

Einer der größten Vorteile von CSS, insbesondere für Designer, besteht in der Möglichkeit, mehrere Stildefinitionen auf alle Elemente des gleichen Typs anzuwenden. Nicht beeindruckt? Dann vielleicht hiervon: Durch das Anpassen nur einer Zeile im CSS-Code können Sie die Farben aller Überschriften gleichzeitig ändern. Sie mögen den verwendeten Blauton nicht? Ändern Sie einfach die betreffende Codezeile und schon sind Ihre Überschriften lila, gelb, rotbraun oder haben eine andere Farbe Ihrer Wahl. Dadurch können Sie sich als Designer auf das Design anstelle der »Drecksarbeit« konzentrieren. Wenn also beim nächsten Meeting irgendetwas die Überschriften in einem anderen Grünton sehen möchte, brauchen Sie bloß Ihre Stildefinition zu ändern und den »Aktualisieren«-Button zu drücken. *Voilà!* Das lässt sich in Sekunden erledigen und die Ergebnisse können sofort gezeigt werden.

Selbstverständlich kann auch CSS nicht alle Ihre Probleme lösen. Die Farbe Ihrer GIFs können Sie leider nicht ändern, aber andere globale Anpassungen werden dennoch erheblich erleichtert. Wir wollen uns nun mit Selektoren und ihrer Beziehung zur Dokumentstruktur befassen.

Grundsätzliche Regeln

Wie bereits erwähnt, besteht eine der großen Stärken von CSS darin, bestimmte Regeln auf einen vollständigen Satz von Elementtypen in einem Dokument anzuwenden. Für das folgende Beispiel soll der Text aller mit h2 markierten Elemente grau dargestellt werden. Nach den HTML-Regeln alter Schule würden Sie dies durch das Einfügen von `...`-Tags in alle h2-Elemente erreichen:

```
<h2><font color="gray">This is h2 text</font></h2>
```

Das ist offensichtlich sehr aufwändig, besonders wenn Ihr Dokument viele h2-Elemente enthält. Schlimmer wird es, wenn alle diese h2-Überschriften später nicht mehr grau, sondern vielleicht grün dargestellt werden sollen und Sie alle Markierungen von Hand wieder ändern müssen.

CSS macht es möglich, einfache und leicht änderbare Regeln aufzustellen, die auf alle von Ihnen definierten Textelemente angewendet werden können. (Im folgenden Abschnitt werden Sie sehen, wie diese Regeln funktionieren.) Um sämtliche h2-Elemente grau einzufärben, benötigen Sie beispielsweise nur diese eine Regel:

```
h2 {color: gray;}
```

Soll nun die Farbe aller h2-Elemente geändert werden, sagen wir einfach mal auf Silbergrau, müssen Sie diese Regel nur entsprechend anpassen:

```
h2 {color: silver;}
```

Struktur der Regeln

Um das Konzept dieser Regeln besser zu verstehen, wollen wir uns deren Struktur einmal genauer ansehen.

Jede Regel besteht aus zwei wesentlichen Teilen: dem *Selektor* und dem *Deklarationsblock*. Der Deklarationsblock besteht seinerseits aus mehreren *Deklarationen*. Jede Deklaration ist unterteilt in eine *Eigenschaft* (Property) und den dazugehörigen *Wert*. Jedes Stylesheet besteht aus einer Reihe von Regeln. Die verschiedenen Teile einer Regel sehen Sie in Abbildung 2-1.

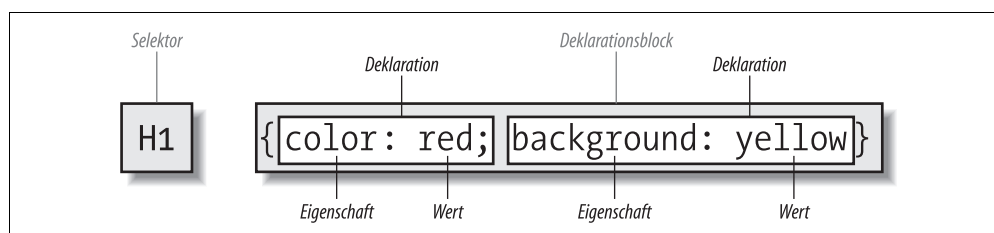


Abbildung 2-1: Die Struktur einer Regel

Der auf der linken Seite stehende Selektor legt fest, auf welche Teile des Dokuments sich die Regel auswirkt. In Abbildung 2-1 wurden h1-Elemente ausgewählt. Wäre p der Selektor, würde sich diese Regel auf alle p-Elemente (Absätze) auswirken.

Die rechte Seite der Regel enthält den Deklarationsblock, der seinerseits eine oder mehrere Deklarationen enthalten kann. Jede Deklaration besteht aus einer CSS-Eigenschaft und einem dazugehörigen Wert. In Abbildung 2-1 enthält der Deklarationsblock demnach zwei Deklarationen. Die erste Deklaration besagt, dass Teile des Dokuments die Farbe (color) Rot (red) erhalten sollen, während die zweite Deklaration dafür sorgt, dass für bestimmte Teile des Dokuments der Hintergrund (background) gelb (yellow) dargestellt werden soll. Sämtliche (durch den Selektor festgelegten) h1-Elemente des Dokuments werden also rot mit einem gelben Hintergrund dargestellt.

Elemente als Selektoren

In den meisten Fällen entspricht ein Selektor einem bestimmten HTML-Element, aber nicht immer. Enthält eine CSS-Datei beispielsweise die Stildefinitionen für ein XML-Dokument, kann ein Selektor auch schon einmal so aussehen:

```
QUOTE {color: gray;}
BIB {color: red;}
BOOKTITLE {color: purple;}
MYElement {color: red;}
```

Anders gesagt: Die Elemente eines Dokuments dienen als grundsätzliche Selektoren. In XML kann ein Selektor so ziemlich alles sein, da XML die Schaffung neuer Auszeichnungssprachen ermöglicht, in denen die Elemente beliebige Namen haben können. Wenn Sie dagegen ein HTML-Dokument mit Stildefinitionen versehen möchten, wird normalerweise ein bestimmtes HTML-Element als Selektor verwendet, wie beispielsweise `p`, `h3`, `em`, `a` oder sogar `html` selbst. Ein Beispiel:

```
html {color: black;}
h1 {color: gray;}
h2 {color: silver;}
```

Die Ergebnisse für dieses Stylesheet sehen Sie in Abbildung 2-2.

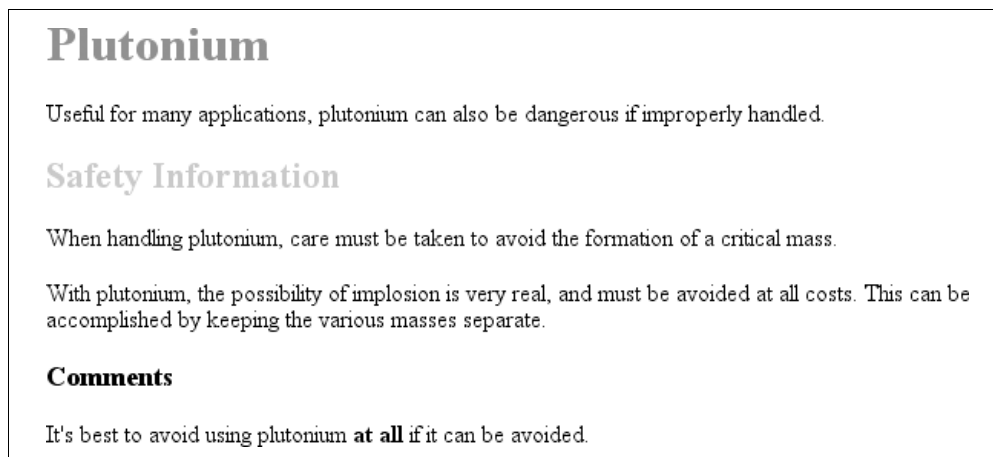


Abbildung 2-2: Einfache Stildefinitionen für ein einfaches Dokument

Haben Sie einmal begonnen, globale Stildefinitionen direkt auf bestimmte Elemente zu beziehen, können Sie diese zwischen den verschiedenen Elementen auch verschieben. Vielleicht wollen Sie, dass nicht die mit `h1` markierten Überschriften aus Abbildung 2-2, sondern der Text in den Absätzen grau dargestellt werden soll. Tauschen Sie hierfür einfach den Selektor `h1` gegen ein `p` aus:

```
html {color: black;}
p {color: gray;}
h2 {color: silver;}
```

Die Ergebnisse sehen Sie in Abbildung 2-3.

Plutonium

Useful for many applications, plutonium can also be dangerous if improperly handled.

Safety Information

When handling plutonium, care must be taken to avoid the formation of a critical mass.

With plutonium, the possibility of implosion is very real, and must be avoided at all costs. This can be accomplished by keeping the various masses separate.

Comments

It's best to avoid using plutonium **at all** if it can be avoided.

Abbildung 2-3: Stildefinitionen zwischen zwei Elementen verschieben

Deklarationen und Schlüsselwörter

Der Deklarationsblock kann eine oder mehrere Deklarationen enthalten. Eine Deklaration besteht dabei immer aus einer Eigenschaft, die mit einem *Wert* durch einen Doppelpunkt verbunden ist. Abgeschlossen wird die Deklaration durch ein nachgestelltes Semikolon (;). Auf den Doppelpunkt und das Semikolon kann eine beliebige Anzahl von Leerzeichen (also auch keines) folgen. Fast immer besteht der Wert aus einem einzelnen Schlüsselwort oder einer durch Leerzeichen getrennten Liste aus mehreren Schlüsselwörtern, die für die genannte Eigenschaft zulässig sind. Benutzen Sie in einer Deklaration eine ungültige Eigenschaft oder einen nicht zulässigen Wert, wird das Ganze einfach ignoriert. Die folgenden zwei Deklarationen würden also nicht funktionieren:

```
brain-size: 2cm; /* unbekante Eigenschaft */  
color: ultraviolet; /* unbekannter Wert */
```

Wenn für den Wert einer Eigenschaft mehrere Dinge angegeben werden können, werden die Schlüsselwörter in der Regel durch Leerzeichen voneinander getrennt. Allerdings akzeptieren nicht alle Eigenschaften mehrere Schlüsselwörter; viele aber schon, wie beispielsweise die Eigenschaft `font`. Wie in Abbildung 2-4 gezeigt, wollen wir nun für den Absatztext den Schrifttyp Helvetica in mittlerer Größe festlegen.

Dafür verwenden wir die folgende Regel:

```
p {font: medium Helvetica;}
```

Beachten Sie das Leerzeichen zwischen `medium` und `Helvetica`, die beide als Schlüsselwort gelten (das erste Schlüsselwort bezieht sich hier auf die Schriftgröße, das zweite auf den verwendeten Schrifttyp). Das Leerzeichen ermöglicht es dem Browser, zwischen beiden

Plutonium

Useful for many applications, plutonium can also be dangerous if improperly handled.

Safety Information

When handling plutonium, care must be taken to avoid the formation of a critical mass.

With plutonium, the possibility of implosion is very real, and must be avoided at all costs. This can be accomplished by keeping the various masses separate.

Comments

It's best to avoid using plutonium **at all** if it can be avoided.

Abbildung 2-4: Die Ergebnisse eines Eigenschaftswerts mit mehreren Schlüsselwörtern

Schlüsselwörtern zu unterscheiden und sie korrekt anzuwenden. Das Semikolon kennzeichnet das Ende der Deklaration.

Diese durch Leerzeichen voneinander getrennten Wörter bezeichnet man als Schlüsselwörter, weil sie zusammen genommen den Wert der fraglichen Eigenschaft bilden. Nehmen Sie beispielsweise folgende fiktive Regel:

```
rainbow: red orange yellow green blue indigo violet;
```

Natürlich gibt es keine Eigenschaft namens `rainbow` und zwei der benutzten Farben sind auch nicht gültig; trotzdem illustriert das Beispiel das zuvor Gesagte recht gut. Der Wert von `rainbow` ist `red orange yellow green blue indigo violet`, wobei die sieben Schlüsselwörter einen einmaligen Wert bilden. Wir können den Wert von `rainbow` daher folgendermaßen neu definieren:

```
rainbow: infrared red orange yellow green blue indigo violet ultraviolet;
```

Jetzt besteht der Wert von `rainbow` aus neun Schlüsselwörtern an Stelle der vorhin benutzten sieben. Auch wenn der Name für die zwei Werte gleich ist, sind die beiden so unterschiedlich wie Null und Eins.



Wie wir gesehen haben, werden CSS-Schlüsselwörter durch Leerzeichen voneinander getrennt – bis auf eine Ausnahme. In der CSS-Eigenschaft `font` gibt es genau eine Stelle, an der ein Schrägstrich (`/`) benutzt werden kann, um zwei Schlüsselwörter voneinander zu trennen. Hier ein Beispiel:

```
h2 {font: large/150% sans-serif;}
```

Der Schrägstrich trennt die Schlüsselwörter, die die Schriftgröße und die Höhe der Zeilen festlegen. Das ist der einzige Ort, an dem der Schrägstrich in einer `font`-Deklaration verwendet werden darf. Alle anderen für `font` erlaubten Schlüsselwörter müssen durch Leerzeichen voneinander getrennt werden.

Das sind die Grundlagen einfacher Deklarationen, die allerdings auch wesentlich komplexer ausfallen können. Im folgenden Abschnitt erhalten Sie einen kleinen Einblick in die Macht von CSS.

