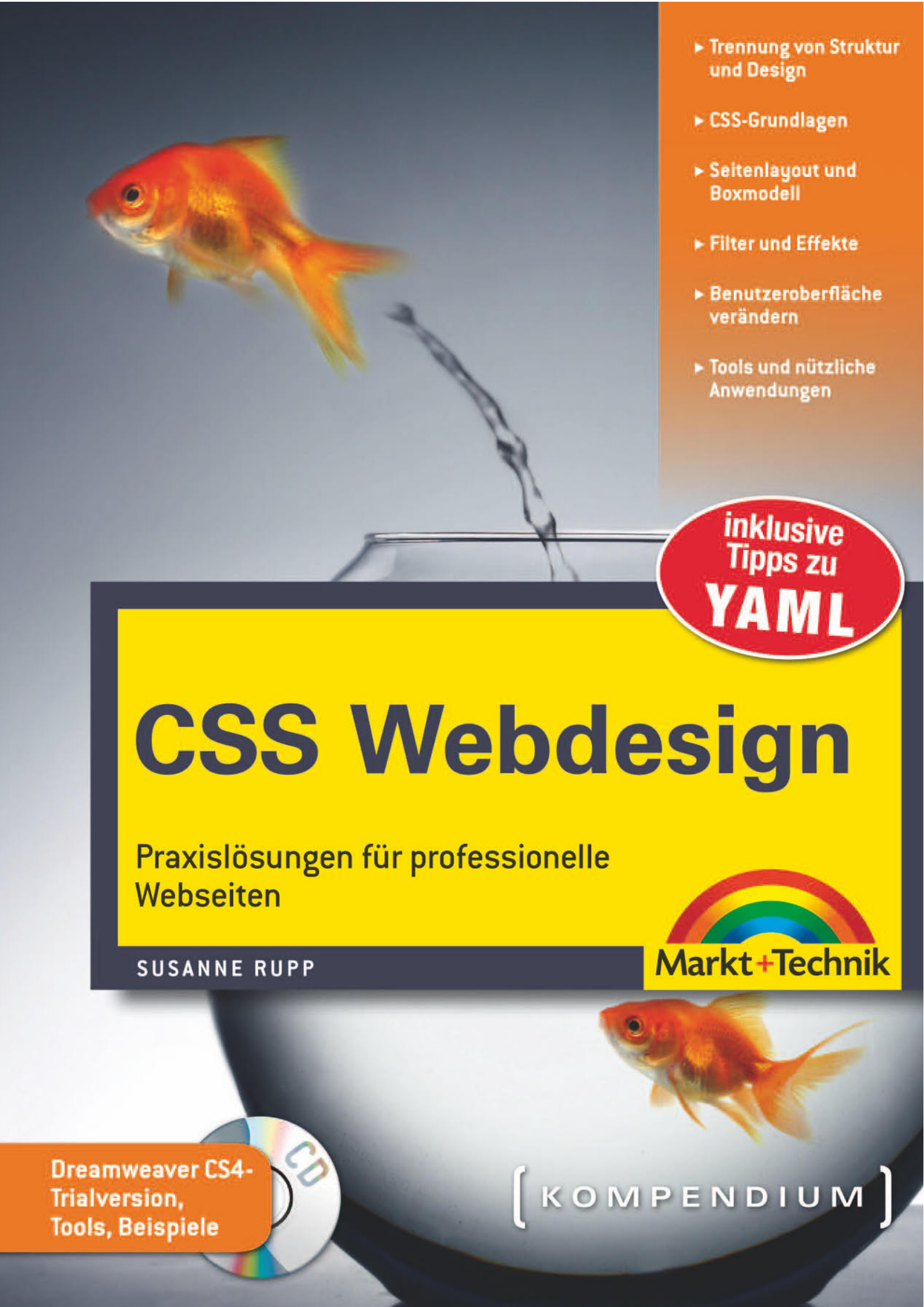


- Trennung von Struktur und Design
- CSS-Grundlagen
- Seitenlayout und Boxmodell
- Filter und Effekte
- Benutzeroberfläche verändern
- Tools und nützliche Anwendungen

inklusive Tipps zu YAML

CSS Webdesign

Praxislösungen für professionelle Webseiten

SUSANNE RUPP

Markt+Technik

Dreamweaver CS4-Trialversion, Tools, Beispiele

CD

KOMPENDIUM

3 Aufbau der Cascading Style Sheets

Definition Cascading Style Sheet

Mit CSS-Regeln bestimmen Sie das Aussehen einzelner Seitenelemente bzw. Inhaltsbereiche, die Sie zuvor mit HTML strukturiert haben. Bei einem Style Sheet handelt es sich also um eine Sammlung unterschiedlicher Stilregeln zur Formatierung einzelner Inhalte. Dabei enthält der **Stil** (`style`) die Deklaration mit den gewünschten visuellen Eigenschaften. So ändern Style Sheets die Voreinstellungen der Webbrowser und anderer Wiedergabegeräte. Das **Sheet** bezeichnet die Sammlung unterschiedlicher Stile, die häufig in einer separaten CSS-Datei gespeichert ist (vgl. Kapitel 3.2.1). Der Begriff **Cascading** bezieht sich auf die gleichzeitige, hierarchische Verwendung mehrerer Stile und Sheets, wobei Reihenfolge und Stilart wichtig sind. Dabei werden sich überschneidende Anweisungen nach bestimmten Regeln angewendet.

Generell besteht ein CSS-Stil aus zwei Teilen, nämlich einem Selektor und der Deklaration in geschweiften Klammern. Dabei wird der Selektor häufig auch als Bezeichner beschrieben. Der Selektor bestimmt, was formatiert wird. Die Deklaration definiert, wie dies geschehen soll. Dabei steht hinter der Eigenschaft durch einen Doppelpunkt getrennt der gewünschte Wert. Für die Übersichtlichkeit können Sie nach dem Doppelpunkt noch ein Leerzeichen in den Code einfügen. Ein solches Attribut endet mit einem Semikolon, das bei der letzten Deklaration auch fehlen kann. Damit stellt ein Selektor die Verbindung zwischen Seitenelement bzw. sichtbarem Inhalt und der in der Stilregel festgelegten Formatierung her.

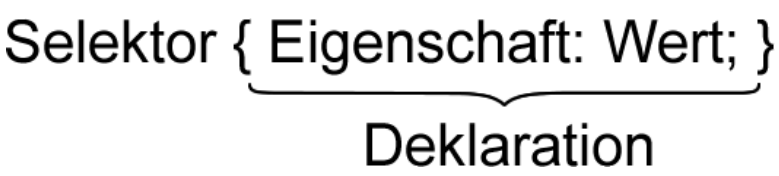


Abbildung 3.1: Aufbau einer CSS-Regel

Einem Selektor können Sie mehrere Deklarationen gleichzeitig zuweisen. Zudem können diese Attribute für mehrere Selektoren gelten, die durch ein Komma voneinander getrennt sind. In diesem Fall spricht man auch von einem zusammengesetzten Ausdruck.

Listing 3.1: Selektor mit mehreren Deklarationen

```
Selektor1, Selektor2 {
  Eigenschaft1: Wert1;
  Eigenschaft2: Wert2;
 }
```

Fehlt das Komma zwischen den einzelnen Selektoren, definieren Sie eine Selektor-Kombination. Die Regel wird erst umgesetzt, wenn die angesprochenen Elemente in einer bestimmten Reihenfolge in den Code eingebunden sind (vgl. Kapitel 3.1.7).

In übersichtlichem Code stehen die Selektoren standardmäßig in einer Zeile über der eingerückten Deklaration. Dabei steht jedes Eigenschaft-Wert-Paar wiederum in einer separaten Zeile. Prinzipiell könnten Sie aber auch den gesamten Code fortlaufend hintereinander schreiben.

Basiert die Webseite auf XHTML, müssen Sie auch bei den Selektoren auf Groß- und Kleinschreibung achten. Am besten schreiben Sie den gesamten Code klein.

Kommentare

Webautoren binden häufig Kommentare mit weiteren Informationen und Erläuterungen in den Quelltext ein und können sich später besser im Code zurechtfinden. Änderungen sind schneller durchführbar. Ein Kommentar steht im CSS-Code zwischen dem Anfangszeichen `/*` und dem Endzeichen `*/`. Dies kann dann beispielsweise folgendermaßen aussehen:

```
/* Das ist ein Kommentar. */
```

Im Folgenden wird die Darstellung einzelner HTML-Elemente sowie weiterer Selektoren mit CSS erläutert. Stilregeln formatieren jedoch auch XML-Inhalte, die ebenfalls über eine geordnete, logische Struktur verfügen.

Die Erläuterung von XSLT würde den Rahmen dieses Buches leider sprengen. Weitere Informationen zu diesem Thema finden Sie in meinem Dreamweaver-Kompendium.

3.1 Unterschiedliche Selektor-Typen verwenden

Bevor Sie ein Style Sheet erstellen, sollten Sie dessen Einsatz konkret planen und den passenden Selektor-Typ wählen. Auf den folgenden Seiten lernen Sie daher zunächst die unterschiedlichen Selektoren und die sich daraus ergebenden Notationen kennen. Dabei beziehen sich Element-Selektoren auf HTML-Tags, denen ein neues Erscheinungsbild zugewiesen wird. Klassen gelten dagegen für unterschiedliche Elemente und sind deutlich flexibler einsetzbar. ID-Selektoren können einem bestimmten Seitenelement zugeordnet werden, das über eine ID eindeutig identifiziert wird. Auch Hyperlink-Zustände werden häufig mit CSS definiert, wodurch interaktive Rollover-Effekte entstehen. Hinzu kommen weitere spezielle Selektor-Typen.

Separate CSS-Datei einbinden

Dabei entspricht die Schreibweise des vorgestellten Codes der in einer separaten CSS-Datei. Möchten Sie die Beispiele der folgenden Seiten gleich selbst ausprobieren, arbeiten Sie folgendermaßen:

1. Speichern Sie den CSS-Stil oder gleich mehrere Regeln in einer Datei mit der Endung *.css* (z. B. *style.css*). Wegen der Übersicht werden Style-Sheet-Dateien häufig in einem separaten Verzeichnis abgelegt.
2. Anschließend öffnen Sie das HTML-Dokument mit den Inhalten, auf die Sie die Regeln anwenden möchten.
3. Fügen Sie einen Verweis auf die CSS-Datei in den `head`-Bereich des HTML-Dokuments nach folgendem Muster ein, wobei Sie *style.css* ggf. durch den zuvor gewählten Dateinamen ersetzen. Achten Sie auch auf die korrekte Verzeichnisangabe.

Listing 3.2: Externe CSS-Datei einbinden

```
<link href="style.css" rel="stylesheet" type="text/css" />
```

Dokument-internes Style Sheet

Alternativ binden Sie die CSS-Regel mit dem `style`-Element in den `head`-Bereich der HTML-Datei ein (Listing 3.3). Diesem Listing entsprechen wegen der besseren Übersicht die meisten Beispiele auf der beiliegenden CD-ROM.

Listing 3.3: CSS-Regel im Dokumentkopf

```
<head>
  <title>…</title>
  <style type="text/css">
    <!--
      Selektor { Eigenschaft: Wert; }
    -->
  </style>
</head>
```


Weitere Informationen zu diesem Thema und alternative Möglichkeiten erläutert Kapitel 3.2. Die entsprechenden Beispieldateien sind auf der CD-ROM im Verzeichnis kap03 *gespeichert. Zudem können Sie eine Deklaration als Inline-Stil innerhalb des* `style`*-Attributs eines HTML-Elements einsetzen. Ein Selektor ist hierbei nicht erforderlich.*

Nachdem Sie die unterschiedlichen Selektoren und ihre Einsatzmöglichkeiten kennen gelernt haben, betrachten wir die zweite Hälfte einer CSS-Regel. Damit wenden wir uns den wichtigsten Werten zu, die in unterschiedlichen CSS-Eigenschaften immer wieder verwendet werden: den unterschiedlichen Maßeinheiten und Farben.

CSS-Stile erstellen Sie mit den gleichen Programmen wie HTML-Dokumente. Auch hier genügt bereits der einfache Text-Editor. Weitere Editoren stellt Kapitel 18 vor.

Tabelle 3.1 zeigt eine Übersicht der auf den folgenden Seiten vorgestellten Selektoren und deren Unterstützung in einzelnen Webbrowsern. Diese finden Sie auch auf der beiliegenden Referenzkarte. Detaillierte Informationen erläutern die einschlägigen Kapitel.

Tabelle 3.1: Referenz – Browserkompatibilität unterschiedlicher Selektoren

Selektor	CSS-Version	IE			FF			Safari			Opera		
		6	7	8	1	2	3	2	3	4	7	8	9
Element	1	j	j	j	j	j	j	j	j	j	j	j	j
Klasse, class	1	j	j	j	j	j	j	j	j	j	j	j	j
ID	1	j	j	j	j	j	j	j	j	j	j	j	j
Universal	2-3	j	j	j	j	j	j	j	j	j	j	j	j
Pseudo-Klassen	1, 2	t	t	j	j	j	j	t	t	t	t	t	t
Pseudo-Elemente	1, 2	t	t	j	j	j	j	t	t	t	j	j	j
Attributselektoren	2-3	n	j	j	j	j	j	j	j	j	t	t	j
kombinierte Selektoren	1, 2	t	j	j	j	j	j	j	j	j	j	j	j

3.1.1 Element-Selektoren für HTML-Tags

Browser geben die HTML-Struktur einer Website nach vordefinierten Standardeinstellungen wieder. Überschriften `<h1>` werden beispielsweise in einer größeren Schrift als Absätze `<p>` dargestellt und in den meisten Browsern mit der Schrift *Times* angezeigt.

Element-Selektoren bestimmen das Erscheinungsbild der kompletten Webseite sowie einzelner HTML-Elemente. Dabei sind Selektor- und Elementname identisch und die definierte Deklaration ist eindeutig dem Seitenelement zuzuordnen. Element-Selektoren werden häufig auch als Typ-Selektoren bezeichnet.

Damit beispielsweise der gesamte Text einer Webseite in der Schrift *Arial* und in schwarzen Zeichen dargestellt wird, weisen Sie diese allgemein gültigen Seiteneigenschaften dem `body`-Element zu. Dieses ist allen Seitenelementen übergeordnet und bestimmt damit deren Aussehen. Die Eigenschaften des `body`-Elements werden damit an untergeordnete Elemente wie Überschriften (`<h1>` bis `<h6>`) oder Absätze (`<p>`) vererbt.

Die Schriftart bestimmt das Attribut `font-family`. Die Zeichenfarbe wird als Hexadezimalcode (z. B.`#000;` oder `#000000;`) für die Eigenschaft `color` angegeben. Alternativ können Sie statt eines Hexadezimalcodes auch den allgemein gültigen Farbnamen entsprechend der CSS2.1-Spezifikation angeben (`black`).

Listing 3.4: Schriftart und Textfarbe einer Seite festlegen

```
body {
  font-family: Arial;
  color: #000;
}
```

Ausführliche Informationen über die Text- und Zeichenformatierung finden Sie in Kapitel 5.

Diese zentralen Seiteneigenschaften können andere CSS-Regeln einschränken oder überschreiben. Weitere Informationen über dieses Kaskadenprinzip und die damit verbundene Vererbung finden Sie in Kapitel 3.3.

Beispielsweise werden alle Überschriften der ersten und zweiten Ordnung (`h1`, `h2`) mit Listing 3.5 in rot und fett hervorgehobenen Zeichen dargestellt, ohne dass weitere Auszeichnungen in den HTML-Tags notwendig sind (Listing 3.6). Da zudem die Zeichengröße einheitlich mit 16 Pixel angegeben ist, erkennen Sie in Abbildung 3.2 gut, dass die Standard-HTML-Einstellungen des Webbrowsers außer Kraft gesetzt sind. Normalerweise wäre `<h1>` nämlich deutlich größer als `<h2>`. Die CSS-Regel gilt also für alle Instanzen dieser beiden Elemente.

Listing 3.5: Stilregel mit mehreren Selektoren

```
h1, h2 {
  color: red;
  font-weight: bold;
  font-size: 16px;
}
```

Listing 3.6: Automatische Formatierung der HTML-Elemente

```
<h1>Das ist eine Überschrift erster Ordnung.</h1>
<p>Dieser Absatz wird normal dargestellt.</p>
<h2>Das ist eine Überschrift zweiter Ordnung.</h2>
<p>Dieser Absatz wird normal dargestellt.</p>
```

Auch anderen Selektor-Typen können Sie auf diese Weise die gleiche CSS-Deklaration zuweisen.

