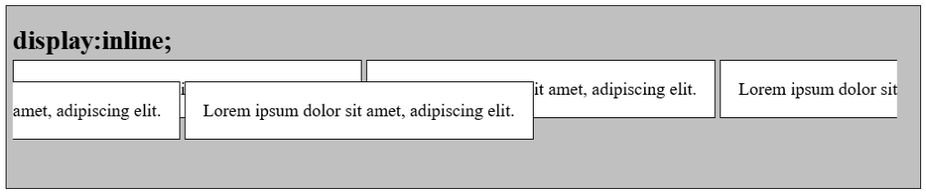

Abbildung 13.50 »inline«-Boxen können sich auch über mehrere Zeilen erstrecken.

Verwenden wir nun dasselbe Beispiel mit `inline-block`:

```
p {
  display: inline-block;
  width: 150px;
  border: 1px solid black;
  background-color: white;
  padding: 1em;
}
```

Listing 13.25 /Beispiele/Kapitel013/13_5_1/index3.html

Eine `inline-block`-Box verhält sich zunächst wie eine `inline`-Box und verläuft über eine Zeile (siehe Abbildung 13.51). Im Gegensatz zur `inline`-Box wird eine `inline-block`-Box allerdings nicht auf der nächsten Zeile fortgesetzt, sondern wird ähnlich wie `float` in die nächsten Zeile

verlegt, wenn die Box nicht mehr in der Bildschirmbreite passt, wie Sie es in Abbildung 13.52 sehen. Ebenfalls berücksichtigt wird bei den `inline-block`-Boxen, im Gegensatz zu den `inline`-Boxen, die Breite, die Sie mit `width` vorgeben.

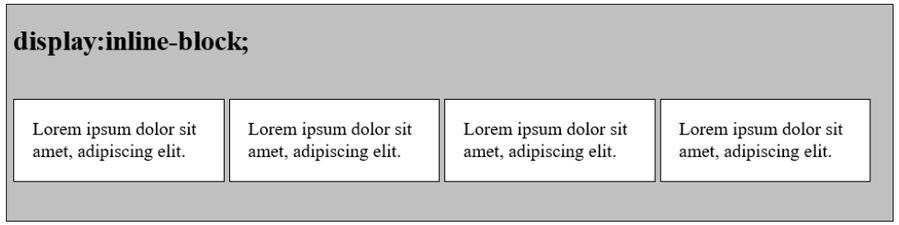

Abbildung 13.51 Hier habe ich das Verhalten der <p>-Elemente auf »display:inline-block;« gesetzt.

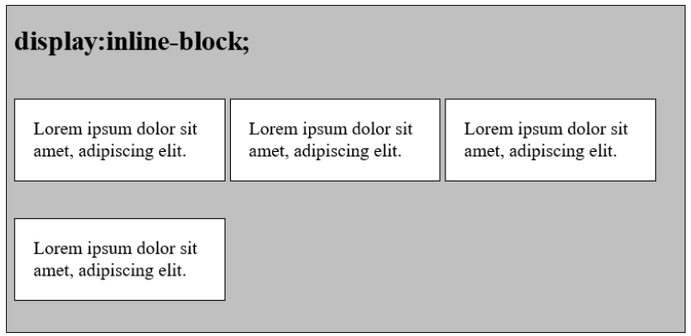

Abbildung 13.52 Eine »inline-block«-Box kann nicht über mehrere Zeilen aufgeteilt werden.

13.5.2 HTML-Elemente als Tabelle mit »display: table«

Eine weitere interessante Form, das Verhalten von Elementen mit `display` zu ändern, ist `table`. Damit können Sie Elemente wie in einer Tabelle anordnen und in der Praxis theoretisch auch ein Layout für eine Website erstellen. Zwar werden Spalten hierfür eher häufiger mit `float` und `display:flex` oder `display:grid` erstellt, aber `display:table` bietet einige Besonderheiten an, die recht nützlich sein können, wenn es Ihnen um das Anordnen von Elementen geht. Sind Sie zum Beispiel auf der Suche nach einem Weg, mehrere Elemente auf einer Zeile mit derselben Höhe anzupassen oder vertikal zu zentrieren, dann finden Sie mit `display:table` eine einfache Lösung dafür. Hierzu ein einfaches Beispiel:

```
…
.table {
  display: table;
}
.row {
```



```
  display: table-row;
}
.article {
  border: 1px solid black;
  padding: 1em;
  width: 33.3333%;
  background-color: lightgray;
  display: table-cell;
}
.bottom {
  vertical-align: bottom;
}
.middle {
  vertical-align: middle;
}
```