Gruppierung

Bisher haben wir verhältnismäßig einfache Techniken kennen gelernt, mit denen eine einzelne Stildefinition einem einzelnen Selektor zugeordnet wurde. Was aber nun, wenn der gleiche Stil auf mehrere Elemente angewendet werden soll? In diesem Fall wollen Sie mehr als einen Selektor benutzen oder mehr als einen Stil auf ein Element oder eine Gruppe von Elementen anwenden.

Selektoren gruppieren

Für dieses Beispiel sollen alle Überschriften der zweiten Ebene (h2) und alle Absätze (p) grau dargestellt werden. Am einfachsten lässt sich das mit folgender Deklaration erreichen:

```
h2, p {color: gray;}
```

Hier stehen *zwei* Selektoren, h2 und p, auf der linken Seite, die durch ein Komma getrennt werden. Damit haben Sie eine Regel definiert, bei der die Stildefinition auf der rechten Seite (color: gray;) auf alle Elemente, auf die sich die beiden Selektoren auf der linken Seite beziehen, angewendet wird. Das Komma zwischen den beiden Selektoren zeigt dem Browser, dass zwei Selektoren an der Regel beteiligt sind. Durch das Weglassen des Kommas würde die Regel eine vollkommen andere Bedeutung erhalten, die wir später im Abschnitt »Kindselektoren« genauer erforschen werden.

Für die Anzahl der Selektoren, die miteinander gruppiert werden können, gibt es keine Obergrenze. Wenn Sie beispielsweise eine große Anzahl von Elementen in Grau darstellen wollen, können Sie die folgende Regel verwenden:

```
body, table, th, td, h1, h2, h3, h4, p, pre, strong, em, b, i {color: gray;}
```

Durch die Gruppierung kann der Autor bestimmte Stilzuweisungen stark zusammenfassen, wodurch auch das Stylesheet selbst kürzer wird. Die folgenden Alternativen erzeugen exakt das gleiche Ergebnis; es ist aber ziemlich offensichtlich, welche der beiden Möglichkeiten leichter einzugeben ist:

```
h1 {color: purple;}
h2 {color: purple;}
h3 {color: purple;}
h4 {color: purple;}
h5 {color: purple;}
h6 {color: purple;}
```

```
h1, h2, h3, h4, h5, h6 {color: purple;}
```

Die Gruppierung gibt Ihnen außerdem ein paar interessante Wahlmöglichkeiten. So sind alle Regelgruppen im folgenden Beispiel gleichwertig und sollen nur unterschiedliche Wege aufzeigen, wie Selektoren und Deklarationen miteinander gruppiert werden können:

```
/* Gruppe 1 */
h1 {color: silver; background: white;}
h2 {color: silver; background: gray;}
h3 {color: white; background: gray;}
h4 {color: silver; background: white;}
b {color: gray; background: white;}

/* Gruppe 2 */
h1, h2, h4 {color: silver;}
h2, h3 {background: gray;}
h1, h4, b {background: white;}
h3 {color: white;}
b {color: gray;}

/* Gruppe 3 */
h1, h4 {color: silver; background: white;}
h2 {color: silver;}
h3 {color: white;}
h2, h3 {background: gray;}
b {color: gray; background: white;}
```

Alle drei Gruppen erzeugen das in Abbildung 2-5 gezeigte Ergebnis. (Zur Definition der Stile werden gruppierte Deklarationen benutzt, auf die wir im folgenden Abschnitt eingehen werden.)

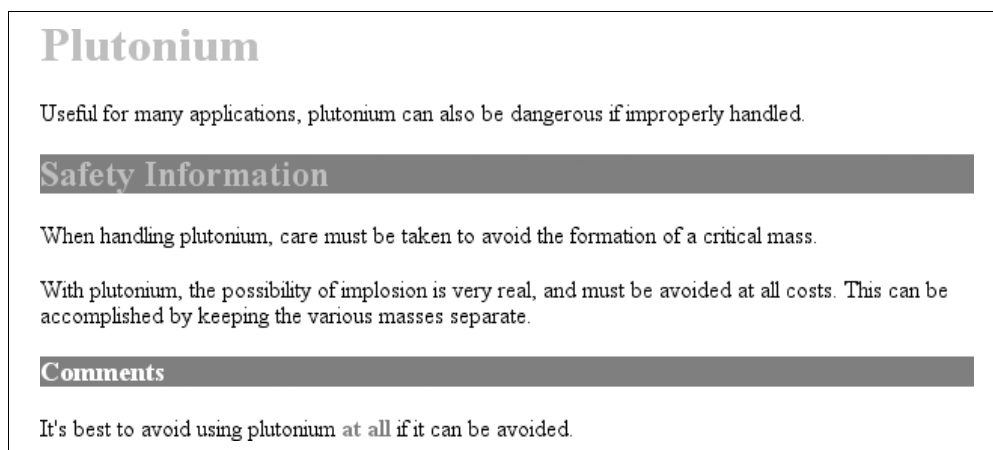


Abbildung 2-5: Das Ergebnis von gleichwertigen Stylesheets

Der universelle Selektor

Mit CSS2 wurde ein neuer einfacher Selektor namens *universeller Selektor* eingeführt. Er wird durch einen Asterisken (*) dargestellt. Dieser Selektor passt auf ein beliebiges Ele-

ment und verhält sich damit so ähnlich wie ein Wildcard-Zeichen. Um beispielsweise alle Elemente in einem Dokument rot einzufärben, bräuchten Sie nur Folgendes schreiben:

```
* {color: red;}
```

Diese Deklaration hat die gleiche Bedeutung wie ein gruppierter Selektor, der jedes im Dokument enthaltene Element einzeln aufführt. Mit dem universellen Selektor können Sie alle Elemente eines Dokuments in einem effizienten Schritt rot anfärben (`color: red`). Sie sollten allerdings bedenken, dass der universelle Selektor trotz seiner Bequemlichkeit einige ungewollte Konsequenzen nach sich ziehen kann, auf die wir im nächsten Kapitel genauer eingehen werden.

Deklarationen gruppieren

Aus der Möglichkeit, Selektoren zu einer einzigen Regel zusammenzufassen, folgt die Möglichkeit, auch Deklarationen gruppieren zu können. Im nächsten Beispiel sollen alle `h1`-Elemente in purpurfarbener, 18 Pixel großer Schrift vom Typ Helvetica vor einem wasserblauen Hintergrund dargestellt werden (wenn es Ihnen nichts ausmacht, Ihre Leser zu blenden). Hierfür könnten Sie folgendes Stylesheet verwenden:

```
h1 {font: 18px Helvetica;}
h1 {color: purple;}
h1 {background: aqua;}
```

Allerdings ist diese Methode nicht besonders effizient. Stellen Sie sich vor, Sie müssten eine solche Liste für ein Element anlegen, das 10 oder sogar 15 Stildefinitionen benötigt. Stattdessen können Sie Ihre Deklarationen in einem gemeinsamen Deklarationsblock zusammenfassen:

```
h1 {font: 18px Helvetica; color: purple; background: aqua;}
```

Diese Schreibweise hat exakt den gleichen Effekt wie das dreizeilige Stylesheet, das wir gerade gesehen haben.

Beachten Sie, dass die Verwendung des Semikolons (;) am Ende jeder Deklaration für die Gruppierung von Deklarationen absolut notwendig ist. Leerzeichen in Stylesheets werden von Browsern ignoriert, allerdings benötigen die User Agents eine korrekte Syntax, um Ihr Stylesheet richtig zu interpretieren. Sie können Ihre Stildefinitionen auf eine der hier gezeigten Arten formatieren:

```
h1 {
  font: 18px Helvetica;
  color: purple;
  background: aqua;
}
```

Wenn allerdings das zweite Semikolon entfernt wird, interpretiert der User Agent das Stylesheet wie folgt:

```
h1 {
  font: 18px Helvetica;
  color: purple background: aqua;
}
```

Da `background` kein gültiger Wert für `color` ist und `color` außerdem nur ein Schlüsselwort als Wert enthalten darf, wird diese Deklaration von `color` (inklusive des `background: aqua`-Teils) vollständig vom Browser ignoriert. Stattdessen wird der Text in `h1` vielleicht purpurfarben, aber ohne den wasserblauen Hintergrund dargestellt. Allerdings ist es wesentlich wahrscheinlicher, dass auch für `h1`-Elemente einfach die Standardfarbe ohne irgendwelche Hintergründe benutzt wird. (Dagegen wird die Deklaration `font: 18px Helvetica` benutzt, weil sie korrekt mit einem Semikolon beendet wurde.)



Zwar ist es technisch gesehen nicht unbedingt notwendig, die letzte Deklaration einer Regel mit einem Semikolon zu beenden, dennoch gilt dieses Vorgehen als guter Stil. Zum einen werden Sie sich so daran gewöhnen, immer Semikola zu benutzen, deren Fehlen einer der häufigsten Fehler beim Interpretieren von Stylesheets ist. Zum anderen müssen Sie beim Hinzufügen einer weiteren Deklaration zu der Regel nicht darauf achten, ob ein zusätzliches Semikolon eingefügt werden muss oder nicht. Und schließlich lassen sich einige ältere Browser wie beispielsweise der Internet Explorer 3.x von einem fehlenden Semikolon am Ende einer Regel leichter verwirren als andere. Um diese Probleme zu vermeiden, sollten Sie eine Deklaration daher immer mit einem Semikolon beenden, unabhängig von ihrer Position innerhalb der Regel.

Wie das Gruppieren von Selektoren bietet auch das Gruppieren von Deklarationen eine bequeme Möglichkeit, um Stildefinitionen kurz, ausdrucksstark und leicht wartbar zu halten.

Alles gruppieren

Sie wissen nun, dass Sie sowohl Selektoren wie auch Deklarationen gruppieren können. Wenn Sie beide Arten der Gruppierung in einer Regel zusammenfassen, können Sie mit wenigen Anweisungen sehr komplexe Stildefinitionen erstellen. Wie gehen Sie also vor, wenn Sie einige komplexe Stilvorgaben für alle Überschriften eines Dokuments gemeinsam verwenden wollen? Hier ein Beispiel:

```
h1, h2, h3, h4, h5, h6 {color: gray; background: white; padding: 0.5em;
border: 1px solid black; font-family: Charcoal, sans-serif;}
```

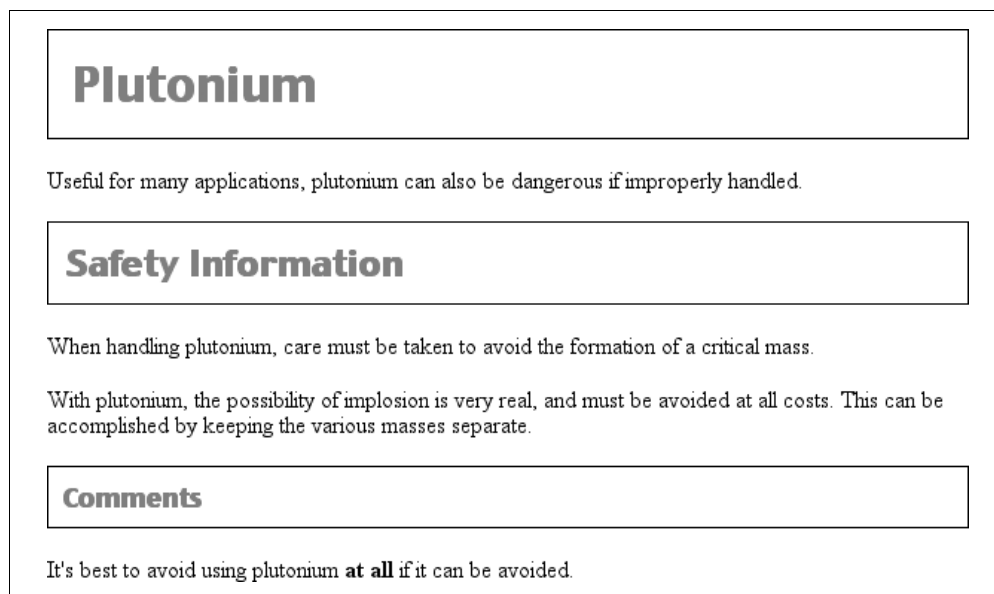
Hier wurden die Selektoren gruppiert, wodurch die Stildefinitionen auf der rechten Seite auf alle in der Liste stehenden Überschriftsmarkierungen angewendet werden. Die Gruppierung der Deklarationen bedeutet, dass alle aufgeführten Stilvorgaben auf die Selektoren auf der linken Seite angewendet werden. Das Ergebnis sehen Sie in Abbildung 2-6.

Dieser Ansatz ist einer Version vorzuziehen, bei der sämtliche Teile einzeln aufgeführt werden. Diese könnte etwa so aussehen:

```
h1 {color: gray;}
h2 {color: gray;}
h3 {color: gray;}
h4 {color: gray;}
```

```
h5 {color: gray;}
h6 {color: gray;}
h1 {background: white;}
h2 {background: white;}
h3 {background: white;}
```

Diese Schreibweise geht noch viele Zeilen so weiter. Sicherlich können Sie Ihre Stildefinitionen auch nacheinander formulieren; allerdings würde ich Ihnen das nicht empfehlen. Ein Bearbeiten Ihrer Stile wäre fast so aufwändig wie die Verwendung von font-Tags!



The image shows a document layout with three distinct sections, each enclosed in a rectangular box. The first section is titled "Plutonium" in a large, bold, dark font. Below the title, a line of text reads: "Useful for many applications, plutonium can also be dangerous if improperly handled." The second section is titled "Safety Information" in a large, bold, dark font. Below the title, there are two lines of text: "When handling plutonium, care must be taken to avoid the formation of a critical mass." and "With plutonium, the possibility of implosion is very real, and must be avoided at all costs. This can be accomplished by keeping the various masses separate." The third section is titled "Comments" in a large, bold, dark font. Below the title, a line of text reads: "It's best to avoid using plutonium **at all** if it can be avoided."

