MyDoku Referenze für CSS Level 2.0

(ein Überblick / 2007)

```
Warum:  die ausführliche Beschreibung umfasst über 300S. REC-CSS2-19980512 (http://www.w3c.org)
        einfache Referenzen entsprechen nicht immer meinem individuellen Wissensbedürfniss
        auf meiner Domain kann ich es schneller finden als herum zu googeln.

        Text wie immer stichpunktartig... ;) / [..] = optional / ➔ = "daraus folgt"
```

- **CSS** steht für **C**ascading **S**tyle **S**heets
- Styles definiert **wie** in HTML Elemente **angezeigt werden**
- **External Style Sheets** erleichtern (in größeren Projekten) ungemein die Arbeit und werden in ***.css** Dateien gespeichert
- **Trennung** von **Layout** (Design) und **Inhalt** (Information und Text)

Wie CSS mit (X)HTML?

Innerhalb von (X)HTML

```
a)  <html-tag style= "[CSS- Deklarationen]">...</html-tag>
b)  <head>…  <style type="text/css">
    </style></head>
```

Ausserhalb von (X)HTML, in einer anderen Datei mit z.B: href="formatvorlage.css"

```
a)  <link rel="stylesheet" type="text/css" href="[url]" […]>
```

Einbinden von anderen CSS-Dateien in eine CSS Datei (ausserhalb (X)HTML):

```
@import url("*.css-Datei") [mediatyp, mediatyp, ...]; zwei Beispiele:
@import url("./mylayout.css");
@import url("../voice/loudvoice.css") aural;
```

Einbinden von anderen CSS-Dateien in eine CSS Datei (innerhalb (X)HTML):

```
<head>
   <link rel="stylesheet" type="text/css" media="print, handheld" href="../css/mysimple.css" />
...
</head>
```

Bestimmte CSS-Eigenschaften sind nur für bestimmte **Medien** vorgesehen. Deshalb kann man Datei umschliessend angeben, ob ein Dokument nur auf einem speziellen Medium ausgegeben werden soll, wie: auf dem Bildschirm, auf Papier, mit einem Sprachsynthesizer, etc...
Jeweils am Anfang und am Ende der *.css Datei mit {} begrenzt wird

```
   @media medientyp {
         ...
   }
```

defniert. Medientyp ist z.B: all, screen, print u.v.m….

Medientypen	**Gerätetypen**
aural	Sprachausgabe (=Sprachsynthesizer)
braille	Elektronische Brailleausgabe
emboss	Brailledrucker
handheld	Handheld
print	Drucker
projection	Projektor
screen	Graphikmonitor
tty	Textmonitor
tv	Fernseher

Einige Angaben zur Syntax

Selector = Bezeichner, Properties = Eigenschaften, Value = "Eigenschafts"-Wert, Deklaration = Property+Value

Werden in einer Deklaration eines Selectors (s.u.) ungültige Angaben gemacht, werden diese vom Parser (Browser) einfach ingnoriert ➔
-es gibt also keine Fehlermeldung
-die gültigen Teile einer fehlerhaften Deklaration werden verwendet, so dass die Fehlersuche mitunter schwierig wird.
-nachfolgende Deklaration eines Selectors oder auch nur teilweise neue Porperty Deklaration überschreiben zuvor gemachte Angaben (z.B wenn mehrer Dateien verwendet werden). Sie hierzu:

Kaskade und Spezifizierung ("Genauigkeit") eines Selectors:
Um den Wert für eine Element/Eigenschaft-Kombination zu ermitteln, verwenden die Browser bestimmte Kaskaderegeln.

Stylesheets können drei verschiedene Quellen haben:
-Author -der "Urheber" liefert eine Quelldokument entsprechend der Vereinbahrungen der Dokumentensprache zum Beispiel mittels eines Entwicklungswerkzeuges

-User -der "Benutzer" schreibt eigene *.css-Dateien oder definiert CSS innerhalb der (X)HTML-Datei. Im Normalfall dürften aber Author und User eins sein, da kein Websurfer seine eigene css-Dateien einbinden wird.)
-User Agent -der "Browser" muß ein 'default style sheet' für alle CSS Elemente haben - in darstellbarer Grunddefinition: (tja, und da gibt es dann unterschiedliche Aufassung diverser Browserhersteller wie diese zu sein hat...)

Stylesheets aus diesen drei Ursprüngen überlappen sich im Gültigkeitsbereich und arbeiten gemäß einer Kaskade zusammen. Jeder Stylsheet-Stilregel wird eine Gewichtung zugeordnet. Gelten mehrere Regeln, setzt sich diejenige mit der höchsten Gewichtung durch:

1) Deklarationen für das entsprechende Element und den Zielmedientyp werden ermittlet
2) Urheber-Stylesheets überschreiben die Benutzer-Stylesheets, die wiederum das Std.-Browser-Stylesheet überschreiben. Ausnahme **!important-Regel**: Benutzer-Deklaration überschreibt Urheber-Deklaration
   z.B: `P { text-indent: 1em !important }`
3) Spezifischere Selektoren überschreiben allgemeinere Selektoren (damit sind dann auch eigene Declarationen möglich) Es überschreiben also, genauer auf einen Bezeichner 'bestimmte Regeln', weniger 'bestimmte Regeln': z.B:
   ```
   a {color:blue;}
   a:link {color:red;}
   ```
   ➔ die Farbe des 'linkes' des Ankers <a> ist rot, weil hier die Declaration genauer spezifiziert, sonst ist <a> blau
4) Reihenfolge: Haben zwei Regeln dieselbe Gewichtung, denselben Ursprung und dieselbe Spezifität, erhält die jeweils zuletzt angegebene den Vorrang.
   ```
   p { color: red; }
   p { color: blue; }
   ```
   ➔ p Elemente werden blau weil zuletzt deklariert

Berechnung (je höher der Wert desto vorrangiger ist die Declaration gegenüber einer anderen):
Pseudoelemente werden ignoriert
(a) Addition der ID-Attribute im Selektor
(b) Addition der anderen Attribute und Pseudo-Klassen im Selektor
(c) Addition der Elementnamen im Selektor

```
*              {}  a=0 b=0 c=0 -> Spezifizierung =   0
LI             {}  a=0 b=0 c=1 -> Spezifizierung =   1
UL LI          {}  a=0 b=0 c=2 -> Spezifizierung =   2
UL OL+LI       {}  a=0 b=0 c=3 -> Spezifizierung =   3
H1 + *[REL=up]{}  a=0 b=1 c=1 -> Spezifizierung =  11
UL OL LI.red   {}  a=0 b=1 c=3 -> Spezifizierung =  13
LI.red.level   {}  a=0 b=2 c=1 -> Spezifizierung =  21
#x34y          {}  a=1 b=0 c=0 -> Spezifizierung = 100
```