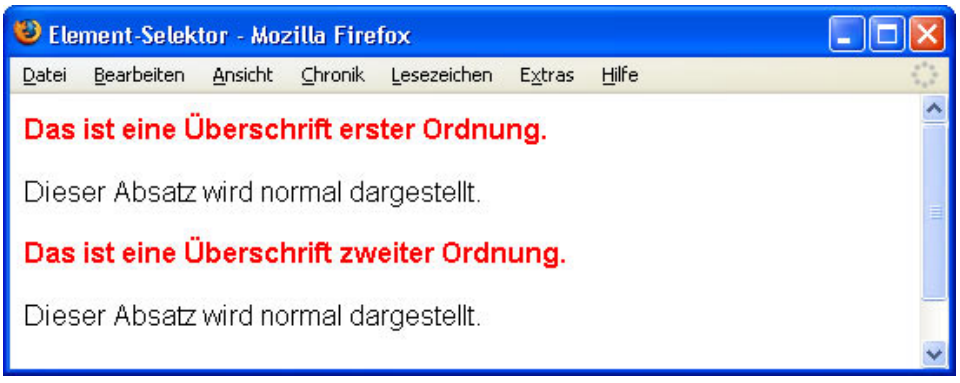


Abbildung 3.2: Anwendung des Element-Stils

TIPP

Definieren Sie die Tags für Formularelemente (<input>*) nicht neu. Verwenden Sie besser benutzerdefinierte Klassen und Kontext-Selektoren, die Sie auf ausgewählte Formularelemente anwenden. Das* input*-Tag wird für unterschiedliche Formularelemente benötigt, beispielsweise Textfelder und Schaltflächen.*

Nicht immer ist ein zu formatierender Seiteninhalt in ein HTML-Element eingebunden, dem Sie einen CSS-Stil zuweisen könnten. Im folgenden Beispiel sollen lediglich die zwei Begriffe »dieses Wort« rot markiert werden, nicht jedoch der komplette Absatz <p>.

```
<p>In diesem Absatz soll dieses Wort hervorgehoben werden.</p>
```

Die logische Auszeichnung mit oder ist dafür jedoch nicht immer geeignet. Die Verwendung solcher Tags sollte nämlich deren inhaltlicher Bedeutung entsprechen (vgl. Kapitel 2.3).

Um diese Aufgabe zu bewältigen, benötigt der HTML-Code weitere Tags zur Kennzeichnung der entsprechenden Textstelle. Seit HTML 4.0 gibt es hierzu das neutrale Inline-Element , das ohne Deklaration die Darstellung im Browser nicht weiter beeinflusst. Start- und End-Tag umschließen den zu formatierenden Inhalt, wobei Sie auf eine korrekte Verschachtelung achten müssen. Erstellen Sie nun eine Stilregel für das span-Element, gilt auch diese für alle entsprechend gekennzeichneten Inhalte. Dies ist jedoch nicht immer sinnvoll. Viel flexibler arbeiten Sie mit CSS-Klassen, die Sie mit dem style-Attribut in das Element integrieren (vgl. Kapitel 3.1.2). Alternativ können Sie natürlich auch einen Inline-Stile mit dem style-Attribut hier einfügen (vgl. Kapitel 3.2.3):

Listing 3.7: Inline-Element mit

```
<p>In diesem Absatz soll
   <span style="color: #F00">dieses Wort</span>
   hervorgehoben werden.</p>
```

Auch ID-Stile, die immer nur für ein einziges Element innerhalb einer Webseite gelten, bilden eine gute Alternative zur allgemeinen Element-Stildefinition (vgl. Kapitel 3.1.3). Über die ID wird das Element eindeutig identifizierbar.

ID-Stile werden primär für div-Elemente verwendet, die ebenfalls mit HTML 4.0 eingeführt wurden. Anders als beim span- handelt es sich hierbei um ein Block-Element, das ohne CSS-Formatierung einen Zeilenumbruch im Browser verursacht. Das div-Element wird häufig für die Aufteilung des Seiteninhaltes in unterschiedliche Gruppen bzw. Container verwendet. Position und Größe der einzelnen Inhalte lassen sich damit eindeutig bestimmen. Das darauf basierende Boxmodell lernen Sie in Kapitel 7 kennen.

DISC

Die Beispiele sind im Verzeichnis kap03 *in der Datei* elemente.htm *gespeichert.*

3.1.2 Klassen für mehrmaligen, flexiblen Einsatz

class="name"

Die in CSS-Klassen festgelegten Format-Eigenschaften sind flexibel auf beliebige HTML-Elemente der Webseite anwendbar. Umschließt kein HTML-Tag die zu formatierende Stelle, verwenden Sie das `span`-Tag als Inline-Element. Die Zuweisung erfolgt mit dem `class`-Attribut im Element. Dabei muss der angegebene Wert exakt dem Namen des Klassen-Selektors entsprechen. In Listing 3.8 wird dadurch eine bestimmte Textstelle markiert und der Klassenname `hervor` zugewiesen. Der Absatz wird dagegen entsprechend den in der Klasse `info` gespeicherten Eigenschaften formatiert.

Listing 3.8: Klassen für `p`-Tag und Inline-Element

```
<p class="info">In diesem Absatz soll
   <span class="hervor">dieses Wort</span>
     hervorgehoben werden.
     </p>
```


.name

Im Style Sheet erkennen Sie den class-Selektor an dem Punkt vor dem Klassen-Namen (Listing 3.9). Basiert die Webseite auf XHTML, müssen Sie zwischen Groß- und Kleinschreibung unterscheiden. Zweckmäßig verwenden Sie nur noch Kleinschreibung. Ansonsten können Sie den Klassen-Namen frei wählen. Dabei muss der Selektorname mit einem Buchstaben beginnen, darf jedoch auch Ziffern und den Bindestrich enthalten.

Notation & Sonderzeichen

Auf Sonderzeichen, wozu auch Umlaute zählen, und den Unterstrich sollten Sie weitgehend verzichten, da diese in der ersten Spezifikation nicht vorgesehen waren und zu Problemen in älteren Browsern führen können. Benötigen Sie dennoch Sonderzeichen, sollten diese aus dem ISO 10646-Zeichensatz stammen und als numerischer Code integriert werden. Damit der Parser diesen dann richtig interpretiert, müssen Sie das Escape-Zeichen \ voranstellen. Theoretisch kann der Bezeichner auch mit einem Unterstrich _ beginnen, wobei die entsprechende Regel von älteren Browsern jedoch nicht erkannt wird, wie etwa dem Internet Explorer 6. Auf diese Weise können Sie jedoch Regeln für spezielle Browser-Generationen entwickeln.

Listing 3.9: CSS-Regeln mit unterschiedlichen Klassen

```
.hervor {
  font-weight: bold;
  color: #F00;
}
.info {
  font-size: 16px;
  color: #00F;
}
```

Wird auf ein Element bereits ein allgemein gültiger Element-Stil angewendet, überschreiben Klassen-Stile diese Formatierungsregeln (vgl. Kapitel 3.3). Dabei können Sie diese Ausnahmeformate beliebig häufig einsetzen.

Elementspez. Selektoren

Soll eine Klasse nur für ein bestimmtes Element gelten, geben Sie den Elementnamen direkt davor, also ohne Leerzeichen, ein. Beispielsweise gilt die Regel in Listing 3.10 nur für einen Absatz, in dem das entsprechende `class`-Attribut ausgewiesen ist. In anderen Elementen wirkt sich ein solcher Regel-Verweis nicht aus. Während beispielsweise die erste Codezeile in Listing 3.11 entsprechend formatiert wird, wirkt sich die Klassen-Angabe in der zweiten Zeile nicht auf die Zeichen-Darstellung aus.

Listing 3.10: Klasse `gelb` für Absätze

```
p.gelb {
  font-size: 12px;
  color: yellow;
}
```

Listing 3.11: Klasse wird im `<span>` ignoriert.

```
<p class="gelb">Hier wird die Klasse angewendet.</p>
<p><span class="gelb">Hier aber nicht.</span></p>
```

Soll die Klasse dagegen für alle Elemente mit dem entsprechenden `class`-Verweis gelten, können Sie auch den Universal-Selektor `*` davor setzen (vgl. Kapitel 3.1.4) oder ihn einfach weglassen. Der Ausdruck `*.gelb` ist also identisch mit der Schreibweise `.gelb`.

Mehrere Selektoren

Zudem können Sie einem HTML-Element eine Klasse bestehend aus mehreren `class`-Selektoren zuweisen und damit eine eigene Syntax realisieren bzw. den CSS-Code übersichtlich strukturieren. Die einzelnen Selektoren stehen ohne Leerzeichen hintereinander (z. B. `.info.gelb`).

Listing 3.12: Regel kombinierter Klassen-Selektoren

```
.info.gelb {
  font-size: 14px;
  color: #00F;
}
```

Das `class`-Attribut im Element enthält dagegen zwischen den Selektornamen ein Leerzeichen. Zudem können Sie dem Element weitere Klassen zuweisen. Dabei ist die Reihenfolge der Nennung unerheblich.

Listing 3.13: Anwendung der Klasse `.info.gelb`

```
<p class="info gelb">Dieser Text ist formatiert.</p>
<p class="gelb info">Dieser Text ist formatiert.</p>
<p class="info headline gelb">Dieser Text ist formatiert.</p>
```

In älteren Browsern kommt es allerdings zu Darstellungsproblemen. Beispielsweise wendet der IE vor der Version 6 die definierte Regel auch auf die Elemente an, denen nur die zuletzt genannte Klasse zugewiesen ist (hier `.gelb`).

Die gezeigten Beispiele sind in der Datei klassen.htm auf der CD-ROM im Verzeichnis kap03 *gespeichert.*

Zur Unterscheidung der einzelnen Themenbereiche gibt es hier zusätzlich die Klasse `.abstand`*, die einen oberen Abstand (*`margin-top`*) von 80 Pixel zum vorherigen Element festlegt. Diese Klasse wurde gemeinsam mit der Klasse* `headline` *auf die untere Überschrift angewendet (*`<h1 class="abstand headline">…</h1>`*). Ausführliche Informationen über solche Kombinationen erhalten Sie in Kapitel 3.1.7.*

3.1.3 ID-Selektoren zur einmaligen Verwendung, ID für einmalige Auszeichnungen

Jedes Seitenelement kann über eine einmalig vergebene ID eindeutig identifiziert werden. Dabei wird dieser Name als Wert des `id`-Attributs in das jeweilige HTML-Element eingefügt. Im folgenden Beispiel soll das `div`-Element über die ID `box1` angesprochen werden. Dieser Name wird auf dieser Webseite kein zweites Mal verwendet, auch nicht von einem anderen Element.

Listing 3.14: CSS-Regel für eine ID

```
<div id="box1">Hier steht der Inhalt.</div>
```

Individualformat

Die Zuweisung erfolgt in der CSS-Regel mit dem ID-Selektor. Dieser besteht aus der Raute #, gefolgt von dem ID-Namen. Für dessen Notation gilt das Gleiche wie für Klassennamen.

Im folgenden Listing werden eine Höhe (`height`) und Breite (`width`) von jeweils 100 Pixel für den Seitencontainer mit der ID `box1` festgelegt.

Listing 3.15: ID-Stil bestimmt die Boxgröße

```
#box1 {
  height: 100px;
  width: 100px;
}
```

In Kapitel 7 bauen Sie Webseiten mit `div`*- und anderen Block-Elementen auf, für deren Platzierung eine eindeutige ID erforderlich ist.*

Damit der Bezeichner sich auf eine ID in einem bestimmten HTML-Element bezieht, geben Sie dieses zusätzlich vor der Raute an. Beispielsweise wird in Listing 3.16 die CSS-Regel `#lead` nur dann angewendet, wenn das `id`-Attribut einem Absatz `<p>` zugewiesen ist.

Listing 3.16: ID für ein bestimmtes HTML-Element

```
p#lead {
  font-family:Arial;
  font-size:24px;
}
[…]
<p id="lead">Diesem Text ist eine ID zugewiesen.</p>
```


IDs werden häufig gemeinsam mit JavaScript verwendet. Beispielsweise können Sie mit der Methode `getElementById()` *den Zugriff auf ein HTML-Element regeln und das Erscheinungsbild neu bestimmen. Auf der CD-ROM finden Sie ein Beispiel (*id_javascript.htm*) im Verzeichnis* kap03.

3.1.4 Universal-Selektor für alle Seitenelemente

Mit dem Universal-Selektor definieren Sie eine zentrale Deklaration für alle Seitenelemente. Diese Bezeichnung besteht nur aus dem Sternchen *, dem so genannten Joker oder Asterisk.

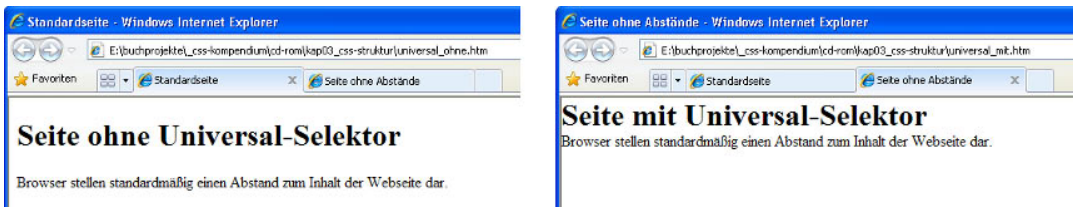
Beispielsweise eliminieren Sie sämtliche Innen- (`padding`) und Außenabstände (`margin`) aller vier Elementseiten mit Listing 3.17. Auf diese Weise verhindern Sie, dass Browser um einige Elemente einen Standardabstand einhalten. Der Universal-Selektor gilt also auch für das `body`-Element, dem die allgemeinen Seiteneigenschaften zugewiesen sind. Anschließend können Sie die gewünschten Abstände für ausgewählte Elemente neu bestimmen.

Häufig wird dieser Selektor auch bei der Kombination unterschiedlicher Selektoren als Platzhalter für eine bestimmte Hierarchiestufe verwendet. Ausführliche Informationen zu diesem Thema finden Sie in Kapitel 3.1.7.

Listing 3.17: Keine Abstände darstellen

```
* {
  margin:0;
  padding:0;
}
```

Abbildung 3.3: Abstände mit dem Universal-Selektor zurücksetzen (rechts)



Die gezeigten Beispiele finden Sie auf der CD-ROM: universal_ohne.htm *enthält keine CSS-Regel. Dagegen wird in* universal_mit.htm *der Universal-Selektor eingesetzt.*

3.1.5 Pseudo-Klassen und -elemente

Pseudo-Selektoren werden in Pseudo-Klassen und Pseudo-Elemente unterschieden:

Werden CSS-Regeln einem Element erst zugewiesen, nachdem es einen bestimmten Status bzw. Zustand erreicht hat, spricht man von einer **Pseudo-Klasse**. Beispielsweise kann das Style Sheet die Darstellung eines Hyperlinks dynamisch entsprechend den User-Aktionen verändern. Damit entstehen interaktive Rollover-Effekte. In diesem Kapitel wenden Sie unterschiedliche Pseudo-Klassen für Hyperlinks an.

Daneben gibt es die strukturellen Selektoren `:first-child` und `:last-child`, die ebenfalls kurz vorgestellt werden. Die Pseudo-Klasse `:lang`, mit der ein Style Sheet für eine bestimmte Sprache definiert werden kann, erläutert Kapitel 13.2. Informationen über die mit CSS2 eingeführten Pseudo-Klassen `:first`, `:left` und `:right` für die Wiedergabe als Papierausdruck finden Sie in Kapitel 16.2. Leider wird die Pseudo-Klasse `:not` noch nicht unterstützt und daher hier vernachlässigt.

Pseudo-Elemente beziehen sich dagegen auf einen bestimmten Teil eines HTML-Elements. Diesen Selektor-Typ verwenden Sie, wenn die CSS-Regel beispielsweise nicht einen kompletten Absatz, sondern nur die ersten Zeile formatieren soll (`:first-line`). Auch hierbei muss die Zeilenlänge und damit der zu formatierende Bereich zuerst im Browser ermittelt werden. Die Zeilenlänge ist von unterschiedlichen Parametern abhängig, wie etwa der Größe des Browserfensters oder der verwendeten Schriftgröße.

Fazit: Pseudo-Selektoren formatieren Elemente außerhalb des Elementbaums (vgl. Kapitel 3.1.7). Diese Bezeichner beginnen stets mit dem HTML-Element, auf das sich die Regel bezieht. Bei einer Pseudo-Klasse für Hyperlinks ist dies beispielsweise das `a`-Element, bei der ersten Zeile eines Absatzes das `p`-Element. Es folgen Doppelpunkt, Schlüsselbegriff und die üblichen Deklaration in geschweiften Klammern.

```
Element: Schlüsselbegriff { Eigenschaft: Wert; }
```

Pseudo-Klassen: Hyperlinks und Rollover-Effekte

Mit einem Klassen-Selektor bestimmen Sie das Erscheinungsbild von Verweisen zu noch nicht besuchten Seiten. Dabei ist zwischen folgenden Zuständen zu unterscheiden:

Zustände

- `:link`: Normalzustand; der User hat noch nicht auf diesen Link geklickt. Die verknüpfte Seite wurde noch nicht besucht.
- `:visited`: Besuchter Link; der User hat das Zieldokument schon einmal im Browser geladen.
- `:hover`: Angesteuerter Link; der Besucher bewegt den Mauszeiger über den Hyperlink.
- `:active`: Aktiver Link; der User klickt auf den Verweis. Sobald er die Maustaste wieder loslässt, verlässt er diesen Status.

