Cascading Style Sheets (CSS)

Mit Stylesheets können Dokumente, die eine bestimmte Grundstruktur besitzen, gesondert formatiert werden. Tatsächlich wurde HTML ursprünglich nur dazu entwickelt, um einem im Internet lesbaren Dokument eine übersichtliche Struktur zu geben: Überschriften, Absätze, Tabellen, etc. Im Laufe der verschiedenen HTML-Versionen hat sich dieser Grundansatz allerdings immer mehr verwässert. Zu den Dokumentenstrukturen sind auch Formatierungsmöglichkeiten hinzugekommen. Ein Beispiel dafür ist das <font>-Tag mit seinen unterschiedlichen Attributen. Das Grundprinzip sieht aber vor, dass Struktur und Formatierung getrennt werden sollen. Ansonsten würde ein HTML-Dokument immer unübersichtlicher und immer schwerer zu pflegen sein. Deshalb werden inzwischen vom W3 Konsortium nach und nach unnötige HTML-Tags, wie z.B. eben auch das <font>-Tag als deprecated, also unerwünscht, verbannt und sollen nicht mehr eingesetzt werden.

Das Arbeiten mit Stylesheets hat sehr viele Vorteile und macht das Arbeiten wesentlich angenehmer. So kann man durch die Trennung von Struktur (HTML) und Format (CSS) für das gesamte Dokument und sogar für das gesamte Projekt (also alle Homepage-Seiten) einheitliche Formatierungen vergeben - mit einer Zeile Text. Man kann außerdem, bei einer geschickt aufgebauten Dokumentenstruktur, Stylesheets einfach austauschen und so die Seiten völlig anders aussehen lassen - ohne die HTML-Dokumente auch nur angefasst zu haben. Außerdem sind die Möglichkeiten, die CSS Formatierungen bieten, wesentlich umfangreicher als das, was HTML alleine bieten kann.

Allerdings ist das Arbeiten mit CSS auch nicht unproblematisch. Die so genannten Browser-Kriege, also die Konkurrenz-Gefechte um die Marktbeherrschung zwischen Internet Explorer, Netscape, Opera, etc. haben dazu geführt, dass einige CSS-Formate in unterschiedlichen Browsern unterschiedlich dargestellt werden. Auch hat kein Browser bisher alle CSS Eigenschaften der mittlerweile bereits zweiten CSS-Version umgesetzt - obwohl CSS2.0 bereits 1998 veröffentlicht wurde. Man muss sich also bei so manchen Formatierungsversuchen auf unangenehme Überraschungen einstellen und dann versuchen, eine Lösung zu finden, die das Problem umgehen kann. Trotzdem bleibt genügend übrig, um mit CSS erst richtig Spaß am Webseiten-Gestalten zu bekommen.

Der Aufbau

Ein CSS Format bezieht sich immer auf ein - oder mehrere - bestimmte HTML-Tags, die es formatiert. CSS erfindet also keine neuen Strukturen, sondern formatiert die vorhandenen. Deshalb muss man sich darüber klar werden, mit welchem Tag und mit welcher CSS Eigenschaft das Erwünschte erreicht werden kann. So kann man z.B. einem Absatz <p> zwar eine bestimmte Schriftfarbe geben, aber ihn nicht anklickbar machen - dafür muss wieder ein Link-Tag <a> hinzugefügt werden.

Die Syntax eines CSS Formates sieht so aus:

```
p {color:#336699; }
```
Selektor {Eigenschaft:Wert;}

In diesem Fall bezieht sich die Eigenschaft color auf das <p>-Tag (den Selektor), dem sie eine blaue Farbe zuweist.

Anders als bei den HTML-Tags, gibt es bei einem CSS-Format keine Anführungszeichen. Eigenschaft und Wert werden mit einem Doppelpunkt getrennt. Werden mehrere Eigenschaften auf den gleichen Selektor angewendet, müssen sie mit einem Semikolon getrennt werden (in HTML-Tags war das ein Leerzeichen).