Listing 13.26 /Beispiele/Kapitel013/13_5_2/css/layout.css

Das HTML-Dokument dazu:

```
…
<div class="table">
  <div class="row">
    <article class="article bottom">…</article>
    <article class="article middle">…</article>
    <article class="article">…</article>
  </div>
  <div class="row">
    <article class="article">… </article>
    <article class="article middle">…</article>
    <article class="article bottom">…</article>
  </div>
</div>
…
```

Listing 13.27 /Beispiele/Kapitel013/13_5_2/index.html

Damit haben Sie eine Tabelle mit zwei Zeilen und drei Spalten erstellt. Für Zeilen können Sie `display:table-row;` und für Spalten `display:table-cell;` verwenden. Der schöne Nebeneffekt hierbei ist auch gleich, dass alle Spalten einer Zeile automatisch gleich hoch sind, und zudem können Sie den Inhalt von `display:table-cell;` ganz komfortabel mit `vertical-align` mit `middle` zentrieren oder mit `bottom` unten ausrichten. Das Beispiel sehen Sie in Abbildung 13.53 bei der Ausführung.

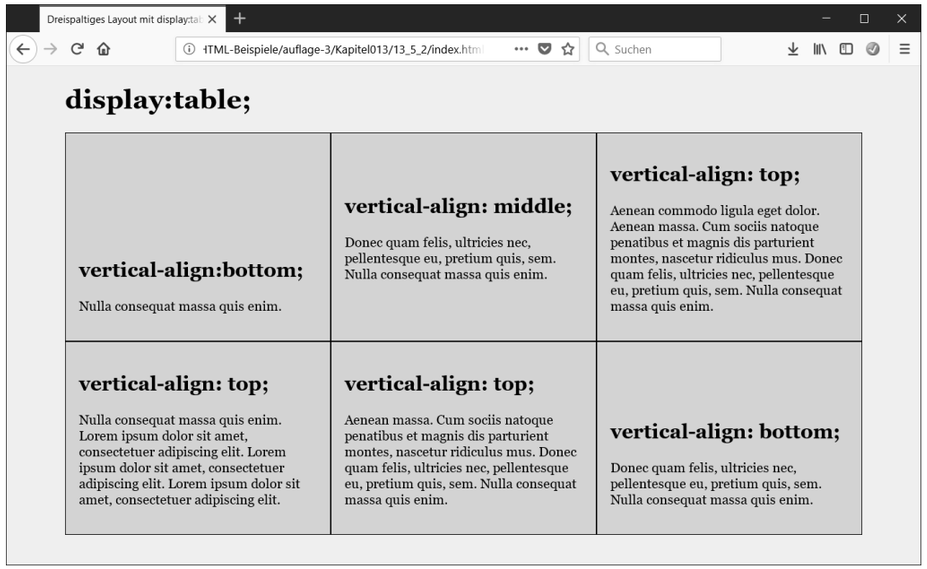

Abbildung 13.53 Tabellen mit »display:table;«

Für das Erstellen von Tabellen in CSS stehen noch spezielle Werte für `display` zur Verfügung. In Tabelle 13.2 finden Sie einen Überblick über die Werte und deren Bedeutung.

display	Bedeutung
`table`	Das Element verhält sich wie `<table>`.
`inline-table`	Das Element verhält sich wie `<table>` in HTML, aber inline.
`table-caption`	Das Element verhält sich wie `<caption>`.
`table-column-group`	Das Element verhält sich wie `<colgroup>`.
`table-header-group`	Das Element verhält sich wie `<thead>`.
`table-footer-group`	Das Element verhält sich wie `<tfoot>`.
`table-row-group`	Das Element verhält sich wie `<tbody>`.
`table-cell`	Das Element verhält sich wie `<td>`.
`table-column`	Das Element verhält sich wie `<col>`.
`table-row`	Das Element verhält sich wie `<tr>`.

Tabelle 13.2 Werte für die Erzeugung von Tabellen in CSS

13.5.3 Elemente verstecken mit »display:none«

Mit `display: none;` können Sie Elemente einfach verstecken, damit sie nicht mehr im Dokument sichtbar sind. Der Webbrowser erstellt für solche Elemente keine Box, und alle Anweisungen zur Positionierung werden ignoriert. Neben `display: none;` gibt es die Möglichkeit, mit `visibility: hidden;` ein Element auszublenden. Im Gegensatz zu `display: none;` bleibt hierbei allerdings die Box erhalten, und das Element behält seine Auswirkungen auf die nachfolgenden Elemente bei. Mit `visibility: hidden;` wird das Element im Grunde nur komplett transparent gemacht.

13.5.4 Weitere Werte für »display«

Es gibt noch einige weitere Werte, mit denen Sie das Verhalten von Elementen mit `display` verändern. Mit `display: flex;` steht Ihnen ein alternatives Modell für die Positionierung von Elementen in Zeilen und Spalten zur Verfügung. `display: grid;` hingegen ist noch flexibler und ermöglicht die Erstellung eines komplexeren Layout-Rasters, in dem Sie jedes Element beliebig platzieren können. Auf `flex` und `grid` bin ich bereits eingegangen.