Diese Pseudo-Klassen werden häufig in Navigationsleisten eingesetzt. Definieren Sie eindeutige CSS-Eigenschaften für die unterschiedlichen Zustände, unterstützen Sie die Usability bzw. die Benutzerfreundlichkeit Ihrer Seite, da sich die Besucher besser zurechtfinden. Pseudo-Klassen sind eine Alternative zu den mit JavaScript definierten Rollovern, bei denen jeweils unterschiedliche Bilder geladen und angezeigt werden. Da JavaScript im Browser ausgeschaltet werden kann und aus Sicherheitsgründen auch recht häufig ausgeschaltet wird, stellen diese CSS-Regeln eine sinnvolle Alternative dar und gewährleisten die Funktionalität definierter Hyperlinks.

`:focus` Zudem führt CSS2 die Pseudo-Klasse `:focus` ein. Diese bezieht sich auf mit der ⇆-Taste angesteuerte Verweise. Diese sind im Browser häufig durch einen gepunkteten Rahmen hervorgehoben. User erkennen dadurch, welcher Hyperlink gerade aktiviert ist, und können durch Drücken der Enter-Taste das Zieldokument aufrufen. Für barrierefreie Webseiten benötigen Sie diese Angabe. Auch aktivierte Formularelemente können mit `:focus` gekennzeichnet werden (vgl. Kapitel 12.2.4).

INFO

In alten Opera-Versionen kann ein Hyperlink nur über eine Zugriffstaste fokussiert werden. Diese geben Sie mit dem HTML-Attribut `accesskey` *an:*

```
<a href="#" accesskey="a">Hyperlink</a>
```

Formularfelder können dagegen auch hier mit der ⇆-Taste aktiviert werden.

Reihenfolge Da durch das Verhalten des Users die unterschiedlichen Pseudo-Klassen aufgerufen werden, wird die vorherige Darstellung entsprechend überschrieben. Für die Funktionalität ist deshalb die Reihenfolge wichtig, in der Sie die einzelnen Pseudo-Stile notieren. Im folgenden Beispiel sind alle Zustände korrekt definiert. Dabei verhindert der Selektor `text-decoration:none;` in der Deklaration des `a:link`-Selektors die standardmäßige Unterstreichung der verlinkten Zeichen im Browser. Diese Eigenschaft wird auf die übrigen Zustände vererbt (vgl. Kapitel 3.3).

Listing 3.18: Reihenfolge unterschiedlicher Pseudoanker

```
a:link {
  color: #FF0000;
  text-decoration: none;
}
a:visited {
  color: #FF00FF;
}
a:focus {
  color: #00F;
}
a:hover {
  color: #00F;
}
```

```
a:active {
  color: #339900;
}
```

Das Listing finden Sie auf der CD-ROM im Verzeichnis kap03*:* pseudo-klassen.htm.

Viele Browser setzen die meisten Pseudo-Klassen problemlos um. Allerdings funktioniert der `:focus`-Zustand in Opera nur mit dem Mauszeiger, nicht jedoch über die ⇆-Taste. Safari kennt diesen Zustand gar nicht und der Internet Explorer erst ab Version 8.

Wie Sie mit CSS eine Navigationsleiste definieren, erfahren Sie in Kapitel 10.

`:focus`, `:hover` und `:active` gelten nicht nur für Hyperlinks, sondern auch für andere Elemente, etwa Überschriften. Dies funktioniert im IE jedoch erst ab Version 7.

```
h1:hover { color: #00F; }
```

Tabelle 3.2 zeigt die Browserunterstützung der hier vorgestellten CSS-Klassen. Die folgenden Abschnitte erläutern die Pseudo-Klassen `:first-child` und `:last-child`.

Die mit CSS3 kommende Pseudo-Klasse `:nth-child()` stellt Kapitel 11.9 vor. Diese bezieht sich auf bestimmte Geschwisterelemente im hierarchischen Dokumentbaum und wird zurzeit nur von wenigen Browsern (Opera, Safari) unterstützt.

Tabelle 3.2: Referenz – Pseudo-Klassen

Selektor	CSS-Version	IE			FF			Safari			Opera		
		6	7	8	1	2	3	2	3	4	7	8	9
`a:link`	1	j	j	j	j	j	j	j	j	j	j	j	j
`a:visited`	1	j	j	j	j	j	j	j	j	j	j	j	j
`a:focus`	2	n	n	j	j	j	j	n	n	n	t	t	t
`a:hover`	2	t	j	j	j	j	j	j	j	j	j	j	j
`a:active`	1/2	t	t	j	j	j	j	j	j	j	j	j	j
`:first-child`	2	n	j	j	j	j	j	j	j	j	j	j	j
`:last-child`	2	n	n	n	j	j	j	j	j	j	n	j	j
`:nth-child()`	3	n	n	n	n	n	n	?	?	j	n	?	j

Pseudo-Klasse: Erstes Kindelement :first-child

Mit der Pseudo-Klasse `:first-child` formatieren Sie das erste Kindelement innerhalb des Strukturbaums (vgl. Kapitel 3.1.7). Haben Sie beispielsweise eine ungeordnete Aufzählung eingefügt (Listing 3.19), formatieren Sie mit diesem Klassen-Selektor das erste Listenelement. Damit `Element 1` in roten Zeichen dargestellt wird, geben Sie folgende CSS-Regel an:

```
li:first-child { color:red; }
```

Das zu formatierende Element steht also vor dem Selektornamen `:first-child`.

Listing 3.19: Ungeordnete Liste

```
<ul>
  <li>Element 1</li>
  <li>Element 2</li>
  <li>Element 3</li>
</ul>
```


Das Listing ist in der Datei first-child.htm *im Verzeichnis* kap03 *auf der CD-ROM gespeichert.*

Die Pseudo-Klasse `:first-child` *verursacht Darstellungsprobleme in Opera 5 und 6 und wird vom Internet Explorer erst ab Version 7 interpretiert umgesetzt.*

In Kapitel 3.1.7, Abschnitt »Kind-Selektor für unmittelbar untergeordnete Elemente (child)«, *bestimmen Sie mehrere zu formatierende Elemente gleichzeitig. Mit diesen und den anderen in diesem Kapitel beschriebenen, verschachtelten und kombinierten Selektoren sind präzisere Angaben möglich.*

Pseudo-Klasse: Letztes Kindelement :last-child

Die Pseudo-Klasse `:last-child` weist CSS-Formate dem letzten Element innerhalb der HTML-Struktur zu (vgl. Kapitel 3.1.7). Beispielsweise können Sie in einer Aufzählung das letzte Listenelement auswählen und formatieren:

```
li:last-child { color: red; }
```


Das Beispiel ist in der Datei last-child.htm *auf der CD-ROM im Verzeichnis* kap03 *gespeichert.*

Die beiden Pseudo-Klassen `:last-child` und `:first-child` können Sie auch gleichzeitig und mit unterschiedlichen Eigenschaften verwenden.

Pseudo-Elemente für Teilbereiche eines Elements

Die im Folgenden vorgestellten Pseudo-Selektoren formatieren nicht das komplette HTML-Element, sondern einen bestimmten Bereich davon.

Mit `:first-letter` formatieren Sie das erste Zeichen in einem Absatz, in einer Überschrift oder in einem vergleichbaren HTML-Element. Satzzeichen werden hierbei nicht berücksichtigt (s. Abbildung 3.4). `:first-letter`

`:first-line` bezieht sich dagegen auf die komplette erste Zeile. Wie viele Zeichen letztendlich damit formatiert werden, hängt von unterschiedlichen Faktoren ab (z. B. Fenster-, Schriftgröße). `:first-line`

In Listing 3.20 wird die erste Textzeile in größeren Zeichen als der übrige Text dargestellt, dessen Formatierung das `body`-Element bestimmt. Zudem wird das erste Zeichen groß und fett hervorgehoben. Die Umsetzung zeigt Abbildung 3.4.

Listing 3.20: Erstes Zeichen und erste Zeile formatieren

```
body {
  font: 12px Arial;
}

:first-letter  {
  font-size: 24px;
  font-weight: bold;
}
:first-line  {
  font-size: 14px;
}
```

Diese Pseudo-Elemente werden von allen modernen Browsern korrekt dargestellt.

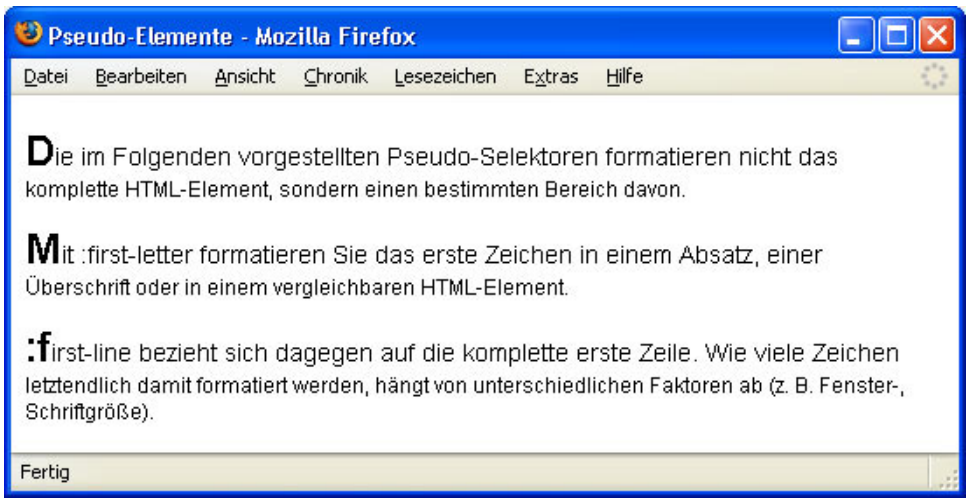


Abbildung 3.4: Zeichenformatierung mit `:first-letter` und `:first-line`

Mit `:before` fügen Sie einen bestimmten Inhalt vor einem Element ein, mit `:after` setzen Sie ihn dahinter. Der Inhalt wird mit dem `content`-Attribut in der Deklaration angegeben (vgl. Kapitel 13). Dabei kann es sich um einen `:before`, `:after`

Text, ein Bild oder um einen Zähler handeln. Auch öffnende und schließende Anführungszeichen können Sie mit den beiden Selektoren definieren.

Generierte Inhalte

Im folgenden Beispiel wird jede `h1`-Überschrift mit dem roten Text *Mitteilung:* eingeleitet. Dahinter wird die Grafik `bild.gif` eingebunden. Dabei wird die Angabe durch den Wert `url` eingeleitet. Die Bilddatei wird dahinter in runden Klammern angegeben. Listing 3.21 enthält zudem zwei Pseudo-Selektoren für die Klasse `.zitat`, wodurch die entsprechende Textstelle in Anführungszeichen (`open-quote` und `close-quote`) eingebunden wird. Abschließend fügt ein `:after`-Element einen Zähler (`counter(name)`) hinter einen mit dem `class`-Attribut `zaehler` ausgezeichneten Bereich. Die Browserdarstellung zeigt Abbildung 3.5.

Listing 3.21: Stilregeln für unterschiedliche Pseudo-Elemente

```
h1:before { content: "Mitteilung: "; color: red; }
h1:after { content: url(bild.gif); }
.zitat:before { content: open-quote; }
.zitat:after { content: close-quote; }
.zaehler:after { content: counter(name); }
```

Abbildung 3.5: Pseudo-Elemente im Browser

Der IE kennt `:before` und `:after` erst ab Version 8. Google Chrome und Safari zeigen keine Anführungszeichen an, stellen aber ansonsten die Anweisungen korrekt dar.

REF

Ausführliche Informationen über die Zeichenformatierung und weitere Formatierungsbeispiele finden Sie in Kapitel 5. CSS3 führt neue Pseudo-Selektoren ein, die in Kapitel 16.2 ausführlich dargestellt werden. Diese werden jedoch nur von wenigen aktuellen Webbrowsern interpretiert.

*Das Listing (*pseudo-elemente.htm*) ist auf der CD-ROM im Verzeichnis* kap03 *gespeichert.*

Tabelle 3.3: Referenz – Pseudo-Elemente

Selektor	CSS-Version	IE			FF			Safari			Opera		
		6	7	8	1	2	3	2	3	4	7	8	9
`:first-letter`	1	j	j	j	j	j	j	j	j	j	j	j	j
`:first-line`	1	j	j	j	j	j	j	j	j	j	j	j	j
`:before`	2	n	n	j	j	j	j	t	t	t	j	j	j
`:after`	2	n	n	j	j	j	j	t	t	t	j	j	j

3.1.6 Attributabhängige Selektoren

Enthält ein HTML-Element ein bestimmtes Attribut, können Sie ihm eine CSS-Regel zuweisen. Hierzu fügen Sie vor der CSS-Deklaration das entsprechende Attribut in eckige Klammern. Dabei spielt es keine Rolle, ob es sich um ein erforderliches oder um ein optionales Attribut handelt (vgl. Kapitel 3.3.2). Auch der zugewiesene Wert ist bei dieser allgemeinen Notation ohne Bedeutung.

```
[Attribut] { Eigenschaft: Wert }
```

Beispielsweise formatiert das folgende Listing die Zeichen aller Elemente mit einem `title`-Attribut rot (`color: #F00;`). Damit wird die im `body`-Bereich eingefügte Überschrift entsprechend eingefärbt.

Listing 3.22: Elemente mit `title`-Attribut formatieren

```
[title] { color: #F00; }
…
<h1 title="Beispielseite">Überschrift mit titel-Attribut</h1>
```

Achten Sie auf die korrekte Groß- und Kleinschreibung.

Notieren Sie vor dem Attribut-Selektor nur im Ausnahmefall den Universal-Selektor `*`, da einige Browser die CSS-Regel dann nicht mehr darstellen (vgl. Kapitel 8). Generell ist `[title]` jedoch identisch mit `* [title]`.

Der Internet Explorer interpretiert Attribut-Selektoren erst ab Version 7.

Die Angaben attributbedingter CSS-Formatierungen können Sie weiter verfeinern. Soll die Regel nur für Elemente mit einem bestimmten Wert gelten, geben Sie diesen folgendermaßen an:

```
[Attribut="Wert"] { Deklaration }
```

Wert bestimmen

Auf diese Weise formatieren Sie beispielsweise einen zentrierten Absatz:

Listing 3.23: Attribut mit spezifischem Wert formatieren

```
[align="center"] { margin: 50px; }
…
<p align="center">Dieser Absatz wird mit HTML zentriert.</p>
```

Enthält einen Wert

Manchmal sind einem HTML-Attribut mehrere Werte bzw. Informationen gleichzeitig zugeordnet, wie etwa bei einem Titel oder einem Sprachencode. Um die CSS-Regel diesem Element zuzuordnen, genügt es, wenn Sie einen dieser Wert-Begriffe nennen. Dabei muss hinter dem Attributnamen die Tilde ~ stehen. Zwischen den einzelnen Wert-Begriffen steht ein Leerzeichen.

```
[Attribut~="Wert1 Wert2 Wert3"] { Deklaration }
```

Das folgende Listing formatiert HTML-Elemente, dessen `title`-Attribut den Begriff `Beispiel` verwendet. Dabei wird die Regel nur auf den ersten Absatz `<p>` angewendet, nicht jedoch auf den zweiten. Hier steht nämlich ein weiteres Zeichen (die `2`) direkt hinter dem Begriff und verhindert daher die Übereinstimmung.

Listing 3.24: Wertspezifische Angabe

```
[title~="Beispiel"] { background-color: #06C; }
…
<p title="Das ist ein Beispiel 1">Zuweisung des Attribut-Selektors</p>
<p title="Das ist Beispiel2">Keine Zuweisung des Attribut-Selektors</p>
```

Erster Wert

Damit der Selektor einem Attribut zugewiesen wird, das mit einem bestimmten Wert beginnt, verwenden Sie das `^`-Zeichen. Dieser Attribut-Selektor wird mit CSS3 eingeführt.

```
[Attribut^="Wert1"] { Deklaration }
```

In folgendem Listing wird die Regel nur auf das erste und letzte HTML-Element angewendet.

