

CSS +  
xhtml

deppensicher

Warum kompliziert, wenn's auch einfach geht?

## INHALT

Über dieses Tutorial & Copyright . . . . .	3
Was sind CSS? . . . . .	4
Wozu CSS? . . . . .	4
CSS - die Vorteile . . . . .	5
Browser und CSS . . . . .	5
XHTML als Brücke zu XML . . . . .	6
Tabellenfrei = Barrierefrei? . . . . .	6
XHTML - Regeln & Syntax . . . . .	7
(X)HTML - Relation von Elementen . . . . .	10
CSS - Syntax & Begriffe . . . . .	11
CSS Selektoren . . . . .	14
Typ Selektoren / Type Selectors . . . . .	14
ID Selektoren / ID Selectors . . . . .	15
Klassen Selektoren / Class Selectors . . . . .	15
Nachfahre Selektoren / Descendant Selectors . . . . .	18
Kind Selektoren / Child Selectors . . . . .	19
Nachbar Selektor / Adjacent Sibling Selectors . . . . .	19
Kombination von Selektoren . . . . .	19
CSS Einbindung . . . . .	21
CSS und Media Types . . . . .	24
CSS Box Modell - margin, padding border . . . . .	25
CSS layout - Position is Everything . . . . .	30
CSS - Vererbung von Eigenschaften . . . . .	33
CSS und Hintergründe . . . . .	34
CSS und Schrift & Text . . . . .	35
CSS und Links - Pseudoklassen . . . . .	41
CSS und Listen . . . . .	43
CSS und Alpha-Transparenz . . . . .	44
CSS Validierung - Gültigkeitsprüfung . . . . .	44
Nützliche Links . . . . .	45

## ÜBER DIESES TUTORIAL & COPYRIGHT

Diese Anleitung zum "deppensicheren" Schreiben von CSS und XHTML will einen umfassenden aber einfachen Überblick über die Gestaltungsmöglichkeiten von Stylesheets für die Erstellung von tabellenfreien Layouts geben.

Dieses Tutorial verzichtet auf in der Praxis wenig verwendete Details und weist auch darauf hin, wenn Style-Anweisungen nicht ausreichend browserkompatibel sind.

### Ziel dieses Tutorials

Dieses Tutorial vermittelt die wichtigsten Regeln von XHTML und CSS, so dass mit dem nächsten Tutorial "Tabellenfreie Layouts mit XHTML + CSS" fortgeschrittene Techniken angewendet werden können.

### Voraussetzungen

Basiskonntnisse in HTML

### Schreibweisen in diesem Tutorial

HTML Code ist blau hervorgehoben, CSS Code ist grün markiert:

```
<h1>Das ist HTML Code</h1>
```

```
/* So sieht CSS Code aus*/  
body {margin: 0px; padding: 0px;}
```

**ACHTUNG:** Besondere Hinweise und Links sind rot markiert.

### Copyright & Nutzungsbedingungen

(c) Elisabeth Wetsch, Wien 2007, [www.elizZZa.net](http://www.elizZZa.net), [elizZZa@elizZZa.net](mailto:elizZZa@elizZZa.net)

Damit weitere "deppensichere" Anleitungen entstehen können, ist dieses Tutorial um 1,90 Euro downloadbar.

**LINK:** <http://www.ebookZZZ.net/>

Dieses Tutorial ist in allen seinen Teilen urheberrechtlich geschützt. Alle Rechte vorbehalten, insbesondere das Recht der Übersetzung, des Vortrags, der Vervielfältigung auf fotomechanischem oder anderem Weg und der Speicherung in elektronischen Medien. Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, übernimmt die Autorin für mögliche Fehler und deren Folgen keine juristische Verantwortung oder irgendeine Haftung. Die in diesem Tutorial wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

## WAS SIND CSS?

Sogenannte Cascading Style Sheets sind idealerweise externe Textdateien mit der Endung .css, welche Angaben zu Position, Formaten, Farben, Hintergründen von HTML-Elementen definieren. „Kaskadierend“ nennt man CSS aus zwei Gründen:

Ein Stylesheet kann über mehrere HTML-Seiten „kaskadieren“ / Einfluß haben. Umgekehrt können aber auch mehrere Stylesheet eine einzige HTML-Seite definieren.

Welches aus mehreren Stylesheets „die Oberhand“ behält, ergibt sich aus der Reihenfolge, wie diese mit dem HTML-Dokument verknüpft oder eingebunden sind. Dazu mehr unter =>CSS EINBINDUNG

## WOZU CSS?

Bevor CSS 1996 „erfunden“ wurde(n), mußten Angaben zu Position, Breiten, Höhen, Farben, Hintergründen, Schrifteseigenschaften etc. bei **jedem** HTML-Element notiert werden. Das führte einerseits zu einer erheblichen Unübersichtlichkeit des HTML-Codes und andererseits zu unnötig „aufgeblasenen“ Dateigrößen. Wer schon mal in „alten“ HTML-Dateien Fehler suchen mußte, weiß davon ein Lied zu singen.

Wollte man z.B. erreichen, daß jedes Heading 2 in in roter Schriftfarbe und einer bestimmten Schriftgröße erscheint, mußte man **jedes** h2-Element wie folgt notieren:

```
<h2 font-color="#FF0000" font-size="3">Headline</h2>
```

Zu allem Überdruß also auch noch eine ziemlich aufwendige Codeschreiberei...

Und wehe, es fiel dem Auftraggeber ein, daß er solche Headlines nun doch lieber grün eingefärbt sehen wollte. **Sämtliche** Vorkommen von h2 in **sämtlichen** Seiten mußten nun entsprechend geändert werden.

Helle Köpfe hatten nun die Idee, daß es doch viel effizienter wäre, in einem separaten Dokument die Anweisung zu schreiben „**alle h2 sind rot und in Schriftgröße X**“.

Für eine Änderung des Aussehens der h2 in **allen** Dokumenten müßte nur noch diese **eine** Zeile geändert werden...

Das war die Geburtsstunde von CSS...

## CSS - DIE VORTEILE

Mit Hilfe von CSS werden Format- und Designangaben **einmal** definiert, was natürlich weit effizienter ist, als in **jeder** HTML-Seite Attribute für **jedes** Element immer wieder zu definieren:

- ☺ Seiten laden schneller, oft bis zu 50% und mehr
- ☺ Sehr viel weniger Code bedeutet schlankere, übersichtlichere HTML-Seiten
- ☺ Designvorgaben bleiben über alle Seiten konsistent, da zentral definiert
- ☺ Updates bzw. Änderungen des Designs sind sehr viel einfacher,
- ☺ die Fehleranfälligkeit ist dadurch weit geringer
- ☺ Gut geschriebenes CSS garantiert auch Menschen mit Behinderungen einen barrierefreien Zugang zu Webseiten
- ☺ CSS erlaubt weit mehr Designoptionen als es in HTML je möglich war

## BROWSER UND CSS

Das **World Wide Web Consortium**, (W3C), ist eine Einrichtung unter Mitwirkung jener Menschen und Institutionen, die an der Entstehung und Entwicklung des World Wide Web maßgeblich beteiligt waren.

Das W3C definiert jene technischen Standards und Sprachen (wie HTML, XML, CSS etc.), nach welchen wir Webproducer arbeiten und welche Browser (und andere Ausgabetechnologien) entsprechend interpretieren sollen.

**LINK** <http://www.w3.org/QA/2002/04/valid-dtd-list.html>

Bisher hat das World Wide Web Consortium zwei wichtige CSS Standards freigegeben, CSS-1 (1996) und CSS-2 (1998), CSS-3 ist in Arbeit.

Seit Browserversionen 4.x wird CSS immer besser interpretiert.

Wenn ein (älterer) Browser CSS nicht versteht, ignoriert er die Angaben völlig. Daher ist es wichtig, für solche Fälle dafür zu sorgen, daß ein Dokument auch ohne Stylesheets gut lesbar ist - das nennt man dann **graceful degradation**.

## XHTML ALS BRÜCKE ZU XML

Zudem forderten seit Jahren neue Ausgabeoptionen (Handheld, Web-TV, Print etc.) unterschiedliche Ausgabeformate. Wäre es da nicht viel effizienter, die Inhalte von Designangaben strikt zu trennen, sodaß Inhalte nur **einmal** eingegeben und nur die Ausgabeoptionen verändert werden müßten?