Abbildung 2-6: Gleichzeitiges Gruppieren von Selektoren und Regeln

Anstatt Stildefinitionen nur auf bestimmte Elemente eines Dokuments zu beziehen, können Sie die Selektoren noch ausdrucksstärker machen, indem Sie sie eher auf bestimmte Informationen anwenden. Für etwas so Mächtiges ist zwar einige Vorarbeit nötig, die sich aber auf jeden Fall lohnt.

Klassen- und ID-Selektoren

Bisher haben wir Selektoren und Deklarationen auf verschiedene Weise miteinander kombiniert; allerdings handelte es sich bisher nur um einfache Selektoren, die sich auf bestimmte Elemente von Dokumenten bezogen. Das ist bis zu einem bestimmten Punkt auch völlig in Ordnung. Manchmal brauchen Sie jedoch etwas Spezielleres.

Zusätzlich zu den rohen Dokument-Elementen gibt es noch zwei weitere Arten von Selektoren: *Klassenselektoren* und *ID-Selektoren*, mit denen sich Stildefinitionen unabhängig von den Elementen eines Dokuments zuweisen lassen. Diese Selektoren können eigen-

ständig oder zusammen mit Elementselektoren verwendet werden. Allerdings ist es hierfür notwendig, die Dokumentteile entsprechend zu markieren. Die Benutzung dieser neuen Selektoren erfordert daher ein wenig Voraussicht und Planung.

Wenn Sie beispielsweise ein Dokument entwerfen, in dem es um die Handhabung von Plutonium geht, so enthält das Dokument vermutlich eine Reihe von Warnungen, wie man mit solch einer gefährlichen Substanz umzugehen hat. Um die Warnungen hervorzuheben, sollen diese in fetter Schrift dargestellt werden. Sie wissen allerdings noch nicht, welche Elemente diese Warnungen enthalten werden. Einige Warnungen könnten aus ganzen Absätzen bestehen, während andere vielleicht nur ein einzelner Eintrag in einer längeren Liste oder ein kleiner Textabschnitt sind. Mit einfachen Selektoren lässt sich eine solche Regel nicht definieren. Folgender Ansatz wäre demnach eine Sackgasse:

```
p {font-weight: bold;}
```

Nun würden *alle* Absätze fett dargestellt, nicht nur solche, die auch wirklich eine Warnung enthalten. Sie benötigen dagegen eine Möglichkeit, nur den Text auszuwählen, der auch wirklich eine Warnung enthält. Genauer gesagt: Es sollen nur die Elemente ausgewählt werden, die als Warnung gelten. Wie machen Sie das? Sie wenden Stildefinitionen auf Teile des Dokuments an, die auf eine bestimmte Art markiert wurden, und zwar unabhängig von den verwendeten Elementen, indem Sie Klassenselektoren verwenden.

Klassenselektoren

Die häufigste Methode, Stile zu verwenden, ohne sich dabei um die beteiligten Elemente kümmern zu müssen, besteht darin, Klassenselektoren zu benutzen. Dafür müssen Sie allerdings zunächst den Code des Dokuments ändern, damit die Klassenselektoren auch funktionieren. Und hier kommt das Attribut `class` ins Spiel:

```
<p class="warning">When handling plutonium, care must be taken to avoid  
the formation of a critical mass.</p>  
<p>With plutonium, <span class="warning">the possibility of implosion is  
very real, and must be avoided at all costs</span>. This can be accomplished  
by keeping the various masses separate.</p>
```

Um die Stildefinitionen eines Klassenselektors mit einem Element zu verbinden, müssen Sie das Attribut `class` mit dem entsprechenden Wert versehen. Im obigen Beispiel wurde zwei Elementen der `class`-Wert `warning` zugewiesen: dem ersten Absatz und dem Element `span` im zweiten Absatz.

Hierfür brauchen Sie nur zu wissen, wie Sie Ihre Stildefinitionen auf die einer Klasse zugeordneten Elemente anwenden können. In HTML-Dokumenten gibt es dafür eine sehr kompakte Schreibweise, bei der dem Namen einer Klasse (`class`) ein Punkt (`.`) vorangestellt wird. Dieser kann nun mit einem einfachen Selektor verbunden werden:

```
*.warning {font-weight: bold;}
```

Wenn Sie diese Anweisung mit dem zuvor gezeigten Markup-Beispiel kombinieren, hat diese einfache Regel den in Abbildung 2-7 gezeigten Effekt. Das heißt, die Stildefinition

font-weight: bold wird auf alle Elemente angewendet (dank des universellen Selektors), deren class-Attribut den Wert warning hat.

Plutonium

Useful for many applications, plutonium can also be dangerous if improperly handled.

Safety Information

When handling plutonium, care must be taken to avoid the formation of a critical mass.

With plutonium, **the possibility of implosion is very real, and must be avoided at all costs.** This can be accomplished by keeping the various masses separate.

Comments

It's best to avoid using plutonium **at all** if it can be avoided.

Abbildung 2-7: Die Benutzung eines Klassenselektors

Wie Sie sehen, bezieht sich ein Klassenselektor direkt auf einen Wert, der im Attribut class eines Elements zu finden ist. Dieser Referenz wird *immer* ein Punkt (.) vorangestellt, der diesen Selektor als Klassenselektor kennzeichnet. Der Punkt hilft dabei, den Klassenselektor von Dingen zu trennen, mit denen er eventuell kombiniert wird, wie etwa einem Elementselektor. Vielleicht wollen Sie bestimmte Teile nur dann fett gedruckt darstellen, wenn ein ganzer Absatz als Warnung markiert wurde:

```
p.warning {font-weight: bold;}
```

Dieser Selektor passt auf alle p-Elemente, deren class-Attribut das Wort warning enthält, aber keine anderen Elemente dieses Typs, seien diese nun klassifiziert oder nicht. Der Selektor p.warning lässt sich so übersetzen: »Ein beliebiger Absatz, dessen class-Attribut das Wort warning enthält.« Da das Element span kein Absatz ist, passt der Selektor für diese Regel nicht mehr, und es wird daher auch nicht fett gedruckt dargestellt.

Wenn Sie für das Element span einen anderen Stil verwenden wollen, könnten Sie hierfür den Selektor span.warning einsetzen:

```
p.warning {font-weight: bold;}  
span.warning {font-style: italic;}
```

In diesem Fall werden als Warnung markierte Absätze fett gedruckt dargestellt, während das als Warnung bezeichnete Element span in kursiver Schrift ausgegeben wird. Jede Regel bezieht sich also auf eine bestimmte Kombination aus Element und Klasse, ohne dabei die Darstellung anderer Elemente zu beeinflussen.

Eine weitere Option besteht in der Kombination eines allgemeinen Klassenselektors mit einem elementspezifischen Klassenselektor, wodurch die Stildefinitionen sogar noch nützlicher werden, wie im folgenden Beispiel zu sehen:

```
.warning {font-style: italic;}  
span.warning {font-weight: bold;}
```

Die Ergebnisse sehen Sie in Abbildung 2-8.

Plutonium

Useful for many applications, plutonium can also be dangerous if improperly handled.

Safety Information

When handling plutonium, care must be taken to avoid the formation of a critical mass.

With plutonium, ***the possibility of implosion is very real, and must be avoided at all costs***. This can be accomplished by keeping the various masses separate.

Comments

It's best to avoid using plutonium **at all** if it can be avoided.

Abbildung 2-8: Die Verwendung von allgemeinen und spezifischen Selektoren zum Kombinieren von Stildefinitionen

Hier werden alle Warnungen kursiv dargestellt, während nur der Text innerhalb eines span-Elements, dessen class-Attribut den Wert warning hat, kursiv und fett gedruckt ausgegeben wird.

Beachten Sie das Format des allgemeinen Klassenselektors im vorigen Beispiel: Er ist einfach ein Klassename, dem ein Punkt, aber kein Elementname vorangestellt wurde. Das heißt: Wenn Sie einfach alle Elemente mit dem gleichen Klassennamen auswählen möchten, können Sie den universellen Selektor auch weglassen, ohne dass dies irgendwelchen Schaden anrichtet.

Mehrfache Klassen

Im vorigen Abschnitt haben wir uns mit Werten für class befasst, die nur ein einzelnes Wort enthielten. In HTML ist es aber auch möglich, eine durch Leerzeichen getrennte Liste von Wörtern als Wert für class anzugeben. Möchten Sie beispielsweise ein bestimmtes Element sowohl als »dringend« (engl. urgent) und als Warnung hervorheben, können Sie Folgendes schreiben:

```
<p class="urgent warning">When handling plutonium, care must be taken to  
avoid the formation of a critical mass.</p>  
<p>With plutonium, <span class="warning">the possibility of implosion is  
very real, and must be avoided at all costs</span>. This can be accomplished  
by keeping the various masses separate.</p>
```

Die Reihenfolge der Wörter spielt hier übrigens keine Rolle; warning urgent würde genauso funktionieren.

Jetzt wollen wir alle Elemente mit dem class-Wert warning fett darstellen und solche mit dem class-Wert urgent kursiv. Elemente, die beide Werte haben, sollen außerdem einen silbergrauen Hintergrund bekommen. Das würde man folgendermaßen schreiben:

```
.warning {font-weight: bold;}
.urgent {font-style: italic;}
.warning.urgent {background: silver;}
```

Indem Sie beide Klassenselektoren miteinander verketteten, können Sie die Auswahl auf Elemente beschränken, in denen beide Klassennamen in beliebiger Reihenfolge vorkommen. Wie Sie sehen, steht im HTML-Quellcode class="urgent warning", während die Schreibweise des CSS-Selektors .warning.urgent lautet. Dennoch wird der Absatz, der mit »When handling plutonium...« beginnt, mit einem silbergrauen Hintergrund versehen (siehe Abbildung 2-9).

Plutonium

Useful for many applications, plutonium can also be dangerous if improperly handled.

Safety Information

When handling plutonium, care must be taken to avoid the formation of a critical mass.

With plutonium, **the possibility of implosion is very real, and must be avoided at all costs.** This can be accomplished by keeping the various masses separate.

Comments

It's best to avoid using plutonium **at all** if it can be avoided.

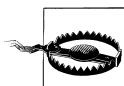
Abbildung 2-9: Auswahl von Elementen mit Hilfe von mehrfachen Klassenselektoren

Enthält der mehrfache Klassenselektor allerdings einen Namen, der sich nicht in der durch Leerzeichen getrennten Liste befindet, wird der Vergleich fehlschlagen. Sehen Sie sich einmal diese Regel an:

```
p.warning.help {background: red;}
```

Wie erwartet, passt dieser Selektor nur auf p-Elemente, deren class-Attribut die Wörter warning und help enthält. Ein p-Element mit den Wörtern warning und urgent als Wert von class wird dagegen nicht beachtet. Auf folgenden Code würde der Selektor jedoch angewendet werden:

```
<p class="urgent warning help">Help me!</p>
```



Internet Explorer hat Probleme beim Umgang mit mehrfachen Selektoren. Während ein einzelner Klassenname aus einer Liste ausgewählt werden kann, ist die Auswahl anhand mehrerer Namen in einer Liste nicht mög-

lich. Der Selektor `p.warning` würde also wie erwartet funktionieren, während der Selektor `p.warning.help` auf alle Elemente angewendet würde, deren `class`-Wert das Wort `help` enthält (weil `help` das letzte Wort im Selektor ist). Hätten Sie stattdessen den Selektor `p.help.warning` verwendet, würde Internet Explorer diesen auf alle Elemente anwenden, deren `class`-Wert das Wort `warning` enthält.

ID-Selektoren

ID-Selektoren ähneln auf viele Arten den Klassenselektoren. Es gibt allerdings ein paar entscheidende Unterschiede. Der erste Unterschied besteht darin, dass ID-Selektoren ein Doppelkreuz (`#`) an Stelle des Punkts vorangestellt wird. Sie könnten also folgende Regel schreiben:

```
*#first-para {font-weight: bold;}
```

Mit dieser Regel wird der Text aller Elemente fett dargestellt, deren `id`-Attribut den Wert `first-para` hat.

Der zweite Unterschied ist, dass ID-Selektoren sich auf Werte des Attributs `id` beziehen und nicht auf die des Attributs `class`, was nicht weiter überrascht. Hier ein Beispiel für den ID-Selektor in Aktion:

```
*#lead-para {font-weight: bold;}
```

```
<p id="lead-para">Dieser Absatz wird fett ausgegeben.</p>  
<p>Dieser Absatz wird NICHT fett dargestellt.</p>
```

Beachten Sie, dass der Wert `lead-para` einem beliebigen Element des Dokuments zugewiesen werden kann. In diesem speziellen Fall wird er dem ersten Absatz zugewiesen. Er hätte aber genauso gut auch im zweiten oder dritten Absatz benutzt werden können.

Wie bei Klassenselektoren kann auch bei ID-Selektoren der universelle Selektor weggelassen werden. Das obige Beispiel kann man also auch so schreiben:

```
#lead-para {font-weight: bold;}
```

Die Wirkung dieses Selektors wäre gleich.

Sinnvoller Einsatz von Klassen und IDs

Wie zuvor gezeigt, können Sie Klassen einer beliebigen Anzahl von Elementen zuweisen. So haben wir den Klassennamen `warning` sowohl einem `p`- als auch einem `span`-Element zugewiesen; es wären aber noch wesentlich mehr Elemente möglich gewesen. IDs werden jedoch nur genau einmal innerhalb eines HTML-Dokuments vergeben. Wenn Sie also ein HTML-Element mit dem `id`-Wert `lead-para` haben, so kann kein anderes Element diesen Wert tragen.