Listing 3.25: Wertspezifische Angabe mit festgelegtem Anfang

```
[title^="Super"] { background-color: #6F9; }
…
<p title="Super, was man alles machen kann.">Zuweisung</p>
<p title="Das ist super">Keine Zuweisung</p>
<p title="Superbeispiel">Zuweisung</p>
```

Letzter Wert

Damit nur die Elemente formatiert werden, in denen ein Attribut mit einem bestimmten Wert endet, verwenden Sie folgende allgemeine Syntax. Dabei steht nach dem Attributnamen das `$`-Zeichen gemäß CSS3-Spezifikation.

```
[Attribut$="Wert"] { Deklaration }
```

Damit lassen sich prima unterschiedliche Hyperlinks formatieren und der User erkennt schnell, welcher Verweis ihn zu einer neuen URL oder zu bestimmten Dokumenten führt. Der erste Selektor formatiert alle Verweise zu HTM-Dokumenten, der zweite solche zu den Startseiten von DE-Domains. Damit werden die Hyperlinks im `body`-Bereich unterschiedlich dargestellt.

Listing 3.26: Hyperlinks formatieren

```
[href$=".htm"] { color: orange; }
[href$=".de"] { color: green; }
…
<p><a href="../index.htm">relativer Verweis</a></p>
<p><a href="http://www.susanne-rupp.de">absoluter Verweis</a></p>
```

Auch dieser und der folgende Selektor-Typ werden mit CSS3 eingeführt und bereits von allen aktuellen Browsern interpretiert.

Wert enthalten

Soll das HTML-Attribut einen Wert enthalten und möchten Sie dessen Position nicht näher bestimmen, verwenden Sie den *-Selektor. Mit dem folgenden Code bestimmen Sie die Hintergrundfarbe (`background-color`) der Elemente, die im `title`-Attribut die Zeichenfolge `Text` enthalten:

```
[title*="Text"] { background-color: orange; }
```

Damit weisen Sie dem ersten Absatz in Listing 3.27 die Hintergrundfarbe zu, während der zweite nicht verändert wird.

Listing 3.27: Attribut-Inhalte identifizieren

```
<p title="Das ist ein Text im Elementtitel">Zuweisung</p>
<p title="Hier ist der Begriff nicht enthalten">keine Zuweisung</p>
```


CSS3 enthält die Pseudo-Klasse `:not`*, mit der Sie die Elemente formatieren, bei denen ein angegebenes Attribut fehlt. Beispielsweise können Sie nach fehlenden* `alt`*-Attributen in Bildern suchen und diese durch einen Rahmen kennzeichnen:*

```
img:not([alt]) { border: 10px #F00 solid;}
```

Dieser Selektor wird bislang leider von keinem Browser unterstützt.

Zusammengesetzter Wert mit Bindestrich

Die Sprachangabe ist für barrierefreie Webseiten besonders wichtig. Diese integrieren Sie beispielsweise mit dem `lang`-Attribut in das jeweilige HTML-Element. Mit dem folgenden Listing können Sie schnell die Richtigkeit dieser Angabe in mehrsprachigen Dokumenten überprüfen und beispielsweise britisches, amerikanisches und australisches Englisch anders formatieren als deutsche Textstellen. Hierzu geben Sie einen senkrechten Strich, den so genannten Vergleichsoperator, hinter dem Attribut an, der festlegt, dass im HTML-Attribut entweder der exakte Wert verwendet wird oder dahinter ein Bindestrich steht.

Listing 3.28: Sprachen formatieren

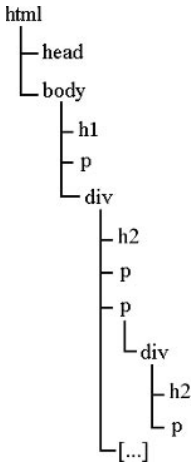
```
[lang|="en"] { color: red; }
[lang|="de"] { color: blue; }
…
<p lang="en-AU">Australian English - wird rot formatiert</p>
<p lang="en-US">American English - wird rot formatiert </p>
<p lang="en-GB">British English - wird rot formatiert </p>
<p lang="de">Deutsch - wird blau formatiert </p>
<p lang="de-CH">Schweizerdeutsch - wird blau formatiert </p>
```


Alle Beispiele finden Sie in der Datei attribut-selektoren.htm *im Verzeichnis* kap03 *auf der CD-ROM. Dabei wiederholt der Beispieltext den jeweiligen CSS-Code, den Sie damit leichter zuordnen können.*

3.1.7 Selektoren verschachteln und kombinieren

Unterschiedliche Selektoren können verschachtelt und miteinander kombiniert werden. Ausschlaggebend hierbei ist die Anordnung der einzelnen Elemente im (X)HTML-Dokument, das eine hierarchische Baumstruktur aufweist. In dieser Struktur sind die einzelnen Elemente miteinander verwandt und stehen in unterschiedlichen Beziehungen zueinander. Dadurch können Selektoren benachbarte oder verschachtelte bzw. untergeordnete Elemente gezielt ansprechen. Übergeordnete Elemente sind in CSS jedoch nicht selektierbar.

Listing 3.29 stellt einen Stammbaum dar. Darin bildet das `html`-Element die Wurzel (root) und ist Elternelement bzw. Vorfahre der Elemente `<head>` und `<body>`. Diese beiden Geschwisterelemente sind dagegen Kinder (*child*) oder Nachfahren (*descendant*) des `html`-Elements. Die daraus resultierenden Selektoren lernen Sie auf den nächsten Seiten näher kennen. Die Randspalte zeigt eine übersichtliche Skizze dieses Stammbaums mit den unterschiedlichen Hierarchie-Ebenen.



Listing 3.29: Struktur eines Webdokuments

```
01    <html>
02      <head>
03      </head>
04      <body>
05        <h1>  </h1>
06        <p>    </p>
07        <div id="inhalt">
08          <h2>  </h2>
09          <p>  </p>
10          <p>  </p>
11          <div id="sub-inhalt">
12            <h2>  </h2>
13            <p>   </p>
14          </div>
15        </div>
16     </body>
17    </html>
```

Nachfahren-Selektor für untergeordnete Elemente (*descendant*)

Einem bestimmten Seitenelement weisen Sie über einen kombinierten Selektor die gewünschte Stilregel zu. Dieser Nachfahren-Selektor listet übergeordnete Elemente durch ein Leerzeichen getrennt auf. Wichtig ist dabei die Nennung in der richtigen Reihenfolge entsprechend des Strukturbaums. Beispielsweise ist `<h3><b>…</b></h3>` nicht das Gleiche wie `<b><h3>…</h3></b>`. Im ersten Fall müsste der Selektor `h3 b` lauten, bei zweiten Beispiel dagegen `b h3`.

Damit in Listing 3.29 alle Absätze innerhalb des `div`-Elements mit der ID `inhalt` grau hinterlegt werden (Zeilen 9, 10 und 13), geben Sie folgenden CSS-Code an:

```
#inhalt p { background-color: #CCC; }
```

Diese Angabe genügt zur eindeutigen Identifikation der entsprechenden Absätze. Sie hätten auch weitere übergeordnete Elemente nennen können, was den Quelltext jedoch unnötig aufblähen und die Arbeit schnell unübersichtlich machen würde. Das folgende Listing ist also identisch mit dem vorherigen:

```
html body #inhalt p { background-color: #CCC; }
```

Hätten Sie dagegen nur den Element-Selektor `p` notiert, wären alle Absätze entsprechend hinterlegt:

```
p { background-color: #CCC; }
```


Trennen Sie die einzelnen Elemente im Selektor durch ein Komma, definieren Sie keine Hierarchie. Die CSS-Regel gilt dann für alle genannten Elemente und Selektoren (vgl. Kapitel 3.1.1). Beispielsweise formatiert folgende CSS-Regel die Absätze `<p>`*, das Element mit der ID* `inhalt` *sowie die mit* `<strong>` *hervorgehobenen Textbereiche. Die Reihenfolge, in der Sie die einzelnen Elemente nennen, ist beliebig.*

```
p, #inhalt, strong { background-color: #CCC; }
```

In der englischen Literatur wird der Nachfahren-Selektor als descendant combinator *bezeichnet. Er wurde bereits mit CSS1 eingeführt und ist auch im CSS Mobile Profile 1.0 verankert. Damit interpretieren auch Mobilgeräte entsprechende CSS-Regeln.*

Mit dem folgenden Selektor weisen Sie dagegen nur dem Absatz in der Codezeile 13 die Stilregel zu. Im folgenden Beispiel wird der entsprechende Text rot formatiert:

```
#inhalt #sub-inhalt p { color: red; }
```

Alternativ können Sie den Universal-Selektor * als Platzhalter einsetzen. Diese Notation wurde mit CSS2 eingeführt. Dabei bedeutet der Platzhalter, dass das zu formatierende Element **mindestens** eine Ebene darunter stehen muss.

```
#inhalt * p { color: red; }
```

Fügen Sie weitere Platzhalter ein, um die Position genauer festzulegen. In Listing 3.30 wird nur die Zeichenfolge `Wort` formatiert, obwohl das Element `<strong>` auch andere Zeichen hervorhebt.

Listing 3.30: Formatierung eines bestimmten Bereichs

```
p * * strong { color: red; }
…
<p>
  <strong>In diesem Absatz wird nur
    <strong>dieses
      <strong>Wort</strong>
    </strong> formatiert.
  </strong>
</p>
```

Auf der CD-Rom finden Sie alle Beispiele in der Datei nachfahren-selektor.htm *(Verzeichnis* kap03*).*

Kind-Selektor für unmittelbar untergeordnete Elemente (*child*)

Spezifischere Angaben als mit dem Nachfahren-Selektor sind mit dem Kind-Selektor möglich, der in der englischen Literatur als *child combinator* bezeichnet wird. Dieser Bezeichner wurde mit CSS2 eingeführt und spricht unmittelbare Nachfahren eines anderen Elements an.

Der Kind-Selektor trennt die einzelnen Elemente mit dem Größer-Zeichen > als Operator. Auch hier wird das übergeordnete Element zuerst genannt; es folgen die untergeordneten in ihrer hierarchischen Reihenfolge. Dabei wird das zuletzt genannte Element formatiert. Weitere Elemente des gleichen Typs, die diesem untergeordnet sind und damit auf einer anderen Verschachtelungsebene liegen, werden nicht formatiert.

Listing 3.31: HTML-Struktur

```
07        <div id="inhalt">
08          <h2>  </h2>
09          <p>  </p>
10          <p>  </p>
11          <div id="sub-inhalt">
12            <h2>  </h2>
13            <p>   </p>
14          </div>
15        </div>
```

Betrachten wir das eingangs erstellte Beispiel. Den jetzt wichtigen Ausschnitt aus Listing 3.29 zeigt Listing 3.31. Damit nur Absätze im `div`-Container `inhalt` rot formatiert werden (Zeilen 9 und 10) und Absätze im untergeordneten `div`-Element (Zeile 13) davon jedoch nicht betroffen sind, erstellen Sie folgenden CSS-Stil:

```
#inhalt > p { color: red; }
```


Der Child-Selektor wird von allen modernen Browsern, jedoch nicht von allen Mobilgeräten umgesetzt. Falls Sie noch für ältere Versionen des Internet Explorers optimieren, beachten Sie, dass Kind-Selektoren erst ab IE 7 unterstützt werden.

Geschwister- bzw. Nachbar-Selektor (*adjacent sibling*)

Der Nachbar-Selektor (*adjacent sibling selector*) bestimmt Elemente, die in einer bestimmten Reihenfolge im Code stehen und unmittelbar aufeinander folgen. In der Literatur wird er deshalb auch als Folgeelement-Selektor bezeichnet. Dieser mit CSS2.1 eingeführte Selektor-Typ verwendet als Operator das +-Zeichen.

Im folgenden Beispiel wird nur der zweite Absatz nach der Überschrift rot eingefärbt (Zeile 10), nicht jedoch der vorherige oder eine andere Textpassage im exemplarischen Quelltext (Listing 3.32).

```
h2 + p + p { color: red; }
```


Auch der Nachbar-Selektor wird vom Internet Explorer erst ab Version 7 interpretiert. Dieser Selektor wird von Mobilgeräten nicht unterstützt und ist nicht Bestandteil der CSS Mobile Profile 1.0.

Listing 3.32: HTML-Struktur

```
04      <body>
05        <h1>  </h1>
06        <p>    </p>
07        <div id="inhalt">
08          <h2>  </h2>
09          <p>  </p>
10          <p>  </p>
11          <div id="sub-inhalt">
12            <h2>  </h2>
13            <p>   </p>
14          </div>
15        </div>
16      </body>
```


Alle Beispiele finden Sie auf der DVD im Verzeichnis kap03 *zu diesem Kapitel:* nachfahren-selektor.htm, kind-selektor.htm, nachbar-selektor.htm.

Mit CSS3 sollen auch Geschwisterelemente selektierbar sein, die nicht unmittelbar aufeinander folgen. Auch hierbei handelt es sich um die gleichen Elternelemente. Zwischen den indirekt benachbarten Elementen steht die Tilde ~ (z. B. `h2 ~ p`*). Der Internet Explorer interpretiert diese Elemente ab Version 7. Ausführliche Informationen über CSS3 finden Sie im Arbeitspapier unter http://www.w3.org/TR/css3-selectors.*

Kapitel 11.9 stellt zudem die Pseudo-Klasse `:nth-child()` *vor, die sich auf bestimmte Geschwisterelemente im hierarchischen Dokumentbaum bezieht.*

3.2 Style Sheets speichern

Stilregeln können an drei unterschiedlichen Orten aufbewahrt bzw. gespeichert werden. Daraus ergeben sich dann auch verschiedene Möglichkeiten der Einbindung in das Webdokument.

- In einer separaten **CSS-Datei** mit der Endung *.css*:
  Eine CSS-Datei speichert zentrale Formate für unterschiedliche Webseiten und gewährleistet so ein einheitliches Erscheinungsbild. Ändern

Sie eine Stilregel, wird die Darstellung von allen damit verbundenen Webseiten automatisch angepasst. Dieser Stiltyp ist der wichtigste und wird wegen seiner Flexibilität und der schnellen Realisierung eines einheitlichen Erscheinungsbildes am häufigsten eingesetzt. Auf eine HTML-Seite können Sie mehrere CSS-Dateien anwenden.

- Im **head-Bereich der HTML-Seite**:

  Die hier gesammelten CSS-Stile formatieren nur das aktuelle Dokument. Häufig werden Stile intern definiert, die ein extern gespeichertes allgemeines Style Sheet ergänzen.

- Als **Inline-Stil** in einem HTML-Element:

  Der Stil wird mit dem `style`-Attribut in das öffnende HTML-Tag eingebunden und formatiert nur dieses. Damit sind diese Stile recht aufwändig zu ändern. Dieser Typ wird in der Regel nur verwendet, um allgemeine Stile für ein bestimmtes Element zu überschreiben. Inline-Stile weisen einzelnen Elementen ein individuelles Erscheinungsbild zu (vgl. Kapitel 3.3.3).

Diese unterschiedlichen Style-Sheet-Arten können in einem Dokument nebeneinander eingesetzt werden. Dabei können sich mehrere Regeln auch widersprechen. Kapitel 3.3 beschreibt, wie solche Konflikte im Webbrowser gelöst werden.

3.2.1 Separate CSS-Datei für flexiblen Einsatz

Speichern Sie häufig benötigte Stilregeln in einer separaten CSS-Datei, realisieren Sie schnell ein einheitliches Erscheinungsbild. Hierzu binden Sie in den einzelnen Webseiten einen entsprechenden Verweis auf diese zentrale Datei in den `head`-Bereich ein. Formatierungen müssen Sie dann nicht mehr in jedem einzelnen Web-Dokument ändern, sondern nur noch in der CSS-Datei. Die Webseiten werden dann automatisch mit den neuen Stilregeln angezeigt.

CSS-Datei speichern

Bei der CSS-Datei handelt es sich um ein einfaches Textdokument mit der Endung *.css*. Neben den einzelnen Stilregeln kann diese Datei auch Kommentare enthalten. Diese beginnen mit der Zeichenfolge `/*` und enden mit `*/`. Dazwischen steht der eigentliche Kommentar-Text mit relevanten Zusatzinformationen:

```
/* Das ist ein Kommentar. */
```

`@charset` Zusätzlich können Sie mit `@charset` die im Dokument verwendete Zeichenkodierung bestimmen. Verwenden Sie den Unicode-Zeichensatz `utf-8`, werden in der Regel alle benötigten Zeichen korrekt kodiert. Diese einmalige Information müssen Sie im Dokument als Erstes nennen.

```
@charset "utf-8";
```

Widerspricht der vom Webserver übermittelte HTTP-Header dieser Definition, verwendet das Wiedergabegerät die Serverkodierung für die CSS-Datei. Die HTML-Datei hat ihre eigene Kodierung (vgl. Kapitel 3.1).

Damit sieht der Inhalt einer CSS-Datei beispielsweise wie in Listing 3.33 aus. Für gute Übersicht stehen Selektoren und Deklarationen jeweils in einer separaten Zeile.