Einfaches Beispiel:

Selector:	Spezifischer Wert ist:	Grund:
p	1	1 (X)HTML selector
div p	2	2 (X)HTML selectors; 1+1
.tree	10	1 class selector
div p.tree	12	2 (X)HTML selectors und ein class selector; 1+1+10)
#bebop	100	1 id selector
body #content .alternative p	112	(X)HTML selector, id selector, class selector, (X)HTML sel. 1+100+10+1

➔ Das ist zwar alles sehr theoretisch, mag aber ein Grund dafür sein, wenn eine Deklaration nicht das tut was man möchte, obwohl man sicht doch der Anwendung der richigen Synthax an einer bestimmten Stelle sicher ist. Es ist also durchaus wichtig die **Rheihenfolge** von Deklarationen zu beachten, insbesondere von komplexeren Stylesheet-Definitionen.

Die wiederholte Definition eines Selectors mit unterschiedlichen Deklarationen kann dazu genutzt werden, bestimmte Angaben für unterschiedliche Aufgaben in unterschiedliche Dateien auszulagern und so eine Kontext bezogenen Übersicht zu schaffen; z.B: eine Datei color.css für alle Farbdefinitionen.

Gruppierung und Einbettung/Verschachtelung
Man kann Selektoren insbesondere Standardselektoren bei der Deklarierung mit Komma gruppieren, um nicht die gleichen Angaben mehrafch zu wiederholen; also z.Bsp:
statt:
```
h1 {
   color: blue;
}
p {
   color: blue;
}
```

geht es auch so:
```
h1, p {
color: blue;
}
```

Andrerseits kam man auch Eigenschaften von Selektoren die sich innerhalb andere Selektoren (im (X)HTML) befinden verschachtelt angeben; Beispiel, alle Überschriften und Absätze innerhalb nur eines bestimmten Bereichs deklarieren:

```
#abschnitt h1 {
   color: blue;
}
#abschnitt p {
   color: green;
}
```

und dann im (X)HTML

```
<div id="abschnitt">
   <h1>Chocolate curry</h1>

   <p>ein Text</p>
   <p>ein weiterer Text</p>
</div>
```

Groß- und Kleinschreibung wird grundsätzlich für CSS typische Bezeichner nicht berücksichtigt
Ausser für CSS **nicht** typische Bezeichner wie u.a. für die URI, oder die selbst vergebenen Namen der Selectoren

Erlaubte **Zeichen** [A-Za-z0-9_] / ISO 10646-Zeichen 161 und höher, (-);
Bezeichnungen dürfen nicht mit einem Trennstrich oder einer Ziffer beginnen.
Komma in Zahlen nur mit "." (engl. Schreibweise)
Backslash-Escapes möglich, z.B um in Strings "\{" für "{" zu schreiben.

Schlüsselwörter wie: (auto, red, none, serife,..) nicht in Anführungszeichen "" stellen wie "none"
Strings und Zuweisungen hingegen schon: z.B: `h2 {font-family: "Courier New";}`

Kommentare: nur /**/ kein: "//", "#", ";" , "--"oder so und nicht verschachteln:
`/* irgendwas */`
Eventuell möglich `<!–irgendwas -->` wie in (X)HTML

Einheiten
relative Längenangaben sind:
'em' = Größe annährend zur Schriftzeichengröße
'px' = Pixel des Bildschirms
'%' = Prozentangabe; z.B des Bereichs eines Fensters, innerhalb eines eingebetteten Blocks
feste Längenangaben sind:
'pt' = Point (1 point = 1/72 inches)
'**pc**' = Picas (1 pica = 12 points)
'**cm**' (centimetres), '**mm**' (millimetres) '**in**' (inches) = weitere Einheiten

'0' = bei Angabe einer 0 Größe kann die Einheit forgelassen werden; z.B: border: 0; - sonst nicht!

Farben:

Standard HTML 4.01 Farben: (nur 16 als Hex und als engl. Name)
Alphabetisch: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow

black (#000000)	silver (#C0C0C0)	gray (#808080)	white (#FFFFFF)
maroon (#800000)	red (#FF0000)	purple (#800080)	fuchsia (#FF00FF)
green (#008000)	lime (#00FF00)	olive (#808000)	yellow (#FFFF00)
navy (#000080)	blue (#0000FF)	teal (#008080)	aqua (#00FFFF)

transparent = Hintergrundfarbe wird sichtbar
rgb(255,0,0) / rgb(100%,0,0) / #FF0000 / #f00 -> wird verdopplet zu #ff0000

```
color: black;                  Vordergrundfarbe
background-color: #FFFFFF;     Hintergrundfarbe: weiß
```

Selector - Allgemeine Schreibweise:

Ein CSS-Stylesheet besteht aus einer Folge von Regeln. Jede Regel beginnt mit Selektoren, die angeben, auf welche HTML-Elemente die Regel anwendbar ist. Auf die Selektoren folgen, in geschweifte Klammern eingeschlossen, Zuweisungen, die angeben, wie die CSS-Parameter für die selektierten Elemente zu setzen sind. Die Parameter beschreiben das äußere Erscheinungsbild des HTML-Elements, etwa seine Farbe oder seine Schrift.

```
/* Kommentar */
@media type
   selector {
      property: value;
    [ property: value;....]
   }
```

```
}
Beispiel: p {text-align:center; color:red}
```

Selectoren sind mit Komma gruppierbar:
```
selector1, selector2, selector3, selector4 {
   color: black;
}
```

die Angaben für Properties kann man auch (wenn sie zusammengehören) zusammenfassen; value=Wert z.B:
```
selector {
     propertyX-top:    valTop;
     propertyX-right:  valRight;
     propertyX-bottom: valBottom;
     propertyX-left:   valLeft;
     propertyX-more:   valMore;
}
```
entspricht ohne Komma (wenn die Werte sich bedingen):
```
selector {propertyX: valTop valRight valBottom valLeft;}
```