Im wirklichen Leben überprüfen die Browser die Einmaligkeit der IDs im HTML-Code eher selten. Sie können ein HTML-Dokument also mit einer Reihe von Elementen versehen, die alle den gleichen Wert für die ID-Attribute verwenden und auf die dadurch auch die gleiche Stilvorgabe angewendet wird. Dieses Verhalten ist zwar nicht korrekt, kommt aber trotzdem vor. Außerdem erschwert die mehrfache Verwendung gleicher ID-Werte die Benutzung von DOM-Skripten. Funktionen wie `getElementById()` verlassen sich darauf, dass es nur exakt ein Element mit einem bestimmten ID-Wert gibt.

Im Gegensatz zu Klassenselektoren können ID-Selektoren nicht kombiniert werden, weil ID-Attribute keine durch Leerzeichen getrennten Wortlisten zulassen.

Auf einer rein syntaktischen Ebene ist nicht gewährleistet, dass die Punkt-Klasse-Schreibweise (z.B. `.warning`) in XML-Dokumenten auch funktioniert. Während dieses Buch geschrieben wird, funktioniert diese Schreibweise bei Dokumenten im HTML-, SVG- und MathML-Format und wird sehr wahrscheinlich auch in zukünftige Sprachen Eingang finden. Es ist jedoch Sache der jeweiligen Spezifikation der betreffenden Sprache, dies zu entscheiden. Die Doppelkreuz-ID-Notation (z.B. `#lead`) funktioniert jedoch in jeder Dokumentensprache, die ein Attribut besitzt, mit dem die Einmaligkeit innerhalb des Dokuments sichergestellt wird. Die Einmaligkeit kann durch ein Attribut namens `id` erreicht werden oder durch irgendetwas anderes, solange der Inhalt des Attributs per Definition innerhalb des Dokuments nur ein einziges Mal verwendet wird.

Ein weiterer Unterschied zwischen den bei `class` und `id` benutzten Namen besteht darin, dass IDs bei der Ermittlung, welche Stilvorgabe für ein bestimmtes Element verwendet werden soll, höhere Priorität genießen. Auf dieses Thema werde ich im folgenden Kapitel sehr detailliert eingehen.

Wie Klassen können auch IDs unabhängig von einem bestimmten Element ausgewählt werden. Es kann passieren, dass Sie wissen, dass ein bestimmter ID-Wert in einem Dokument vorkommt, aber nicht, in welchem Element er auftaucht (wie bei den Plutonium-Warnungen). Also möchten Sie ID-Selektoren verwenden, die für sich stehen können. Stellen Sie sich vor, Sie wissen von einem Element mit dem ID-Wert `mostImportant`. Sie wissen aber nicht, ob diese überaus wichtige Sache ein Absatz ist, eine kurze Textpassage, ein Listenelement oder eine Abschnittsüberschrift. Sie wissen nur, dass dieses Element in jedem Dokument und nicht öfter als einmal vorkommt. In diesem Fall könnten Sie folgende Regel schreiben:

```
#mostImportant {color: red; background: yellow;}
```

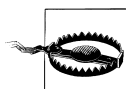
Diese Regel würde auf die folgenden Elemente passen (die, wie ich bereits zuvor gesagt habe, *nicht* im gleichen Dokument vorkommen dürfen, da sie den gleichen ID-Wert besitzen):

```
<h1 id="mostImportant">This is important!</h1>
<em id="mostImportant">This is important!</em>
<ul id="mostImportant">This is important!</ul>
```

Bedenken Sie bitte auch, dass die ID-Selektoren je nach verwendeter Dokumentensprache zwischen Groß- und Kleinschreibung unterscheiden. Für HTML und XHTML ist festgelegt, dass für die Werte von Klassen und IDs zwischen Groß- und Kleinschreibung unterschieden wird. Die Schreibweise der Werte muss also mit derjenigen im Dokument exakt übereinstimmen. Bei der folgenden Kombination von CSS und HTML wird das Element demnach nicht fett gedruckt ausgegeben:

```
p.criticalInfo {font-weight: bold;}  
  
<p class="criticalinfo">Don't look down.</p>
```

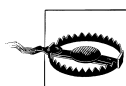
Durch die unterschiedliche Schreibweise des Buchstaben »I« passt der Selektor nicht auf das gezeigte Element.



Einige ältere Browser unterscheiden nicht zwischen Groß- und Kleinschreibung bei Klassen- und ID-Namen. Zurzeit ist diese Unterscheidung aber bei allen aktuellen Browsern korrekt implementiert.

Attributselektoren

Bei Klassen- und ID-Selektoren findet die Zuordnung eigentlich anhand der Werte von Attributen statt. Die in den vorigen zwei Abschnitten verwendete Schreibweise kommt dabei (während dieses Buch geschrieben wird) bei Dokumenten im HTML-, SVG- und MathML-Format zum Einsatz. In anderen Markup-Sprachen stehen diese Klassen- und ID-Selektoren vielleicht nicht zur Verfügung. Als Lösung für dieses Problem wurden mit CSS2 so genannte *Attributselektoren* eingeführt, die in der Lage sind, Elemente anhand ihrer Attribute und deren Werten auszuwählen. Es gibt insgesamt vier Arten von Attributselektoren.



Attributselektoren werden von Opera und von Gecko-basierten Browsern unterstützt; nicht jedoch von Internet Explorer 5 (Macintosh) bis hin zu Internet Explorer 6 (Windows).

Einfache Auswahl anhand von Attributen

Wollen Sie Elemente anhand bestimmter Attribute, aber unabhängig von deren Werten auswählen, so können Sie einen einfachen Attributselektor verwenden. In diesem Beispiel soll der Text aller mit h1 markierten Elemente, die ein class-Attribut mit beliebigem Wert besitzen, silbergrau eingefärbt werden. Hierfür können Sie Folgendes schreiben:

```
h1[class] {color: silver;}
```

Bei folgendem Code erhalten Sie das in Abbildung 2-10 gezeigte Ergebnis.

```
<h1 class="hoopla">Hello</h1>  
<h1 class="severe">Serenity</h1>  
<h1 class="fancy">Fooling</h1>
```



Abbildung 2-10: Elemente anhand ihrer Attribute auswählen

Diese Arbeitsweise ist besonders bei XML-Dokumenten nützlich, da gerade XML-Sprachen Element- und Attributnamen benutzen, die sich stark an ihrem jeweiligen Zweck orientieren. Nehmen wir als Beispiel eine XML-Sprache, mit der die Planeten des Sonnensystems beschrieben werden (wir nennen sie PlanetML). Angenommen, Sie wollen alle Elemente mit dem Namen `planet` auswählen, die ein Attribut namens `moons` besitzen. Diese Elemente sollen in fetter Schrift dargestellt werden, um die Planeten besonders hervorzuheben, die Monde besitzen. Hierfür können Sie Folgendes schreiben:

```
planet[moons] {font-weight: bold;}
```

Durch diese Regel wird der Text des zweiten und dritten Elements im unten stehenden Markup-Code fett dargestellt, das erste Element jedoch nicht:

```
<planet>Venus</planet>  
<planet moons="1">Earth</planet>  
<planet moons="2">Mars</planet>
```

In HTML-Dokumenten können Sie dieses Merkmal auf eine Vielzahl kreativer Arten verwenden. So ist es beispielsweise möglich, allen Bildern mit einem `alt`-Attribut eine bestimmte Stilvorgabe zuzuweisen und so die Bilder hervorzuheben, die korrekt formatiert sind:

```
img[alt] {border: 3px solid red;}
```

(Dieses Beispiel kann besonders für Diagnosezwecke hilfreich sein, also wenn es darum geht herauszufinden, ob die Bilder korrekt formatiert sind, und eignet sich weniger für Designzwecke.)

Wenn Sie alle Elemente fett darstellen wollen, die ein `title`-Attribut tragen (die von den meisten Browsern als eine Art Kurzbeschreibung oder »Tooltip« angezeigt werden, wenn der Mauszeiger darüber bewegt wird), könnten Sie Folgendes schreiben:

```
*[title] {font-weight: bold;}
```

Auf ähnliche Weise könnten Sie auch alle Anker (a-Elemente), die ein `href`-Attribut besitzen, mit einem eigenen Stil versehen.

Es ist auch möglich, die Auswahl von mehr als einem Attribut abhängig zu machen. Dazu müssen Sie nur die entsprechenden Attributselektoren miteinander verketteten. Um etwa den Text von Hyperlinks fett darzustellen, die sowohl ein href- als auch ein title-Attribut besitzen, könnten Sie Folgendes schreiben:

```
a[href][title] {font-weight: bold;}
```

Dadurch würde der erste Link fett hervorgehoben, der zweite und der dritte aber nicht:

```
<a href="http://www.w3.org/" title="W3C Home">W3C</a><br />
<a href="http://www.webstandards.org">Standards Info</a><br />
<a title="Not a link">das ist kein Link</a>
```

Auswahl basierend auf einem exakten Attributwert

Zusätzlich zu der Auswahl von Elementen anhand ihrer Attribute können Sie den Auswahlprozess auch auf solche Elemente beschränken, deren Attribute einen bestimmten Wert haben. Vielleicht wollen Sie nur Hyperlinks fett hervorheben, die auf einen ganz bestimmten Webserver verweisen. Das sähe dann etwa so aus:

```
a[href="http://www.css-discuss.org/about.html"] {font-weight: bold;}
```

Für alle Elemente kann die Kombination beliebiger Attribute und Werte angegeben werden. Erscheint diese Kombination jedoch nicht im Dokument, wird der Selektor auch keinen Treffer erzielen. Auch von dieser Art der Stildefinition können besonders XML-Sprachen profitieren. Wir wollen nun zu unserem PlanetML-Beispiel zurückkommen. Es sollen alle planet-Elemente ausgewählt werden, deren Attribut moons den Wert 1 trägt:

```
planet[moons="1"] {font-weight: bold;}
```

Durch diese Regel würde der Text des zweiten Elements in fetter Schrift erscheinen, während das erste und dritte Element von der Regel unberührt bleiben:

```
<planet>Venus</planet>
<planet moons="1">Earth</planet>
<planet moons="2">Mars</planet>
```

Wie bei der Auswahl anhand der Attribute ist es auch hier möglich, mehrere Attributwerte anzugeben, um ein ganz bestimmtes Element auszuwählen. Um die Textgröße von Hyperlinks zu verdoppeln, die ein href-Attribut mit dem Wert `http://www.w3.org` und ein title-Attribut mit dem Wert `W3C Home` besitzen, könnten Sie folgende Regel verwenden:

```
a[href="http://www.w3.org/"][title="W3C Home"] {font-size: 200%;}
```

Hierdurch wird die Textgröße des ersten Links im unten stehenden Markup-Fragment verdoppelt, nicht aber die des zweiten oder dritten:

```
<a href="http://www.w3.org/" title="W3C Home">W3C</a><br />
<a href="http://www.webstandards.org"
  title="Web Standards Organization">Standards Info</a><br />
<a href="http://www.example.org/" title="W3C Home">dead.link</a>
```

Die Ergebnisse sehen Sie in Abbildung 2-11.



Abbildung 2-11: Auswahl von Elementen anhand ihrer Attribute und deren Werten

Bedenken Sie, dass für dieses Format eine exakte Entsprechung für den Wert des Attributs notwendig ist. Das Auffinden der richtigen Treffer wird wichtig, sobald die Werte wiederum durch Leerzeichen getrennte Wertelisten enthalten können (wie beispielsweise das HTML-Attribut `class`). Sehen Sie sich hierfür einmal folgendes Markup-Fragment an:

```
<planet type="barren rocky">Mercury</planet>
```

Die einzige Möglichkeit, um dieses Element basierend auf seinem exakten Attributwert zu finden, besteht in folgender Formulierung:

```
planet[type="barren rocky"] {font-weight: bold;}
```

Hätten Sie jedoch `planet[type="barren"]` geschrieben, hätte die Regel unser Beispiel-Markup nicht gefunden und wäre demnach fehlgeschlagen. Das gilt sogar für das HTML-Attribut `class`. Noch ein Beispiel:

```
<p class="urgent warning">When handling plutonium, care must be taken to  
avoid the formation of a critical mass.</p>
```

Um dieses Element anhand seines exakten Attributwerts auszuwählen, müssten Sie Folgendes schreiben:

```
p[class="urgent warning"] {font-weight: bold;}
```

Hierbei handelt es sich nicht um die Punkt-Klasse-Schreibweise, über die wir zuvor gesprochen haben.

Es ist wichtig, zwischen ID-Selektoren und Attributselektoren, die sich auf das `id`-Attribut beziehen, korrekt zu unterscheiden. Es gibt einen subtilen, aber schwer wiegenden Unterschied zwischen `h1#page-title` und `h1[id="page-title"]`, der im folgenden Kapitel erklärt wird.

Auswahl basierend auf partiellen Attributwerten

Bei allen Attributen, die eine durch Leerzeichen getrennte Wortliste als Wert besitzen können, ist es möglich, die Auswahl vom Vorkommen eines dieser Wörter abhängig zu machen. Auch hier ist das klassische Beispiel das HTML-Attribut `class`, bei dem mehr als ein Wort als Wert angegeben werden kann. Hier noch einmal unser übliches Beispiel:

```
<p class="urgent warning">When handling plutonium, care must be taken to  
avoid the formation of a critical mass.</p>
```

Um alle Elemente auszuwählen, deren Attribut `class` das Wort `warning` enthält, können Sie folgenden Selektor verwenden:

```
p[class~="warning"] {font-weight: bold;}
```

Die Tilde (~) spielt in diesem Selektor eine wichtige Rolle. Sie ist der Schlüssel zur Auswahl anhand eines bestimmten Wortes in einer als Attributwert benutzten Wortliste. Lassen Sie die Tilde weg, haben Sie wieder die Suche nach einem exakten Treffer, wie im vorigen Abschnitt besprochen.