Listing 3.33: Beispielinhalt einer CSS-Datei

```
@charset "utf-8";
/* Code von S. Rupp */
* {
  margin: 0;    /* alle Außenabstände zurücksetzen */
  padding: 0;    /* alle Innenabstände zurücksetzen */
}
body {
  font: 100.1% Arial, Helvetica, sans-serif;
  background: #fff;
}
h1 {
  color: red;
  font-size: 1em;    /* relative Angabe */
}
p {
  color: black;
  font-size: 0.9em;    /* relative Angabe */
}
```


Externes Style Sheet mit HTML anfügen

Die Stilregeln werden erst visualisiert, wenn Sie auf Elemente einer Webseite angewendet werden. Dazu fügen Sie an einer beliebigen Stelle im `head`-Bereich des entsprechenden Webdokuments einen HTML-Verweis auf die CSS-Datei ein:

Listing 3.34: Relativer Verweis auf siteinterne CSS-Datei

```
<link href="../style.css" rel="stylesheet" type="text/css" />
```

`href` **= Hypertext-Referenz**

Das `href`-Attribut im `link`-Element enthält den Verweis auf die CSS-Datei. Hierbei kann es sich um eine relative (Listing 3.34) oder um eine absolute Verknüpfung (Listing 3.35) handeln, wie sie auch bei Bildern und Hyperlinks benutzt wird.

Listing 3.35: Absoluter Verweis auf siteexterne CSS-Datei

```
<link href="http://www.susanne-rupp.de/style.css"
 rel="stylesheet" type="text/css" />
```

Das Attribut `rel` stellt den Bezug zu einem `Stylesheet` mit dem dahinter angegebenen MIME-Typ `text/css` her.

Auf diese Weise integrieren Sie die gleiche CSS-Datei in beliebig viele Web-Dokumente und realisieren ein einheitliches Erscheinungsbild. Da Browser die CSS-Informationen nur einmal laden, sind die anschließend geöffneten Webseiten schneller sichtbar, als wenn Sie die Formatierung hierin jeweils aufs Neue durchführen. Durch das Auslagern der Style Sheets sind Design-Änderungen schnell durchgeführt. Soll die Schrift eines Absatzes beispielsweise nicht mehr schwarz, sondern blau dargestellt werden, müssen Sie nur die entsprechende Deklaration für den richtigen Selektor aktualisieren und die Datei neu abspeichern:

Listing 3.36: Stilregeln sind schnell geändert.

```
p {
  color: blue;    /* geänderte Angabe */
  font-size: 0.9em;    /* relative Angabe */
}
```


INFO

Die Regeln eines auf diese Weise verlinkten Style Sheets werden von allen Browsern umgesetzt. Daneben gibt es die `@import`*-Anweisung als Alternative. Diese CSS-Syntax funktioniert jedoch nicht einheitlich in allen Ausgabegeräten (vgl. Abschnitt @import-Anweisung als Browsersperre).*

Bevorzugtes und alternatives Style Sheet einbinden

Standard-Style-Sheet

Sie können mehrere CSS-Dateien in ein HTML-Dokument einbinden. Dabei benötigt jeder Verweis ein separates `link`-Element. Verweisen Sie auf mehrere Standard-Style-Sheets, ergänzen sich die meisten Formatanweisungen oder werden entsprechend dem Kaskadenprinzip angewendet. Standard-Style-Sheets werden teilweise auch als dauerhafte Style Sheets bezeichnet und von dem englischen Begriff *persistent* abgeleitet.

```
<link href="style1.css" rel="stylesheet" type="text/css" />
```

Sie können jedoch auch unterschiedliche, sich widersprechende Style Sheets für verschiedene Benutzerbedürfnisse erstellen. Unterschiedliche Kontraste, Farben oder Schriftgrößen zeichnen solche Lösungen aus. Dabei ist zwischen bevorzugten und alternativen Style Sheets zu unterscheiden.

HALT

Verweisen Sie auf Standard-Style-Sheets stets zuletzt im Code. Nur dann werden diese Regeln im Browser umgesetzt und durch die zusätzlich angegebenen Stil-Typen ergänzt.

Bevorzugtes Style Sheet

Ein angegebenes Standard-Style-Sheet wandeln Sie in ein bevorzugtes Style Sheet um, indem Sie das Attribut `title` im `link`-Element angeben. Dessen Wert besteht aus einer beliebigen Zeichenfolge, die den Stil möglichst treffend beschreibt (z. B. `Cool - Winter`). Diesen Text zeigen Endgeräte später im Menü zur Auswahl an (vgl. Abbildung 3.6). Neue Versionen des Internet Explorers kennzeichnen diese Style Sheets im Menü als STANDARD. Anwendungsprogramme setzen das als bevorzugt definierte Style Sheet nach dem

Laden der Webseite um. Dieser Style-Sheet-Typ wird auch als *preferred* bezeichnet. Haben Sie mehrere bevorzugte Style Sheets definiert, verwendet die Anwendung das zuerst definierte als Standard und bietet die anderen als Alternative an.

Listing 3.37: Verweis auf das bevorzugte Style Sheet

```
<link href="style2.css" rel="stylesheet" type="text/css"
 title="Cool - Winter" />
```

Alternativ-Style-Sheets

Sind Alternativ-Style-Sheets mit dem Dokument verknüpft, können User diese ebenfalls über das ANSICHT-Menü im Webbrowser auswählen (Abbildung 3.6). Statt der bevorzugten Formate werden die Seitenelemente nun mit den entsprechenden Alternativ-Einstellungen angezeigt. Alternative Style Sheets enthalten ebenfalls das `title`-Attribut im `link`-Element. Zudem müssen Sie den `rel`-Wert erweitern: Dieser muss bei alternativen Style-Sheet-Verweisen `alternate stylesheet` lauten.

Listing 3.38: Verweis auf ein alternatives Style Sheet

```
<link href="style3.css" rel="alternate stylesheet" type="text/css"
 title="Hot - Summer" />
```

Aktualisieren Sie die Webseite im Browser, zeigen einige Anwendungen (z. B. Firefox) anschließend statt des alternativen Style Sheets wieder die Formate des bevorzugten Style Sheets an.

Auf der CD-ROM enthält die Datei regel-einbinden.htm *drei unterschiedliche Verweise zu externen Style Sheets.* style1.css *ist das Standard-Style-Sheet,* style2.css *enthält die bevorzugten und* style3.css *die alternativen Regeln.*

DISC

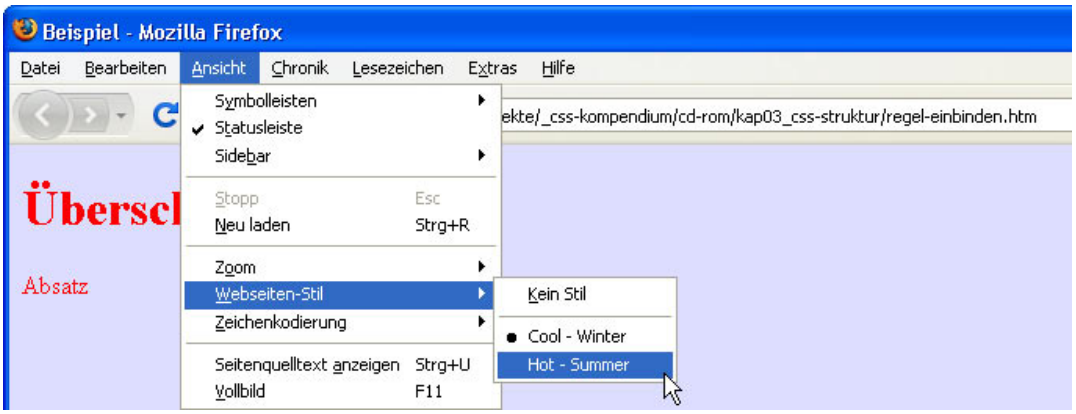


Abbildung 3.6: Stil in Firefox auswählen

Im Webbrowser können Sie Style Sheets auch komplett ausblenden und die Inhalte entsprechend ihrer logischen HTML-Struktur sehen. Mit dieser Einstellung können Sie gut die Syntax des Webdokuments überprüfen.

TIPP

Medienspezifische Style Sheets mit optimierten Formatierungen

Bildschirm vs. Druck

Nicht jedes Style Sheet ist für die Wiedergabe in jedem Medium geeignet. Ist hierin beispielsweise ein weißer Text auf einem dunklen Hintergrund definiert, sieht die Seite auf einem Bildschirm oft sehr schön aus. In unserem Beispiel heißt die Datei *screen.css*. Möchte der User die Informationen dann jedoch ausdrucken, bekommt er ein Problem, denn der Hintergrund wird in der Regel nicht mit ausgegeben. Weißer Text auf einem weißen Hintergrund…

Die Lösung des Problems ist recht einfach: Sie erstellen ein weiteres Style Sheet, das einen dunklen Text auf einem hellen Hintergrund setzt (*print.css*). Anschließend teilen Sie dem Browser mit, dass diese Regeln nur für den Drucker gelten. Hierzu binden Sie das `media`-Attribut in das `link`-Element ein. Als Wert verwenden Sie den Schlüsselbegriff `print` (vgl. Tabelle 3.4). Im gezeigten Beispiel wurde zudem die Schriftart von serifenloser Arial auf Times geändert, die im Ausdruck besser lesbar ist (vgl. Abbildung 3.7).

Dagegen ist das Style Sheet *screen.css* für die Bildschirmausgabe optimiert. Daher geben Sie `screen` als `media`-Wert an.

Listing 3.39: Style Sheets für unterschiedliche Wiedergabegeräte

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

Abbildung 3.7: Bildschirmseite und Druckvorschau (vorne)

DISC

Auf der CD-ROM finden Sie im Verzeichnis kap03 *die Datei* media_druck.htm, *in der unterschiedliche Style Sheets für die Bildschirmwiedergabe und den Ausdruck eingebunden sind* (screen.css, print.css).

Weitere Medien

Die Wiedergabe der Webseite kann auch noch durch weitere Medien erfolgen, wie etwa einem Mobiltelefon oder einem Fernseher. Wenn Sie medienspezifische Style Sheets hinterlegen, sollten Sie auch festlegen, wie die Seite von diesen Medien dargestellt wird. Dabei müssen Sie das Style Sheet nicht erneut einbinden. Vielmehr sollten Sie den entsprechenden Schlüsselbegriff aus Tabelle 3.4 in ein bereits vorhandenes `media`-Attribut einfügen. Dabei müssen die einzelnen Begriffe durch Kommata getrennt sein:

```
media="print, handheld"
```

Grundlegende Formate

Speichern Sie grundlegende Stilregeln, die für alle Medien gelten, in einer separaten CSS-Datei und verwenden Sie den Medientyp `all`. Wenn Sie kein `media`-Attribut angeben, gilt das Style Sheet für alle Ausgabegeräte.

Damit ist die folgende Verknüpfung …

```
<link href="style.css" rel="stylesheet" type="text/css" />
```

… identisch mit dieser:

```
<link href="style.css" rel="stylesheet" type="text/css" media="all" />
```

Tabelle 3.4: Mögliche Ausgabegeräte nach CSS2-Spezifikation

Wert	Ausgabegerät
`all`	Alle Medienarten
`aural`	Sprachausgabe
`braille`	Braillezeile (Der Seiteninhalt wird als Blindenschrift ausgegeben.)
`embossed`	Braille-Drucker (Der Seiteninhalt wird in ein Material gestanzt und abtastbar.)
`handheld`	PDAs, Smartphones, Mobiltelefone, u. ä.
`print`	Drucker
`protection`	Beamer, Overhead-Projektor
`screen`	Bildschirm
`tty`	Fernschreiber, Textbrowser (Lynx) und andere nicht-grafische Medien mit fester Zeichenbreite
`tv`	Fernsehgerät, Ausgabemedium mit geringer Auflösung und mangelhafter Scrollfunktion

Mit CSS3 soll der Medientyp `speech` *anstelle* `aural` *verwendet werden.* `aural` *gilt daher jetzt schon als* deprecated*, also als missbilligt. Die CSS-Spezifikation sieht zudem die* `@page`*-Anweisung für weitere Seiteneigenschaften vor (vgl. Kapitel 16.2). Auch ein erweiterter Ausdruck* `@media` *für die exakte Beschreibung der Anforderungen an das Ausgabegerät soll es geben.*

Trotz Angabe des `media`*-Typs funktioniert die Zuordnung nicht immer. Insbesondere bei Mobilgeräten kommt es häufig zu Problemen. Zudem interpretiert der Internet Explorer das* `media`*-Attribut erst ab Version 7 für* `screen` *und* `print` *weitgehend korrekt.*

Die im folgenden Abschnitt vorgestellte `@import`*-Anweisung ist mit weit mehr Fehlern behaftet. Eine entsprechende Übersicht der Browserunterstützung finden Sie unter http://www.codestyle.org/css/media/print-BrowserSummary.shtml.*

@import-Anweisung als Browsersperre

CSS-Syntax

Alternativ zum HTML-Verweis mit `<link>` können Sie externe Style-Sheet-Dateien auch mit `@import` in das Webdokument einbinden. Bei dieser @-Regel handelt es sich um eine CSS-Syntax, die früher verwendet wurde, wenn man die Stilregeln vor dem Netscape Navigator 4.x verbergen wollte. Dieser mittlerweile unbedeutende Browser konnte nämlich keine CSS-Formate interpretieren und zeigte den Code für die Formatierung neben anderen Seiteninhalten an. Mit diesem Trick wurde dann wenigstens der normale Seiteninhalt sauber dargestellt.

Heute wird die `@import`-Anweisung primär verwendet, um ein Style Sheet in eine CSS-Anweisung einzubinden und diese miteinander zu verschachteln. Diese Formatangaben können entweder im `head`-Bereich der Webseite oder in einer externen CSS-Datei gespeichert sein.

In CSS-Datei

In einem Style Sheet muss die `@import`-Anweisung stets über allen anderen CSS-Regeln stehen. Darüber darf also nur die `@charset`-Regel mit der Nennung der Zeichenkodierung stehen. Zudem können Sie mehrere Import-Direktiven untereinander einfügen. Allerdings darf die Anweisung nicht innerhalb einer anderen @-Anweisung stehen.

Die Syntax in einer separaten CSS-Datei verdeutlicht Listing 3.40. Dabei kann eine CSS-Datei entweder wie in Zeile 1 oder mit dem Schlüsselwort `url` und dem Dateinamen in Klammern angegeben werden (Zeile 3). Das Ausgabemedium notieren Sie bei Bedarf mit einem Leerzeichen dahinter (Zeile 2 und 3). Weitere Medien fügen Sie durch Komma getrennt hinzu. Anders als bei der HTML-Syntax wird hier kein `media`-Attribut benötigt. Jede `@import`-Anweisung endet mit einem Semikolon.

Listing 3.40: Verschiedene `@import`-Anweisungen

```
01    @import "dateiname1.css";
02    @import "dateiname2.css" screen, projector;
03    @import url("dateiname3.css") print;
```

In HTML-Datei

Die CSS-Datei binden Sie mit dem `style`-Element in den `head`-Bereich einer HTML-Seite. Ausführliche Informationen finden Sie in Kapitel 3.2.2. Dabei können Sie den Medientyp entweder mit HTML, also im `style`-Element (Listing 3.41), oder in der CSS-Syntax bestimmen (Listing 3.42).

Listing 3.41: Medientyp mit HTML festlegen

```
<style type="text/css" media="screen, projector">
  <!--
    @import url("screen.css");
  -->
</style>
```

Listing 3.42: Medientyp mit CSS festlegen

```
<style type="text/css">
  <!--
    @import url("screen.css") screen, projector;
  -->
</style>
```


Fügen Sie ein Style Sheet in eine CSS-Datei ein, sind die hier definierten Regeln den bereits bestehenden untergeordnet und ergänzen diese lediglich (Abbildung 3.8). Weitere Informationen über dieses Kaskadenprinzip erhalten Sie in Kapitel 3.3.

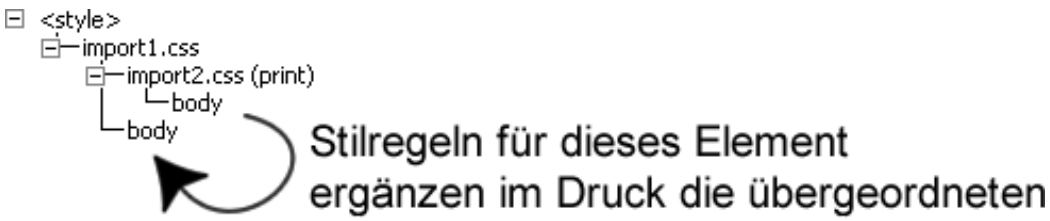


Abbildung 3.8: Kaskade: Stilregeln für gleiche Elemente

Die `@import`*-Anweisung wird nahezu von allen Browsern unterstützt. Allerdings gibt es bei älteren Versionen des Internet Explorers Probleme bei einigen Medientypen. So unterstützt IE 5 nur* `screen` *und* `all`*.*

Auch wenn dies im CSS Mobile Profile 1.0 vorgesehen ist, gibt es Probleme bei zahlreichen Mobilgeräten. Daher sollten Sie Style Sheets für den Medientyp `handheld` *besser mit dem HTML-Element* `link` *einbinden (vgl. Abschnitt »Externes Style Sheet mit HTML anfügen«).*

Kapitel 8 stellt neben dem `media`*-Attribut weitere Browser-Hacks vor, mit denen Sie Darstellungsprobleme lösen.*

3.2.2 Style Sheet im <head> eines HTML-Dokuments

Mit dem `style`-Element integrieren Sie unterschiedliche CSS-Stilregeln in den `head`-Bereich einer HTML-Seite. Diese Stilregeln gelten dann nur in diesem Dokument. Solche dokumentinternen Definitionen sollten Sie nur verwenden, um allgemeine extern gespeichert CSS-Stile zu überschreiben oder zu ergänzen. Der Aktualisierungsaufwand ist bei dokumentinternen Stilregeln recht aufwändig, da Änderungen in jeder einzelnen Webdatei durchgeführt werden müssen.