CSS einbinden:

Es gibt drei verschiedene Methoden, CSS-Formate in eine HTML-Datei einzubinden.

1. direkt im Element
Es gibt für CSS ein eigenes Attribut, das in fast jedem HTML-Tag auftauchen darf (außer in <html> und innerhalb des Kopf-Bereichs). Das ist das Attribut style="". In ihm wird als Wert das CSS-Format deffiniert:

```
<p style="color:#336699;">Hier ist Text</p>
```

2. Zentral im Dokument:
Für den Head-Bereich gibt es ein eigenes <style>-Tag, das dann die Formate umschließt. Formate, die hier angegeben werden, können für das gesamte Dokument gültig sein. Im Beispiel also für alle Absätze:

```
<style type="text/css">
<!--
p {color:#336699; }
-->
</style>
```

Die <!-- Kommentare --> um die Formate sorgen dafür, dass Browser, die kein CSS darstellen können, diese Angaben nicht als Text auf dem Bildschirm ausgeben

3. in externer Datei:
Die beste, weil unabhängigste Methode, ist, die Stylesheets in einer externen Datei auszulagern. Auf diese Weise kann man zentral für das gesamte Web-Projekt Formatierungen vornehmen. Eine solche Datei ist eine reine Textdatei, die die Dateiendung .css erhält. In ihr werden die jeweiligen Formate einfach untereinander nach dem oben kennen gelernten Schema

```
p {color:#336699; }
```

aufgelistet. Eine solche Datei wird ebenfalls im Head des HTML-Dokuments mit der folgenden Angabe eingebunden (der Dateiname ist dabei frei wählbar):

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

Es können auch alle drei Methoden gleichzeitig verwandt werden! Das macht mitunter Sinn, wenn eine einzelne Seite - oder dort ein einzelner Bereich - aus irgendeinem Grund anders formatiert werden soll als der Rest des Projektes. Dabei haben die drei Methoden aber eine Rangfolge in der Wertgkeit. So wird verhindert, dass bei drei verschiedenen Formatierungen z.B. des <p>-Tags ein Konflikt auftritt. Die jeweils höherrangige Methode bestimmt also das Format. Im Folgenden die Wertigkeit von Höchstrangig bis Niedrigstrangig:

1. im Element
2. im Dokument
3. in externer Datei

Grundlegende CSS Formate

Werte in Klammern sind Alternativwerte, die auch, aber nicht gleichzeitig gesetzt werden dürfen.

```
p {color:#336699; }
```
Selektor {Eigenschaft:Wert;}

Eigenschaft	Wert	Bedeutung
/* Kommentar */		gilt sowohl in externer Datei als auch innerhalb des <style></style> Elements im head
font-family:	Georgia, "Times New Roman", Times, serif (Beispiel)	mehrere **Schriftarten** werden mit einfachem Komma getrennt, enthält ein Schriftname Leerzeichen, muss er in Anführungszeichen gesetzt werden
color:	blue (#336699)	**Schriftfarbe**
font-style:	italic (normal)	**Schriftstil:** gilt nur für kursiv
font-weight:	bold (bolder, lighter, 100 - 900, normal)	**Schriftgewicht:** gilt nur für fett
font-size:	12px; (pt, em, cm, ...)	**Schriftgröße**: Achtung: Dezimalstellen mit . (Punkt) trennen (0.9em)!!
text-decoration:	underline (overline)	**Textdekoration**: unterstrichen
text-align:	left; (right, center, justify)	**Ausrichtung**: lässt sich auch auf die meisten Inline-Elemente, wie <img> oder <a> anwenden (der IE lässt auch Block-Elemente zu)
margin: (margin-top, margin-right, margin-bottom, margin-left)	20px; (auch auto)	**Außenabstand**: Mit den Eigenschaften in Klammern lassen sich die Abstände separat angeben
padding (padding-top, padding-right, padding-bottom, padding-left)	10px; (auch auto)	**Innenabstand**: siehe margin
border-width:	2px	**Rahmendicke**
border-style:	solid (dashed, dotted, double)	**Rahmenstil**: durchgängige Linie, gestrichelt, gepunktet, doppelte Linie
border-color:	blue (#336699)	**Rahmenfarbe**