Gedacht - getan: Das World Wide Web Consortium machte sich mit der Entwicklung der Standards **XML** (e**X**tensible **M**arkup Language) nicht nur um uns Webproducer verdient, indem es uns kilometerlanges Codieren ersparte. Mit der Einführung von XML wurde eine Scriptsprache zur **Strukturierung** von Inhalten entwickelt, die es erlaubt, die **gleichen** Inhalte in **unterschiedlichsten** Medien auszugeben und darzustellen.

Für das Web baute man uns Webproducern eine Brücke namens **XHTML** (e**X**tensible **H**ypertext **M**arkup Language. XHTML wird HTML ersetzen und soll uns schlicht und einfach dazu erziehen und darauf vorbereiten,

- ☞ Inhalte verstärkt mit Hilfe von strukturellen Tags zu gestalten (z.B. Headings, Paragraphs, Lists etc.),
- ☞ Inhalte möglichst strikt von Design zu trennen,
- ☞ Syntax (also vorgeschriebene Schreibweisen) strenger zu befolgen

## TABELLENFREI = BARRIEREFREI?

Und weil man gleich Nägel mit Zöpfen machen wollte, überlegte man auch, wie man all diese Segnungen für das Thema „Barrierefreiheit“ nützen könnte. Darunter versteht man die Zugänglichkeit von Informationen im Web auch für Menschen mit Behinderungen.

Frames und Tabellen waren Verfechtern der „Accessibility“ immer schon ein Dorn im Auge - konnte man darauf verzichten?

Und so etablierte man als dritte Komponente zeitgemäßen Webdesigns sogen. DIV-Container (auch DIVs) als wesentliches Konstruktionselement von Webseiten und ersetzte damit auch Tabellen.

Somit lautete die neue Erfolgsformel im Webdesign:

**XHTML - (Tabellen + Frames) + DIVs + CSS = Usability + Accessibility**

Das Thema „Barrierefreiheit“ ist mit der Einhaltung dieser Formel zwar keinesfalls abgehakt (dazu wird es ein eigenes Tutorial geben), aber es werden damit die wichtigsten Voraussetzungen geschaffen.

## XHTML - REGELN &amp; SYNTAX

Bevor wir uns CSS widmen, vorweg die (wenigen) Regeln, die für "validen" (gültigen) XHTML-Standard zu beachten sind. Bestehen die Seiten die Gültigkeitsprüfung durch das W3C, dürfen sie sich mit einer entsprechenden Plakette schmücken.



Mehr dazu unter =>CSS VALIDIERUNG - GÜLTIGKEITSPRÜFUNG

### DTD - Doc Type Definition

---

Den Anfang eines XHTML-Dokuments bildet die sogen. Doc Type Definition. Eine solche DTD informiert die Ausgabetechnik (z.B. ein Browser) darüber, nach welchen Standards das Dokument geschrieben wurde und dient der Interpretierbarkeit desselben.

Der "Vorspann" einer gültigen XHTML-Seite muß also eine DTD vor dem html-Tag aufweisen. Eine Liste aller möglichen DTDs findest Du unter

LINK: <http://www.w3.org/QA/2002/04/valid-dtd-list.html>

Beispiele

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

## Well-Formed

Ein XHTML Dokument muß well-formed sein, d.h. die nachfolgenden vier Regeln müssen unbedingt befolgt werden.

### Elemente korrekt verschachteln

Während HTML das falsche Verschachteln von Tags verzieht, ist dies in XHTML schlicht ein Fehler. Verschachtelungen dürfen sich also nicht überschneiden.

```
<strong><em>Das ist falsch</strong></em>
```

```
  |      | ----- |-----|
  |-----|
```

```
<strong><em>So ist es richtig</em></strong>
```

```
  |      | ----- |
  |-----|
```

### Tags & Attribute in Kleinbuchstaben

Schrieb man HTML-Tags und Attribute eine Zeitlang in Großbuchstaben, ist dies in XHTML verboten.

```
<BODY>Das ist falsch</BODY>
```

```
<body>So ist es richtig</body>
```

### Ende-Tags zwingend

Ist für ein Element ein Ende-Tag vorgesehen, muß dies auch eingesetzt werden, z.B.

```
<p>Das ist ein Absatz.</p>
```

### Elemente ohne Ende-Tag in sich schließen

Elemente ohne eigenes Ende-Tag (z.B. br, img, meta, link) müssen in sich geschlossen werden - vor der spitzen Klammer am Ende Leerzeichen und Slash:

```
<br />
```

## Werte in Anführungszeichen

---

Werte von Attributen müssen immer in Anführungszeichen stehen, z.B.

```
<td width=100>FALSCH</td><td width="100">RICHTIG!</td>
```

## Attribute immer mit Wert

---

In HTML gibt es Attribute, die ohne Wert notiert werden, das ist in XHTML verboten, z.B.

```
<td nowrap>Das ist falsch</td>  
<td nowrap="nowrap">So ist es richtig</td>
```

## Werte mit codierten Sonderzeichen

---

Benötige ich in einem Wert Sonderzeichen, so sind diese als Code zu schreiben, z.B.

```
  

```

## Stylesheets & Scripts separieren

---

Stylesheets und Scripts werden als separate Dateien mit dem link-Tag mit HTML-Seiten im head-Bereich verknüpft. Javascripts dürfen auch im body-Bereich eingefügt werden. Wenn es der Funktion nicht widerspricht, ist aber die Einbindung im head-Bereich vorzuziehen.

## id anstatt name (Empfehlung)

---

Da in XML anstatt „name“ grundsätzlich „id“ verwendet wird, sollten wir uns das jetzt schon angewöhnen.

## (X)HTML - RELATION VON ELEMENTEN

Ähnlich einem Stammbaum kann man die Abhängigkeiten von Elementen in einer HTML-Datei nach Verwandtschaftsgrad definieren. Das Verständnis für die hierarchische Struktur eines HTML-Dokuments ist wichtig, da manche CSS Attribute „vererbt“ werden - mehr dazu unter =>CSS - VERERBUNG von EIGENSCHAFTEN





Beispiel einer HTML-Struktur (ohne DTD, html, head)

```
<body>
  <div id="content">
    <h1>Headline</h1>
    <p>Der erste Absatz</p>
    <p>Der zweite Absatz</p>
    <hr />
  </div>
  <div id="nav">
    <ul>
      <li>Menu Item 1</li>
      <li>Menu Item 2</li>
      <li>Menu Item 3</li>
    </ul>
  </div>
</body>
```

In einer HTML-Seite ist der body immer der „Ancestor“ (Vorfahre) aller Elemente der Seite. Alle anderen Elemente sind „Descendants“ (Abkömmlinge) des body Elements.

- ☞ Gleichzeitig sind aber auch z.B. die li-Elemente Descendants des div#nav
- ☞ Das div#content ist „Parent“ (Elternelement) von h1, p, p, hr
- ☞ Das div#nav ist Parent des ul-Elements, welches seinerseits Parent der li-Elemente ist.
- ☞ Die li-Elemente sind also „Childs“ (Kinder) des ul-Elements.
- ☞ Zueinander sind die li-Elemente wiederum „Siblings“ (Geschwister), da sie das gleiche Elternelement teilen.

Somit kann jedes der Elemente in unserem Beispiel unterschiedliche Verwandtschaftsverhältnisse zu unterschiedlichen Elementen haben. So ist das `div#nav` in unserem Beispiel gleichzeitig

-  Child des `body`-Elements
-  Sibling des `div#content`
-  Parent des `ul`-Elements
-  Ancestor von `ul`- und `list`-Element

## CSS SYNTAX UND BEGRIFFE

Die CSS Syntax (Schreibweise) wird sehr streng ausgelegt. Ein fehlender Strichpunkt am Ende einer Deklaration kann dazu führen, daß der Rest des Stylesheets nicht mehr ausgeführt wird...

### CSS Rules (Anweisungen) bzw. Rule Sets

---

#### Rule (Selector mit einer Declaration)

```
body {color: #333333;}
```

```
selector {attribute: value;}
```

#### Rule Set (Selector mit mehreren Declarations)

```
body { color: #333333; margin: 0px; padding: 0px;}
```

```
selector {attribute: value; attribute: value;}
```

oder

```
selector {declaration; declaration;}
```

oder

```
selector {declaration block}
```

Ein Selektor wählt das gewünschte Element bzw. die gewünschten Elemente aus.

Attribut+ Wert werden als Deklaration bezeichnet.

Eine Declaration sagt dem Browser, wie er das gewählte Element darstellen soll.

