Kapitel 9: Cascading Style Sheets

Literatur:

- Erik Wilde: World Wide Web — Technische Grundlagen. Springer, 1999, ISBN 3-540-64700-7, 641 Seiten.
- Eric Ladd, Jim O'Donnell, et al.: Using HTML 4, XML, and Java 1.2, Platinum Edition. QUE, 1999, ISBN 0-7897-1759-X, 1282 pages.
- W3C: Web Style Sheets Home Page. `http://www.w3.org/Style/`
- W3C: Cascading Style Sheets Home Page. `http://www.w3.org/Style/CSS/`
- Håkon Wium Lie, Bert Bos: Cascading Style Sheets, level 1. W3C Recommendation. `http://www.w3.org/TR/REC-CSS1`
- Bert Bos, Håkon Lie, Chris Lilley, Ian Jacobs: Cascading Style Sheets, level 2. W3C Recommendation. `http://www.w3.org/TR/REC-CSS2/`
- Håkon Wium Lie, Bert Bos: Cascading Style Sheets, Designing for the Web. Addison Wesley, 2nd Edition, 1999, ISBN 0-201-59625-3, 413 pages.
- Dave Raggett, Arnaud Le Hors, Ian Jacobs (Eds.): HTML 4.01 Specification. W3C, 24.12.1999. `http://www.w3.org/TR/html4/`
- Stefan Münz: HTML-Dateien selbst erstellen — SELFHTML. `http://www.selfhtml.org/`
- Wikipedia: `http://en.wikipedia.org/wiki/Cascading_Style_Sheets`

Lernziele

- Verständnis für den Nutzen von Style Sheets.
- Eigene Entwicklung von Style Sheets, zumindest in relativ einfachen Fällen.
- Übersicht über Literatur/Quellen für weitere Informationen.

Inhalt

1. Allgemeines
2. Einbindung in HTML
3. Auswahl von betroffenen Elementen
4. Eigenschaften der Darstellung
5. Konfliktlösung, Bedingungen

Motivation (1)

- Dokumente im WWW können auf unterschiedliche Arten verwendet werden:
  - ◇ Darstellung im Browser-Fenster.
  - ◇ Darstellung in kleinem Display (PDA).
  - ◇ Ausdruck auf Papier.
  - ◇ Ausgabe in Blindenschrift, Sprachausgabe.
  - ◇ Eintragung in Suchmaschinen.
  - ◇ Verarbeitung durch Programme (Shopbots, etc.)
- Jede Anwendung benötigt unterschiedliche Angaben zur Darstellung.

Motivation (2)

- HTML hatte anfangs vor allem inhalts-orientierte Elemente.
- Dann führten aber die Browser-Hersteller viele neue Elemente und Attribute ein, die das Aussehen der Seiten im Browser-Fenster beschreiben.

  Dies hatte auch den Zweck, Benutzer an einen Browser zu binden. Auf den Seiten stand oft „`Best viewed with ...`“. Es hatte auch etwas damit zu tun, dass das Web zuerst für Wissenschaftler entwickelt wurde, aber dann auch von Firmen verwendet wurde.
- Das Markup wurde also immer mehr darstellungs-orientiert.

Motivation (3)

- So wurde HTML immer umfangreicher, undurchschaubarer, und schwieriger für andere Medien zu verarbeiten.

  HTML war so auch einer schnellen Änderung unterworfen.
- Deswegen wurde bei HTML 4 eine Trennung eingeführt:
  - ◇ HTML selbst sollte nur den Inhalt beschreiben,
  - ◇ das Aussehen sollte mit Hilfe von Style Sheets festgelegt werden.

Motivation (4)

- Die Angaben zum Aussehen sind häufig willkürlich (Geschmackssache). Auch deswegen ist eine Trennung vom eigentlichen Inhalt wünschenswert.
- Für Anwendungen wie Suchmaschinen sind Angaben zum Aussehen irrelevant.
- Die Einführung besserer Gestaltungsmöglichkeiten war auch nötig, weil HTML-Elemente wie Tabellen und `BLOCKQUOTE` missbraucht wurden, und viel Text als Bild in die Seiten eingefügt wurde.

Motivation (5)

- Für andere Medien (Ausdrucke, Sprachausgaben) sind auch andere Festlegungen zum Aussehen notwendig, als für einen Web-Browser.
- Eventuell möchten verschiedene Benutzer den Text auf verschiedene Arten angezeigt bekommen.
- Es kann daher zu einer HTML-Datei verschiedene Style Sheets geben.
- Man kann auch umgekehrt in vielen HTML-Dateien das gleiche Style Sheet verwenden.

  Alle Seiten einer Webpräsenz (“Site”) sollten einheitlich aussehen.