`border:`	`1px solid #336699;`	**Rahmen gesamt**: auf diese Weise lassen sich alle Einzelwerte zusammen angeben - die Reihenfolge muss eingehalten werden.
`background-color:`	`#FFDDDD (yellow)`	**Hintergrundfarbe**
`background-image:`	`url(../background2.gif);`	**Hintergrundbild**: Hier gehört die Klammer zum Wert dazu, in ihr wird das Bild referenziert
`list-style-image:`	`url(stern.gif);`	eigene **Bullet Grafik** einbinden
`list-style-type:`	`decimal (upper-roman, lower-alpha, disc, circle, square, none)`	**Listen-Typ**: auf Unterschiede zwischen ul und ol achten
`float:`	`left; (right)`	**Textfluss**: bei float:left steht das Element links und der Text fließt rechts vorbei (manchmal muss noch eine width Angabe dazu genommen werden); eine margin-Angabe für den Abstand zum umfließenden Text ist hier sinnvoll
`clear:`	`left (right, both)`	**Textfluss beenden**: ab hier läuft der Text wieder unterhalb des Elements weiter (clear:left bei float:left etc, - oder clear:both)

Pseudoklassen für Verweise:

mit diesen Pseudoklassen kann man Links gezielt formatieren

`a:link {}` - noch nicht besuchte Seiten
`a:visited {}` - besuchte Seiten
`a:focus {}` - beim Benutzen der Tabulator-Taste
`a:hover {}` - wenn man die Maus darüber hält
`a:active {}` - gerade angeklickt

Achtung: die hier angegebene Reihenfolge muss unbedingt eingehalten werden, sonst funktioniert es nicht!!

Es lassen sich auch auf der gleichen Seite unterschiedliche Linkformatierungen anwenden. Dazu bekommen die Links eine Klasse, die dann in die entsprechende Pseudoklasse übernommen wird:

```
<a href="neueseite.htm" class="anders">hier dr&uuml;cken</a>
```

```
a.anders:link {}
```

Klassen und IDs

Es können nicht nur allgemeine Tags wie <p> oder <table> formatiert werden, sondern auch ganz gezielt ein oder mehrere bestimmte Tags herausgegriffen werden. Soll z.B. in einem Fließtext, der aus vielen Absätzen besteht, lediglich ein bestimmter Absatz einen gelben Hintergrund bekommen, kann man ihm eine eigene **id** geben:

```
<p id="gelb">Text</p>
```

Im externen Stylesheet wird diese id dann so angegeben:

```
p#gelb { background-color:#FFFFCC;}
```

man muss noch nicht einmal das Tag angeben, auch das geht:

```
#gelb { background-color:#FFFFCC;}
```

Eine id ist allerdings absolute einmalig und darf kein zweites Mal vergeben werden. Wenn nun z.B. eine Tabellenzelle ebenfalls diese Hintergrundfarbe erhalten soll, weist man beiden eine **Klasse** zu. Klassen dürfen mehrfach vorkommen. So sieht das dann aus:

```
<p class="gelb">Text</p>

<table>
 <tr>
  <td class="gelb">Inhalt</td>
 </tr>
</table>
```

.gelb {} - Klasse
#text {} - id

Im externen Stylesheet wird die Klasse aber nur einmal vergeben und wirkt auf beide Tags gleichermaßen:

```
.gelb { background-color:#FFFFCC;}
```

Mit DIV layouten

Das ** Tag** ist ein Tag ohne Eigenschaften. Erst mit Formatierungen bekommt es einen Zweck. Ganz ohne Eigenschaften ist es aber dennoch nicht. Völlig unformatiert hat es die beiden Eigenschaften:

- es erstreckt sich über die ganze ihm zur Verfügung stehende Breite
- Elemente, die nach ihm oder vor ihm kommen, werden darunter oder darüber angezeigt.