Ein Attribut wählt eine Eigenschaft des Elements, ein Value weist dieser Eigenschaft einen bestimmten Wert zu.

### Values

Werte müssen - im Gegensatz zu HTML - immer eine (Maß)einheit haben. Ausgenommen davon ist der Wert 0, bei dem es gleichgültig ist, ob er eine Maßeinheit besitzt, denn Null ist immer Null. Dennoch empfiehlt sich, auch beim Wert 0 eine Maßeinheit zu setzen, möchte man später vielleicht den Wert auf ungleich 0 ändern, kann es nicht passieren, daß man vergißt, eine Maßeinheit hinzuzufügen.

### Units

In CSS kommen folgenden Einheiten zur Anwendung:

**Maßeinheiten:** em, ex, pc, pt, px, cm, mm, in

**Keywords:** smaller, larger, xx-small, x-small, small, medium, large, x-large, xx-large

**Prozentangaben:** %

### Colors

Farben werden üblicherweise als 6-stelliger HEXcode mit einer Raute davor notiert.

Zulässig sind auch Farbangaben als RGB Werte und als benannte Farben (named colors). Beim HEXCode stehen je zwei Stellen für R (rot), G (grün), B (blau). Farben als HEXcode dürfen "verkürzt" geschrieben werden, wenn die Wertepaare jeweils gleich sind.

Es wird jedoch empfohlen, Farben immer als HEXcode und unverkürzt zu schreiben. Dies ist übersichtlicher und erleichtert die Fehlersuche in einem Stylesheet. Die "Zeitersparnis" durch die verkürzte Schreibweise ist in diesem Fall vernachlässigbar.

### Kommentare

---

```
/* hier ein Kommentar in CSS */  
body {margin: 0px; padding: 0px; /* color: #333333; */ background: #FF0000;}
```

### Allgemeine Tipps zur Schreibweise von CSS

Eine Gruppe von Deklarationen wird als Deklarationsblock bezeichnet. Die einzelnen Deklarationen werden durch Strichpunkt getrennt. Obwohl die letzte Deklaration einer Anweisung (Rule Set) nicht mit einem Strichpunkt beendet werden müßte, wird das doch ausdrücklich empfohlen. Es vermeidet Fehler, wenn man später vielleicht noch eine Deklaration anfügt...

Leerzeichen dienen der besseren Übersichtlichkeit der Anweisungen.  
Auch **Zeilenvorschübe** und **Einrückungen** dürfen eingesetzt werden, um die Lesbarkeit zu verbessern.

Klassische Schreibweise

```
body {  
  color: #333333;  
  margin: 0px;  
  padding: 0px;  
}
```

Alternative Schreibweise

```
body {color: #333333; margin: 0px; padding: 0px;}
```

Selektoren gruppieren

Wenn mehrere Selektoren die gleiche(n) Eigenschaft(en) haben sollen, dürfen die Selektoren wie folgt gruppiert werden.

```
h1, h2, h3 {font-weight: normal;}  
#nav ul, #nav a {color: #FF0000;}
```

**ACHTUNG:** Ein häufiger Fehler ist hier, daß weitere Selektoren im gleichen Element nicht korrekt angeführt werden oder nach dem letzten Selektor ein Beistrich gesetzt wird, z.B.

```
#nav ul, a {color: #FF0000;}  
#nav ul, a, {color: #FF0000;}
```

Verkürzte Schreibweise

Diese sind bei den jeweiligen Attributen angeführt.

## CSS SELEKTOREN

Für die Auswahl der Elemente, die mit Eigenschaften versehen werden sollen, stehen verschiedene Selektoren/Selectors zur Verfügung:

### Typ Selektoren (auch HTML Selektoren) - Type Selectors

---

Im CSS wird einfach der HTML-Tag ohne Zusätze geschrieben.

Die Anweisung für einen HTML-Selektor definiert sämtliche Vorkommen des Tags im HTML-Code.

```
body {background: #FF0000;}  
p {text-align: justify;}  
hr {border-bottom: 1px dashed #FF0000;}
```

#### Resultat

Der Seitenhintergrund ist rot.

Alle Absätze (Paragraphs) werden mit Blocksatz versehen.

Alle horizontalen Linien (horizontal Rules) mit Rand unten, 1 Pixel stark, strichliert und rot.

### Typ Selektor a - Link Pseudoklassen

---

Mit den Pseudoklassen a:link, a:visited, a:hover, a:active werden Linkfarben und -verhalten definiert. Mehr dazu unter =>CSS und LINKS

**ACHTUNG:** Die angeführte Reihenfolge ist einzuhalten, da sonst die Attribute für a:hover und a:active nicht korrekt angezeigt werden.

```
a {font-weight: bold;}  
a:link {color: #FF0000;}  
a:visited {color: #FF9900; text-decoration: none;}  
a:hover {color: #FFFFFF; background: #00FF00; text-decoration: none;}  
a:active {border: 1px solid #0000FF;}
```

## Universal Selector - Universal Selektor

---

Das Sternchen als Selektor bedeutet, daß eine Anweisung auf sämtliche Elemente einer HTML-Seite zutrifft.

```
* {color: #FF0000;}
```

### Resultat

Sämtliche Elemente erscheinen in roter Schriftfarbe

## Pseudo-Elemente ::first-letter und ::first-line

---

Definieren den ersten Buchstaben oder die erste Zeile eines Textes/Absatzes.

**ACHTUNG:** Doppelte Doppelpunkte!

```
#box p::first-letter {}
```

```
#box p::first-line {}
```

## ID Selektoren - ID Selectors

---

ID Selektoren unterscheiden sich von Klassen Selektoren in zwei Punkten:

- ☞ Eine ID darf auf pro HTML-Seite nur einmal vergeben werden.
- ☞ Eine ID ist "gewichtiger" als eine Klasse. Gibt es also einen Konflikt zwischen einer Anweisung zu einer Klasse und einer Anweisung zu einer ID, so "gewinnt" die Anweisung für die ID.

```
<div id="rotebox">Hier beliebiger Inhalt</div>
```

```
#rotebox {.....}
```

Eine ID wird im Stylesheet mit einer Raute # vor dem id-Namen referenziert.

## Klassen Selektoren - Class Selectors

---

Im Gegensatz zum Type Selector definiert ein Class Selector nur jene Elemente in der HTML-Seite, die mit der entsprechenden class versehen sind. Ein Klassen Selektor darf (im Gegensatz zu ID Selektoren) auf einer HTML-Seite beliebig oft eingesetzt werden.

```
<p class="grosseschrift">Schrift größer dargestellt</p>
<ul>
<li class="grosseschrift">List Item größer dargestellt.</li>
<li>Dieses ohne Anweisung</li>
</ul>
```

```
.grosseschrift {font-size: larger;}
```

Eine Klasse wird im Stylesheet mit einem Punkt . vor dem Klassennamen referenziert.

#### Resultat

Die Schrift sämtlicher Elemente mit dem Class Selector .grosseschrift wird größer dargestellt.

#### Class Selector versus ID Selector

Es gibt oft Diskussionen darüber, ob man lieber Klassen Selektoren oder ID Selektoren einsetzen soll. Nun, die Entscheidung ist nicht allzu schwierig, bedenkt man die folgenden Voraussetzungen:

- ☞ Ein Klassen Selektor darf pro Seite beliebig oft eingesetzt werden
- ☞ Ein ID Selektor darf auf einer HTML-Seite nur einmal erscheinen
- ☞ Einem HTML-Element darf nur ein ID Selektor zugewiesen werden
- ☞ Einem HTML-Element dürfen mehrere Klassen Selektoren zugewiesen werden
- ☞ ID Selektoren haben eine höhere Priorität als Klassen Selektoren, gibt es einen Konflikt zwischen der Anweisung eines ID Selektors und der Anweisung eines Klassen Selektors, gewinnt die Anweisung des ID Selektors.

#### DENKEN statt KLASSEN!

Bevor man eine Klasse anlegt, sollte man überlegen:

- ☞ Gibt es einen HTML-Selektor, den wir verwenden können?
- ☞ Kann eine vorhandene ID oder Klasse verwendet werden, so daß wir keine neue Klasse anlegen müssen?

Ein häufiger Fehler bei Anfängern ist das Anlegen von zu vielen Klassen. Viele Klassen sind überflüssig, wenn man die Regeln zur Schreibweise von Klassen und Kombinationen befolgt.