Anderes Beispiel:
```
#mybox { margin: 0 10 5 3; }
   #mybox { margin: 10 3;}      Was ist das? ➔ "Top/Bottom" und "Right/Left" werden zusammengefasst
   #mybox { margin: 5;}         Was ist das? ➔ "Top/Right/Bottom/Left" haben alle den Wert = 5
```

entspricht mit Komma (wenn zusätzliche mögliche Werte):
```
selector { propertyX: valTop valRight valBottom valLeft, valMore;}
```

Außerdem können alle Properties eines Selectors mit ";" getrennt in eine Zeile gebracht werden:

➔ Man kann also eine ganze Menge zusammenhackeln, ohne dass ein Anderer weiß was gemeint ist, bevor er sich nicht über die jeweilgen Properties des entsprechenden Selectors informiert oder diese kennt. Ich würde im Allgemeinen immer davon abraten alles in eine Zeile zu bringen, es sei denn es macht ausnahmsweise Sinn; z.B bei font:
(font: `font-style font-variant font-weight font-size/line-height font-family;`)

Anderes Beispiel:
```
.myFont {
   color: #A1A;
   font: oblique small-caps 900 12px/14px arial;
   font-family: arial, "lucida console", sans-serif;
}
```

Definition "." und "#"

(x)HTML definiert eine Reihe von Attributen, die nahezu jedes HTML-Element aufweisen darf. Diese Attribute sind:
id= das Attribut 'id' gibt Elementen einen innerhalb des Dokuments eindeutigen Identifikator.
class= das Attribut 'class' ermöglicht es, feiner zwischen den Rollen von Elementen zu differenzieren als es die Elementnamen allein zulassen oder neue Rollen von Elementen zu definieren, die voher nicht vorgesehen waren.
style= Mit dem Attribut 'style' können Formatvorgaben für einzelne Elemente getroffen werden.
title= das Attribut 'title' beinhaltet einen Namen oder zusätzlichen Erläuterungstext für das Element.
lang= das Attribut 'lang' erlaubt es, die Sprache des Elementtextes zu spezifizieren.
dir= mit dem Attribut DIR läßt sich die Schreibrichtung (links/rechts) in der Ausgabe festlegen.

➔ in CSS:
"." = class Selector: zur mehrfachen Benutzung. Wenn es mehrfach gebraucht wird z.B: .center {text-align: center}
`.sinnvollerClassName {...}`
in der (X)HTML-Dateil: `<p class="sinnvollerClassName"> texxxxxxt </p>`
Angabe mehrer Klassen an einer Stelle in der (X)HTML-Dateil: `<div class="purple mr1 mb2 bp5"></div>`

Beispiel mit Type Selector (s.u.) `p.colGreen {color: green;}`
in der (X)HTML-Dateil: `<p class="colGreen"> grüner Text</p>`
Wird **kein** Type Selector verwendet wie: `p.sinnvollerClassName` dann entspricht
`.sinnvollerClassName == *.sinnvollerClassName`
und "*" entspricht allen bekannten Type Selectoren (s.u.)

"Unterklassen" möglich: `.sinnvollerClassName1.sinnvollerClassName2`

"#" = id Selector: ein Attribute vom Typ ID wird benutzt, um ein Element eindeutig zu identifizieren.
Zwei solcher Attribute dürfen nicht den selben Wert haben (und Mozilla/Firefox irgnoriert id's die mit einer Nr. beginnen)
`#sinnvollerIDName {...}`
in der (X)HTML-Dateil: `<div id="sinnvollerIDName "></div>`

Definition Unversal Selector

steht für alle Elemente: a) alle b) alle einer Klasse/ID c) alle einer Sprache
a) *

```
b) *.myclass oder *#myid
c) *[LANG=de] oder [LANG=de]
```

Definition Type Selector:

einfach ein Selector der einen bestimmten (X)HTML-Typ meint; z.B: h1
h1 {color: blue;}

Einige Properties von Type Selectoren werden vererbt andere aber nicht! Dies ist beim Deklarieren von Selectoren zu berücksichtigen, da je nach Bedarf bestimmte Properties überschrieben werden, oder eine Änderung in einem 'abgeleiteten' Element einfach nichts bewirkt.

Andererseits können Deklarieren auch allgemeingültig für alle (X)HTML Elemente gemacht werden; wie typisch am Anfang die Randabstände und border von allen HTML-Elementen auf Null setzten, um sie dann explizit wenn benötigt zu setzten:

```
* { margin:  0;
    padding: 0;
    font-size: 100.01%;
}
fieldset, img { border:0; }
```

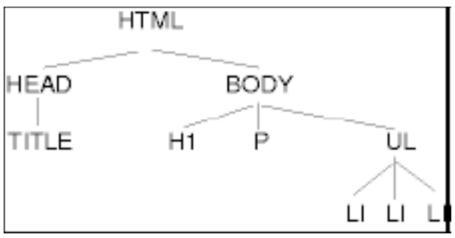
Hier werden alle Elemente mit font Größen in "em" gesetzt und haben eine Zeilenhöhe von 1.3em:

```
h1, h2, h3, h4, h5, h6, p, blockquote, form, label, ul, ol, dl, fieldset, address {
   margin: 0.5em 0;
}
```

Dokumentenbaum

Definition Element = die primären syntaktischen Konstrukte der Dokumentenauszeichnungssprache,
z.B: "p", "table" oder "ul" – im Dokument (Webseite) auch als "Tags" bezeichnet

Ein Dokumentenbaum ist der hierachische Baum der im (X)HTML-Dokument codierten Elemente. Jedes Element in diesem Baum hat genau ein übergeordnetes Element, mit Ausnahme des Wurzel-Elements:



Ein (X)HTML-Dokument ist nach dem folgenden Muster aufgebaut:

```
<html>
   <head>...</head>
   <body>...</body>
</html>
```

Es hat also ein Wurzelelement mit Namen HTML und zwei Kindelemente, HEAD und BODY. Das Element HEAD enthält Verwaltungsinformation zum Dokument. Der eigentliche Inhalt eines HTML-Dokuments befindet sich im Element BODY.