Diese Form des Selektors entspricht der Punkt-Klasse-Schreibweise, die wir bereits besprochen haben. Die Formulierungen `p.warning` und `p[class~="warning"]` haben die gleiche Bedeutung, wenn sie auf HTML-Dokumente angewendet werden. Wir wollen einen weiteren Blick auf unser XML-Beispiel von vorhin werfen:

```
<planet type="barren rocky">Mercury</planet>
<planet type="barren cloudy">Venus</planet>
<planet type="life-bearing cloudy">Earth</planet>
```

Um alle Elemente, deren Attribut `type` das Wort `barren` enthält, kursiv auszugeben, schreiben Sie:

```
planet[type~="barren"] {font-style: italic;}
```

Der Selektor dieser Regel passt auf die ersten zwei Elemente im Beispiel-XML-Code und sorgt dafür, dass ihr Text kursiv dargestellt wird (siehe Abbildung 2-12).

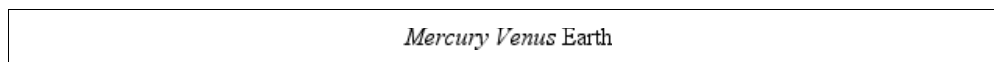


Abbildung 2-12: Die Auswahl von Elementen basierend auf Teilen des Attributwerts

Diese Art von Attributselektor kann sogar in HTML nützlich sein. Vielleicht haben Sie ein Dokument mit Grafiken, von denen aber nur einige als »Abbildungen« gelten. Wenn Sie den Selektor für partielle Werte auf den `title`-Text anwenden, ist es möglich, nur die Grafiken auszuwählen, die eine Abbildung enthalten:

```
img[title~="Abbildung"] {border: 1px solid gray;}
```

Diese Regel wählt alle Bilder aus, deren `title`-Text das Wort `Abbildung` enthält. Solange der Text im `title`-Attribut etwas enthält wie »Abbildung 4. Ein glatzköpfiger alter Staatsmann«, wird die obige Regel auf diese Bilder angewendet. Deshalb wird die Regel `img[title~="Abbildung"]` auch einen Treffer erzielen, wenn der `title`-Wert etwa »Der Unterschied zwischen einer Abbildung und einem Schaubild« lautet. Bilder, die kein `title`-Attribut besitzen oder deren `title`-Wert nicht das Wort »Abbildung« enthält, werden von der Regel nicht gefunden.

Eine Sonderform des Attributselektors

Es gibt noch eine vierte Art von Attributselektor, der eine gewisse Besonderheit aufweist. Es ist leichter, diesen Selektor zu zeigen, als ihn zu erklären. Nehmen Sie die folgende Regel:

```
*[lang="en"] {color: white;}
```

Diese Regel wählt Elemente aus, deren Attribut lang dem Wert en entspricht oder mit den Buchstaben en beginnt. Dementsprechend würden die ersten drei Elemente aus dem unten stehenden Beispiel-Markup ausgewählt, während die letzten beiden nicht beachtet würden:

```
<h1 lang="en">Hello!</h1>
<p lang="en-us">Greetings!</p>
<div lang="en-au">G'day!</div>
<p lang="fr">Bonjour!</p>
<h4 lang="cy-en">Jrooana!</h4>
```

Im Allgemeinen kann die Form [attribut|= "wert"] für beliebige Attribute und deren Werte benutzt werden. Nehmen wir einmal an, Sie hätten eine Reihe von Abbildungen in einem HTML-Dokument, die alle Dateinamen tragen wie *abbildung-1.gif* und *abbildung-3.jpg*. Alle diese Bilder können von folgendem Selektor gefunden werden:

```
img[src|= "abbildung"] {border: 1px solid gray;}
```

Am häufigsten wird dieser Attributselektor verwendet, um auf natürliche Sprachen bezogene Werte zu finden, wie wir später in diesem Kapitel sehen werden.

Die Verwendung der Dokumentstruktur

Wie ich bereits erwähnt habe, ist CSS so mächtig, weil es die Struktur von HTML-Dokumenten benutzt, um die passenden Stilvorgaben zu finden und diese anzuwenden. Das ist aber nur ein Teil des Ganzen, da CSS die Dokumentstruktur bei weitem nicht nur auf diese Weise nutzt. Die Struktur spielt eine wesentlich größere Rolle bei der Anwendung von Stildefinitionen auf ein Dokument. Wir wollen uns jetzt einen Moment Zeit nehmen, um diese Struktur genau zu betrachten, bevor wir uns mit mächtigeren Auswahlmöglichkeiten befassen.

Die Eltern-Kind-Beziehung verstehen

Um die Beziehung zwischen Selektoren und Dokumenten zu verstehen, müssen Sie ein weiteres Mal untersuchen, wie Dokumente strukturiert sind. Als Beispiel soll uns dieses sehr einfache HTML-Dokument dienen:

```
<html>
<head>
  <base href="http://www.meerkat.web/">
  <title>Meerkat Central</title>
</head>
<body>
  <h1>Meerkat <em>Central</em></h1>
  <p>
    Welcome to Meerkat <em>Central</em>, the <strong>best meerkat web site
    on <a href="inet.html">the <em>entire</em> Internet</a></strong>!</p>
  <ul>
    <li>We offer:
  </ul>
```

```

<ul>
<li><strong>Detailed information</strong> on how to adopt a meerkat</li>
<li>Tips for living with a meerkat</li>
<li><em>Fun</em> things to do with a meerkat, including:
<ol>
<li>Playing fetch</li>
<li>Digging for food</li>
<li>Hide and seek</li>
</ol>
</li>
</ul>
</li>
<li>...and so much more!</li>
</ul>
<p>
Questions? <a href="mailto:suricate@meerkat.web">Contact us!</a>
</p>
</body>
</html>

```

Ein großer Teil der Fähigkeiten von CSS basiert auf der *Eltern-Kind-Beziehung* der Elemente. HTML-Dokumente (eigentlich sogar die meisten strukturierten Dokumente) basieren auf einer Hierarchie der Elemente, die am einfachsten durch ein Baumdiagramm des Dokuments verdeutlicht wird (Abbildung 2-13). In dieser Hierarchie hat jedes Element seinen Platz in der Gesamtstruktur des Dokuments. Jedes Element ist entweder das *Elternelement* oder das *Kindelement* eines anderen Elements und oftmals sogar beides auf einmal.

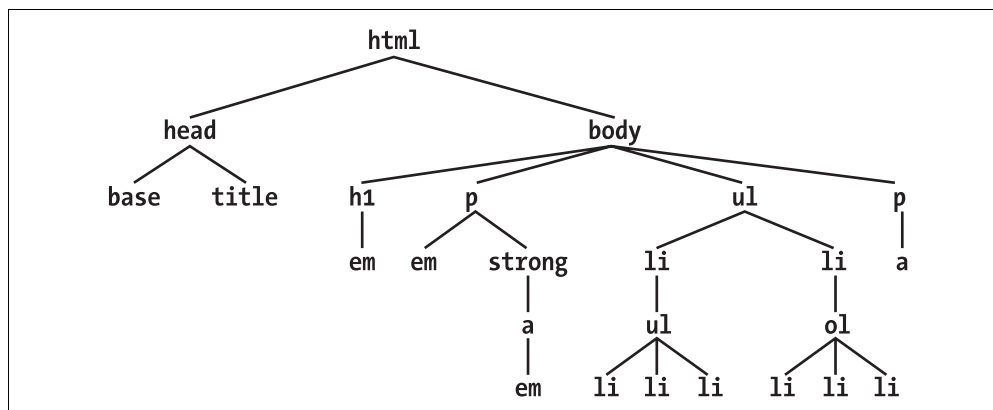


Abbildung 2-13: Die Baumstruktur eines Dokuments

Man spricht von einem Elternelement, wenn sich dieses in der Dokumenthierarchie direkt oberhalb eines anderen Elements befindet. So ist in Abbildung 2-13 das erste p-Element ein Elternelement für ein em- und ein strong-Element, während strong seinerseits ein Elternelement für ein Anker-Element (a) ist. Der Anker ist wiederum ein Elternelement für ein em-Element. Andererseits spricht man von einem Kindelement, wenn dieses

in der Hierarchie direkt unterhalb eines anderen Elements steht. Daraus folgt: Das Anker-Element in Abbildung 2-13 ist ein Kindelement von `strong`, das wiederum ein Kindelement des Absatzes ist und so weiter.

Die Bezeichnungen Eltern und Kind sind spezifische Anwendungen der Begriffe *Vorfahre* und *Nachfahre*. Es gibt einen wichtigen Unterschied zwischen diesen beiden Begriffspaaren: Wenn sich ein Element genau eine Ebene über oder unter einem anderen Element befindet, haben diese beiden Elemente eine Eltern-Kind-Beziehung. Durchläuft der Pfad von einem Element zu einem anderen jedoch zwei oder mehr Ebenen, haben diese zwei Elemente zwar eine Vorfahre-Nachfahre-Beziehung, aber kein Eltern-Kind-Verhältnis. (Selbstverständlich sind Eltern auch Vorfahren und Kinder auch Nachfahren.) In Abbildung 2-13 ist das erste `ul`-Element ein Elternelement für zwei `li`-Elemente. Gleichzeitig ist das `ul` aber auch ein Vorfahre für alle Elemente, die von seinen `li`-Elementen »abstammen«, bis hinunter zu den am tiefsten verschachtelten `li`-Elementen.

In Abbildung 2-13 findet sich außerdem ein Anker-Element, das ein Kindelement von `strong` ist, gleichzeitig aber auch ein Nachfahre des Absatz-Elements (`p`) sowie der `body`- und `html`-Elemente. Das Element `body` ist ein Vorfahre für alles, was der Browser standardmäßig anzeigt; das Element `html` ist der Vorfahre für das gesamte Dokument. Deshalb wird das Element `html` auch als *Wurzelement* (root element) bezeichnet.

Selektoren für Nachfahren

Der erste Vorteil, der sich aus dem Verständnis dieses Modells ergibt, ist die Möglichkeit, *Selektoren für Nachfahren* (auch als Kontext-Selektoren bezeichnet) zu definieren. Die Definition von Selektoren für Nachfahren besteht aus dem Festlegen von Regeln, die nur für bestimmte hierarchische Strukturen gelten. Als Beispiel wollen wir diejenigen `em`-Elemente mit einer Stildefinition versehen, die von `h1`-Elementen »abstammen«. Hierfür könnten Sie jetzt jedes `em`, das sich innerhalb eines `h1`-Elements befindet, mit einem `class`-Attribut versehen. Das wäre aber fast genauso zeitaufwändig wie die Benutzung von `font`-Tags. Es ist offensichtlich wesentlich effizienter, eine Regel zu definieren, die nur auf `em`-Elemente angewendet wird, die sich innerhalb von `h1`-Elementen befinden.

Dazu schreiben Sie Folgendes:

```
h1 em {color: gray;}
```

Diese Regel sorgt dafür, dass der gesamte Text eines `em`-Elements grau dargestellt wird – vorausgesetzt, es handelt sich um einen Nachkommen eines `h1`-Elements. Andere mit `em` markierte Textteile, die sich etwa in Absätzen oder einem Zitat befinden, werden von dieser Regel nicht ausgewählt. Abbildung 2-14 verdeutlicht das.



Abbildung 2-14: Die Auswahl eines Elements basierend auf seinem Kontext

Bei einem Selektor für Nachkommen besteht die linke (Selektoren-)Seite aus zwei oder mehr durch Leerzeichen getrennten Selektoren. Der Zwischenraum zwischen den beiden Selektoren ist ein Beispiel für einen so genannten *Kombinator*. Der Leerzeichen-Kombinator kann gelesen werden als »zu finden innerhalb von« oder »ist Teil von« oder auch »ist ein Nachfahre von«, aber nur, wenn Sie die Regel von rechts nach links lesen. Die Formulierung `h1 em` lautet also übersetzt: »Ein beliebiges `em`-Element, das ein Nachfahre eines `h1`-Elements ist«. (Um den Selektor andersherum zu lesen, könnten Sie etwa sagen »Bei jedem `h1`, das `em`-Elemente enthält, sollen die folgenden Stilvorgaben für `em` verwendet werden«.)

Hierbei sind Sie selbstverständlich nicht auf zwei Selektoren beschränkt, wie hier gezeigt:

```
ul ol ul em {color: gray;}
```

Wie in Abbildung 2-15 zu sehen, werden mit dieser Regel `em`-Elemente grau dargestellt, sofern sie Teil einer ungeordneten Liste sind, die ihrerseits Teil einer geordneten Liste ist, die wiederum Teil einer ungeordneten Liste ist (ja, das stimmt schon so). Das ist offensichtlich ein sehr spezielles Auswahlkriterium.

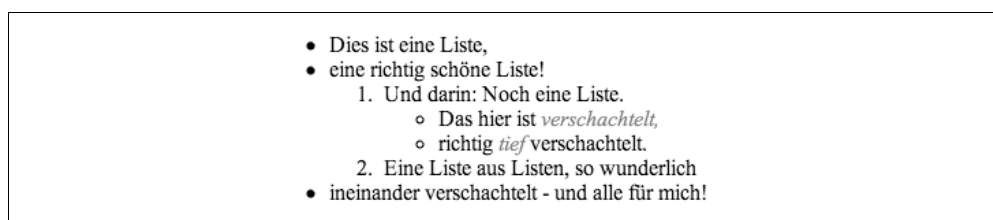


Abbildung 2-15: Ein sehr spezieller Selektor für Nachfahren

Selektoren für Nachfahren können sehr mächtig sein. Sie ermöglichen Dinge, die in HTML nie möglich waren – zumindest nicht ohne Unmengen von `font`-Tags. Lassen Sie uns ein häufig vorkommendes Beispiel betrachten: Stellen Sie sich vor, Sie haben ein Dokument mit einer Seitenleiste und einem Hauptbereich. Die Seitenleiste hat einen blauen Hintergrund, während der Hauptteil einen weißen Hintergrund hat; in beiden befinden sich Links. Sie können nicht einfach alle Links blau einfärben, weil die Links in der Seitenleiste dadurch unlesbar würden.