Beispiel

```
<div id="sidebar">
<h2 class="sideheading">Headline</h2>
<ul class="sidelist">
  <li class="sidelistitem">Menupunkt1</li>
  <li class="sidelistitem"><a href="....." class="sidelistlink">Verlinkter Menupunkt</a></li>
</ul>
</div>
```

```
#sidebar {.....}
.sideheading {.....}
.sidelistitem {.....}
.sidelistlink {.....}
```

Besser

```
<div id="sidebar">
<h2>Headline</h2>
<ul>
  <li>Menupunkt1</li>
  <li><a href=".....">Verlinkter Menupunkt</a></li>
</ul>
</div>
```

```
#sidebar {.....}
#sidebar h2 {.....}
#sidebar ul {.....}
#sidebar li {.....}
#sidebar li a {.....}
```

Strukturelle HTML-Elemente bevorzugen

Anstatt z.B. eine Klasse `.headline` einzuführen und entsprechend zu formatieren, sollten nach Möglichkeit immer strukturelle HTML-Elemente bevorzugt werden, in diesem Fall würde man ein Heading (`h1` bis `h6`) verwenden.

Sollte ein Browser Stylesheets nicht erkennen/einsetzen, behält unsere HTML-Dokument in der Ansicht immer noch eine gewisse strukturelle Gliederung.

Der Aspekt der Strukturierung ist auch der Grund, warum Navigationsleisten bevorzugt als Listen angelegt werden, auch wenn diese horizontal erscheinen sollen.

#### Als strukturelle HTML-Elemente gelten

- ☺ h1 bis h6 Headings/Überschriften
- ☺ ul, ol, dl Listen
- ☺ label Beschriftung von Formularfeldern
- ☺ caption Überschriften von Datentabellen
- ☺ th Table Headers Kopfzeile einer Datentabelle

#### Nachfahre Selektoren - Descendant Selectors

---

Mit Nachfahre Selektoren wählen Elemente, die Nachfahren eines Elements sind (also innerhalb desselben liegen).

```
<h1>Heading <em>here</em></h1>
```

```
<p>Und hier der Inhalt <em>eines Absatzes</em>.</p>
```

```
p em {color: #333333;}
```

Formatiert nur em-Typ Selektoren, die innerhalb eines p-Elements liegen (also „Nachfahren eines p-Elements sind). Dabei dürfen Ebenen übersprungen werden:

```
<p>Lorem ipsum dolor <em>sit</em> amet.</p>
```

```
<ul>
```

```
<li>item 1</li>
```

```
<li>item 2</li>
```

```
<li><em>item 3</em></li>
```

```
</ul>
```

```
ul em {color: blue; }
```

## Kind Selektoren - Child Selectors

---

Im Gegensatz zu Descendant Selectors wählen Child Selectors nur die erste Nachfahren-Generation, als die direkten Kinder aus:

```
<h1>Heading <em>text</em></h1>
<div>
  This is some <em>text</em>
  <p>This is a paragraph of <em>text</em></p>
</div>

div>em { color: blue; }
```

## Nachbar Selektoren - Adjacent Sibling Selectors

---

Es werden Selektoren ausgewählt, die einem angegebenen Selektor unmittelbar folgen.

```
<h2>Heading 2 <em>text</em></h2>
<h3>Heading 3 text</h3>
<p>This is <em>text</em> and more <strong>text</strong></p>

h2 + h3 {margin: 0px; }
```

## Kombination von Selektoren

---

Eine mächtige Technik ist die Kombination von Selektoren. Dabei können praktisch alle hier angeführten Arten von Selectors miteinander kombiniert werden.

### Kombination von Class und Type Selectors (Contextual Style)

Noch gezielter kann man vorgehen, wenn man Type Selectors und Class Selectors kombiniert:

```
.grosseschrift {font-size: larger;}
p.grosseschrift {color: #FF0000;}
li.grosseschrift {font-weight: bold;}
```

### Resultat

Alle Elemente der Seite mit der Klasse `.grosseschrift` erhalten eine größere Schrift.  
 Absätze (`p`) mit der gleichen Klasse haben zusätzlich rote Textfarbe.  
 Listenelemente (`li`) sind zusätzlich mit fetter Schrift versehen.

Diese mächtige Technik hilft uns, daß wir nicht uferlos neue Klassen erfinden müssen.  
 Möchten wir z.B. erreichen, daß alle fett geschriebenen Texte in Absätzen auch noch rot unterlegt sind, so schreiben wir:

```
p strong {background: #FF0000;}
```

Dazu werden die Selektoren in absteigender Reihenfolge nacheinander durch Leerzeichen getrennt geschrieben.  
 Diese Technik verwenden wir z.B. auch dann, wenn wir für unterschiedliche Seitenbereiche unterschiedliche Linkfarben wünschen:

```
#nav a {color: #FF9900;}
```

Wenn obiges `#nav` die ID eines DIVs ist, dann dürfen wir auch schreiben

```
div#nav a {color: #FF9900;}
```

Diese Schreibweise wird von manchen CSS Autoren wegen der besseren Übersichtlichkeit verwendet.

### Kombination mehrerer Klassen

Sehr mächtig ist auch die Möglichkeit, einem Element mehrere Klassen zuzuweisen. Dabei werden die Klassen durch Leerzeichen getrennt.

```
<p class="grosseschrift bgred">Schrift größer und mit roten Hintergrund</p>
```

```
.grosseschrift {font-size: larger;}
.bgred {background: #FF0000;}
```

Diese Vorgangsweise erlaubt eine bessere Modularisierung unserer Anweisungen.

## CSS EINBINDUNG

Es stehen verschiedene Möglichkeiten zur Verfügung, StyleSheets bzw. Style-Anweisungen mit unseren HTML-Seiten zu verknüpfen. Zusätzlich können praktisch alle der nachfolgend angeführten Arten der Einbindung miteinander kombiniert werden.

### Link zu externen StyleSheets

---

Separate StyleSheets werden mit dem Link-Tag in HTML-Dokumente eingebunden. De facto können beliebig viele externe StyleSheets mit einem HTML-Dokument verknüpft werden:

```
<head>
<link rel="stylesheet" href="styles.css" type="text/css" />
<link rel="stylesheet" href="st_farben.css" type="text/css" />
</head>
```

Dabei wird angegeben:

rel(ation) - in welcher Beziehung die verknüpfte Datei zu dem HTML File stehen soll

href (http reference) - WO liegt das StyleSheet (relativer oder absoluter Pfad)

type (mime-type) - wie das verknüpfte Dokument interpretiert werden soll

### Embedded Styles

---

Erlaubt ist es auch, Styles im Head eines HTML-Dokuments anzugeben. Diese Methode sollte allerdings nur in Ausnahmefällen eingesetzt werden.

```
<head>
<style type="text/css"><!--
h1 {color: #FF0000;}
p.green {background: green;}
--></style>
</head>
```

Diese Anweisung im Head eines HTML-Dokuments wirkt nur in diesem Dokument.

Das Einbetten der CSS Anweisungen in Kommentaren verhindert, daß ältere Browser diese falsch interpretieren.

## Import von externen Stylesheets

---

Muss immer am Anfang von StyleSheet oder am Anfang eines embedded Styles notiert sein. Diese Methode wurde ursprünglich eingeführt, um modulare StyleSheets in ein HTML-Dokument oder ein StyleSheet einzufügen.

Später wurde diese Methode auch verwendet, um Netscape (und Browser unter 5.x) auszutricksen. Netscape versteht die Anweisung @import nicht. Daher kann man einfache CSS (die auch Netscape versteht) mit Link verknüpfen und raffiniertere CSS mittels @import (was von Netscape ignoriert wird).

### Beispiel - Import in HTML Datei

```
<html>
<head>
<title>Meine WebSite</title>
<style type="text/css"><!--
@import url(styles.css);
--></style>
</head>
<body>
Inhalte meiner WebSite
</body>
</html>
```

### Beispiel - Import in CSS Datei

```
@import url(farben.css);
@import url(hintergruende.css);
body {margin: 0px;}
```

## Inline Styles

---

Ein Style darf aber auch einem HTML-Element direkt angefügt werden. Auch diese Methode sollte nur in Ausnahmefällen verwendet werden, da diese Technik ja allem, was wir über die Vorteile von CSS vorausgeschickt haben, widerspricht.

```
<h1 style="color: #FF0000; font-weight: bold;">Das ist eine fette, rote Headline</h1>
```

## span-Tag

Wie der Name sagt, „spannt“ dieser Tag ein angegebenes CSS über ein Element oder Teile dessen, also z.B. über einen Textbereich, ein Wort etc.