Das sind die allgemeinen Eigenschaften eines **Block-Elements** – im Gegensatz zu einem **Inline-Element**, das die genau gegenteiligen Eigenschaften hat.

Beispiele für **Block-Elemente**: `<div>, <p>, <table>`

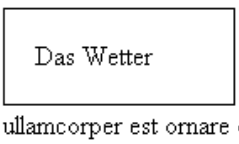
Beispiele für **Inline-Elemente**: `<a>, <img>, <b>`

Der Zweck eines besteht nun darin, als Container zu fungieren. Man kann in ein unterschiedliche andere Elemente zusammenfassen. So können z.B. Überschriften, Absätze, Bilder etc. zusammen in ein gruppiert und so beispielsweise als Spalte in einem mehrspaltigen Layout angelegt werden.

Floaten

Um Block-Elemente, die von ihrer Natur her immer untereinander liegen, dazu zu bekommen, nebeneinander zu liegen, muss man sie **floaten**. Um ein Element zu floaten, muss dieses Element wenigstens eine **Breite** bekommen, da es ansonsten – ebenfalls von seiner ursprünglichen Natur her – über die gesamte ihm zur Verfügung stehende Breite gehen würde. Eine Float-Angabe würde dabei z.B. so aussehen:

```
div {
    width: 300px;
    float: left;
    }
```



Integer elementum dapibus neque vel tempus. Duis tincidunt arcu in urna mattis feugiat. Nam vel nisi ullamcorper est ornare dictum sit amet at eros. Quisque elit leo, vestibulum vel feugiat id, placerat non dui. Cras arcu elit, sollicitudin ut pharetra eu, auctor vel mauris. Suspendisse potenti. Cras vel lacus mattis lectus lobortis sollicitudin tincidunt sed magna! Sed porta, velit quis commodo rutrum, mi ante dapibus dolor, ac venenatis orci metus quis nunc. In diam arcu, sodales a ultrices sit amet, consequat ac neque? Nulla placerat rutrum vulputate.

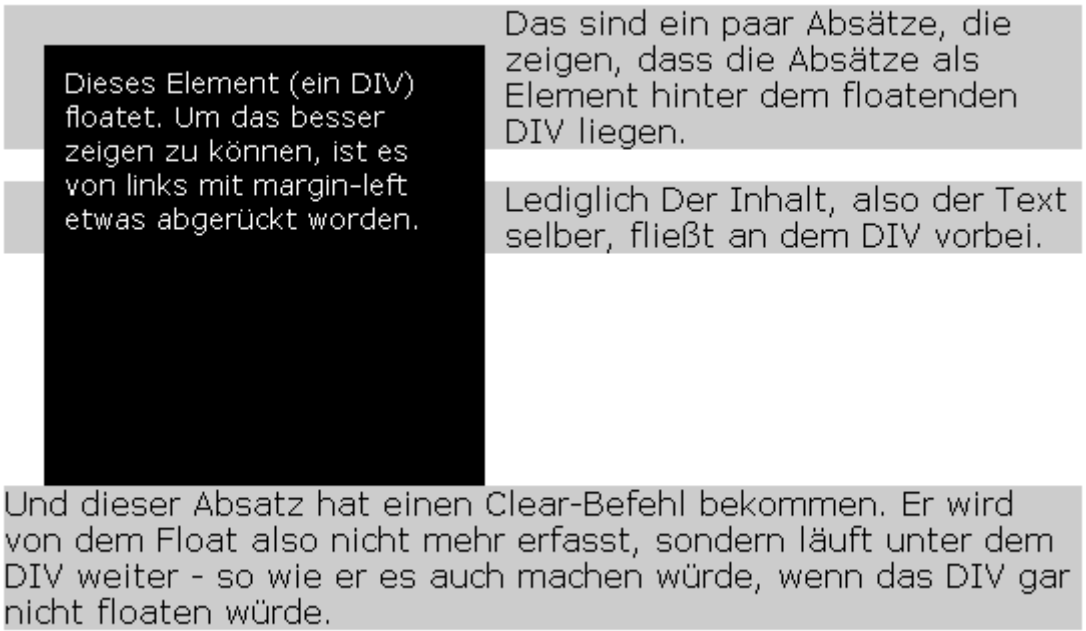
Im Ergebnis liegt das Element nun links und alle auf das Element folgende Element (im Beispiel oben ein Absatz) fließen daran vorbei.