Übersicht XHTML-1.0 Tags*:
In alphabetischer Reihenfolge. NN: Netscape / IE: Internet Explorer / **DTD**: S=Strict, T=Transitional, and F=Frameset
Elem.: **Block-Element** (B), **Inline-Elemente** (I) **Tabellen-Elemente** (T) **Meta-Element** (M) **Gerüst** (G)

Tag	Elem	Beschreibung [EN]	NN	IE	DTD
<!--...-->		Bestimmt einen Kommentar	3.0	3.0	STF
<!DOCTYPE>		Bestimmt den Dokumententyp			STF
<a>	I	Bestimmt einen (Hyperlink-)Anker	3.0	3.0	STF
<abbr>	I	Bestimmt eine Abkürzung	6.2		STF
<acronym>	I	Bestimmt ein Akronym	6.2	4.0	STF
<address>	B	Bestimmt ein Adress-Element	4.0	4.0	STF
<applet>		Veraltet. Bestimmt ein (Java-)Applet	2.0	3.0	TF
<area>	M	Bestimmt einen Bereich (area) innerhab einer ImageMap	3.0	3.0	STF
<b>	I	Bestimmt fette Textausgabe	3.0	3.0	STF
<base>	M	Setzt eine Basis-URL für alle relativen Links der Seite	3.0	3.0	STF
<basefont>	I	Veraltet. Bestimmt einen Basisfont	3.0	3.0	TF
<bdo>	I	Bestimmt die Anzeigerichtuing des Textes	6.2	5.0	STF
<big>	I	Bestimmt big text	3.0	3.0	STF
<blockquote>	B	Bestimmt ein long quotation	3.0	3.0	STF
<body>	G	Bestimmt das body Element	3.0	3.0	STF
 	I	Inserts ein single line break	3.0	3.0	STF
<button>	I	Bestimmt ein push button	6.2	4.0	STF
<caption>	B	Bestimmt eineTabellen-Titelzeile	3.0	3.0	STF
	B	Veraltet. Bestimmt zentrierten Text	3.0	3.0	TF
<cite>	I	Bestimmt ein citation	3.0	3.0	STF
<code>	I	Bestimmt computer code text	3.0	3.0	STF
<col>	M	Bestimmt attribut für Tabellenspalten		3.0	STF
<colgroup>	M	Bestimmt Gruppen von Tabellenspalten		3.0	STF
<dd>	B	Bestimmt eine definition description	3.0	3.0	STF
<del>	I/B	Bestimmt deleted Text	6.2	4.0	STF
<dir>	B	Veraltet. Bestimmt ein directory list	3.0	3.0	TF
	B	Bestimmt eine Section/Abschnitt im Dokument	3.0	3.0	STF
<dfn>	I	Bestimmt ein definition term		3.0	STF
<dl>	B	Bestimmt ein definition list	3.0	3.0	STF
<dt>	B	Bestimmt ein definition term	3.0	3.0	STF
	I	Bestimmt emphasized text	3.0	3.0	STF
<fieldset>	B	Bestimmt ein fieldset	6.2	4.0	STF
<font>	I	Veraltet. Bestimmt text font, size, and color	3.0	3.0	TF
<form>	B	Bestimmt ein Formular	3.0	3.0	STF
<frame>		Bestimmt ein Unterfenster im B. (ein frame)	3.0	3.0	F
<frameset>		Bestimmt ein Set von Frames	3.0	3.0	F
<h1> to <h6>	B	Bestimmt header 1 to header 6	3.0	3.0	STF
<head>		Bestimmt information about the document	3.0	3.0	STF
<hr>	B	Bestimmt eine horizontale rule	3.0	3.0	STF
<html>		Bestimmt das Dokument als HTML-Dokument	3.0	3.0	STF
<i>	I	Bestimmt kursiven Text (italic)	3.0	3.0	STF
<iframe>		Bestimmt ein inline Unterfenster im Browser	6.0	4.0	TF
<img>	I	Bestimmt ein image	3.0	3.0	STF
<input>	I	Bestimmt ein Eingabefeld	3.0	3.0	STF
<ins>		Bestimmt inserted text	6.2	4.0	STF
<isindex>	B	Veraltet. Bestimmt ein single-line input field	3.0	3.0	TF
<kbd>	I	Bestimmt keyboard text	3.0	3.0	STF
<label>	I	Bestimmt ein label for ein form control	6.2	4.0	STF
<legend>		Bestimmt ein title in ein fieldset	6.2	4.0	STF
		Bestimmt ein list item	3.0	3.0	STF
<link>		Bestimmt ein resource reference	4.0	3.0	STF
<map>	M	Bestimmt ein image map	3.0	3.0	STF
<menu>	B	Veraltet. Bestimmt ein menu list	3.0	3.0	TF
<meta>	M	Bestimmt meta information	3.0	3.0	STF
<noframes>	B	Bestimmt ein noframe section	3.0	3.0	TF
<noscript>	B	Bestimmt ein noscript section	3.0	3.0	STF
<object>	I	Bestimmt ein embedded object		3.0	STF
	B	Bestimmt ein ordered list	3.0	3.0	STF
<optgroup>		Bestimmt ein option group	6.0	6.0	STF
<option>		Bestimmt eine Option in einer drop-down Liste	3.0	3.0	STF
<p>	B	Bestimmt einen Paragraphen / Absatz	3.0	3.0	STF
<param>		Bestimmt ein parameter for ein object	3.0	3.0	STF
<pre>	B	Bestimmt pre-formattierten Text	3.0	3.0	STF
<q>	I	Bestimmt ein Zitat "short quotation"	6.2		STF
<s>		Veraltet. Bestimmt durchgestrichenen Text	3.0	3.0	TF

<samp>	I	Bestimmt sample computer code	3.0	3.0	STF
<script>	I	Bestimmt ein Skript	3.0	3.0	STF
<select>	I	Bestimmt ein selectable list	3.0	3.0	STF
<small>	I	Bestimmt small/kleinen Text	3.0	3.0	STF
	I	Bestimmt Section/Abschnitt in einem Document	4.0	3.0	STF
<strike>	I	Veraltet. Bestimmt strikethrough text	3.0	3.0	TF
	I	Bestimmt strong text	3.0	3.0	STF
<style>	M	Bestimmt ein style definition	4.0	3.0	STF
<sub>	I	Bestimmt subscripted text	3.0	3.0	STF
<sup>		Bestimmt superscripted text	3.0	3.0	STF
<table>	B	Bestimmt ein table	3.0	3.0	STF
<tbody>	T	Bestimmt ein table body		4.0	STF
<td>	T	Bestimmt ein table cell	3.0	3.0	STF
<textarea>	I	Bestimmt ein text area	3.0	3.0	STF
<tfoot>	T	Bestimmt ein table footer		4.0	STF
<th>	T	Bestimmt ein table header	3.0	3.0	STF
<thead>	T	Bestimmt ein table header		4.0	STF
<title>	M	Bestimmt the document title	3.0	3.0	STF
<tr>	T	Bestimmt ein table row	3.0	3.0	STF
<tt>	I	Bestimmt teletype text	3.0	3.0	STF
<u>		Veraltet. Bestimmt underlined text	3.0	3.0	TF
	B	Bestimmt ein unordered list	3.0	3.0	STF
<var>	I	Bestimmt ein Variable	3.0	3.0	STF
<xmp>		Veraltet. Bestimmt preformatted text	3.0	3.0	