Cascading Style Sheets (1)

- HTML ist nicht an eine bestimmte Sprache für Style Sheets gebunden.

  Man muss daher in HTML deklarieren, dass man "Cascading Style Sheets" (CSS) verwendet.

- CSS ist nicht nur für HTML geeignet, sondern auch für XML.

  Man kann sich fragen, warum das W3C zwei Style Sheet Ansätze standardisiert hat (CSS und XSL). XSLT hat aber eine ganz andere Aufgabe als CSS (Transformation, Umsortierung/Umstrukturierung von Elementen). Insbesondere kann man mit XSLT auch in HTML+CSS übersetzen. CSS und XSL FO basieren auf dem gleichen zugrundeliegenden Formatierungs-Modell.

Cascading Style Sheets (2)

- Das Wort "Cascading" drückt aus, dass man mehrere Style Sheets kombinieren / vermischen kann (d.h. Angaben verschiedener Prioritäten haben).

  Zum Beispiel könnte die Firma ein Style Sheet für ihre „Corporate Identity“ vorgeben, das eventuell für jede Abteilung leicht modifiziert werden kann, anschließend für jedes Dokument, und dann kann es noch Ausnahmen in den einzelnen Elementen im Text geben.

- Insbesondere kann auch der Leser der Dokumente (Benutzer des Browsers) Wünsche für die Darstellung äußern, nicht nur der Autor des Dokumentes.

  Dies führte bei der Vorstellung des Vorschlags auf der zweiten WWW-Konferenz zu Diskussionen über Rechte von Autoren und Lesern.

Cascading Style Sheets (3)

- Mit CSS1 kann man z.B. definieren:
  - ◇ Schriftart, Schriftgröße
  - ◇ Farben
  - ◇ Einrückungen, horizontaler/vertikaler Leerplatz
  - ◇ Begrenzungslinien, Ränder
  - ◇ Ausrichtung, z.B. Zentrierung
- Insgesamt bekommt man mehr Möglichkeiten, als HTML jemals an darstellungs-orientierten Elementen und Attributen hatte.

Cascading Style Sheets (4)

- Man kann mit Style Sheets auch das übliche Verhalten der Browser definieren („Default Style Sheet“).

  Im CSS1 Standard ist ein Default Style Sheet für HTML 2.0 angegeben. Im CSS2 Standard gibt es eins für HTML 4. Das Verhalten einiger weniger Elemente kann aber noch nicht in CSS 2 definiert werden (z.B. `img`, Applets, Frames, Formulare).

- Die Darstellung der HTML-Elemente ist dann nicht mehr fest in den Browser eingebaut, sondern über Daten definiert.

  Tatsächlich war schon der erste Browser von Tim Berners-Lee so aufgebaut, dass die Darstellung der Elemente über ein Style Sheet gesteuert werden konnte. Die Style Sheet Sprache wurde damals aber nicht veröffentlicht.

Geschichte (1)

- Ein erster Entwurf für "Cascading HTML Style Sheets" von Håkon Wium Lie erschien 1994.

  Der Vorschlag wurde auf der WWW Konferenz im November 1994 vorgestellt. Er wurde kombiniert mit Ideen von Bert Bos, der eine Style Sheet Sprache in seinem Browser "Argo" implementiert hatte.

- Cascading Style Sheets, level 1 (CSS1) wurde vom W3C am 17.12.1996 verabschiedet.

  Eine überarbeitete Version wurde am 11.01.1999 veröffentlicht.

- Cascading Style Sheets, level 2 (CSS2) wurde am 12.05.1998 verabschiedet.

Geschichte (2)

- CSS 2.1 ("CSS Level 2 Revision 1") ist ein W3C Standard seit dem 07.06.2011.

  [https://www.w3.org/TR/CSS2/]

  Es enthält die Features, die zum Zeitpunkt der Veröffentlichung allgemein implementiert sind. Damit werden auch einige Features aus CSS2 weggelassen (auf CSS3 verschoben). Es gibt aber auch einige neue Möglichkeiten.

  Tatsächlich war CSS 2.1 schon einmal eine Cadidate Recommendation (seit 25.02.2004), aber wurde am 13.06.2005 in den "Working Draft" Status zurückgestuft. "Candidate Recommendation" war es ab 19.07.2007. Auch die Version vom 08.09.2009 war noch "Candidate Recommendation".

  CSS Level 2 ist nur noch von historischem Interesse. Es wäre nach den neueren Regeln nicht mehr als Standard verabschiedet worden.

Geschichte (3)

- Der erste Browser, der einen Teil von CSS unterstützte, war Internet Explorer 3 (August 1996).

  Es fehlten große Teile vom "Box Model".