Zu erwähnen wäre hier noch `list-item`, womit Sie das Element als Liste darstellen. Damit werden für ein Element zwei Boxen erzeugt. Eine Box wird dabei für den Listenpunkt und die andere Box für das Listenelement verwendet.

Es gibt noch andere Werte für `display`, die allerdings eher selten eingesetzt werden oder zum Teil gar nicht richtig oder nie implementiert wurden.

13.6 Berechnungen mit CSS und der »calc()«-Funktion

Gerade im responsiven Webdesign kann es hilfreich sein, sich die einzelnen Angaben abhängig von der Medieneigenschaft errechnen und darstellen zu lassen. Eine solche Funktion steht in CSS mit `calc()` zur Verfügung, womit Sie die die Grundrechenarten Plus (+), Minus (–), Multiplikation (*) und Division (/) durchführen können. Hierbei können Sie auch Einheiten mischen und beispielsweise Pixel mit Prozent multiplizieren. Wichtig ist zudem, dass beim Plus und Minus vorher und nachher ein Leerzeichen stehen muss. Bei Multiplikation und Division ist dies nicht nötig.

Um `calc()` zu demonstrieren, erstellen wir ein einfaches Gestaltungsraster mit acht Sektionen. In der Standardversion für Bildschirme über 640 Pixel sollen hierbei jeweils vier Sektionen in einer Reihe verteilt werden (4 × 2). Bei Bildschirmen, die weniger als 640 Pixel breit sind, soll ein Layoutumbruch erfolgen, bei dem nur noch zwei Spalten pro Zeile zu sehen sind (2 × 4). Bei jedem Layoutumbruch wird die Anzahl der Spalten passend zur Breite neu berechnet. Hierfür wird im folgenden Beispiel die Klasse `.spalte` verwendet. Bei einer Bildschirmbreite von weniger als 480 Pixeln wird noch eine Spalte pro Zeile verwendet (1 × 8). Hierzu das Beispiel:

```
…
img {
  width: 100%;
  display: block;
}
.spalte {
  float: left;
  padding: 10px;
  width: calc(100% / 4);
}

@media screen and (max-width: 40em) {
  .spalte {
    width: calc(100% / 2);
  }
}

@media screen and (max-width: 30em) {
  .spalte {
    width: 100%;
  }
}
```

Listing 13.28 /Beispiele/Kapitel013/13_6/css/layout.css

Das HTML dafür:

```
…
<section class="spalte">1</section>
<section class="spalte">2</section>
<section class="spalte">3</section>
<section class="spalte">4</section>
<section class="spalte">5</section>
<section class="spalte">6</section>
<section class="spalte">7</section>
<section class="spalte">8</section>
…
```

Listing 13.29 /Beispiele/Kapitel013/13_6/index.html

In den folgenden Abbildungen sehen Sie sehr schön, wie mit `calc()` bei jedem Layoutumbruch ein neues Gestaltungsraster berechnet wird. Hierbei wird stets nur die Klasse `.spalte` verwendet.

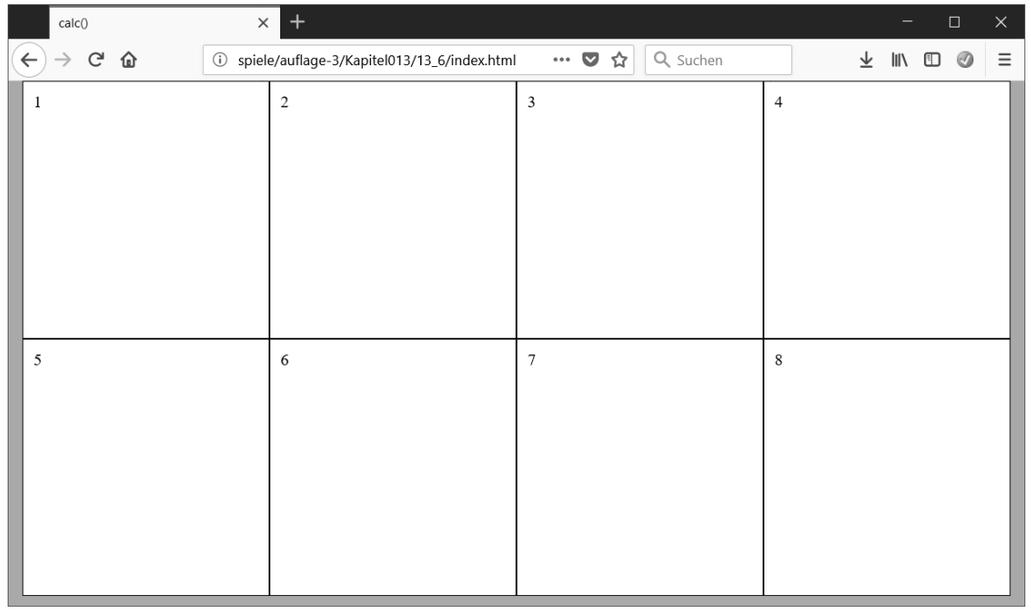

Abbildung 13.54 4-spaltiges Layout mit »width: calc(100% / 4);« für einen Viewport über 640 Pixel