CSS-Layout

Gut, man kann also mit CSS eine ganze Reihe an den Standard-Tags (Type Selectoren) herumbiegen und das Design einer Seite verändern, vor allem aber zentral verändern. Wie macht man nun ein Layout mit CSS-Befehlen? Hierzu folgenden Überlegungen:

Um die Seite (das Dokument) einzuteilen, Kopfbereich, Seitenkannten z.B für Steuerlemente, Inhaltsbereich, Fußbereich verwendet man das ** - Tag** und definiert seine Eigenschaften. Hierbei gilt es zahlreiche Bedingungen und Überlegungen zu berücksichtigen:

Man muss sich das Dokument / die Webseite als einen durchlaufenden Fluß an Kästchen (**Box**) vorstellen. Dabei gibt es zwei unterschiedliche Typen von Kästchen/Boxen: a) welche die hintereinander fließen und und Zeilen bilden, wobei die Zeilen wegen der Begrenzung durch das Browserfenster auf mehrer Zeilen aufgeteilt werden können, und b) welche die einen Umbruch erzwingen und somit untereinander fließen. (Hintereinander fliesende Elemente heißen **Inline-Boxen**, bilden in einem größeren Absatz mehrere vertikal hinterinander fliessende **Zeilenboxen**, und untereinder fließende Elemente heißen **Block-Boxen**)

Bei Tabellen gibt es noch den Typ der **Compaktbox**, die sich entweder als Inline-Box oder als BockBox verhält (näheres W3C-Referenz).

Es ist ausserdem nützlich, alle **(X)HTML-Elemente** in Gruppen zu unterteilen:
Meta-Elemente / **Block-Elemente** + **Inline-Elemente**

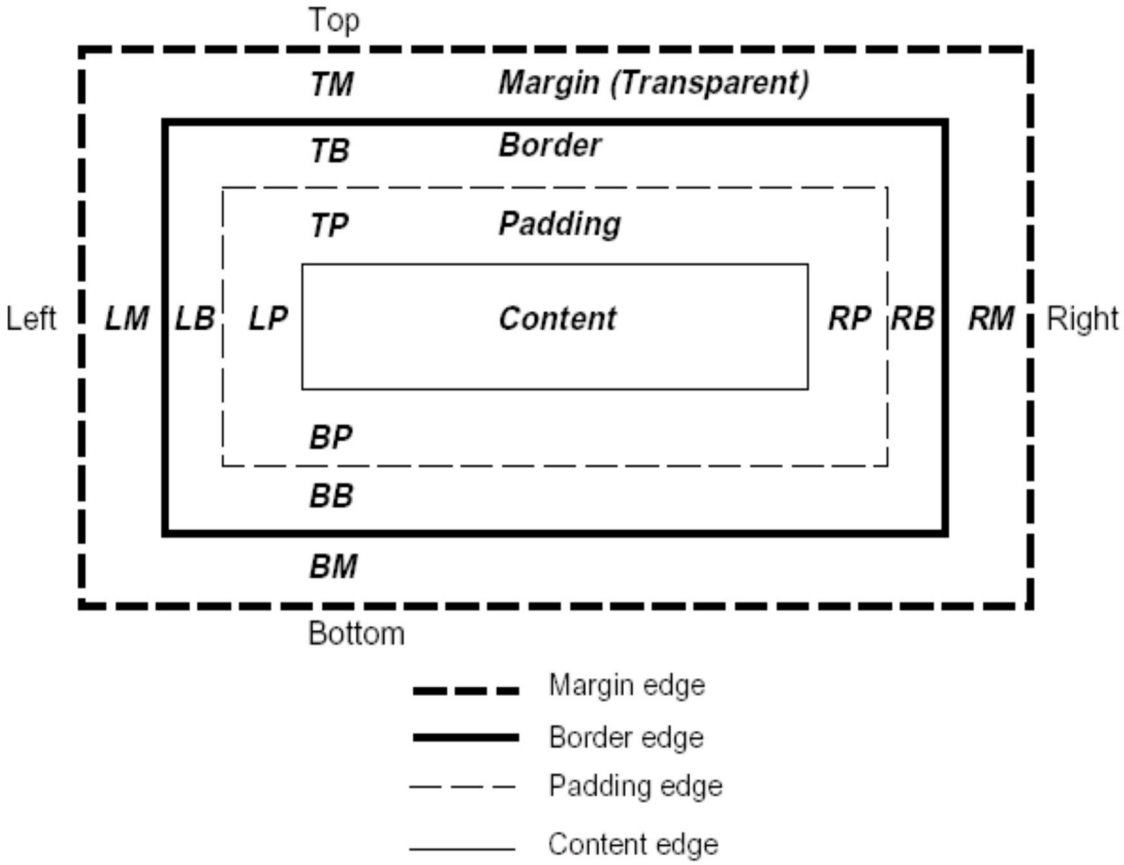
Die (X)HTML-Elemente, die nichts zum Inhalt eines (X)HTML-Dokumentes beitragen, gehören den Metaelementen an oder, wenn sie das äußere Gerüst festlegen, der Gerüstebene. Die (X)HTML-Elemente, die zum Inhalt eines HTML-Dokuments beitragen, gehören den Blockelementen, der Inlineelementen oder der Tabellenelementen an. Bei einigen wenigen Elementen hängt die Zuordnung vom Kontext ab, in dem sie stehen.

Einige Block-Elemente können nur als untergeordnete Kindelemente anderer Block-Elemente auftreten (z.B. dd), man nennt diese Elemente zweitstufige Blockelemente und die übrigen erststufige Blockelemente. Inlineelemente sind alle erststufig.