- Netscape Navigator 4 unterstützte im Prinzip einen großen Teil von CSS1, aber die CSS Implementierung enthielt noch viele Fehler.

  Netscape 3.x verstand noch keine Style Sheets.

- Auch Internet Explorer 4 unterstützte relativ viel von CSS1, aber mit Fehlern.

Geschichte (4)

- Opera 3.5 (Nov. 1998) unterstützte den größten Teil von CSS1.
- Internet Explorer 5.0 für den Mac (März 2000) war der erste Browser, der CSS1 fast vollständig unterstützte (mehr als 99%).
- Anfang 2010 gab es noch keinen Browser, der CSS2 vollständig unterstützte.

  [http://www.css4you.de/browsercomp.html/standardbrowser/]

- Die vielen Fehler und Inkompatibilitäten waren ein ernstes Hindernis in der Verbreitung von CSS.

Geschichte (5)

- Es wird schon länger an CSS Level 3 (CSS3) gearbeitet, aber dieser Standard ist in viele Module aufgeteilt, die unterschiedlich weit im Standardisierungsprozess sind.

  Z.B. ist “Selectors Level 3” (Teil von CSS 3) eine “W3C Recommendation” seit 29.09.2011. Übersicht über den Stand der verschiedenen Teile (ca. 50): [https://www.w3.org/Style/CSS/current-work]. Z.B. ist das “CSS Syntax Module Level 3” im Juli 2016 noch eine “Candidate Recommendation” (vom 20.02.2014).

- Die Idee ist, dass diese Module Teile der CSS 2.1 Spezifikation ersetzen.

  Aktueller Stand: [https://www.w3.org/TR/CSS/].

Inhalt

1. Allgemeines
2. Einbindung in HTML
3. Auswahl von betroffenen Elementen
4. Eigenschaften der Darstellung
5. Konfliktlösung, Bedingungen

Einbindung in HTML (1)

- Style Sheet Regeln können ein drei Stellen stehen:
  - ◇ In einer vom HTML-Dokument getrennten Datei, auf die mit `link` verwiesen wird.

    Ein Style Sheet kann auch mit dem HTTP-Header `Default-Style` (oder dem entsprechenden `meta`-Element) festgesetzt werden.
  - ◇ Innerhalb des Elementes `style` im Dokument-Kopf.
  - ◇ Jeweils in dem betroffenen Element im Attribut `style`.
- Die Vor- und Nachteile werden auf den folgenden Folien erläutert.

Einbindung in HTML (2)

Style Sheet in eigener Datei:

- Vorteile:
  - ◇ Das gleiche Style Sheet kann für mehrere Seiten verwendet werden.
  - ◇ Nur benötigte Stylesheets werden geladen.

    Wenn es mehrere alternative Style Sheets gibt.
- Nachteile:
  - ◇ Der Browser muss das Style Sheet extra anfordern und kann den Text erst danach darstellen.

    Wenn das gleiche Style Sheet in mehreren Seiten eingesetzt wird, betrifft dieses Problem nur die erste Seite (Caching im Browser).

Einbindung in HTML (3)

Style Sheet im Dokumentkopf:

- Vorteile:
  - ◇ Wird gleich mit dem Dokument mit geladen, die HTML Seite kann daher inkrementell aufgebaut werden (während die Daten noch kommen).
- Nachteile:
  - ◇ Keine Mehrfachverwendung von Stylesheets.

    Dadurch Redundanzen und Gefahr von Inkonsistenzen.
  - ◇ Verwirrt möglicherweise ganz alte Browser, die `style` nicht kennen.

Einbindung in HTML (4)

Angaben im `style`-Attribut bei betroffenen Elementen:

- Vorteile:
  - ◇ Einfach, direkt.

    Es steht direkt beim betroffenen Element, man braucht keinen Selektor anzugeben.
- Nachteil:
  - ◇ Man kann keine alternativen Angaben machen.

    Zum Beispiel für verschiedene Ausgabemedien.
  - ◇ Inkonsistenzen durch sehr lokale Angaben.

Einbindung in HTML (5)

Beispiel (Style Sheet in eigener Datei):

- Z.B.: `h1` Überschriften sollen in roter 14pt Schrift erscheinen.
- Man legt eine Datei, z.B. `my.css`, mit folgendem Inhalt an:

```
h1 { color: red; font-size: 14pt}
```

- Diese kann man nun mit einem `link`-Element referenzieren:

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


Einbindung in HTML (6)

Beispiel (Style Sheet im Kopf des Dokumentes):

- Hier steht das Style Sheet im Element `style`:

```
<style type="text/css">
    h1 { color: red; font-size: 14pt}
</style>
```