Die Lösung lautet: Selektoren für Nachfahren. In diesem Fall weisen Sie der Zelle, in der sich Ihre Seitenleiste befindet, die Klasse `sidebar` zu, während dem Hauptbereich die Klasse `main` zugewiesen wird. Nun können Sie folgende Stildefinitionen verwenden:

```
td.sidebar {background: blue;}
td.main {background: white;}
td.sidebar a:link {color: white;}
td.main a:link {color: blue;}
```

Das Ergebnis sehen Sie in Abbildung 2-16.



Abbildung 2-16: Die Verwendung von Selektoren für Nachfahren, um gleichartigen Elementen verschiedene Stile zuzuweisen



Die Bezeichnung `:link` bezieht sich auf Ressourcen, die noch nicht besucht wurden. Wir werden uns später in diesem Kapitel genauer damit befassen.

Hier ein weiteres Beispiel: Nehmen wir an, Sie wollen allen Text innerhalb von `b`-Elementen grau einfärben, der sich innerhalb von `blockquote`-Tags oder normalen Absätzen befindet:

```
blockquote b, p b {color: gray;}
```

Als Ergebnis werden `b`-Elemente, die von Absätzen oder `blockquotes` abstammen, grau dargestellt.

Ein leicht zu übersehender Aspekt der Selektoren für Nachfahren besteht darin, dass die beiden Elemente im Prinzip unendlich weit voneinander entfernt sein können. Wenn Sie beispielsweise `ul em` schreiben, wählt diese Schreibweise jedes `em`-Element aus, das von einem `ul`-Element abstammt, egal wie tief verschachtelt `em` hierbei ist. Deshalb würde die Formulierung `ul em` auch das `em` im folgenden Markup-Beispiel finden:

```
<ul>
  <li>Listenelement 1
  <ol>
    <li>Listenelement 1-1</li>
    <li>Listenelement 1-2</li>
    <li>Listenelement 1-3
      <ol>
        <li>Listenelement 1-3-1</li>
        <li>Listenelement <em>1-3-2</em></li>
        <li>Listenelement 1-3-3</li>
      </ol></li>
    <li>Listenelement 1-4</li>
  </ol></li>
</ul>
```

Kindelemente auswählen

In manchen Fällen ist es nicht wünschenswert, ein beliebiges abstammendes Element auszuwählen. Stattdessen wollen Sie den Auswahlbereich nur auf direkte Nachkommen, also

Kindelemente, einschränken. So soll hier nur dann ein `strong`-Element ausgewählt werden, wenn es ein Kindelement (also kein einfacher Nachfahre) eines `h1`-Elements ist. Hierfür verwenden Sie das Kombinatorzeichen für Kindselektoren, das Größer-als-Zeichen (`>`):

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

Aufgrund der oben stehenden Regel wird das `strong`-Element innerhalb der ersten `h1`-Zeile rot eingefärbt, nicht aber das in der zweiten Zeile:

```
<h1>This is <strong>very</strong> important.</h1>  
<h1>This is <em>really <strong>very</strong></em> important.</h1>
```

Von rechts nach links gelesen, bedeutet dieser Selektor: »Wähle `strong`-Elemente, die Kindelemente eines `h1`-Elements sind.« Der Kind-Kombinator kann optional von Leerzeichen umgeben sein. Die Schreibweise `h1 > strong` ist also gleichbedeutend mit `h1> strong` und `h1>strong`. Sie können Leerzeichen nach eigenem Belieben benutzen oder weglassen.

Wenn wir noch einmal einen Blick auf die Baumstruktur des Dokuments werfen, wird schnell klar, dass ein Kindselektor seine Treffer auf Elemente beschränkt, die im Baum direkt miteinander verbunden sind. In Abbildung 2-17 sehen Sie einen Teil eines Dokumentbaums.

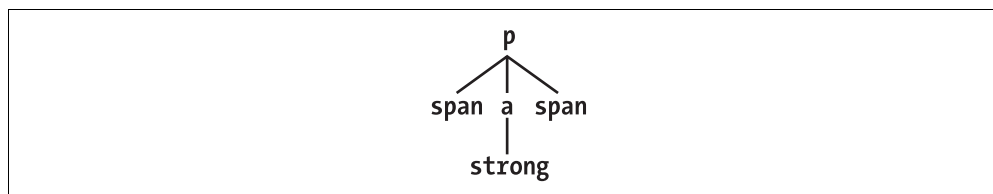


Abbildung 2-17: Ein Teil eines Dokumentbaums

In diesem Baumfragment lassen sich leicht die Eltern-Kind-Beziehungen erkennen. So ist `a` ein Elternelement für `strong`, aber gleichzeitig Kindelement für das `p`-Element. Auf diese Elemente würden daher die Selektoren `p > a` und `a > strong` passen, nicht aber `p > strong`, weil `strong` zwar ein Nachfahre von `p` ist, aber nicht dessen Kind.

Kombinatoren für Nachfahren und Kindelemente können übrigens im gleichen Selektor gemeinsam verwendet werden. `table.summary td > p` sorgt also für die Auswahl von `p`-Elementen, die Kindelemente eines `td`-Elements sind. Hierbei muss `td` von einem `table`-Element abstammen, dessen Attribut `class` das Wort `summary` enthält.

Benachbarte Geschwisterelemente auswählen

Nehmen wir an, Sie wollen einen Stil für einen Absatz definieren, der direkt auf eine Überschrift folgt, oder eine Liste mit einem besonderen Rand versehen, wenn diese direkt auf einen Absatz folgt. Um ein Element auszuwählen, das direkt auf ein anderes Element folgt und dabei vom gleichen Elternelement abstammt, benutzen Sie das *Kombinatorzeichen für Selektoren für benachbarte Elemente* (adjacent sibling combinator), das als ein

Pluszeichen (+) dargestellt wird. Wie beim Kindselektor-Kombinatorzeichen, können links und rechts des Zeichens beliebig viele Leerzeichen eingefügt werden.

Um den oberen Rand von einem Absatz zu entfernen, der direkt auf ein h1-Element folgt, schreiben Sie:

```
h1 + p {margin-top: 0;}
```

Dieser Selektor liest sich: »Wähle einen beliebigen Absatz aus, der direkt auf ein h1-Element folgt, das das gleiche Elternelement wie der Absatz besitzt.«

Um sich leichter vorstellen zu können, wie dieser Selektor funktioniert, ist es auch hier am einfachsten, sich einen Teil des Dokumentbaums anzusehen (siehe Abbildung 2-18).

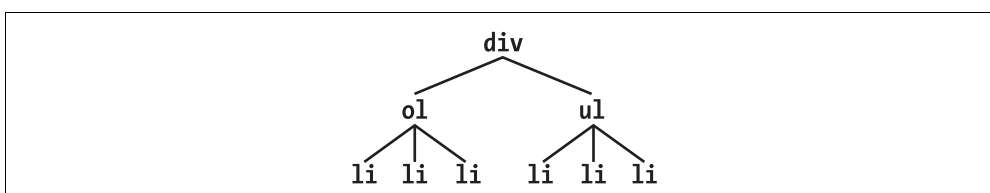


Abbildung 2-18: Ein weiteres Dokumentbaum-Fragment

In diesem Fragment stammen eine geordnete und eine ungeordnete Liste von einem div-Element ab. Beide Listen enthalten jeweils drei Elemente. Beide Listen haben ein nachbarschaftliches Verhältnis und dabei das gleiche Elternelement. Genauso verhält es sich mit den einzelnen Einträgen der beiden Listen, die ebenfalls benachbarte Kinder gleicher Eltern sind. Dagegen sind die Einträge der ersten Liste *keine* benachbarten Geschwister der Einträge der zweiten Liste, da sie verschiedene Eltern haben. (Sie sind höchstens Cousins oder Cousinen.)

Bedenken Sie, dass Sie nur das zweite von zwei benachbarten Geschwistern mit einem einzelnen Kombinator auswählen können. Wenn Sie also `li + li {font-weight: bold;}` schreiben, werden nur der zweite und der dritte Listeneintrag fett dargestellt (siehe Abbildung 2-19).

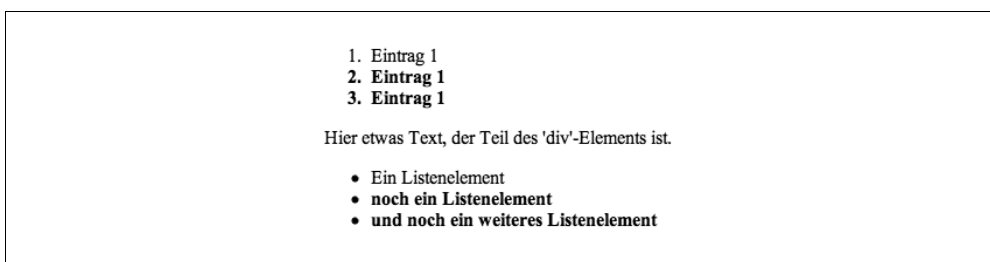


Abbildung 2-19: Benachbarte Geschwister auswählen

Um richtig zu funktionieren, verlangt CSS, dass die beiden Elemente in »Quellcode-Reihenfolge« aufgeführt werden. In unserem Beispiel folgt ein ul- auf ein ol-Element. Sie

könnten das zweite Element daher mit der Formulierung `ol + ul` auswählen, aber nicht das erste. Um mit der Schreibweise `ul + ol` einen Treffer zu erzielen, müsste eine geordnete Liste direkt auf eine ungeordnete Liste folgen.

Steht zwischen zwei benachbarten Kindelementen Text, so verhindert dieser nicht, dass das Kombinatorzeichen für Selektoren für benachbarte Elemente funktioniert. Sehen Sie sich einmal folgendes Markup-Fragment an, dessen Baumansicht Abbildung 2-18 entspricht:

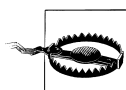
```
<div>
<ol>
<li>Eintrag 1</li>
<li>Eintrag 1</li>
<li>Eintrag 1</li>
</ol>
Hier etwas Text, der Teil des 'div'-Elements ist.
<ul>
<li>Ein Listenelement</li>
<li>noch ein Listenelement</li>
<li>und noch ein weiteres Listenelement</li>
</ul>
</div>
```

Obwohl sich zwischen den beiden Listen Text befindet, kann die zweite Liste mit dem Selektor `ol + ul` gefunden werden, weil der Text nicht in einem eigenen Kindelement steht, sondern Teil des Elternelements `div` ist. Hätten Sie den Text in einen eigenen Absatz gepackt, würde die Schreibweise `ol + ul` nicht funktionieren. Stattdessen bräuchten Sie nun eine Formulierung wie `ol + p + ul`.

Wie das folgende Beispiel zeigt, kann der Nachbarschafts-Kombinator mit anderen Kombinatoren gemeinsam verwendet werden:

```
html > body table + ul{margin-top: 1.5em;}
```

Dieser Selektor liest sich: »Wähle ein `ul`-Element, das direkt auf ein benachbartes `table`-Element folgt, das von einem `body`-Element abstammt, welches wiederum ein direkter Nachkomme (Kind) eines `html`-Elements ist.«



Internet Explorer für Windows inklusive der Version IE6 (die beim Schreiben dieses Buches aktuellste Version) bietet keine Unterstützung für Kind- und Nachbarschaftsselectoren.

Pseudoklassen und Pseudoelemente

Richtig interessant werden die Dinge erst mit Selektoren für *Pseudoklassen* und *Pseudo-elemente*. Mit diesen Selektoren können Sie Stildefinitionen auf Strukturen zuweisen, die nicht unbedingt im Dokument vorkommen müssen, oder auch auf Phantomklassen, die vom Zustand bestimmter Elemente oder sogar vom Zustand des Dokuments selbst

abhängen. Anders gesagt, werden die Stile, basierend auf etwas anderem als der Struktur des Dokuments, auf Teile des Dokuments angewendet. Die Art der Anwendung kann hierbei nicht durch ein Studium des Dokument-Markup abgeleitet werden.

Das klingt, als ob ich Stile jetzt nach dem Zufallsprinzip anwenden wollte; das ist aber nicht der Fall. Stattdessen richtet sich die Zuweisung der Stildefinitionen nach anderen, flüchtigen Bedingungen, die sich nicht voraussagen lassen. Allerdings sind die Umstände sehr genau definiert, unter denen die Stildefinitionen auftauchen. Stellen Sie sich das so vor: Immer wenn während eines Fußballspiels die Heimatmannschaft ein Tor schießt, werden die Fans jubeln. Sie wissen aber nicht genau, wann das passiert, sondern nur, dass die Fans wie vorausgesagt jubeln werden.

Selektoren für Pseudoklassen

Lassen Sie uns mit den Selektoren für Pseudoklassen beginnen, für die es eine bessere Browser-Unterstützung gibt und die daher auch häufiger benutzt werden.