Das Verhalten einer Box bzw. des Elementes (inline/block) kann mit der 'display'-Eingenschaft geändert werden. Welche Eingeschaft ein Element als Standard hat: siehe oben in **Übersicht XHTML-1.0 Tags** gekennzeichnet mit B oder I (bzw. M/T/G).

Das Box-Modell

Das Box-Modell beschreibt die rechteckigen "Kästchen" (Box) die für jedes Element eines Dokumentenbaums imaginär exestieren und gemäß dem visuellen Formatierungsmodell über die Seite (in der Standardsituation) von oben nach unten aneinander gereiht werden, sozusagen "fliessen". Zu beobachten wenn man mit der Maus das Fenster vergrößert oder verkleinert und keine absoluten festen Größen verwendet wurden.



content edge - Die **Inhalt**skante umschließt den dargestellten Inhalt des Elements (width/height).
padding edge - Die Kante der **Polsterung** umschließt die Polsterung der Box. Hat die Polsterung die Breite 0, fällt die Kante der Polsterung mit der Inhaltskante zusammen.
border edge - Die **Rahmen**kante umschließt den Rahmen der Box. Hat der Rahmen die Breite 0, fällt die Rahmenkante mit der Kante der Polsterung zusammen.
margin edge - Die **Rand**kante umschließt den Rand der Box. Hat der Rand die Breite 0, fällt die Randkante mit der Rahmenkante zusammen. Die Randkante ist nie sichtbar.

Jeder Teil einer Seite ist eine individuelle Box, die dorthin plaziert werden kann, wo immer man es braucht abhängig von der Box des Elternelementes (es sei denn das Element ist inline, ein Inline-Box-Element, z.B.: Text).

Viele Positionen und Größen von Boxen werden relativ zu den Kanten des rechteckigen, der Box umschließenden, Blockes berechnet. Eine Box ist jedoch nicht auf diesen umschließenden Block begrenzt und kann überlaufen.

Die Wurzel (Root) des Dokumentbaums erzeugt eine Box, die als umschließender **Ausgangsblock** für ein nachfolgendes Layout dient und kann nicht positioniert oder gefloatet werden.

Wenn eine Block-Box (z.B eine Box) eine weitere Block-Box z.B. <p> enthält, wird erzwungen, dass die umschliessende Block-Box nur weitere Block-Boxen enthält, indem alle ev. weiteren Inline-Boxen in sogenannte **Anonyme Block-Boxen** einhüllt werden.

Anonyme Inline-Boxen: z.B.: `<p>Irgendein <em>betonter</em> Text</p>`
Die Box die für die Betonung () erzeugt wird, ist eine inzeilige Box, aber die für die anderen beiden Inline-Boxen [Irgendein] und [Text] erzeugete Boxen sind Inline-Boxen, die von einem Element auf Blockebene erzeugt wurden, nämlich <p>. Die letzteren werden, weil ihnen kein entsprechendes Element auf Inline-Ebene explizit zugeordnet ist, auch als anonyme Inline-Boxen bezeichnet. Solche anonyme Inline-Boxen erben vererbbare Eigenschaften von ihrer übergeordneten Block-Box.

Visuelles Formatierungsmodell

Jedes Element im Dokumentbaum erzeugt gemäß dem Box-Modell null oder mehr Boxen.

Das **Layout der Boxen** wird bestimmt durch:
- Box Abmessungen und Box-Typ (Inline/Block).
- Positionierung: Normaler Fluß, fließend, absolute und relative Position
  (engl.: position as: normal flow, float, absolute and relative).
- Beziehungen zwischen den Elementen untereinander gemäß der Dokumentenbaumhierachie.
- Externe Beeinflußungen (z.B. Viewport-Größe, eigene Größe von Bildern usw)

Zu beachten: die Darstellung auf Endlosmedien oder auf Seitenmedien (anderer Randeigenschaften).

Viewport == Browserfenster durch den das Dokument betrachtet wird
-Größenänderungen bewirken Darstellungsänderung des Dokumentes
-kleiner als der "Umschliesende Ausgangsblock" des Dokumentes z.B: <body> bewirkt Scrollfunktion

Box-Abmessungen:

`width:` gibt die Inhaltsbreite von Boxen an, die auf Blockebene erzeugt wurden, sowie die Inhaltsbreite von ersetzten Elementen. Feste Breite, Prozentuale Breite oder 'auto', d.h. die Breite ist von den Werten anderer Eigenschaften abhängig.
Die Breite von Zeilen-Boxen wird durch ihren umschließenden Block vorgegeben

`min-width: / max-width:` (nicht < IE6.0 / FireFox ok) Maximale oder Minimale Breite einer Block-Box

`height:` dito: Höhe als feste Breite, Prozent oder auto
Bezieht sich nicht auf 'nicht ersetzte' inzeilige Elemente (z.B: Text) Änderungen bei diesen Elemten mit 'line-height'-Wert des Inline-Elements

`hight: nn%` Höhenangaben, die in Prozent angegeben werden, beziehen sich auf die Höhe des umschließenden Blockes (dem Elternelement im Dokumentenbaum). Wird für diesen umschließenden Block keine Höhe angegeben, so wird der angegebene Prozentwert als auto interpretiert. Das Element wird trotz Höhenangabe nur so hoch, wie der Inhalt es erfordert.
➔ Es muss allen Elternelementen eine bestimmte Höhe zugewiesen werden;
auch html und body sind Elternelemente

Box-Typ: Die Angabe der display: xxxx Eigenschaft definiert den Box-Typ für das jeweilige Element:

`display:`

`inline` inline-Elemente folgen dem natürlichen Dokumentenfluß und erzeugen keine neue Zeile
Inline-Elemente dürfen Text und weitere Inline-Elemente aber keine Block-Elemente enthalten.
Inline-Elemente unterteilen sich in ersetzende und nichtersetzende Elemente:

ersetzende Elemnte: werden meist durch etwas ersetzt wie bei `img, input und textarea`
width: und height: können angegeben werden

nicht ersetztende Elemente: hier ist eine **Angabe von Höhe und Breite unwirksam**
Beispiel:
`a, abbr, acronym, b, bdo, big, br, button, cite, code, dfn, em, i, img, input, kbd, label, map, object, q, samp, script, select, small, span, strike, strong, sub, sup, textarea, tt, var,` *`basefont`*, *`font`*,…