`<p>Um die Klasse .bgred an einem ausgewählten Textbereich anzuwenden, <span class="bgred">würden wir hier entsprechend den span-Tag einsetzen.</span>. Nach dem span-Tag erscheint der Text wieder ohne die Attribute dieser Klasse </p>`

```
.bgred {background: #FF0000;}
```

## Kaskadieren - and the Winner is...

Welche Style-Anweisung sich “durchsetzt”, hängt einerseits von der Position derselben ab, andererseits gibt es auch eine “Spezifität” von Klassen und Styles, welche die Prioritäten bestimmen. Für den Anfang genügt es, die Positionen zu beachten:

**Faustregel**

Je näher eine Anweisung an einem Element ist, desto “gewichtiger”.

**Oder anders ausgedrückt:**

Stylesheets werden Zeile für Zeile von oben nach unten durchgeführt,  
beginnend mit externen Stylesheets (in der Reihenfolge der Verlinkung),  
darin evt. importierte Stylesheets an erster Stelle  
danach importierte Stylesheets in der HTML-Datei  
danach embedded Styles, die im head einer HTML-Datei notiert sind  
danach Styles in einem span-Tag  
danach style Attribut eines HTML-Elements (inline Styles)

**Somit gewinnt**

inline Styles über  
span-Tags über  
Embedded Styles über  
Import Stylesheets im HTML über  
(importierte Stylesheets in) Externen Stylesheets

## CSS UND MEDIA TYPES

Eine der großen Stärken von CSS ist die Möglichkeit, verschiedene Stylesheets für verschiedene Ausgabeoptionen zu definieren. Diese Media-Stylesheets werden vom Browser bzw. dem Ausgabegerät automatisch ausgewählt.

#### Erlaubte Werte

all, aural, braille, embossed, handheld, print, projection, screen, tty, tv

Auch bei der Definition von Media-Stylesheets stehen verschiedene Optionen zur Verfügung. Und auch hier sind praktisch alle Kombinationen erlaubt:

```
<head>
<link rel="stylesheet" type="text/css" href="ink.css" media="print">
<style type="text/css" media="braille, embossed"><!--
@import "../tactile.css";
--></style>
<style type="text/css"><!--
@media print {
  h1 {font-size: 22pt; background: white; }
}
--></style>
</head>
```

**ACHTUNG:** Die Anweisungen für die Media Type Styles werden in geschwungenen Klammern zusammengefaßt.

```
<head>
<style><!--
@media screen
{
p.test {font-family:verdana,sans-serif; font-size:14px}
}
@media print
{
p.test {font-family:times,serif; font-size:10px}
}
```

```

}
@media screen,print
{
p.test {font-weight:bold}
}
--></style>
</head>

```

### Überlegungen zu Print Stylesheet

Es hat sich als vorteilhaft erwiesen, sämtliche Content-Elemente nochmals mit einem div#wrapper zu umhüllen. Das erlaubt das Definieren von Rändern und Ausrichtung für alle Elemente einer Seite in einer Anweisung.

Beim Drucken würde man z.B. die Breite der Content-Elemente auf cm oder % reduzieren und Bilder ausblenden. Zusätzlich könnte man auch die Schriftgröße erhöhen.

```

* {font-size: 14pt;}
h1, h2, h3, h4, h5, h6 {font-size: 18pt;}
#wrapper {width: 90%;}
img {display: none;}

```

## CSS BOX MODELL – MARGIN, PADDING, BORDER

Das Verstehen des sogen. Box-Modells ist unumgänglich für das sichere und präzise Layouten mit CSS.

Grundsätzlich wird in (X)HTML unterschieden zwischen

### Block Level Elementen

Elemente wie z.B. das body-Element, DIVs, Absätze, Überschriften, Listen, Tabellen.

### Inline Elementen

Elemente wie z.B. a, em, span und fast alle weiteren Formatierungselemente, so wie auch Bilder und Formularfelder. Sie können Eigenschaften von allen anderen Elementen erben.

### List Item Elementen

Diese Elemente ordnen oder sortieren Aufzählungen.

- ☛ Block Level Elemente nehmen standardmäßig eine Breite von 100%, also die ganze Fensterbreite, ein.
- ☛ Block Level Elemente sind meist von Zeilenvorschüben umschlossen, beginnen also eine neue Zeile.
- ☛ Bilder und Formularfelder sind normalerweise keine Block-level Elemente, können es aber in bestimmten Fällen sein.
- ☛ Viele Eigenschaften von Block Level Elementen werden auf alle weiteren Elemente vererbt und durch ein solches Element wird oft eine neue Zeile im Browser angefangen.
- ☛ Zusätzlich können Block Level Elemente unter bestimmten Bedingungen Eigenschaften von anderen Block Level Elementen erben.

Das derzeit wichtigste und mächtigste Block-Level-Element einer HTML-Seite ist der div-Tag (von DIVision, auch div-Container). Es kann

- ☛ ganze Seiten beinhalten
- ☛ andere Elemente aufnehmen
- ☛ Tabellen ersetzen

### display

Die Art des Elements kann mit dem Attribut display geändert werden Mehr zum Einsatz von div-Containern und dem display-Attribut unter =>CSS und LAYOUT

## Das Box Modell

---

Alle HTML Block-Level-Elemente haben zwei Formatoptionen und drei Abstandsoptionen:

**width, height, margin, padding, border**

Ein solches Element bildet mit den angeführten Format- und Abstandsoptionen eine sogen. Box. Daher spricht man vom Box-Modell.

Sind margin, padding und border auf Null gesetzt, ist das Element so groß, wie durch width und height definiert bzw. so groß wie der Inhalt der "Box".

padding und border werden zu width und height dazugezählt. Nicht so im Internet Explorer 5.x => IE Box Modell Hack

**ACHTUNG:** Die meisten Browser haben ein Standard-Stylesheet sozusagen eingebaut. Dieses definiert meist Randabstände des Body, Linkfarben, Headings, Listen, Formularfelder etc. Werden solche Werte von uns nicht definiert, so wirken die Browserstyles (die von uns definierten Styles nennt man "Autorstyles").

## width - Breite von Elementen

---

Mit Maßangaben werden absolute Breiten angegeben, mit Prozentangaben prozentuelle Werte in Relation zum Elternelement.

### Anwendbar auf

Alle Elemente außer einige Inline-Elemente wie em, strong, span

### Erlaubte Werte

Längenangaben und Prozentangaben

```
#onebox {width: 100px;}
```

```
#twobox {width: 50%;}
```

**Nicht IE kompatibel sind die Attribute**

min-width, max-width, min-height, max-height

## margin - Aussenabstand von Elementen

---

Mit margin wird der Aussenabstand eines Elements definiert.

### Erlaubte Attribute

margin, margin-top, margin-right, margin-bottom, margin-left

### Erlaubte Werte

Maßeinheiten, Prozentangaben, inherit, none

Negative Werte sind erlaubt

### Verkürzte Schreibweise, Beispiele

```
#onebox {margin: 10px 20px 5px 30px;}
```

Das Element hat Aussenabstände oben 10px, rechts 20px, unten 5px, links 30px.

```
#twobox {margin: 10px 20px;}
```

Das Element erhält Aussenabstände oben und unten 10px, rechts und links 20px.

```
#threebox {margin: 10px 20px 30px;}
```

Das Element hat Aussenabstände oben 10px, links und rechts 20px und unten 30px.

## padding - Innenabstand eines Elements

---

Das Attribut padding definiert den Innenabstand eines Elements.

### Erlaubte Attribute

padding, padding-top, padding-right, padding-bottom, padding-left

### Erlaubte Werte

Maßeinheiten, Prozentangaben, inherit, none

Negative Werte sind nicht erlaubt

### Verkürzte Schreibweise, Beispiele

```
#onebox {padding: 10px 20px 5px 30px;}
```

Das Element hat Innenabstände oben 10px, rechts 20px, unten 5px, links 30px.

```
#twobox {padding: 10px 20px;}
```

Das Element erhält Innenabstände oben und unten 10px, rechts und links 20px.

```
threebox {padding: 10px 20px 30px;}
```

Das Element hat Innenabstände oben 10px, links und rechts 20px und unten 30px.

## border - Rahmen eines Elements

---

Das Attribut border definiert die Rahmen eines Elements.

### Erlaubte Attribute

border, border-width, border-style, border-color, border-style, border-color, border-top, border-right, border-bottom, border-left, border-top-width etc., border-top-style etc., border-top-color etc.