Die Regeln in einem internen Style Sheet stehen zwischen dem öffnenden und schließenden `style`-Element. Dabei sollten Sie diesen Quelltext als HTML-Kommentar einbinden (`<!—Kommentar -->`) und damit vor alten Browsern verbergen, die das `style`-Element nicht kennen und CSS nicht interpretieren. Die Regeln selbst unterscheiden sich nicht von denen in einer externen Datei.

Listing 3.43: Style Sheet im `head`-Bereich

```
<head>
 <style type="text/css">
  <!--
  body {
    font-family: Arial;
    color: #FFF;
    background: #003;
  }
  h1, h2{
    color: red;
  }
  -->
 </style>
 […]
</head>
```

Innerhalb des `head`-Bereichs können Sie mehrere Style Sheets definieren, was für die Angabe unterschiedlicher Medientypen sinnvoll sein kann. Dabei geben Sie das `media`-Attribut mit dem gewünschten Schlüsselbegriff im `style`-Element an:

Listing 3.44: Internes Style Sheet mit Ausgabemedium

```
<style type="text/css" media="print">
<!--
 body {
   font-family: Arial;
   color: #000;
   background: #FFF;
 }
-->
</style>
```

Alternativ können Sie den Medientyp innerhalb des Style Sheets auch mit `@media` bestimmen. Innerhalb des gleichen `style`-Bereichs sind also mehrere Medienstile möglich. Damit stehen die entsprechenden Stilregeln in geschweiften Klammern:

Listing 3.45: Medienspezifische Regel

```
@media print, handheld {
  body {
    font-family: "Times New Roman", Times, serif;
     color: #000;
 }
}
```

Auch dokumentinterne Style Sheets können Sie als bevorzugt oder alternativ kennzeichnen. Integrieren Sie `rel`*- und* `title`*-Attribut mit der gewünschten Beschriftung in das* `style`*-Element (vgl. Kapitel 3.2.1, Abschnitt »Bevorzugtes und alternatives Style Sheet einbinden«).*

Den in diesem Kapitel präsentierten Beispiel-Code finden Sie auf der CD-ROM im Verzeichnis kap03*:* internes_stylesheet.htm.

3.2.3 Inline-Stile für einzelne Elemente

Bei einem Inline-Stil wird der Selektor nicht angegeben, sondern Eigenschaft und Wert werden mit dem `style`-Attribut in das öffnende Tag integriert. Dieser Stiltyp wird hauptsächlich verwendet, wenn allgemeine, bereits angewendete Stile überschrieben bzw. außer Kraft gesetzt werden sollen (vgl. Kapitel 3.3). Da die Formatierung jedem Element einzeln zugewiesen wird, sind Inline-Stile besonders aufwändig zu aktualisieren.

Die allgemeine Notation verdeutlicht Abbildung 3.9, wobei auch hier mehrere Deklarationen hintereinander definiert werden können.

Abbildung 3.9: Aufbau eines Inline-Stils

Möchten Sie beispielsweise die auf der Seite verwendete Schriftart und deren Farbe direkt dem `body`-Element zuweisen, fügen Sie die Eigenschaften `font-family` und `color` mit dem `style`-Attribut in das HTML-Tag ein (Listing 3.46). Analog dazu formatieren Sie einzelne Textstellen innerhalb von Absätzen (`<p>`), Überschriften (`<h1>`, `<h2>`,..., `<h6>`), `span`-, `div`- oder andere, bereits vorhandene Elemente.

Listing 3.46: Inline-Stil im `body`-Element

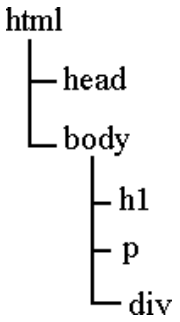
```
<body style="font-family: Arial; color: #000;">
    [Seiteninhalt]
</body>
```


Platzieren Sie in die Nähe von Inline-Stilen einen Kommentar, können Sie spätere Aktualisierungen mit der Suchfunktion im Editor leichter durchführen. Eine solche Kennzeichnung enthält also immer den gleichen Text. Ein HTML-Kommentar sieht beispielsweise folgendermaßen aus:

```
<!— Inline-CSS folgt… -->
```

3.3 Vererbung, Kaskade und Konflikte

Vererbung



Eigenschaften übergeordneter Elemente werden häufig auf untergeordnete und verwandte Elemente vererbt bzw. weitergegeben. Diese Information ist in den einzelnen Referenztabellen in der Spalte *vererbbar* eingetragen. Den dabei berücksichtigten Strukturbaum mit den unterschiedlichen Element-Hierarchien erläutert Kapitel 3.1.7.

Ist beispielsweise die Schriftart *Arial* für das übergeordnete `body`-Element definiert, werden auch Überschriften oder Absätze in dieser Schrift dargestellt. Da Sie die CSS-Regel nicht erneut für diese Elemente oder andere Selektoren definieren müssen, vermeiden Sie unnötigen Quelltext, und das Style Sheet bleibt übersichtlicher.

Dabei müssen Sie nur für die Elemente eine neue CSS-Regel definieren, denen Sie andere Attribute zuweisen möchten. Das ist etwa dann der Fall, wenn Sie einen Absatz in einer *Times* statt in *Arial* darstellen möchten. Die allgemeine Regel wird damit durch die spezifische überschrieben. Zu diesem Kaskadenprinzip kommen wir gleich.

`inherit`

Nicht alle Attribute werden auf alle Elemente vererbt. Ist beispielsweise eine Schriftgröße für das `body`-Element bestimmt, wird diese zwar an Absätze, nicht jedoch an Überschriften vererbt. Vielmehr stellen Browser diese weiterhin etwas größer dar und bilden damit die zugrunde liegende logische Struktur ab. Erstellen Sie nun eine separate CSS-Regel für das `h1`-Element, können Sie entweder einen separaten Wert in der Deklaration angeben oder mit dem Wert `inherit` die Vererbung erzwingen. Mit dem folgenden Code übernimmt die Überschrift `h1` also den entsprechenden Wert des übergeordneten Elements:

```
h1 { font-size: inherit; }
```


Leider funktioniert dies im Internet Explorer erst ab Version 8. In älteren Versionen bleiben Größenunterschiede für Überschriften und Absätze erhalten. Hier können Sie das Problem nur lösen, indem Sie die Schriftgröße der Überschriften mit einem separaten Wert angeben.

Auch bei der Übernahme von Abständen und Rändern kann es hier zu Problemen kommen (vgl. Kapitel 8*).*

In standardkonformen Browsern werden relative Schriftgrößen kumulativ auf untergeordnete Elemente angewendet. Legen Sie beispielsweise die Größe von Absätzen und `span`*-Elementen mit* `90 %` *fest, fällt die Schriftgröße entsprechend gekennzeichneter Inline-Elemente kleiner aus: Statt 90 % beträgt hier die Zeichengröße nur 81%, bezogen auf das übergeordnete Element.*

Listing 3.47: Vererbung der Zeichengröße

```
p, span { font-size: 90%; }
…
<p>In diesem Text <span>sind diese Zeichen kleiner</span>.</p>
```

Weitere Informationen über unterschiedliche Größen und Einheiten finden Sie in Kapitel 3.4.2.

Der Wert `inherit` *ist auf fast alle Attribute anwendbar.*

Bestimmen Sie grundlegende Formate zentral für das `html`- *oder* `body`*-Element. Damit diese für alle Elemente gelten, verwenden Sie den Universal-Selektor *. Damit werden etwa Außen- und Innenabstände zurückgesetzt (vgl. Kapitel 3.1.4). Auch Angaben über Höhe, Ausschnitt oder den Rahmen eines Elements werden nicht an die ihm untergeordneten weitergegeben.*

Informationen zur Vererbung finden Sie im jeweiligen Kapitel. In der Literatur wird für die Vererbung auch der englische Begriff inheritance *verwendet.*

Die Vererbung dokumentiert das Dokument vererbung.htm *im Verzeichnis* kap03 *auf der CD-ROM.*

Kaskade

Auf ein Element können unterschiedliche Selektoren gleichzeitig zugreifen und dessen Formatierung bestimmen. Diese zugewiesenen CSS-Regeln können sich jedoch widersprechen. Listing 3.48 weist dem Absatz zwei unterschiedliche Schriften zu: Einmal die *Arial* über das `body`-Element und dann noch die *Times* über das `p`-Element.

Listing 3.48: Diese Regeln widersprechen sich.

```
body { font-family: Arial, Helvetica, sans-serif; }
p { font-family: "Times New Roman", Times, serif; }
```

Doch woher weiß der Browser, wie er das Element formatieren und anzeigen soll? CSS2.1 sieht zur Lösung dieses Konfliktes die Kaskade vor, bei der Deklarationen auf mehreren Stufen überprüft werden.

Cascading Style Sheets: Der Begriff Kaskade kommt vom italienischen *cascare* und bedeutet *fallen*. Im Allgemeinen versteht man hierunter einen Wasserfall, der über mehrere Stufen plätschert. Und genauso verhält es sich bei der Konfliktlösung im Ausgabemedium bzw. im Webbrowser. Dabei wird die Deklaration in vier Stufen mit unterschiedlichen Kriterien überprüft. Ist eine Kaskade durchlaufen, wird auf der nächsten geprüft. Sind dagegen bestimmte Kriterien erfüllt, werden Deklaration und Stilregel auf ein Element angewendet und die Prüfung abgebrochen.

3.3.1 Stufe 1: Zuordnung Regel–Element

Das vom Server abgerufene Dokument wird im Browser (bzw. Ausgabegerät) eingelesen und analysiert. Durch dieses Parsen prüft der Browser die Syntax und baut die Baumstruktur auf. Dabei werden eindeutige Formatanweisungen auf entsprechende Elemente angewendet.

Enthält das Style Sheet Syntaxfehler, ist die Regel ungültig. Ist die Syntax eines Selektors innerhalb einer Gruppe falsch, wird die Deklaration auch bei den übrigen Selektoren ignoriert. Im folgenden Beispiel ist das Element `h2` *nicht korrekt geschrieben:*

```
h1, 2h, p { font-family: arial; }
```

Das ist die Theorie. Die Praxis sieht jedoch etwas anders aus, denn viele Browser wenden die Stilregel dennoch auf die übrigen Elemente an (z. B. IE bis Version 7). Dennoch sollten Sie auf korrekten, also validen CSS-Code achten. Unter http://jigsaw.w3.org/css-validator/ können Sie diesen überprüfen (vgl. Kapitel 18).

Sind alle Stile angewendet und keine Konflikte vorhanden, wird die Webseite im Browser angezeigt. Ansonsten wird Kriterium 2 überprüft:

3.3.2 Stufe 2: Ursprung des Style Sheets ermitteln

Gibt es für ein Element mehrere anzuwendende Stilregeln, wird auf dieser Stufe deren Ursprung ermittelt. Dabei gibt es drei unterschiedliche Quellen mit verschiedenen Prioritäten. Die folgende Auflistung nennt Style Sheets mit abnehmendem Einfluss:

- **Autoren-Style-Sheet**: Diese CSS-Regeln erstellen Sie als Webentwickler. Ein solches Style Sheet ist entweder in die Webseite selbst eingebunden oder in einer separaten CSS-Datei gespeichert (vgl. Kapitel 3.2).
- **Benutzer-Style-Sheet**: Benutzer können eigene Style Sheets erstellen und als CSS-Datei in den Browser einbinden. Damit werden diese Regeln auf alle Webseiten angewendet.

  Im Internet Explorer 8 geht dies beispielsweise über EXTRAS / INTERNETOPTIONEN. Im Register ALLGEMEIN drücken Sie dann auf die Schaltfläche BARRIEREFREIHEIT (Abbildung 3.10). In Opera wählen Sie dagegen EXTRAS / EINSTELLUNGEN / ERWEITERT / INHALTE / DARSTELLUNGS-OPTIONEN (vgl. Abbildung 3.11).

  Hinsichtlich der Barrierefreiheit bieten Benutzer-Style-Sheets enorme Vorteile, da User beispielsweise den Kontrast der Webseiten selbst bestimmen können. Auch Spielereien auf der Programmoberfläche, wie geänderte Scrollbalken, können hier verhindert und mit geeigneten `!important`-Regeln überschrieben werden (vgl. Kapitel 15.3).

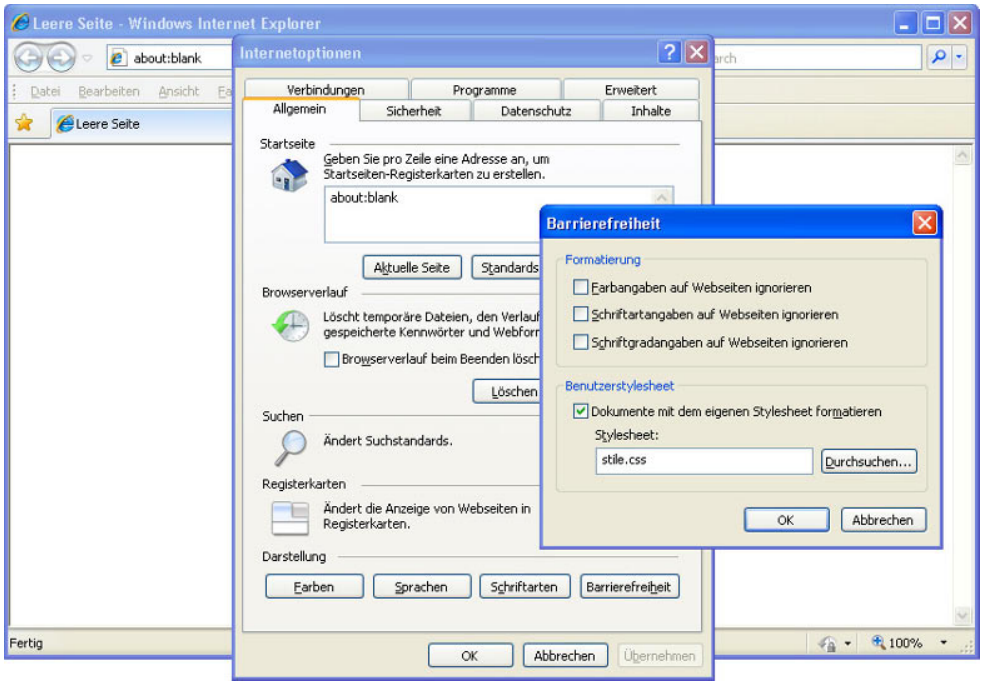


Abbildung 3.10: Benutzer-Style-Sheet im IE 8 angeben

- **Browser-Style-Sheet, Regeln des Wiedergabemediums**: Fehlen Formatierungsregeln, werden die Standardwerte (*default*) des Browsers bzw. des Anwenderprogramms verwendet. Ansonsten werden diese Werte entsprechend überschrieben.

Autoren-Style-Sheets haben die höchste Priorität und überschreiben die Regeln des Benutzer-Style-Sheets. Beide Quellen ersetzen zudem die rangniedrigeren Regeln des Anwenderprogramms.

Dabei werden nur widersprüchliche Angaben überschrieben. Miteinander vereinbare Formatierungen ergänzen sich dagegen. Angenommen ein Autoren-Style-Sheet legt für Absatz die Schrift `Arial` fest und ein Benutzer-Style-Sheet sieht hierfür die Farbe `blau` vor, dann zeigt der Browser einen blauen Text in der Schrift Arial.