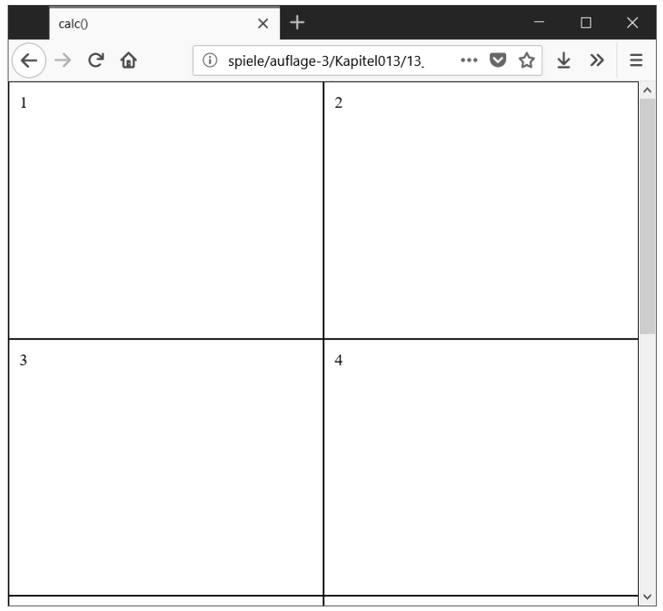

Abbildung 13.55 2-spaltiges Layout mit »width: calc(100% / 2);« für einen Viewport unter 640 Pixel

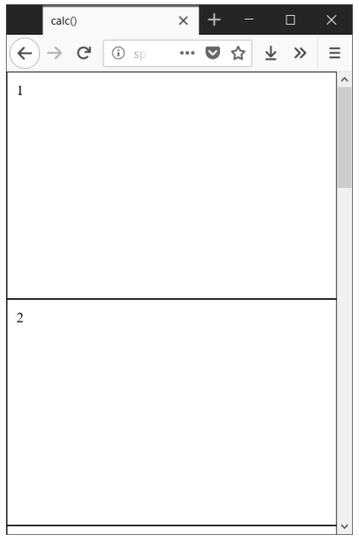

Abbildung 13.56 ... und ein einspaltiges Layout mit »width: 100%;« für den Viewport unter 480 Pixel

Das Anwendungsgebiet von `calc()` geht natürlich weit über dieses Beispiel hier hinaus. So lässt es sich zum Beispiel auch sehr gut für die absolute Positionierung auf dem Bildschirm oder das automatische Anpassen einzelner Elemente innerhalb des Eltern-Elementes verwenden. Auch eine flexible Anpassung der Schriftgröße lässt sich mit `calc()` berechnen.

13.7 Zusammenfassung und »Da geht noch (viel) mehr ...«

An dieser Stelle angekommen, wünscht man sich als Autor, noch weitere Details zum Thema »responsives Layout mit CSS« zu liefern. Bei diesem Buch handelt es sich allerdings nicht um ein reines CSS-Handbuch oder ein Buch zum responsiven Webdesign. Aber Sie kennen jetzt die Grundlagen zum responsiven Webdesign und sind mit den Medienabfragen vertraut. Sie kennen nun auch nützliche Bausteine, mit denen Sie selbst kreativ werden und einfache Layouts entwerfen.

Ein paar Worte noch zu den Beispielen in diesem Kapitel, die Sie nur als einfache Anregungen verstehen sollten. Die Möglichkeiten, mit CSS Layouts zu erstellen, sind extrem vielfältig und in der Praxis nicht selten aufwendiger. Aber gerade das ist das Schöne an der Webentwicklung: Sie können kreativ sein und selbst etwas Einzigartiges erschaffen. Um etwas »Spezielles« zu erstellen, müssen Sie Erfahrungen sammeln und möglichst viel ausprobieren und sich Schritt für Schritt heranarbeiten. Einen Königsweg nach dem Motto »Mach es so, dann ist es perfekt!« gibt es selten, weil dies von den Bedürfnissen (und auch den Kenntnissen) eines jeden Einzelnen abhängt. Es kommt ganz darauf an, was Sie vorhaben zu erstellen.

Auf einen Blick

Inhalt