### Erlaubte Werte

border: Werte für width, style, color - oder inherit

border-width: Maßangaben, thin, medium, thick, inherit

border-style: solid, dashed, dotted, double, groove, ridge, inset, outset, inherit, none

border-color: Farbangabe, inherit

### Verkürzte Schreibweise, Beispiele

```
#onebox {border: 1px solid #FF0000;}
```

```
#twobox {
border: 10px 5px;
border-style: dashed;
border-color: #FF0000;
}
```

Die Attribute padding und border werden üblicherweise zur Breite eines Elements addiert. Nicht so im Internet Explorer 5.x, dieser zieht padding und border von der Breite ab =>

### IE5 Box Model Hack für störrische Browser

---

Beim folgenden Beispiel für eine Lösung sind die Leerzeichen exakt einzuhalten.

Hier eine Lösung am Beispiel einer Box mit einer Breite von 100px

```
#onebox {
padding: 10px;
border: 5px solid #FF0000;
width: 130px;
width/* *//**/100px;
width: /**/100px;
}
```

Die erste Breite gilt für den IE 5.x

Die zweite Breite ist für Browser außer IE 5.0 für Win und IE5.5 für Win und Mac

Die dritte Breitenangabe ist für Browser außer IE5.5 und IE 5.0 für Win

## CSS LAYOUT - POSITION IS EVERYTHING

Um in CSS präzise Layouts zu erstellen, bedient man sich des div-Elements. Dieses kann man als eine Art "Container" sehen, das verschiedene andere Elemente beinhalten darf.

Das div-Element ist ein Block-Level-Element und wurde explizit dafür vorgesehen, Tabellen zu ersetzen. Das div-Element hat selber praktisch keine Eigenschaften (außer align / left, center, right). Eigenschaften werden erst über CSS definiert.

### display - Art der Anzeige

---

Das display Attribut definiert die Art der Anzeige und wird eingesetzt, um z.B. Block Level Elemente in Inline Elemente umzuwandeln. Wird gerne verwendet, um z.B. eine vertikale Liste in eine horizontale Navigationsleiste zu ändern.

#### Erlaubte Werte

block, inline, inline-block, list-item, compact, run-in, table, inline-table, (weitere table-.... Werte) inherit, none

```
<h1>Kurzbeschreibung für Google</h1>
```

```
h1 {display: none;}
```

### position und top, right, bottom, left

---

Das Attribut position erlaubt in Kombination von Angaben für top, right, bottom, left das präzise Platzieren von Elementen.

#### Erlaubte Werte

absolute, relative, static, fixed, inherit

#### absolute

Positioniert ein Element mit x- und y-Koordinaten absolut zum Fenster oder zu einem positionierten Elternelement. Position absolute zum body führt dazu, daß das positionierte Element mitscrollt. Ein absolut positioniertes Element liegt über den anderen Elementen und beeinflußt deren Position nicht.

```
#rotebox {position: absolute; top: 10px; right: 100px; background: #FF0000;}
```

**relative**

Positioniert Elemente relativ zu der Position, wo sie eigentlich erschienen wären (hätte man sie nicht positioniert). Andere Elemente verhalten sich so, als wäre das Element nicht verschoben.

**static**

Dient dazu, vorherige position-Angaben aufzuheben und führt zur "normalen" Positionierung im Elementefluß.

**fixed**

Wie absolute, aber das positionierte Element scrollt nicht mit.

**top, right, bottom, left**

---

Diese Attribute sind nur in Verbindung mit dem Attribut position (absolute, relative oder fixed) wirksam. Die Angaben beziehen sich auf die entsprechenden Ränder von Elternelementen.

**Erlaubte Werte**

Maßeinheiten, Prozentangaben

```
<div id="boxaussen">  
  <div id="boxinnen">Hier der Inhalt</div>  
</div>  
#boxinnen {position: absolute; top: 30px; right: 40px;}
```

**Resultat**

Das div#boxinnen ist vom div#boxaussen oben 30px und rechts 40px eingerückt.

**float - "schwebende" Elemente**

---

Das float Attribut nimmt Elemente aus dem normalen Fluss heraus - die Elemente "schweben". Umgebende Elemente umfließen das float Element.

Das Floaten von DIVs wird u.a. verwendet, um Spalten zu erzeugen.

### Erlaubte Werte

left, right, inherit, none

## clear - Aufheben von float

---

Das clear Attribut beendet das Floaten von Elementen. Das Element mit clear ist jenes, das nicht mehr umflossen wird.

### Erlaubte Werte

left, right, both, inherit, none

```
.clear {clear: both;}
```

## z-index - Stapelhöhe von Elementen

---

Das Attribut z-index definiert die "Stapelhöhe" eines Elements. Ein Element mit z-index: 200; liegt über einem Element mit z-index: 100;

### Anwendbar auf

Positionierte Elemente

### Erlaubte Werte

Ganze Zahl, auto, inherit

## visibility - Sichtbarkeit von Elementen

---

Das Attribut visibility definiert die Sichtbarkeit eines Elements unter Beibehaltung der Raumforderung desselben. D.h. ein Element, das auf visibility: hidden; gesetzt ist, nimmt dennoch den Platz ein, als wäre es sichtbar gesetzt.

### Erlaubte Werte

visible, hidden, (collapse), inherit

## overflow - Überlauf

---

Das Attribut overflow steuert wie Inhalte eines Elements, die größer sind als das Element, angezeigt werden.

### Erlaubte Werte

visible, hidden, scroll, auto, Inherit

## vertical-align - Vertikale Ausrichtung

---

Definiert die vertikale Ausrichtung eines Elements in Bezug auf sein Eltern-Element.

### Erlaubte Werte

top, bottom, text-top, text-bottom, baseline, middle, sub, super

```
img {vertical-align: top; margin: 10px;}
```

## Horizontal zentrieren

---

Um Elemente browserkompatibel horizontal zu zentrieren, setzt man die seitlichen Ränder auf auto und definiert eine Textausrichtung center. Für inliegende Elemente ist zu beachten, daß die Textausrichtung wieder aufgehoben werden muß, da diese vererbt wird.

```
body {text-align: center;}  
#wrapper {width: 600px; margin: 0px auto;}
```

## CSS - VERERBUNG VON EIGENSCHAFTEN

Wie schon im Abschnitt HTML RELATIONEN von ELEMENTEN angesprochen, findet eine Vererbung bestimmter Eigenschaften statt.

Nicht vererbt werden Eigenschaften von margin, padding, border, background.

Die meisten Attribute erlauben die Eingabe des Wertes "inherit", um ein Erben der Eigenschaften eines Elternelements zu erzwingen.

Eine Liste von Eigenschaften und Vererbungsregeln findet sich unter  
[LINK: http://www.w3.org/TR/REC-CSS2/propidx.html](http://www.w3.org/TR/REC-CSS2/propidx.html)

### Sonderfall - Vererbung von Schriftgrößen

Dieser Sonderfall tritt dann ein, wenn relative Schriftgrößen eingesetzt werden, also z.B. Prozent- oder em-Angaben. Werte für Schriftgrößen werden an Kind-Elemente vererbt, jedoch nicht an weitere Descendants. Demgegenüber werden Schriftgrößen z.B. von einem div-Element zu einem darin liegenden div-Element vererbt und somit multipliziert.

```
<div id="erstebox">Schrift mit 80% einer Basisschriftgröße  
  <div id="innenbox">Schriftdarstellung nur 64%.</div>  
</div>
```

```
#erstebox {font-size: 80%;}
```

#### Resultat

Die Schriftgröße des div#erstebox wird auf 80% gesetzt und in das div#innenbox vererbt. Damit erscheint die Schrift des div#innenbox als  $80\% \times 80\% = 64\%$

#### Lösung z.B.

```
#erstebox, #innenbox {font-size: 80%;}
```

## CSS UND HINTERGRÜNDE

Im Gegensatz zu HTML bietet CSS erweiterte Möglichkeiten zur Darstellung von Hintergrundfarben und -bildern.

#### Erlaubte Attribute

background-image, background-color, background-repeat, background-position, background-attachment

#### Erlaubte Werte

background-image - relative oder absolute URL, inherit, none  
background-color - Farbwerte, inherit

background-repeat - no-repeat, repeat, repeat-x, repeat-y, Inherit  
 background-position - left, right, center, top, bottom, center, Maßangaben, Prozent, Kombinationen aus diesen, inherit  
 background-attachment - fixed, scroll, Inherit

Verkürzte Schreibweise Beispiel

```
#onebox {background: url(image.gif) no-repeat fixed top 10px right 40px #FF0000;}
```

## CSS UND SCHRIFT & TEXT

Auch mit Text und Schrift gibt uns CSS weitreichende Möglichkeiten der Formatierung.