Tatsächlich fließt aber nur der **Inhalt** der folgenden Elemente daran vorbei. Die folgenden Elemente selbst (also die Absätze - <p>) liegen nun **hinter** dem floatenden Element. Denn float heißt tatsächlich **schweben**. Das floatende Element – hier also das - **schwebt über den folgenden Elementen**.

Das muss einem klar sein, wenn man mit floats arbeitet. Außerdem muss man sich bewusst sein, dass **alle** auf ein floatendes Element folgenden Elemente **hinter dem floatenden Element** zu liegen kommen, wenn sie dazu noch Platz haben. Dieses Verhalten kann man mit dem **clear**-Befehl stoppen:

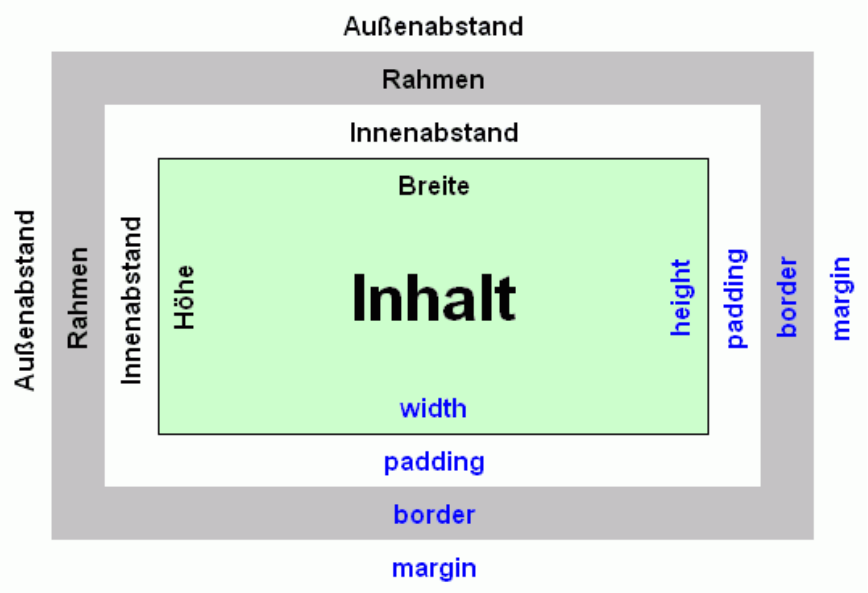
```
p {
   clear: both;
   }
```

Ab dem Element, das den clear-Befehl bekommt, fließt nun nichts mehr an dem floatenden Element vorbei.



Das Box-Model

Als Box-Model wird bezeichnet, wie Block-Elemente berechnet werden. Solange Block-Elemente nicht formatiert sind, erstrecken sie sich über die ganze ihnen zur Verfügung stehenden Breite. Gibt man ihnen jedoch eine feste Breite, greift das Box-Model. Ein Block-Element setzt sich dann aus der Summe aus Breite, Außenabstand, Rahmen und Innenabstand zusammen.



Beispiel:

```
div {
      width: 200px;
      margin: 30px;
      padding: 20px;
      border: 10px solid #cccccc;
      background-color: yellow;
      }
```

Auch wenn das (gelb) sichtbare DIV nur 200px breit zu sein scheint, nimmt es tatsächlich den Raum über insgesamt 320px ein:

200px + (2 x 30px) + (2 x 20px) + (2 x 10px) = **320px**

Wenn ein Block-Element also nur eine fest bestimmte Breite zur Verfügung hat (weil es sich z.B. in einer Spalte mit einer festen Breite befindet), muss man aufpassen, dass man es nicht z.B. mit einem Padding verbreitert und damit den Platz, in dem es sich befindet, sprengt.