`block` block-Elemente **erzwingen** einen **Zeilenumbruch Vor und Nach dem Element**
Block-Elemente können Text, Inline- und je nach Element weitere Block-Elemente enthalten
Beispiel:
`address, blockquote,` **`div`**`, dl, fieldset, form, h1-h6, hr, noframes, noscript, ol, p, pre, table, ul`*`, center, dir, isindex, menu,`* …

`none` block-Elemente werden nicht angezeigt (z.B: @media print { .myNavBar {display: none;}}
oder mit JavaSkript verstekte Elemente werden erst nach klick auf n.n. anzeigen)

Positionierung: eine Box kann gemäß den drei folgenden Positionierungsschemata ausgelegt werden:
-Normaler Fluß (siehe oben)
-Floats (Im Float-Modell wird eine Box zunächst gemäß dem normalen Fluss ausgelegt, dann aus dem Fluss entfernt und so weit wie möglich nach links oder rechts verschoben. Weiterer Inhalt kann entlang der Seite eines Floats fließen.)
-Absolute Positionierung (Die Box wird vollständig aus dem normalen Fluss entfernt (sie hat keinen Einfluss auf spätere gleichrangige Elemente), und es wird ihr eine Position relativ zu ihrem umschließenden Block zugewiesen.)

`position:`

`static` ist die Standardvorgabe und rendert das Element im normalen HTML-FLuß
Angaben von `left, top, right, bottom` sind unwirksam.

`relative` rendert auch das Element im normalen HTML-FLuß, kann aber zusätzlich zu seiner normalen Position einen `Offset` haben der mit `top, right, bottom und left` angegeben wird.
Die nachfolgenden Elemente verhalten sich so, als wäre das Element nicht verschoben

absolute entbindet das Element dem normalen Fluß und setzt eine feste Position: **top, right, bottom, left**
Die absolute Position wird relativ zu den Rändern des Eltern-Elements berechnet, **wenn** dieses ebenfalls positioniert ist, oder wenn es der Seiteninhalt (<body>) ist.
Da sich der Seiteninhalt scrollen lässt, werden absolut positionierte Elemente mitgescrollt.

fixed entbindet das Element dem normalen Fluß und zeigt es mittels Angabe von top, right, bottom und left an einer bestimmten Stelle des Browserfenster unabhängig von anderen Elementen (nicht < IE6.0 / FireFox ok)

float:
- left; Block-Box links ausgerichtet - der Inline-Boxen fließen an der rechten Seite der Box vorbei, beginnend oben
- right; Block-Box rechts ausgerichtet - der Inline-Boxen fließen an der linken Seite der Box vorbei, beginnend oben
- none; Die Box gleitet nicht

width: 80px; Eine Floating-Box muss eine explizite Breite haben, die über die 'width'-Eigenschaft zugewiesen wird!

Falls **nicht genügend horizontaler Platz** in der aktuellen Zeile für die Float-Box vorhanden ist, wird sie Zeile für Zeile nach unten verschoben, es genügend Platz gibt; z.B dann wenn man das Browserfenster zusammenschiebt.

Nicht positionierte Block-Boxen, die vor und nach der Floating-Box erzeugt werden, fliessen an der Float-Box vertikal vorbei, weil sich ein Float nicht im Fluss befindet.

clear:
- left; [right; both;] beendet für ein Block-Element die Float-Eigenschaft d.h. man gibt damit an, welche Seiten der Boxen eines Elements nicht neben einer vorhergehenden Floating-Box stehen dürfen.
- none; keine Beschränkungen für die Position des Elemtentes zu der Float-Box – quasi wie nicht angegeben.

display, position und float: arbeiten wie folgt zusammen (wenn nicht 1 dann 2 dann 3 etc..):
1) display = none; ➔ position und float wird ignoriet ➔ es gibt keine Box
2) position = absolute; oder fixed; ➔ display wird auf block gesetzt und float wird ignoriert u. auf none gesetzt
3) float = anderer Wert als none ➔ display wird auf block gesetzt und die Box gleitet
4) = es gelten andernfalls die restlichen 'display'-Eigenschaften wie angegeben

Der umschließende Block für eine positionierte Box wird durch den nächsten positionierten Vorfahren eingerichtet oder, falls es keinen solchen gibt, durch den umschließenden Ausgangsblock des Dokumentenbaumes

Absolute positionierte Box als eine untergeordnetes Elemente einer relativ positionierten Box:
Obwohl eine übergeordnete relative Box nicht wirklich verschoben wird, ist es möglich, dass diese Box als umschließender Block für positionierte Nachkommen dienen kann, da ihre 'position'-Eigenschaft 'relative' gesetzt ist.

Umschließender Block / Verschachtelung
Position und die Größe der Boxen eines Elements werden relativ zu einem bestimmten Rechteck berechnet, dem umschließenden Block. Es gelten folgende Bedingungen:

a) Der umschließender Ausgangsblock, in dem sich das Wurzelelement befindet, wird vom Benutzerprogramm ausgewählt.
b) Für andere Elemente wird der umschließende Block durch die Inhaltskante der nächsten benachbarten Box auf Blockebene gebildet, es sei denn, das Element ist absolut positioniert.
c) 'position: **fixed'** dann wird der umschließende Block durch den Viewport eingerichtet.
d) 'position: **absolute**' dann wird der umschließende Block durch den nächsten Vorfahren der eine anderen 'position' als 'static' hat eingerichtet, und zwar wie folgt:
Falls es einen Vorfahren auf Blockebene gibt, wird der umschließende Block durch die Polsterungskante (padding) des Vorfahren gebildet.
Falls der Vorfahre auf Inline-Ebene liegt, ist der umschließende Block von der 'direction'-Eigenschaft des Vorfahren abhängig.
Falls es keinen solchen Vorfahren gibt, richtet die Inhaltskante der Box des Wurzelelements den umschließenden Block ein.

Ebenen (Layer)
Eine Webseite die mit CSS erzeugt wird ist dreidimensional. Es gibt eine weitere z-Achse oder Tiefe in der Seite ausser der x- und y-Achse. Deshalb können die Box'en übereinander geschichtet (gelayert) werden und somit teilweise komplizierte zusammengesetzt visuelle Effekte erzeugt werden.