### color - Textfarbe/Vordergrundfarbe

---

Es ist zwar erlaubt, Text auch als Farbnamen zu definieren.

LINK: <http://halflife.ukrpack.net/csfiles/help/colors.shtml>

dies wird aber nicht empfohlen, da es eine Fehlersuche im CSS Dokument erschwert.

```
h1 {color: #FF0000;}
```

Auch die Verkürzung von HEX Codes für Farben auf 3 Stellen ist erlaubt, so steht #FFF für weiß (anstatt #FFFFFF) oder #FOO für #FF0000. Aber auch diese Methode ist nicht zu empfehlen - die „Zeitersparnis“ Farben verkürzt zu schreiben ist minimal, die Unübersichtlichkeit bei evt.Fehlersuche jedoch erheblich.

### font-family - Schriftfamilie

---

Wir geben in CSS Schriftgruppen nach Präferenz an.

```
h1 {font-family: Verdana, Helvetica, sans-serif; color: #FF0000;}
```

Damit sagen wir dem Browser eines Besuchers unserer Seiten, daß wir bevorzugt die Schrift Verdana verwenden möchten, wenn diese nicht verfügbar ist, die Schrift Helvetica und wenn auch diese auf dem PC des Users nicht verfügbar ist, möge eine andere serifenlose Schrift verwendet werden.

**ACHTUNG:** Besteht der Name einer Schrift aus mehr als einem Wort, wie z.B. bei Century Gothic, Trebuchet MS, Times New Roman, so müssen wir diese in Anführungszeichen setzen.

```
h1 {font-family: „Trebuchet MS“, Helvetica, sans-serif; color: #FF0000;}
```

### Lesefreundlichkeit von Schriften

Vom typographischen Standpunkt sollte man für Laufschrift/Basisschrift eine serifenlose Schrift wählen, weil diese in kleinen Grössen besser lesbar ist.

Nicht empfehlenswert ist die Verwendung von Arial, Helvetica. Obwohl diese Schriften im Web am häufigsten verwendet werden, sind sie viel zu eng geschnitten, um lesefreundlich oder ästhetisch zu erscheinen.

Für Überschriften dürfen Serifen-Schriften eingesetzt werden. Diese sind in grösseren Schriftgrößen leidlich lesbar.

### Microsoft Screen Fonts

Microsoft hat speziell für den Bildschirm sogen. "Screenfonts" erstellt, die in gängigen Betriebssystemen standardmäßig vorhanden sind. Diese Schriften erhöhen die Lesefreundlichkeit von Texten am Bildschirm. Nachfolgend gültige MS Screen Fonts:

#### Fixed Width

Andale Mono, Courier New

#### Serif

Times New Roman, Georgia

#### Fantasie

Comic Sans MS

#### Headline

Impact

#### Serifenlos

Arial, Arial Black, Trebuchet MS, Verdana

#### Symbolschrift

Webdings

## Typographische Aspekte

Die typographischen Möglichkeiten sind im Web derzeit noch dürftig. Vor allem verhindert eine inkonsistente Interpretation von Schriftgrößen und Schriftschnitten in verschiedenen Browsern nach-wie-vor eine einheitliche Darstellung von Schriften...

### font-size - Schriftgröße

---

Da die Frage der Schriftgrößen für Usability (Benutzerfreundlichkeit) und Accessibility (Barrierefreiheit) ein essentielles Thema sind, wird diese hier eingehender beleuchtet.

Grundsätzlich ist es erlaubt, Schriftgrößen anzugeben mit

Maßeinheiten: ems (em), points (pt), pixels (px); seltener: inches (in), centimetres (cm), millimetres (mm), picas (pc), x-Höhe (ex)

Keywords: xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger

Prozentangaben: %

**ACHTUNG:** Die Angabe von Schriftgrößen in pt und px ist unter Profis verpönt, da diese Methode Menschen mit nicht 100%iger Sehkraft benachteiligt (die Schriftgröße ist in manchen Browsern nicht oder nicht einfach adaptierbar).

Die Angabe von Schriftgrößen in Keywords führt zu schwer kontrollierbaren Ergebnissen, ist also auch nicht empfehlenswert.

Das gleiche gilt für die Angabe von Schriftgrößen in %, mit der Ausnahme, die in Methode 2 nachfolgend erklärt wird.

Profis bevorzugen die Verwendung von em, da sich diese typographische Maßeinheit relativ an der gewählten Schrift orientiert. Ein em ist so hoch wie das kleine m einer Schrift breit.

Ich persönlich bevorzuge die folgenden zwei Methoden, beide berücksichtigen Menschen mit nicht 100%iger Sehkraft.

#### Methode 1

Im CSS wird im body generell die Schriftgröße auf 1.0em gesetzt:

```
body {font-size: 1.0em;}
```

**ACHTUNG:** Die Kommastelle muß als Punkt notiert werden, nicht als Komma!

Das bewirkt, daß die Schriftgröße in unserem Dokument mit der voreingestellten Standardschriftgröße im Browser des Users „gleichgeschaltet“ wird.

Danach können dann einzelne Elemente (ausgehend von der Basis-Schriftgröße im Browser des Users) mit größerer oder kleinerer Schrift definiert werden (kleinere Schrift hier wirklich nur in Ausnahmefällen einsetzen, z.B. für Copyright oder weniger wichtiger Info).

```
body {font-size: 1.0em;}  
h1 {font-size: 1.4em;}  
.smallprint {font-size: 0.9em;}
```

### Methode 2

Der legendäre CSS Guru Richard Rutter hat getüftelt und getüftelt, wie man wohl Schriftgrößen in allen Browsern so ähnlich wie möglich darstellen kann und hat die Formel entwickelt, daß eine Schriftgröße von 62.5% im body in allen Browsern etwa 10 Pixel ergibt (was das Rechnen anderer Schriftgrößen erleichtert).

Wer Richard's Überlegungen im Detail nachlesen möchte, der liest „How to size text using ems“ LINK: <http://clagnut.com/blog/348/>

Ich setze also im body generell die Schriftgröße auf 62.5% (entspricht 10px).

Gleichzeitig bleiben bei dieser Methode die Schriften für User flexibel und können über den Browser an die Sehkapazität eines Users angepaßt werden.

**ACHTUNG:** Welche Methode auch immer angewendet wird, ist zu beachten, daß Schriftgrößen zu Kind-Elementen vererbt und bei relativen Schriftgrößen dabei verkleinert werden. Ein praktischer Rechner zu diesem Thema findet sich hier: LINK: <http://riddle.pl/emcalc/>

## font-style - Schriftschnitt

---

### Erlaubte Werte

italic, oblique, normal

```
h2 {font-style: italic;}
```

„normal“ muß nur dann verwendet werden, wenn eine vorherige Anweisung außer Kraft gesetzt werden soll.

Italic und oblique haben den gleichen Effekt (zeigen die Schrift kursiv), der Unterschied besteht nur, da manche Schriften den kursiven Schriftschnitt als oblique bezeichnen. Meist liegt man aber mit italic richtig, oblique wird nur selten verwendet.

## font-weight - Schriftstärke

---

Obwohl font-weight auch in Schritten zwischen 100 und 900 angegeben werden kann (400 ist normale Schriftstärke), wird diese Option nicht von allen Browsern unterstützt. Wir beschränken uns daher auf folgende

### Erlaubte Werte

normal, bold, (bolder, lighter, 100 - 900), inherit

```
.wichtig {font-weight: bold; color: #FF0000;}
```

## font-variant - Schriftvariante

---

### Erlaubte Werte

small-caps, normal, Inherit

## text-transform - Text umwandeln

---

### Erlaubte Werte

lowercase, uppercase, capitalize, none, inherit

## text-decoration - Textauszeichnung

---

### Erlaubte Werte

underline, overline, line-through, none, blink

**ACHTUNG:** Der Wert blink nervt und sollte generell nicht verwendet werden!

## text-align - Schriftausrichtung

---

### Erlaubte Werte

left, center, right, justify

**ACHTUNG:** Funktioniert nur in Block-Level-Elementen.

## text-indent - Einrückung

---

### Erlaubte Werte

Maßangaben, Prozent, inherit

```
p {text-indent: 10px;}
```

Rückt jeden neuen Absatz 10 Pixel ein. Negative Werte sind erlaubt, um Text herauszurücken (negative Werte werden nicht von allen Browsern unterstützt).