Für dieses Beispiel benutzen wir ein Anker-Element (a), das in HTML und XHTML einen Link von einem Dokument zu einem anderen herstellt. Manche Anker beziehen sich auf Seiten, die bereits besucht wurden, während andere Anker auf Seiten zeigen, die erst noch besucht werden müssen. Durch ein einfaches Betrachten des HTML-Codes lässt sich jedoch kein Unterschied feststellen; alle Anker haben das gleiche Aussehen. Die einzige Möglichkeit, um festzustellen, welche Links bereits besucht wurden, besteht darin, sie mit den Links im »Verlauf« des Browsers zu vergleichen. Es gibt also zwei grundsätzliche Arten von Ankern: besuchte und nicht besuchte. Diese Typen werden als *Pseudoklassen* bezeichnet und die dazugehörigen Selektoren heißen *Pseudoklassen-Selektoren*.

Um diese Klassen und Selektoren besser zu verstehen, sollten Sie sich vor Augen führen, wie Browser sich bezüglich Links verhalten. Nach den Konventionen von Mosaic wurde einmal festgelegt, dass Seiten, die noch nicht besucht wurden, blau markiert werden sollen, während bereits besuchte Links rot angezeigt werden sollten (in späteren Browsern wie Internet Explorer wurde aus Rot Lila). Wenn Sie jetzt eigene Klassen für die unterschiedlichen Anker hätten, könnten Sie festlegen, dass Links der Klasse »besucht« (visited) rot eingefärbt werden sollen:

```
a.visited {color: red;}
```

```
<a href="http://www.w3.org/" class="visited">W3C Website</a>
```

Dieser Ansatz erfordert allerdings, dass die Klassen für die Anker sich jedes Mal verändern, sobald Sie eine neue Seite besucht haben, was ein wenig albern ist. Stattdessen definiert CSS Pseudoklassen, die dafür sorgen, dass Anker für besuchte Seiten sich verhalten, als gehörten sie der Klasse »besucht« an:

```
a:visited {color: red;}
```

Jetzt wird jeder Anker, der auf eine besuchte Seite verweist, rot eingefärbt, ohne dass Sie die Anker mit class-Attributen versehen müssten. Beachten Sie den als Trennzeichen ver-

wendeten Doppelpunkt (:) zwischen a und visited, der für die Kennzeichnung von Pseudoklassen und Pseudoelementen verwendet wird.

Pseudoklassen für Links

In CSS2.1 werden zwei Pseudoklassen definiert, die sich ausschließlich auf Hyperlinks beziehen. In HTML sowie XHTML 1.0 und 1.1 sind dies alle a-Elemente, die ein href-Attribut besitzen; in XML-Sprachen wären dies alle Elemente, die als Link auf eine andere Ressource verwendet werden. Diese beiden Pseudoklassen werden in Tabelle 2-1 beschrieben.

Tabelle 2-1: Pseudoklassen für Links

Name	Beschreibung
:link	Bezieht sich auf Anker, die als Hyperlink benutzt werden (d.h. ein href-Attribut besitzen) und noch nicht besucht wurden. Vorsicht: Manche Browser beziehen :link fälschlicherweise auf beliebige Hyperlinks, besucht oder nicht.
:visited	Bezieht sich auf Anker, die als Hyperlink verwendet werden und auf eine bereits besuchte Adresse verweisen.

Die erste der beiden Pseudoklassen in Tabelle 2-1 kommt Ihnen vielleicht redundant vor. Schließlich ist ein Anker, der noch nicht besucht wurde, offensichtlich »unbesucht«, oder? Wenn das der Fall ist, bräuchten wir eigentlich nur etwas wie das hier:

```
a {color: blue;}
a:visited {color: red;}
```

Auch wenn dieses Format auf den ersten Blick völlig ausreichend erscheint, ist es das aber nicht. Die erste hier gezeigte Regel bezieht sich nämlich nicht nur auf unbesuchte Links, sondern verweist auch auf Anker wie den folgenden:

```
<a name="section4">4. The Lives of Meerkats</a>
```

Der ausgegebene Text würde blau eingefärbt, weil die Regel `a {color: blue;}` auch auf das einfache Element `a` zutrifft. Um zu vermeiden, dass Ihre Stildefinitionen auch auf Anker angewendet werden, die keine Hyperlinks sind, benutzen Sie die Pseudoklasse `:link`.

```
a:link {color: blue;} /* unbesuchte Links sind blau */
a:visited {color: red;} /* besuchte Links sind rot */
```

Vermutlich ist es Ihnen schon aufgefallen: Die Pseudoklassen `:link` und `:visited` haben die gleiche Funktion wie die `body`-Attribute `link` und `vlink`. Nehmen wir an, ein Autor möchte die Anker unbesuchter Seiten lila und die Anker für besuchte Seiten silbergrau darstellen lassen. In HTML 3.2 könnte er das so angeben:

```
<body link="purple" vlink="silver">
```

In CSS erreichen Sie den gleichen Effekt mit:

```
a:link {color: purple;}
a:visited {color: silver;}
```

Im Fall der CSS-Pseudoklassen können Sie selbstverständlich wesentlich mehr ändern als nur die Farben. Vielleicht wollen Sie besuchte Links kursiv darstellen, die neben ihrer silbergrauen Farbe auch noch durchgestrichen werden sollen, wie in Abbildung 2-20 gezeigt.



Abbildung 2-20: Mehrere Stildefinitionen auf einen besuchten Link anwenden

Dafür benötigen Sie nur die folgende Regel:

```
a:visited {color: silver; text-decoration: line-through; font-style: italic;}
```

Diese Stelle bietet sich an, um einen weiteren Blick auf die Klassenselektoren zu werfen und zu zeigen, wie diese mit Pseudoklassen gemeinsam verwendet werden können. Vielleicht wollen Sie die Farbe von Links ändern, die auf Seiten außerhalb Ihrer eigenen Website verweisen. Wenn Sie jeden dieser Anker mit einem Klassenattribut versehen, ist das einfach:

```
<a href="http://www.mysite.net/">My home page</a>  
<a href="http://www.site.net/" class="external">Another home page</a>
```

Um dem externen Link einen eigenen Stil zuzuweisen, brauchen Sie nur eine Regel wie diese hier:

```
a.external:link, a.external:visited {color: maroon;}
```

Diese Regel färbt den zweiten Link im obigen Markup-Code rotbraun ein, während der erste Anker die Standardfarbe für Hyperlinks behält (üblicherweise Blau).

Die gleiche allgemeine Schreibweise wird auch für ID-Selektoren benutzt:

```
a#footer-copyright:link{font-weight: bold;}  
a#footer-copyright:visited {font-weight: normal;}
```

Obwohl `:link` und `:visited` sehr nützlich sind, bleiben sie statisch, d.h., nachdem ein Dokument einmal angezeigt wurde, ändern sie den Stil eines Dokuments nicht mehr. In CSS2.1 gibt es jedoch einige Pseudoklassen, die nicht ganz so statisch sind und mit denen wir uns als Nächstes befassen wollen.

Dynamische Pseudoklassen

CSS2.1 definiert drei Pseudoklassen, die das Erscheinungsbild eines Dokuments als Reaktion auf bestimmte Aktionen des Benutzers verändern können. Diese dynamischen Pseudoklassen wurden traditionell dazu verwendet, um Hyperlinks mit Stilinformationen zu versehen; ihre Möglichkeiten sind jedoch noch wesentlich weiter gefasst. In Tabelle 2-2 finden Sie eine Aufstellung dieser Pseudoklassen.

Tabelle 2-2: Dynamische Pseudoklassen

Name	Beschreibung
:focus	Bezieht sich auf ein Element, das gerade den Eingabefokus besitzt. Das heißt, das Element kann Eingaben von der Tastatur entgegennehmen oder sonst wie aktiviert werden.
:hover	Bezieht sich auf ein Element, über dem gerade der Mauszeiger platziert wurde, z. B. ein Hyperlink, über dem sich gerade der Cursor befindet.
:active	Bezieht sich auf ein Element, das durch eine Benutzereingabe aktiviert wurde, z. B. einen Hyperlink, auf den ein Benutzer gerade klickt, während die Maustaste heruntergedrückt ist.

Wie `:link` und `:visited` sind diese Pseudoklassen am beliebtesten im Zusammenhang mit Hyperlinks. Viele Webseiten benutzen Stildefinitionen wie diese:

```
a:link {color: navy;}
a:visited {color: gray;}
a:hover {color: red;}
a:active {color: yellow;}
```

Die ersten beiden Regeln verwenden Pseudoklassen für Links, während die letzten beiden mit dynamischen Pseudoklassen arbeiten. `:active` verhält sich hierbei analog zum Attribut `alink` in HTML 3.2. Bei der Verwendung der Pseudoklasse können Sie aktiven Links auf diese Weise neben einer anderen Farbe eine beliebige Anzahl eigener Stildefinitionen zuweisen.



Die Reihenfolge der Pseudoklassen ist wichtiger, als es auf den ersten Blick scheint. Die übliche Empfehlung lautet »link-visited-hover-active«, wurde aber inzwischen geändert in »link-visited-focus-hover-active«. Das kommende Kapitel erklärt, warum diese Reihenfolge so wichtig ist, und zeigt auf, warum Sie die empfohlene Reihenfolge vielleicht ändern oder sogar komplett ignorieren wollen.

Die dynamischen Pseudoklassen können übrigens auf beliebige Elemente angewendet werden. Das ist besonders praktisch, wenn Sie dynamische Stile vielleicht auch für Elemente benutzen möchten, die keine Links sind. Nehmen wir beispielsweise folgende Regel:

```
input:focus {background: silver; font-weight: bold;}
```

Mit diesem Code können Sie ein Formularelement hervorheben, das bereit ist, Tastatureingaben entgegenzunehmen, wie in Abbildung 2-21 gezeigt.

Name

Title

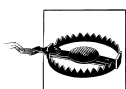
E-mail

Abbildung 2-21: Ein Formularelement hervorheben, das gerade den Fokus hat

Durch die Anwendung von Pseudoklassen auf beliebige Elemente lassen sich auch einige seltsame Dinge anstellen. Vielleicht möchten Sie Ihre Leser durch folgenden Code mit einem Hervorhebungseffekt überraschen:

```
body *:hover {background: yellow;}
```

Dadurch wird jedes Element, das vom body-Element abstammt, mit einem gelben Hintergrund versehen, sobald Sie mit dem Mauszeiger darüber fahren (»hover«) – das betrifft Überschriften, Absätze, Listen, Tabellen, Bilder und alles andere, was sich im body befindet. Sie könnten auch den Schrifttyp ändern, das Element, über dem sich der Cursor befindet, mit einem Rahmen versehen oder was auch immer Ihr Browser Sie tun lässt.



Internet Explorer für Windows inklusive der Version IE6 (beim Schreiben dieses Buches die aktuellste Version) gestattet die Verwendung von dynamischen Pseudoklassen nur für Hyperlinks.

Dynamische Stildefinitionen im wahren Leben

Dynamische Pseudoklassen bringen einige interessante Eigenwilligkeiten mit sich. So ist es beispielsweise möglich, besuchten und unbesuchten Links die gleiche Schriftgröße zuzuweisen, während Links, über denen sich der Mauszeiger befindet, größer dargestellt werden (siehe Abbildung 2-22):

```
a:link, a:visited {font-size: 13px;}  
a:hover {font-size: 20px;}
```



Abbildung 2-22: Das Layout mit dynamischen Pseudoklassen ändern

Wie Sie sehen, verändert der Browser die Textgröße eines Ankers, während sich der Mauszeiger über dem Link befindet. Ein Browser, der dieses Verhalten unterstützt, muss hierfür das Dokument neu aufbauen, während sich ein Anker im hover-Zustand befindet. Dadurch werden unter Umständen alle Elemente neu angeordnet, die auf den Link folgen.

In den CSS-Spezifikationen ist jedoch festgelegt, dass von User Agents nicht erwartet wird, dass sie ein Dokument neu darstellen müssen, nachdem es einmal für die Anzeige aufbereitet wurde. Sie können sich also nicht darauf verlassen, dass der von Ihnen gewünschte Effekt auch tatsächlich ausgeführt wird. Ich empfehle Ihnen, Designs, die sich auf ein solches Verhalten stützen, auf jeden Fall zu vermeiden.

Das erste Kindelement auswählen

Eine weitere statische Pseudoklasse namens `:first-child` wird verwendet, um Elemente auszuwählen, die man als »Erstgeborene« eines anderen Elements bezeichnen könnte. Diese Pseudoklasse wird leicht missverstanden, daher werden wir ihr ein umfangreiches Beispiel widmen. Sehen Sie sich dazu den folgenden Markup-Code an:

```
<div>
<p>These are the necessary steps:</p>
<ul>
<li>Insert key</li>
<li>Turn key <strong>clockwise</strong></li>
<li>Push accelerator</li>
</ul>
<p>
Do <em>not</em> push the brake at the same time as the accelerator.
</p>
</div>
```

In diesem Beispiel sind die ersten Kindelemente das erste `p`, das erste `li` sowie die Elemente `strong` und `em`. Durch die Anwendung folgender Regeln

```
p:first-child {font-weight: bold;}
li:first-child {text-transform: uppercase;}
```

erhalten Sie das in Abbildung 2-23 gezeigte Ergebnis.

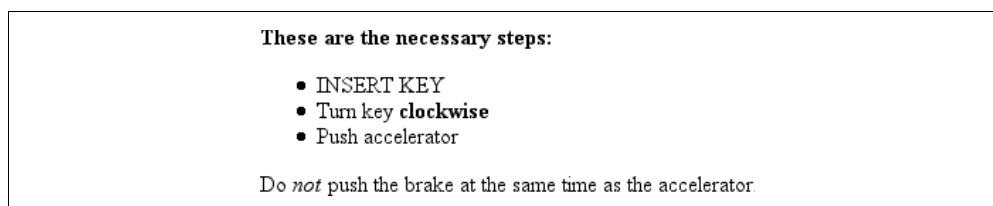


Abbildung 2-23: Stildefinitionen für erste Kindelemente

Die erste Regel gibt jedes `p`-Element fett aus, das das erste Kindelement eines anderen Elements ist. Die zweite Regel gibt alle `li`-Elemente in Großbuchstaben aus, die erste Kindelemente eines anderen Elements sind (in HTML also entweder ein `ol`- oder ein `ul`-Element).