`!important`

Eine Ausnahme bei diesen Gewichtungsregeln stellt das Konstrukt `!important` dar, das Anwendungsprogramme ebenfalls berücksichtigen müssen. Dieses kennzeichnet wichtige Darstellungsregeln und steht direkt hinter der entsprechenden CSS-Eigenschaft. Das abschließende Semikolon steht hinter dem Konstrukt:

```
Selektor { Eigenschaft: Wert !important; }
```

Markiert `!important` eine Regel sowohl im Autoren- als auch im Benutzer-Style-Sheet, wird ab CSS2.1 die Regel des Benutzer-Style-Sheets angewendet. Damit haben die Bedürfnisse der Benutzer Vorrang. (In CSS1 war dies noch umgekehrt.) Damit sieht die Hierarchie mit abnehmender Gewichtung folgendermaßen aus:

- Benutzer-Style-Sheet **mit** `!important`
- Autoren-Style-Sheet **mit** `!important`

- Autoren-Style-Sheet **ohne** `!important`
- Benutzer-Style-Sheet **ohne** `!important`
- Browser-Style-Sheet

Die mit `!important` gekennzeichneten Regeln überschreiben also andere und sind damit höher gewichtet. Damit werden einzelne Deklarationen eingefügt. Beinhaltet die Regel weitere Deklarationen ohne Konstrukt, interpretieren die meisten Browser diese ebenfalls wie `!important`.

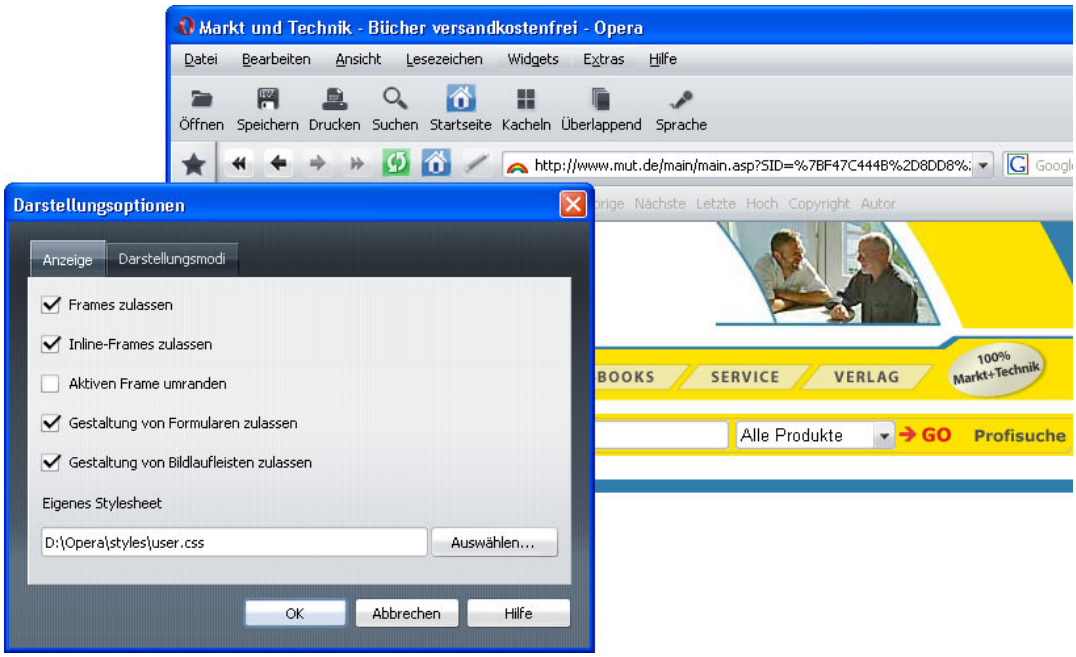
In Listing 3.49 ist der Überschrift zunächst die Farbe Blau zugewiesen. Diese wird durch die Klasse .rot jedoch überschrieben. Da `!important` hinter der Deklaration des Elements `h1` eingefügt ist, wird der Text dennoch blau angezeigt. Das Beispiel finden Sie auf der CD-ROM (*important.htm*).

Listing 3.49: `!important` überschreibt andere Regeln

```
h1 { color: blue !important; }
.rot{color: red;}
…
<h1 class="rot">Das ist eine Überschrift</h1>
```

Kann das Anwendungsprogramm nun alle Elemente formatieren und die Stilregeln zuordnen, wird die Webseite dargestellt und die Prüfung abgebrochen. Ansonsten wird Kriterium 3 überprüft.

Abbildung 3.11: Style Sheet in Opera einbinden



3.3.3 Stufe 3: Bestimmung der Selektor-Spezifität

Sind mehrere CSS-Regeln einem Element zugeordnet, sortiert das Anwendungsprogramm diese entsprechend ihrer Spezifität. Darunter versteht man eine Gewichtung der einzelnen Selektor-Typen, die die Besonderheit des jeweiligen Bezeichners ausdrückt und eine Rangfolge ermöglicht. Dabei

erhält ein mit dem `style`-Attribut eingebundener Inline-Stil stets den höchsten Rang und überschreibt andere Formatanweisungen für das entsprechende Element.

Die Wertigkeit eines Selektors setzt sich aus einer Kette von vier Ziffern zusammen. Hierfür werden die vier Buchstaben a bis d als Platzhalter eingesetzt:

a – b – c – d

Dabei ergibt sich die Zahlenfolge aus der Anzahl unterschiedlicher Selektor-Typen. Ist ein solcher Selektor vorhanden, nimmt der entsprechende Platzhalter den Wert 1 an.

a = Inline-Stile (`style="…"`)

b = ID-Attribute im Selektor (`#id`)

c = andere Attribute (Klassen) und Pseudo-Klassen (`.klasse`, `element.klasse`, `:pseudoklasse`)

d = Anzahl der Elementnamen und Pseudo-Elemente (z. B. `p`, `h1`, `div`, `:first-line`)

Je höher der ermittelte vierstellige Wert, desto spezifischer ist die CSS-Regel. Regeln mit spezifischen Selektoren überschreiben allgemeine. Anwendungsprogramme setzen also die Regeln mit höchster Spezifität um.

In Listing 3.50 sind für eine Überschrift `h1` sowohl ein Element- als auch ein Klassenstil definiert. Während die erste Stilregel den Text blau färbt, formatiert ihn die Klasse rot. Berechnen Sie nun die Wertigkeit der beiden Stilregeln, können Sie entscheiden, welcher Stil höherwertig ist und von den Browsern umgesetzt wird. Die jeweilige Wertigkeit steht im Listing als Kommentar hinter der entsprechenden Stilregel. Dabei erhält der Element-Selektor `h1` einen Wert für d und für die Klasse `.rot` ergibt sich ein c-Wert von 1. Damit ist die Klasse mit 0010 höherwertig als das Element mit 0001 und der Text wird rot dargestellt.

Listing 3.50: Errechnung der Wertigkeit

```
h1 { color: blue; }    /* a=0, b=0, c=0, d=1 */
.rot{ color: red; }    /* a=0, b=0, c=1, d=0 à wird angewendet */
…
<h1 class="rot">Das ist eine Überschrift</h1>
```

Auch hier werden nur widersprüchliche Angaben überschrieben. Miteinander vereinbare Formatierungen ergänzen sich dagegen. Wäre für das `h1`-Element zusätzlich die Schriftart `Arial` festgelegt, würde der Text in Arial und rot erscheinen.

Listing 3.51: Sich ergänzende Attribute

```
h1 { font-family: Arial }
.rot { color: red; }
```

Bei einer Selektor-Kombination analysieren Sie jede einzelne Stilregel. In der linken Spalte von Tabelle 3.5 sind hierzu alle Selektoren aus Listing 3.52 aufgeführt. Dabei dürfen Sie auch Inline-Selektoren, die mit dem `class`-Attribut in ein Element eingebunden sind, nicht vergessen. In den Spalten rechts davon tragen Sie dann die Werte a bis d ein. Gilt eine Klasse nur für ein bestimmtes Element (`a.verweis`), zählen Sie sowohl das Element (d = 1) als auch die Klasse (c = 1). Analog dazu ergeben sich aus einer Pseudo-Klasse für das `a`-Element ein Wert für die Ziffer c und einer für d. `a:link` entspricht also der Wertfolge 0011. Auch bei Pseudo-Elementen wird doppelt gezählt. In Tabelle 3.5 entspricht die Anordnung der einzelnen Selektoren deren Wichtigkeit.

Listing 3.52: Kombinierte Selektoren und ihre Wertigkeiten

```
#beispiel a.verweis { color: orange; }   /*  a=0, b=1, c=1, d=1  */
#beispiel li a { color: black; }   /*  a=0, b=1, c=0, d=2  */
a:link { color: red; }   /*  a=0, b=0, c=1, d=1  */
li a { color: green; }   /*  a=0, b=0, c=0, d=2  */
.verweis { color: blue; }   /*  a=0, b=0, c=1, d=0  */
p { color: green; }   /*  a=0, b=0, c=0, d=1  */
…
<p id="beispiel">Das ist ein <a href="#" class="verweis">Hyperlink</a>
<span style="color: #00F;">Text</span> in einem Absatz. </p>
```


Bei einer Selektor-Gruppe sind mehrere Elemente zusammengefasst. Zwischen den einzelnen Elementen steht ein Komma. Da Sie diese auch separat notieren können, zählt die gesamte Gruppe nur einmal.

`h1, h2, p { … }` *erzielt damit eine Wertigkeit von 0001.*

Tabelle 3.5: Wertigkeit errechnen

Selektor (Kombination, Inline-Stil)	a	b	c	d
`class="verweis"`	1	0	0	0
`#beispiel a.verweis`	0	1	1	1
`#beispiel li a`	0	1	0	2
`a:link`	0	0	1	1
`li a`	0	0	0	2
`.verweis`	0	0	1	0
`p`	0	0	0	1

Das gezeigte Beispiel finden Sie auf der CD-ROM im Verzeichnis kap03. *Öffnen Sie die Datei* spezifitaet.htm *und überprüfen Sie die Umsetzung der einzelnen Stile im Webbrowser.*

Kann die Anwendung keine eindeutige Reihenfolge ermitteln und ist damit mindestens zwei Elementen die gleiche Spezifität einzuräumen, wird Kriterium 4 überprüft. Ansonsten wird die Webseite im Browser angezeigt.

3.3.4 Stufe 4: Reihenfolge der Stilregeln

Innerhalb der Kaskade bildet die Position der Stilregel das letzte Prüfkriterium. Dabei überschreibt die zuletzt in einem Style Sheet definierte Regel die vorherige. In Listing 3.53 widersprechen sich die beiden gleichzeitig auf die Absätze angewendeten Klassen `.rot` und `.blau`. Da die Regel der Klasse `.blau` im Code weiter unten steht, wird der Text blau dargestellt. Unerheblich ist dabei die Reihenfolge der im `class`-Attribut genannten Werte.

Listing 3.53: Anwendung der zuletzt genannten Klasse

```
.rot { color:red; }
.blau { color:blue; }
…
<p class="rot blau">Dieser Text ist blau.</p>
<p class="blau rot">Dieser Text ist blau.</p>
```

Sind unterschiedliche Style Sheets in den `head`-Bereich eines Webdokuments eingebunden, haben dokumentinterne Regeln Vorrang vor externen, die in separaten CSS-Dateien eingebunden sind. Dies gilt jedoch nur, wenn der Verweis auf die CSS-Datei vor den dokumentinternen CSS-Definitionen steht (Listing 3.54) – ansonsten ist es umgekehrt (Listing 3.55). Im gezeigten Beispiel enthält die CSS-Datei die Klasse `.green`, mit der eine grüne Zeichenfarbe festgelegt wird (`.green { color: green; }`).

Damit gilt auch hier: Je weiter unten eine CSS-Regel im Quelltext der Webseite platziert ist, desto höher ist ihre Priorität. Widersprechen sich zwei Regeln, wird die im Code weiter unten stehende Regel umgesetzt.

Binden Sie in HTML eine externe CSS-Datei ein, in die wiederum eine andere CSS-Datei mit der `@import`*-Anweisung integriert ist, sind die Regeln dieser Datei allen anderen untergeordnet.*

Listing 3.54: Anwendung der internen Regel

```
<link href="green.css" rel="stylesheet" type="text/css" />
<style type="text/css">
  <!--
    .blau { color: blue; }
  -->
</style>
…
<p class="rot blau green">Dieser Text ist blau.</p>
```

Listing 3.55: Anwendung der externen Regel

```
<style type="text/css">
  <!--
    .blau { color: blue; }
  -->
</style>
<link href="green.css" rel="stylesheet" type="text/css" />
…
<p class="rot blau green">Dieser Text ist grün.</p>
```

!important

Auch hier können Sie mit `!important` CSS-Formate überschreiben. Die Position im Quelltext spielt dann keine Rolle mehr. Steht folgender Code in der CSS-Datei, wird der Text immer grün dargestellt:

```
.green { color: green !important; }
```

Dabei muss das Anwendungsprogramm allerdings die Position der Regel im Code beachten: Eine mit `!important` gekennzeichnete Regel kann durch eine andere überschrieben werden, wenn diese im Code weiter unten steht. So wird der in Listing 3.54 formatierte Text nicht blau (`color: blue;`), sondern grün (`color: green`) dargestellt. Der zuerst genannte durch `!important` gekennzeichnete Stil hat also Vorrang vor dem zuletzt genannten.

Das Beispiel ist in der Datei reihenfolge.htm *in* kap03 *gespeichert. Hier finden Sie auch das Style Sheet* green.css.

3.4 Deklaration: Attribute und Werte

Auf den vorherigen Seiten haben Sie bereits häufig benötigte Deklarationen kennen gelernt. Beispielsweise wissen Sie bereits, dass das Attribut `font-family` die Schriftart bestimmt und `color` die Textfarbe. Darüber hinaus gibt es natürlich weit mehr Attribute, die Sie in den weiteren Kapiteln noch detailliert kennen lernen werden, und mit denen Sie den Initialwert ändern. Auch die entsprechenden Schlüsselbegriffe bzw. Wertzuweisungen kommen hier selbstverständlich nicht zu kurz.

Vorneweg betrachten wir in diesem Kapitel häufig verwendete Werte, die für unterschiedliche Attribute benötigt werden. Hierzu zählen Farbangaben, die nicht nur Zeichen, sondern auch den Hintergrund oder Rahmen eines Elements formatieren. Ebenso werden absolute oder relative Maßeinheiten für die Bestimmung unterschiedlicher Attribute eingesetzt.

Abschließend stelle ich Ihnen verschiedene Schreibweisen vor: Die CSS-Syntax sieht nämlich neben einer langen, ausführlichen Notation der Deklaration auch eine Kurzform vor, die bestimmte Anweisungen zusammenfasst.

3.4.1 Farben unterschiedlich definieren

Farben werden für Textzeichen, im Hintergrund der Webseite oder eines Objektes oder für unterschiedliche Rahmen benötigt. Dabei können Sie in CSS entweder den offiziellen Farbnamen, einen Hexadezimal-Wert oder verschiedene Prozentwerte angeben. Grundlage dieser relativen Angaben ist das Additionsverfahren, bei dem die drei Farben Rot, Grün und Blau gemischt werden.

Farbnamen verwenden

CSS1 benennt 16 Farben, die auf allen Systemen und in allen Browsern gleich dargestellt werden. Mit CSS2.1 wurde diese VGA-Palette um die Farbe Orange ergänzt. Die in Tabelle 3.6 aufgelisteten Namen dieser Grundfarben können jeweils als Schlüsselbegriff in der CSS-Deklaration verwendet werden. Dabei sieht die allgemeine Notation folgendermaßen aus:

```
Selektor { Eigenschaft: Farbname; }
```

Beispielsweise färben Sie mit dem folgenden Code eine Überschrift `h1` rot:

```
h1 { color: red; }
```

Farbname	Übersetzung / Beschreibung
aqua	Aquamarin
black	Schwarz
blue	Blau
fuchsia	Pink, Purpurrot
gray, grey	Grau
green	Grün
lime	Hellgrün
maroon	Kastanienbraun
navy	Marineblau
olive	Olivgrün
orange	Orange
purple	Lila, Violett
red	Rot
silver	Silber
teal	Aquamarin, See-, Smaragdgrün
white	Weiß
yellow	Gelb

Tabelle 3.6: Grundfarben nach CSS1 und 2.1 (alphabetisch sortiert)

Benutzeroberfläche

CSS2.0 führt weitere Farbnamen ein, die sich auf die Systemoberfläche beziehen. Da Benutzer den Arbeitsbereich individuell konfigurieren können, wissen Sie mit dieser Angabe nicht, wie das formatierte Element letzten Endes dargestellt wird. Zudem werden diese Farbangaben nicht von allen Browsern interpretiert und umgesetzt. Diese Notation wird bereits abgelehnt (*deprecated*) und soll mit CSS3 komplett verschwinden.

X11-Farbnamen

Stattdessen werden weitere Farben hinzukommen, die an die W3C-Spezifikation SVG 1.0 angepasst sind und X11-Farbnamen enthalten. Die meisten Farben stammen aus der Palette des Netscape-Browsers. Insgesamt beinhaltet die neue Palette 140 Farben, die noch nicht dem aktuellen Standard entsprechen und nicht komplett in allen Browsern dargestellt werden.