## line-height - Zeilendurchschuß

---

### Erlaubte Werte

Maßangaben, Prozent, inherit

```
p {line-height: 2.0em;}
```

## letter-spacing - Spationierung

---

Dieses Attribut definiert den Abstand zwischen Buchstaben.

### Erlaubte Werte

Maßangaben, Prozent

```
h1 {letter-spacing: 0.1em;}
```

## Text Pseudo Elemente

Ein winziger Lichtblick für typographisch orientierte Webproducer ist die Einführung der Pseudo-Elemente first-letter und first-line, welche die Gestaltung von Initialen und die Formatierung der ersten Zeile eines Absatzes erlauben.

```
p::first-line {font-size: 3.0em; color: #FF0000;}
p::first-letter {font-style: italic;}
```

**ACHTUNG:** Doppelter Doppelpunkt zwischen Selektor und Pseudo-Element!!!

font - Ausführliche Schreibweise

```
p {font-weight: bold; font-style: italic; font-size: 1.0em; line-height: 2.0em; font-family: Verdana, sans-serif;}
```

font - Verkürzte Schreibweise

```
p {font: bold italic 1.0em/2.0em Verdana, sans-serif;}
```

Die Formel für die verkürzte Schreibweise von font lautet

font: (weight) (style) (variant) size(/line-height) family;

Attribute in Klammern sind optional, die fett gedruckten MÜSSEN bei der verkürzten Schreibweise definiert sein.

## CSS UND LINKS - PSEUDOKLASSEN

Es gibt vier Optionen, das Aussehen von Links zu definieren.

**ACHTUNG:** Die Reihenfolge dieser sogen. „Pseudoklassen“ MUSS genau so eingehalten werden!!! Andernfalls kann es passieren, daß sämtliche Link-Anweisungen im Dokument nicht mehr korrekt funktionieren.

**ESELSBRÜCKE:** Lieber Vetter Hau Ab!

```
a {font-weight: bold;}
a:link {color: #FF0000;}
a:visited {color: #980000; text-decoration: none;}
a:hover {background: #FFCC00;}
a:active {border-bottom: 1px dashed #980000;}
```

a definiert Angaben für Links in allen Zuständen

a:link definiert einen noch nicht angeklickten Link

a:visited definiert einen bereits besuchten Link

a:hover definiert das Verhalten, wenn der Cursor über den Link bewegt wird (rollover)

a:active reagiert auf MouseDown, wird eher wenig verwendet, da nicht sehr hilfreich

**HINWEIS:** hover funktioniert auch auf anderen Elementen, z.B. p:hover  
Allerdings nicht voll browserkompatibel.

Möchte man in speziellen Bereichen abweichende Angaben zu Links definieren, so kombiniert man z.B. ID-Selektoren wie folgt:

```
a#nav:link {color: #FF0000;}
a#nav:visited {color: #980000; text-decoration: none;}
```

**ACHTUNG:** Bei der Definition der unterschiedlichen Link-Zustände ist zu beachten, daß keine "hüpfenden" Texte entstehen (z.B. ein Zustand in font-weight: normal; und einer in font-weight: bold;)

## Links auf Images

---

haben standardmäßig einen unschönen 1 Pixel starken blauen Rahmen. Das ist einfach zu verhindern.

Lösung

```
a img {border: none;}
```

oder

```
a img {border: 0px;}
```

Listen dienen der besseren Gliederung (und damit der erhöhten Lesefreundlichkeit) von Texten, werden aber auch eingesetzt, um Navigationslisten zu erstellen.

### list-style-type - Stil der Listenelemente

---

Definiert das Aussehen der Listenelemente

#### Erlaubte Werte

unordered list - disc, square, circle, none

ordered list - decimal, lower-alpha, upper-alpha, lower-roman, upper-roman, lower-greek, decimal-leading-zero, hebrew, armenian, georgian, cjk-ideographic, hiragana, katakana, hiragana-iroha, katakana-iroha

### list-style-position

---

Bestimmt, ob der Listenelement außerhalb oder innerhalb des Abstands der Liste liegt und ob der Text von mehrzeiligen Listenelementen einrückt oder nicht.

#### Erlaubte Werte

inside, outside, inherit

### list-style-image

---

erlaubt es, Bilder anstatt der Listenelemente zu verwenden.

Meist wird allerdings die Methode bevorzugt, Hintergrundbilder zu verwenden, da hier die Positionierung präziser gelingt.

#### Verkürzte Schreibweise

Es ist erlaubt, list-style Attribute nach folgender Regel zu schreiben:

```
ul {list-style: type position image}
```

```
ul {list-style: square inside url(images/bullet.gif);}
```

## CSS UND ALPHA-TRANSPARENZ

Effekte mit transparenten Bildern (evt. mit hover) erlaubt folgender CSS3 Trick. Es müssen immer alle drei Angaben notiert werden, damit der Effekt weitgehend browserkompatibel ist.

```
filter: alpha(opacity=60); /*Angabe für IE*/  
opacity: 0.6; /*Angabe für CSS3 Browser*/  
-moz-opacity: 0.6; /*Angabe für Firefox*/
```

## CSS VALIDIERUNG – GÜLTIGKEITSPRÜFUNG

Das W3C stellt verschiedene Validatoren zur Verfügung, um zu prüfen, ob man gültiges HTML, XHTML, CSS geschrieben hat. Möglich ist sowohl die Angabe einer URL einer zu prüfenden Seite, die Eingabe des Codes oder der Upload einer Datei.

(X)HTML-Validator [LINK](http://validator.w3.org/): <http://validator.w3.org/>

CSS-Validator [LINK](http://jigsaw.w3.org/css-validator/): <http://jigsaw.w3.org/css-validator/>

LINK-Checker [LINK](http://validator.w3.org/checklink): <http://validator.w3.org/checklink>

## CSS BASICS

**CSS CHEAT SHEETS (E)**

Wer alle CSS Infos auf einem "Schummelzettel" vereint sehen will, ist hier richtig:

**LINK:** <http://www.ilovejackdaniels.com/cheat-sheets/css-cheat-sheet/>

**CSS4YOU (D)**

Eine kompakte Übersicht über Kompatibilität von CSS Selektoren und Eigenschaften

**LINK:** <http://www.css4you.de/browsercomp.html>

**SELFHTML (D)**

Die umfassendste Dokumentation über CSS

Kurzreferenz **LINK:** <http://de.selfhtml.org/navigation/css.htm>

Detail Dokumentation **LINK:** <http://de.selfhtml.org/css/index.htm>

**MAXDESIGN (E)**

Kompakte Erläuterungen zu CSS Selektoren, Floating, Listen in Englisch

Selectutorial **LINK:** <http://css.maxdesign.com.au/selectutorial/>

Floatutorial **LINK:** <http://css.maxdesign.com.au/floatutorial/>

Listutorial **LINK:** <http://css.maxdesign.com.au/listutorial/>

Und eine umfangreiche Anleitung zu vielfältigsten Möglichkeiten, Navigationsleisten mit Hilfe von Listen zu erstellen:

Listamatic1 **LINK:** <http://css.maxdesign.com.au/listamatic/>

Listamatic2 **LINK:** <http://css.maxdesign.com.au/listamatic2/>

## CSS DESIGN

---

### CSSELITE (E)

Wer hier vorgestellt wird, schwebt im 7. CSS-Himmel >;o))

**LINK:** <http://www.csselite.com/>

### CSSZENGARDEN (E)

Die legendäre CSS Spielwiese, wo die Herausforderung darin besteht, aus immer der gleichen HTML-Seite die abenteuerlichsten Designs zu gestalten - ausschließlich durch Änderung der Stylesheets.

**LINK:** <http://www.csszengarden.com/>

### CSSBEAUTY (E)

Auch dies eine Site für CSS-Addicts.

**LINK:** <http://www.cssbeauty.com/>

## CSS Tricks

---

### CSSPLAY (E)

Erstaunlich, was man mit CSS so alles erreichen kann

**LINK:** <http://www.cssplay.co.uk/>

### ALISTAPART (E)

Hier schreiben die wahren CSS Gurus

**LINK:** <http://www.alistapart.com/topics/code/css/>

## DIVERSE

---

### BrowserBugs Internet Explorer (E)

**LINK:** <http://www.positioniseverything.net/explorer.html>