Der häufigste Fehler besteht in der Verwendung eines Selektors wie `p:first-child`, um das erste Kindelement eines `p`-Elements auszuwählen. Vergessen Sie dabei nicht die Natur der Pseudoklassen, die eine Art »Phantomklasse« für das mit ihnen verbundene Element vergeben. Fügen Sie dem Code ein tatsächliches Klassenattribut hinzu, ergibt sich Folgendes:

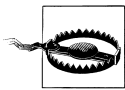
```
<div>
<p class="first-child">These are the necessary steps:</p>
<ul>
<li class="first-child">Insert key</li>
```

```

<li>Turn key <strong class="first-child">clockwise</strong></li>
<li>Push accelerator</li>
</ul>
<p>
Do <em class="first-child">not</em> push the brake at the same time as the accelerator.
</p>
</div>

```

Wenn Sie also ein `em`-Element auswählen möchten, das das erste Kindelement eines anderen Elements ist, so schreiben Sie `em:first-child`. Mit diesem Selektor können Sie beispielsweise den ersten Eintrag einer Liste mit einem besonderen Stil versehen oder den ersten Absatz eines `div`-Elements oder auch das erste `td`-Element in einer Tabellenspalte.



Internet Explorer für Windows inklusive der Version IE6 (beim Schreiben dieses Buches die aktuellste Version) bietet keine Unterstützung für `:first-child`.

Sprachbasierte Auswahl

Für die Auswahl eines Elements anhand seiner Sprache gibt es die Pseudoklasse `:lang()`. Was die Mustererkennung angeht, verhält sich `:lang()` genau wie der Attributselektor `|=`. Um beispielsweise alle französischsprachigen Elemente kursiv darzustellen, können Sie die folgende Regel verwenden:

```
*:lang(fr) {font-style: italic;}
```

Der wesentliche Unterschied zwischen dem Pseudoselektor und dem Attributselektor besteht darin, dass die Information über die verwendete Sprache aus verschiedenen Quellen stammen kann, die sich auch außerhalb des Dokuments befinden können. Die Spezifikation von CSS2.1 sagt hierzu:

In HTML wird die Sprache durch eine Kombination des Attributs »lang«, dem META-Element und möglicherweise Informationen aus dem Übertragungsprotokoll (wie beispielsweise einem HTTP-Header) ermittelt. XML benutzt das Attribut `xml:lang`. Und es kann noch andere für die Dokumentensprache spezifische Methoden geben, nach denen die Sprache ermittelt wird.

Aus diesem Grund ist die Verwendung der Pseudoklasse etwas verlässlicher als der Attributselektor und in den meisten Fällen, in denen sprachspezifische Stildefinitionen benötigt werden, die bessere Wahl.

Pseudoklassen kombinieren

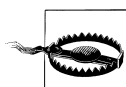
Seit der Einführung von CSS2.1 können Sie mehrere Pseudoklassen in einem Selektor zusammenfassen. So ist es möglich, unbesuchte Links rot einzufärben, wenn der Mauszeiger sich darüber befindet, während bereits besuchte Links rotbraun dargestellt werden:

```
a:link:hover {color: red;}
a:visited:hover {color: maroon;}
```

In diesem Fall spielt die Reihenfolge keine Rolle. Sie könnten also auch `a:hover:link` schreiben, um den gleichen Effekt zu erzielen. Es ist auch möglich, Links (besuchte und unbesuchte), über denen sich der Mauszeiger befindet, abhängig von der Sprache einzufärben, zum Beispiel Deutsch:

```
a:link:hover:lang(de) {color: gray;}
a:visited:hover:lang(de) {color: silver;}
```

Passen Sie aber auf, dass Sie keine Pseudoklassen kombinieren, die einander ausschließen. Ein Link kann beispielsweise nicht gleichzeitig »besucht« und »unbesucht« sein. Der Selektor `a:link:visited` hat also keinen Sinn. Ein derartiger Selektor (und damit die gesamte Regel) wird normalerweise von User Agents ignoriert.



Internet Explorer für Windows inklusive der Version IE6 (beim Schreiben dieses Buches die aktuellste Version) erkennt kombinierte Pseudoklassen nicht korrekt. Wie bei den Kombinationen aus Klasse und Wert wird stattdessen nur die letzte der kombinierten Pseudoklassen beachtet. Bei einem Selektor wie `a:link:hover` wird demnach zwar der `:hover`-Teil beachtet, nicht aber der `:link`-Teil des Selektors.

Selektoren für Pseudoelemente

So wie Pseudoklassen Anker mit fiktiven Klassenattributen versehen, fügen Pseudoelemente fiktive Elemente in ein Dokument ein, um bestimmte Effekte zu erzielen. In CSS2.1 werden vier Pseudoelemente definiert: Stilangaben für den ersten Buchstaben (`first-letter`), die erste Zeile (`first-line`) und Stilangaben vor (`before`) und nach (`after`) bestimmten Elementen.

Stilangaben für den ersten Buchstaben

Das erste Pseudoelement versieht nur den ersten Buchstaben eines Block-Elements mit einer eigenen Stildefinition:

```
p:first-letter {color: red;}
```

Durch diese Regel wird der erste Buchstabe jedes Absatzes rot dargestellt. Alternativ dazu können Sie auch den ersten Buchstaben jeder mit `h2` markierten Überschrift doppelt so groß darstellen wie den Rest:

```
h2:first-letter {font-size: 200%;}
```

Das Ergebnis sehen Sie in Abbildung 2-24.

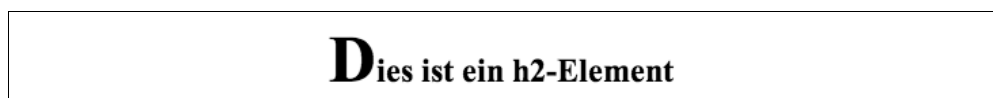


Abbildung 2-24: Das Pseudoelement `:first-letter` in Aktion

Wie ich bereits gesagt habe, erzeugt der User Agent durch diese Regel ein fiktives Element, das den ersten Buchstaben jedes h2-Elements umgibt. In HTML würde das etwa so aussehen:

```
<h2><h2:first-letter>D</h2:first-letter>ies ist ein h2-Element</h2>
```

Die Stildefinitionen von `:first-letter` werden nur auf das im Beispiel gezeigte Element angewendet. Das Element `<h2:first-letter>` erscheint *nicht* im Quellcode des Dokuments. Stattdessen wird es spontan vom Browser zum Leben erweckt und benutzt, um anschließend die mit `:first-letter` definierten Stilvorgaben auf die entsprechenden Textteile anzuwenden. Anders gesagt: `<h2:first-letter>` ist ein Pseudoelement. Sie brauchen hierfür keinerlei zusätzliche Tags in Ihr Dokument einzufügen; der User Agent erledigt das für Sie.

Stildefinitionen für die erste Zeile

Auf ähnliche Weise kann `:first-line` benutzt werden, um die erste Textzeile eines Elements mit einer eigenen Stilvorgabe zu versorgen. Vielleicht möchten Sie die erste Zeile aller Absätze in Ihrem Dokument lila einfärben:

```
p:first-line {color: purple;}
```

In Abbildung 2-25 wird die Stildefinition auf die erste Zeile des angezeigten Textes jedes Absatzes angewendet. Das geschieht unabhängig von der tatsächlichen Darstellungsbreite. Enthält die erste Zeile nur fünf Wörter, werden auch nur diese fünf Wörter lila eingefärbt. Enthält die erste Zeile die ersten 30 Wörter dieses Elements, werden alle 30 Wörter lila eingefärbt.

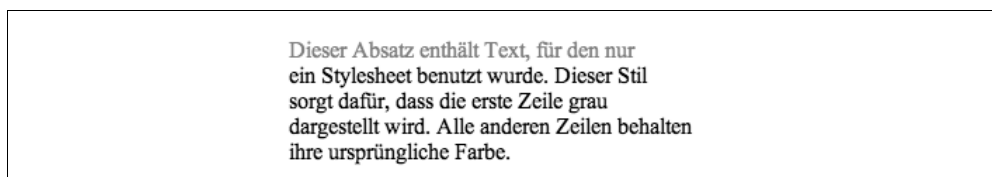


Abbildung 2-25: Das Pseudoelement `:first-line` in Aktion

Um den Text von »Dieser« bis »nur« lila darzustellen, verwendet der User Agent ein fiktives Markup, das in etwa so aussieht:

```
<p><p:first-line>Dieser Absatz enthält Text, für den nur</p:first-line>  
ein Stylesheet benutzt wurde. Dieser Stil  
sorgt dafür, dass die erste Zeile grau ...
```

Wenn die erste Textzeile so geändert wird, dass sie nur die ersten fünf Wörter enthält, dann würde das fiktive Tag `</p:first-line>` so verschoben, dass es direkt nach dem Wort »für« erscheint.

Einschränkungen für :first-letter und :first-line

In CSS2 können die Pseudoelemente :first-letter und :first-line nur auf Block-Elemente wie Überschriften (h1, h2 usw.) und Absätze, nicht aber auf Inline-Elemente wie beispielsweise Hyperlinks angewendet werden. In CSS2.1 wirkt :first-letter auf alle Elemente. Zudem gibt es einige Einschränkungen bei den CSS-Eigenschaften, die für :first-letter und :first-line benutzt werden können. Eine Liste dieser Einschränkungen sehen Sie in Tabelle 2-3.

Tabelle 2-3: In Pseudoelementen erlaubte Eigenschaften

:first-letter	:first-line
Alle font-Eigenschaften	Alle font-Eigenschaften
Alle color-Eigenschaften	Alle color-Eigenschaften
Alle background-Eigenschaften	Alle background-Eigenschaften
Alle margin-Eigenschaften	word-spacing
Alle padding-Eigenschaften	letter-spacing
Alle border-Eigenschaften	text-decoration
text-decoration	vertical-align
vertical-align (wenn float den Wert none hat)	text-transform
text-transform	line-height
line-height	clear (nur CSS2; entfernt in CSS2.1)
float	text-shadow (nur CSS2)
letter-spacing (neu seit CSS2.1)	
word-spacing (neu seit CSS2.1)	
clear (nur CSS2; entfernt in CSS2.1)	
text-shadow (nur CSS2)	

Zudem müssen alle Pseudoelemente am Ende des Selektors platziert werden, in dem sie erscheinen. Es ist also nicht erlaubt, Dinge zu schreiben wie `p:first-line em`, weil das Pseudoelement hier vor dem Subjekt (dem letzten Element der Liste) steht. Die gleiche Regel gilt für die anderen zwei in CSS2.1 definierten Pseudoelemente.

Stildefinitionen vor und nach Elementen

Im nächsten Beispiel wollen wir jedem h2-Element ein Paar silbergrauer eckiger Klammern voranstellen, um einen typographischen Effekt zu erzielen:

```
h2:before {content: "]]"; color: silver;}
```

Seit CSS2.1 ist es möglich, so genannte *generierte Inhalte* in das Dokument einzufügen und diese mittels der Pseudoelemente `:before` und `:after` direkt mit Stildefinitionen zu versehen. Ein Beispiel sehen Sie in Abbildung 2-26.

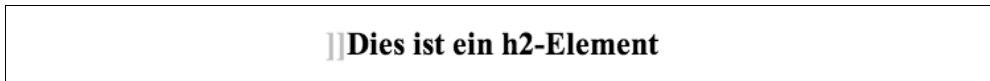


Abbildung 2-26: Generierte Inhalte vor einem Element einfügen

Mit Hilfe des Pseudoelements können Sie generierte Inhalte einfügen und mit Stildefinitionen versehen. Um Dinge nach einem Element einzufügen, benutzen Sie das Pseudoelement `:after`. Um Ihre Dokumente angemessen zu beenden, wäre folgende Regel denkbar:

```
body:after {content: " The End.";}
```

Generierte Inhalte bilden ein eigenständiges Thema, das (inklusive weiterer Details zu `:before` und `:after`) in Kapitel 12 behandelt wird.

Zusammenfassung

Die Verwendung von Selektoren, die auf der Sprache des Dokuments basieren, ermöglicht es Autoren, genauso einfach CSS-Regeln aufzustellen, die sich auf eine große Anzahl ähnlicher Elemente anwenden lassen, wie Regeln für sehr klar definierte Umstände zu definieren. Durch die Möglichkeit, Selektoren und Regeln zu gruppieren, können Stylesheets kompakt und flexibel gehalten werden, was wiederum zu kleineren Dateigrößen und so zu schnelleren Download-Zeiten führt.

User Agents müssen in der Lage sein, Selektoren korrekt zu handhaben, denn ohne diese Fähigkeit kann ein Browser nur recht wenig mit CSS anfangen. Andererseits ist es absolut notwendig, dass Autoren ihre Selektoren korrekt schreiben, da Fehler in den Regeln dafür sorgen, dass ein User Agent die Stile nicht richtig auf das Dokument anwendet. Um Selektoren und ihre Kombinationsmöglichkeiten richtig zu begreifen, müssen Sie verstehen, wie Selektoren sich auf die Dokumentstruktur beziehen und welche Rolle andere Mechanismen, wie Vererbung und die Kaskadierung selbst, bei der Anwendung der Stildefinitionen auf das Dokument spielen. Diese Dinge sind das Thema des nächsten Kapitels.