Angewendet auf positionierte Elemente:
a) Z-Achsen-Positionen sind vor allem relevant, wenn sich Boxen visuell überlappen
b) Jede Box gehört zu einem Stapelkontext. Jede Box in einem bestimmten Stapelkontext hat eine ganzzahlige Stapelebene, die ihre Position auf der Z-Achse relativ zu anderen Boxen im selben Stapelkontext angibt.
c) Boxen mit höheren Stapelebenen werden immer vor Boxen mit niedrigeren Stapelebenen positioniert.
d) Boxen können negative Stapelebenen haben.
e) Boxen mit derselben Stapelebene in einem Stapelkontext werden von unten nach oben der Reihenfolge im Dokumentbaum entsprechend gestapelt.

f) Standardmäßig verhält sich eine Box so, dass dahinterliegende Boxen durch die transparenten Bereiche in ihrem Inhalt sichtbar sind. Dieses Verhalten kann mit Hilfe einer der existierenden Hintergrundeigenschaften überschrieben werden.

`inherit` die Werte des Elternelementes werden geerbt
(jedoch bugy [2006]: <= IE6.0 vererbt nicht immer / FireFox gibt das Erbe an Kindelemente weiter)

Überlauf, Abschneiden und Sichtbarkeit
Im Allgemeinen wird der Inhalt einer Block-Box auf die Inhaltskanten der Box begrenzt. In bestimmten Fällen kann es einen Überlauf einer Box geben, das heißt, ihr Inhalt liegt teilweise oder ganz außerhalb der Box, z.B: wenn eine Zeile nicht umbrochen werden kann, eine Box auf Blockebene ist zu breit für den umschließenden Block oder eine Verschiebung nach Ausserhalb des umschließenden Block durch absolute Positionierung der untergeordneten Box. Eine eingebette Box hat negative Ränder, was dazu führt, dass Teile außerhalb der umgebenden Box liegen.

`overflow:`
- `visibel;` der Inhalt wird nicht abgeschnitten; d.h. er wird ausserhalb der Block-Box angezeigt
  je nach Browser anderes verhalten: IE 6.0 vergrößert die Box trotz fester Angaben, FF überschreibt
- `hidden;` der Inhalt wird an der definierten Block-Box-Grenze abgeschnitten
- `scoll;` wie 'hidden' aber es werden scroll-Balken erzeugt und der Inhalt kann in einem Unterfenster gescrollt werden
- `auto;` vom Browser abhängig (meistens wie scroll)

`clip:` bezieht sich auf Elemente, deren 'overflow'-Eigenschaft einen anderen Wert als 'visible' hat und definiert, welcher Teil des dargestellten Inhalts eines Elements sichtbar ist.

`visibility:` gibt an, ob die durch ein Element erzeugten Boxen angezeigt werden. Unsichtbare Boxen haben weiterhin Einfluss auf das Layout; ggf. `display: none;` verwenden
- `visible;` die Box ist sichtbar
- `hidden;` unsichtbar (völlig transparent), beeinflusst aber das Layout.
- `collapse;` für Zeilen oder Spalten bei Tabellen, ansonsten hat es dieselbe Bedeutung wie 'hidden'.

Erzeugter Inhalt, automatische Nummerierung und Listen

In einigen Fällen sollen Inhalte dargestellt werden die nicht aus dem Dokumentenbaum stammen. Inhalt kann mit hilfe mehrer Mechanismen erzeugt werden:
a) Die 'content'-Eigenschaft, in Kombination mit den Pseudoelementen :before und :after.
c) Die akustischen Eigenschaften 'cue-before' und 'cue-after'
d) Elemente mit dem 'list-item' Wert für die 'display'-Eigenschaft.

Pseudoelemente :before und :after

:before und :after geben die Position von Inhalt vor und hinter dem Dokumentbauminhalt eines Elements an.
Die 'content'-Eigenschaft gibt in Kombination mit diesen Pseudoelementen an, was eingefügt wird.

Automatische Zähler und Nummerierung

Zum Beispiel für die Möglichkeit, Kapitel und Abschnitte mit „Kapitel 1", „1.1", „1.2" usw. zu nummerieren.

```
H1:before {
   content: "Kapitel " counter(chapter) ". ";
   counter-increment: chapter;                 /* 1 zu chapter addieren */
   counter-reset: section;                     /* section auf 0 setzen */
}
H2:before {
   content: counter(chapter) "." counter(section) " ";
   counter-increment: section;
}
```

u.s.w. siehe W3C-Referenz

Markierungen (Marker) und Listen
Eine Hauptblock-Box (für den Inhalt eines Elements) und eine separate Markierungs-Box (für Dekorationen, wie beispielsweise einen Punkt, ein Bild oder eine Nummer). Die Position der Haup-Box wird nicht beeinflußt.

zum Beispiel Punkte hinter jeder Nummer einer geordneten Liste einfügen:

```
LI:before {
   display: marker;
   content: counter(mycounter, lower-roman) ".";
   counter-increment: mycounter;
}
```
mit
```
<BODY>
      <OL>
         <LI>Das erste Element.
         <LI>Das zweite Element.
         <LI>Das dritte Element.
      </OL>
</BODY>
```

erzeugt:

i. Das erste Element.
ii. Das zweite Element.
iii. Das dritte Element.

Listen
Die Listeneigenschaften erlauben eine grundlegende visuelle Formatierung von Listen.

`display: list-item;` erzeugt eine Haupt-Box für den Elementinhalt und eine optionale Markierungs-Box.

`list-style-type` Erscheinungsbild, z.B: circle, square, decimal, lupper-roman, lower-alpha, lower-latin

oder z.B. Bild für die Listenelementmarkierung:
```
ul {
   list-style-image: url("http://mydomain.tld/ellipse.png")
}
```

`list-style-position` gibt die Position der Markierungs-Box in der Hauptblock-Box
- `inside` Die Markierungs-Box ist die erste inzeilige Box in der Hauptblock-Box, welcher der Elementinhalt folgt.
- `outside` Die Markierungs-Box liegt außerhalb der Hauptblock-Box.

`list-style` Beispiel:
- `UL { list-style: upper-roman inside }` Jede UL inside
- `UL UL { list-style: circle outside }` Jeder UL-Nachfahre einer UL und für diesen 'outside'

list-style: ist eine zusammenfassende Notation, mit der die drei Eigenschaften zusammen angegeben werden können.

==========

Einige Dinge die mich im Moment nicht interessiert haben, aber für Andere relevant sein können hier nicht oder noch nicht erwähnt worden sein ➔ dann einfach in der Orignal Doku nachschauen: REC-CSS2-19980512 (http://www.w3c.org)