INFO

Statt `gray` *können Sie auch* `grey` *schreiben. Die Farben Fuchsia und Magenta sind identisch, ebenso Aqua und Cyan.*

DISC

Auf der CD-ROM im Verzeichnis kap03 *ist die VGA-Farbpalette in der Datei* farben.htm *gespeichert. Die Farbe ist mit ihrem Namen sowie als Hexadezimal- und Dezimal-Code angegeben. Zudem listet die Seite (veraltete) benutzerdefinierte Kolorierungen und die zukünftigen CSS3-Farben auf.*

Absoluten und relativen Farbanteil bestimmen

Das optische Modell der additiven Farbmischung basiert auf den drei Grundfarben Rot, Grün und Blau, weswegen man auch von RGB-Farben spricht. Beispielsweise entsteht Gelb, wenn rotes und grünes Licht zusammentreffen. Monitore, Fernseher und andere vergleichbaren Geräte arbeiten nach diesem Verfahren.

Der Anteil der jeweiligen Farbe wird auf einer Skala von 0 bis 255 gemessen. Insgesamt können 16.777.216 unterschiedliche Farben ($=256^3$) definiert werden. Je höher der angegeben Wert, desto höher ist der entsprechende Farbanteil. Damit können Sie einen Farbwert entweder absolut oder relativ angeben, wobei Nachkommastellen bei der Prozentangabe zulässig sind. Änderungen im Farbwert machen sich jedoch nur bei einem Abstand von etwa 0,4 % bemerkbar.

Die einzelnen, durch Komma getrennten RGB-Werte sind in Klammer angegeben, vor der `rgb` stehen muss (Listing 3.56). Die Großbuchstaben RGB sind Platzhalter. Die Leerzeichen sind optional und dienen lediglich der besseren Lesbarkeit (vgl. Listing 3.57).

Listing 3.56: Absolute und relative Dezimalangaben

```
Selektor { Eigenschaft: rgb(RRR, GGG, BBB; }
Selektor { Eigenschaft: rgb(RRR,R%, GGG,G%, BBB,B%; }
```

Überschneiden sich alle drei Farben in voller Intensität, entsteht die Tertiärfarbe Weiß. RGB betragen also jeweils 255 bzw. 100 %.

Listing 3.57: Unterschiedliche Definitionen von Weiß

```
Selektor { Eigenschaft: rgb(255,255,255;}
Selektor { Eigenschaft: rgb(100%,100%,100%;}
```

Schwarz stellt das andere Extrem dar: Die drei RGB-Farben haben daran keinen Anteil:

Listing 3.58: Unterschiedliche Definitionen von Schwarz

```
Selektor { Eigenschaft: rgb(0,0,0;}
Selektor { Eigenschaft: rgb(0%,0%,0%;}
```

Ein reines Blau erhalten Sie beispielsweise, wenn kein Anteil von Rot und Grün vorhanden ist `rgb(0,0,255)`.

Die Bestimmung der gewünschten Zwischenfarbe ist mit einem Webeditor, wie etwa Dreamweaver, recht einfach. Über einen Schieberegler ist hier schnell der gewünschte Farbton zusammengemischt und Sie können die entsprechenden Farbwerte in den Feldern darunter ablesen.

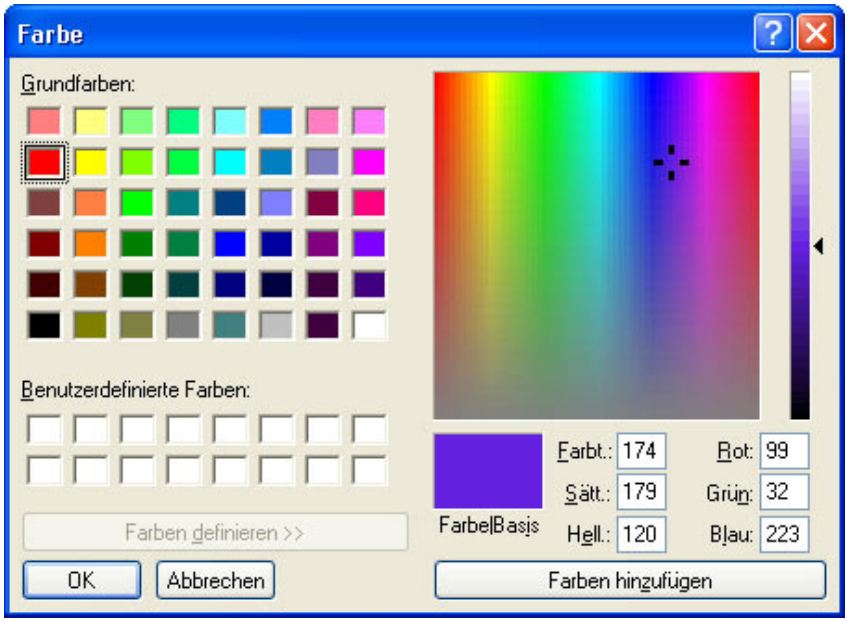


Abbildung 3.12: Farbe mischen

Der Druckbereich basiert nicht auf der additiven, sondern auf der subtraktiven Farbmischung, bei der vier verschiedene Farben verwendet werden: Gelb, Magenta, Cyan und Schwarz. Benötigen Sie einen Farbwert für die Printversion einer Seite (`media="print"`), müssen Sie dennoch die entsprechenden RGB-Werte angeben. Dies gilt übrigens für alle Medien-Style-Sheets.

Hexadezimalwert einsetzen

Weit gebräuchlicher als das Dezimalsystem ist die Angabe des Farbwertes als Hexadezimalcode, der auf dem Ausgangswert 16 basiert. Dieser Wertebereich der RGB-Farben liegt zwischen `00` und `ff`, wobei Sie Buchstaben auch groß schreiben können. Für jeden RGB-Wert sind also zwei Stellen im

sechsstelligen Hexadezimalcode reserviert, vor dem immer das Lattenkreuz # steht. Damit sieht die allgemeine Schreibweise folgendermaßen aus:

```
Selektor { Eigenschaft: #RRGGBB; }
```

Tabelle 3.7 listet die Hexadezimalwerte von 0 bis FF auf. Im Dezimalsystem entspricht dies dem Wertebereich von 0 bis 255. Folglich ergibt sich für Schwarz ein Hexadezimalcode von #000000 und für Weiß ein Wert von #FFFFFF. Der Wert für reines Blau lautet demnach #0000FF.

Auch mit diesem System gibt es pro Farbe 256 mögliche Werte, so dass insgesamt ebenfalls 16,8 Mio. Farben darstellbar sind.

Tabelle 3.7: Hexadezimal-Zahlen

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Kurzform

Neben der sechsstelligen langen Schreibweise gibt es auch eine Kurzform, bei der gleiche Zeichen zusammengefasst werden. Dies erfolgt für die Zahlenpaare Rot, Grün und Blau jeweils separat. Beispielsweise können Sie statt #0000FF auch #00F schreiben. Dann wird Weiß #FFF und Schwarz #000 kodiert. Auch bei einem Grauton bestehen alle drei Ziffern aus dem gleichen Wert (z. B. #CCC). Die allgemeine Syntax sieht damit folgendermaßen aus:

```
Selektor { Eigenschaft: #RGB; }
```

Für den Blauton #004080 gibt es dagegen keine Kurzform, da die letzten beiden Farbwerte (G = 40, B = 80) uneinheitlich sind.

HALT

Einige CSS-Filter funktionieren nur, wenn Sie statt der Kurzform den ausführlichen Hexadezimalcode angeben. Diese Effekte beschreibt Kapitel 14.

INFO

Auch für Eigenschaften gibt es eine Kurzschreibweise, die Kapitel 3.4.3 erläutert.

Websichere Farben

Ebenso können alle so genannten websicheren Farben in Kurzform geschrieben werden. Dabei handelt es sich um Kombinationen der Werte 00, 33, 66, 99, CC und FF. Diese 216 Farben wurden bereits früher auf allen Systemen (Windows, Macintosh), in allen Browsern und bei einer damals üblichen Farbtiefe von 8 Bit gleich dargestellt. Für diese Farben gibt es nur die Hexadezimalschreibweise. Mittlerweile gilt diese Mitte der 90er Jahre entwickelte Farbskala als veraltet.

Mit dem Farbwert `transparent` *legen Sie ausdrücklich fest, dass ein dahinter liegendes Objekt sichtbar sein soll. Da dies ohnehin der Standardeinstellung im Browser entspricht, hat diese Notation keine praktische Bedeutung. Tatsächliche Transparenzeffekte mit fließenden Übergängen zwischen Vorder- und Hintergrund erzielen Sie dagegen im Internet Explorer mit diversen CSS-Filtern (vgl. Kapitel 14).*

3.4.2 Absolute und relative Maßeinheiten angeben

Schriftgrößen, Höhe und Breite von Objekten sowie Ränder und Abstände werden mit numerischen Werten definiert. Diese können positiv und negativ sein sowie Nachkommastellen haben. Dabei verwenden Sie jedoch statt des Kommas einen Punkt.

Der Einsatz eines negativen Wertes für absolute oder relative Einheiten ist bei bestimmten Eigenschaften nicht erlaubt und auch sonst nicht immer sinnvoll. Beispielsweise sollten Sie die Zeichengröße stets mit positiven Werten definieren. Zudem können nicht alle Browser negative Einheiten interpretieren. Stattdessen wird dann der nächstgelegene Wert, also der Wert 0 umgesetzt. Basiert das Layout auf negativen Werten, kann es dann zu Problemen kommen.

Handelt es sich nicht gerade um den Wert 0, müssen Sie direkt hinter dem Wert eine Einheit angeben. Dabei ist zwischen absoluten und relativen Angaben zu unterscheiden:

Absolute Angaben

Absoluten Einheiten ist immer eine feste Größe zugewiesen. Ein Zentimeter ist eben immer zehn Millimeter lang – oder etwa nicht?

Auf dem Bildschirm ist die Umsetzung eines absoluten Wertes von der Monitorbreite und der eingestellten Auflösung abhängig. Dabei liegt die Auflösung bei den meisten Geräten zwischen 72 und 130 dpi. Auch die Monitorgröße wird berücksichtigt. Folglich kann ein zehn Zentimeter breites Element auch mal etwas kleiner oder größer ausfallen. Auch bei Ihnen gut lesbare Texte können woanders viel kleiner und unleserlich sein. Damit sind die in Tabelle 3.8 aufgeführten absoluten Angaben für die Bildschirmdarstellung kaum geeignet. Hier werden ohnehin Pixelwerte benötigt, so dass der absolute Wert zuerst umgerechnet werden muss.

Angenommen Sie arbeiten an einem Monitor mit einer Auflösung von 1.440 x 900 Pixel und 72 dpi. Bei einer Breite von 36,8 cm bzw. 14,5 Zoll ergibt sich eine tatsächliche Auflösung von 99 dpi (= 1.440/14,5). Soll der Browser nun ein Element mit dem Wert 1 Zoll darstellen, wird die eingestellte Auflösung von 72 dpi zugrunde gelegt. Damit ist auf diesem Monitor 1 Zoll tatsächlich etwas kleiner, nämlich 0,7 Zoll (= 72/99). Ein 10 Zentimeter breites Element wird also nicht mit 3,9 Zoll, sondern nur mit 2,8 Zoll dargestellt. Dies entspricht 7,1 Zentimetern.

Definieren Sie dagegen ein Style Sheet für den Ausdruck auf Papier, sind absolute Werte durchaus sinnvoll, da die Papiergröße bekannt ist und keine Umrechnung stattfindet. Die Einheiten Point (pt) und Pica (pc) stammen ohnehin aus dem Druckwesen.

Tabelle 3.8: Absolute Einheiten

CSS-Code	Beschreibung
`in`	Inch, Zoll, 1 in = 2,54 cm, Längeneinheit
`cm`	Zentimeter, Längeneinheit, = 10 mm
`mm`	Millimeter, Längeneinheit, = 0,1 cm
`pt`	Point, Punkt, 1 pt = 1/72 Zoll = 0,35 mm, typografische Maßeinheit
`pc`	Pica, 1 pc = 12 Punkt = 1/6 Zoll, typografische Maßeinheit

Bei einer Auflösung von 72 dpi erfolgt die Umrechnung von Pixel in cm bzw. umgekehrt nach folgenden Formeln:

Umrechnung Pixel in cm: $y\ cm = (x\ Pixel \cdot 2{,}54) / 72\ dpi$
Umrechnung cm in Pixel: $y\ Pixel = (x\ cm / 2{,}54) \cdot 72\ dpi$

Zu umständlich? Unter http://de.selfhtml.org/helferlein/ptinchmm.htm können Sie schnell eine Einheit in eine andere umrechnen.

Relative Einheiten

Relative Einheiten ergeben sich aus der Größe eines übergeordneten Elements und haben damit immer einen Bezugspunkt. Zudem hängt die Größe eines Pixels von der Auflösung des Ausgabemediums ab. Je höher die Auflösung, desto kleiner erscheint hier der Pixel.

Benutzer können im Browser die beiden Einheiten em und ex beeinflussen, indem sie die Schriftgröße im ANSICHT-Menü neu einstellen. em steht für *equal M* und bezieht sich auf die Breite des Buchstabens M, die der Höhe der Gevierts und damit der Schriftgröße entspricht. Dabei ist der Standardschriftgröße des Browsers das Maß 1 em zugewiesen. Dagegen bezieht sich die Einheit ex auf die Höhe des Kleinbuchstabens x und ist damit von der verwendeten Schriftart abhängig. Die Einheiten em und ex werden vom Elternelement geerbt und beeinflussen die entsprechend definierte Schriftgröße eines untergeordneten Elements (vgl. Kapitel 5.1.5).

Tabelle 3.9: Relative Einheiten

CSS-Code	Beschreibung
`px`	Pixel, Bildpunktgröße ist von der Auflösung des Ausgabemediums abhängig, 16 px ~ 1 em
`em`	*equal M*, bezieht sich auf das große M der aktuellen Schriftart, typografische Maßeinheit, 1 em entspricht der Standardschriftgröße im Browser (~ 16 Pixel)
`ex`	Bezieht sich auf das kleine x der aktuellen Schriftart. 1 ex ~ 1/2 em, typografische Maßeinheit
`%`	Prozent, bezieht sich auf die Größe des übergeordneten Elements bzw. auf die Seitenbreite

CSS2.1 bestimmt, wie relative Werte für die Darstellung umgerechnet werden: Zunächst ermittelt das Ausgabegerät die festgelegten Werte und wendet diese wenn möglich direkt auf das jeweilige Element an. Dies kann auch ein von einem Elternelement vererbter Wert sein. Anschließend werden relative Werte in numerische umgerechnet und ggf. an Kindelemente weitergegeben. Der tatsächlich verwendete Pixelwert bestimmt die Darstellung am Bildschirm. Dabei kann die Berechnung einen gerundeten Wert ergeben.

Kaskade

3.4.3 Kurz- und Langschreibweise

Die CSS-Syntax sieht neben einer ausführlichen, langen Schreibweise eine kurze Schreibweise vor. Wie Sie einen Hexadezimal-Code zusammenziehen und damit Wert-Paare verkürzen, haben Sie schon in Kapitel 3.4.1 gesehen. Die Kurzschreibweise erlaubt zusätzlich die Zusammenfassung mehrerer Attribute und Werte in einer einzigen Deklaration. Dabei müssen Sie den Attributnamen ändern und die einzelnen Werte in einer bestimmten Reihenfolge nennen.

Beispielsweise fasst das Attribut `font` die Werte der Attribute `font-family`, `font-size`, `font-size-adjust`, `font-stretch`, `font-style`, `font-variant` und `font-weight` zusammen.

Die folgenden beiden Listings zeigen die gleiche Formatanweisung in Lang- und Kurzform. Statt der unterschiedlichen Einzelattribute verwenden Sie nur die `font`-Eigenschaft. Während die Schriftart stets hinter der Größe stehen sollte, ist die Reihenfolge der anderen Attribute unerheblich.

Listing 3.59: CSS-Langschrift

```
p {
    font-family: Arial;
    font-size: 0.9em;
    font-style: italic;
  }
```

Listing 3.60: CSS-Kurzschrift

```
p { font: italic 0.9em Arial; }
```

Durch die Kurzschreibweise wird der Code deutlich kürzer und übersichtlicher. Allerdings interpretieren Browser die Kurzform teilweise recht uneinheitlich. Geben Sie eine Eigenschaft nicht an, wird häufig der Standardwert verwendet. Dabei können die ausgelassenen Kurzschrift-Eigenschaften die in anderen CSS-Regeln (lange Form) definierten Attribute außer Kraft setzen.

Auch für Rahmenstile, -breiten oder Abstände gibt es neben einer ausführlichen Schreibweise eine Kurzform. Dabei werden die Werte der vier einzelnen Objektseiten zusammengefasst wiedergegeben. Ausführliche Informationen finden Sie im jeweiligen Kapitel. Ebenso lernen Sie in Kapitel 4 unterschiedliche Eigenschaften eines Hintergrundes sowie die entsprechende Kurznotation kennen.