- Alte Browser, die `style` nicht kennen, nehmen an, dass hier bereits der `body` beginnt.
- Deswegen wird die Stylesheet-Information manchmal in einen HTML-Kommentar eingeschlossen:

```
<style type="text/css"><!--
    h1 { color: red; font-size: 14pt}
--></style>
```

Einbindung in HTML (7)

Style Sheet Angabe im Kopf, Forts.:

- Das Element `style` ist folgendermaßen deklariert:

```
<!ELEMENT style - -    CDATA>
<!ATTLIST style %i18n;
                type   %ContentType; #REQUIRED
                media  %MediaDesc;   #IMPLIED
                title  %Text;        #IMPLIED >
```

- Da der Inhalt als `CDATA` deklariert ist, wird Markup nicht ausgewertet (insbesondere keine Kommentare gelöscht). In XHTML dagegen `PCDATA`.

Kommentare in Stylesheets schreibt man wie in C: `/*...*/`.

Einbindung in HTML (8)

Beispiel (Style Sheet Angabe im Attribut):

- Alternativ kann man die Angabe auch direkt bei den betroffenen Überschriften machen:

```
<h1 style="color: red; font-size: 14pt">
```

- Das Attribut `style` ist Bestandteil von `%attrs;`. Es ist bei praktisch jedem Body-Element möglich.
- Der HTML Standard verlangt, dass die Style Sheet-Sprache immer deklariert ist. Oben geschieht dies mit dem Attribut `type`, hier mit einem Meta-Tag:

```
<meta http-equiv="Content-Style-Type"
      content="text/css" />
```


Alternative Style Sheets (1)

- Man kann auch für jedes Ausgabemedium ein anderes Stylesheet auswählen:

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

Gibt man nichts an, wird das Stylesheet für alle Medien verwendet.

- Das geht auch beim `style`-Element:

```
<style type="text/css" media="print">
    ...
</style>
```

- Bei der dritten Variante (`style`-Attribut) kann man keine alternativen Style Sheets angeben.

Alternative Style Sheets (2)

- Ausgabemedien:
  - ◇ `screen` (normaler Computer-Bildschirm),
  - ◇ `tty` (festes Raster von Zeichen),
  - ◇ `tv` (kleine Auflösung, kaum scrollen),
  - ◇ `projection` (Beamer),
  - ◇ `handheld` (kleiner Bildschirm, schwarzweiss, ...),
  - ◇ `print` (eingeteilt in Seiten),
  - ◇ `braille` (Schrift zum Tasten)
  - ◇ `aural` (CSS2)/`speech` (CSS2.1) (Sprachausgabe),
  - ◇ `all` (geeignet für alle Geräte).

Alternative Style Sheets (3)

- Smartphone-Browser sollen sich nicht als `handheld` verstehen, um nicht reduzierten Inhalt anzuzeigen.

  [https://wiki.selfhtml.org/wiki/CSS/Media_Queries].

- In "Media Queries Level 4" wird verlangt, dass Angaben für `tty`, `tv`, `projection`, `handheld`, `braille`, `embossed`, `aural` ignoriert werden.

  Das Dokument ist nur ein Entwurf. Media Queries wurden mit CSS3 eingeführt und im letzten Abschnitt des Kapitels genauer besprochen.

- Damit bleiben als sinnvoll verwendbaren Medienarten nur `screen`, `print`, `all`, `speech`.

Alternative Stylesheets (4)

- Es ist möglich, mehrere alternative Stylesheets im Dokument anzugeben, und dem Benutzer die Auswahl zu überlassen.

  Falls der Browser das unterstützt.

- Dazu hat man mehrere `link`-Elemente mit Verweisen auf Stylesheet-Dateien.
- Der vom Autor vorgeschlagene Default ("preferred stylesheet") verwendet `rel="stylesheet"`, alle anderen verwenden

```
rel="alternate stylesheet"
```


Alternative Stylesheets (5)

- In diesem Fall (mehrere alternative Style Sheets) muss das `title`-Attribut aller dieser `link`-Elemente gesetzt werden (der Browser kann dann diese Namen dem Benutzer zur Auswahl anbieten).

  Style Sheets mit gleichem `title` sind möglich (immer zusammen).

- Wird für ein `link`-Element kein `title` angegeben, und ist `rel="stylesheet"`, so ist es "persistent" und muss immer angewendet werden (ggf. zusätzlich zur Benutzerauswahl).

  Aber nur, wenn die `media`-Auswahl passt. Außerdem können Browser erlauben, Stylesheets ganz abzuschalten.

Alternative Style Sheets (6)

- Zumindest bei CSS ist es möglich, Angaben aus mehreren Stylesheets zu mischen, z.B.
  - ◇ mehrere passende `link`-Elemente, oder
  - ◇ `link` und zusätzlich `style` etc.
- Die Prioritätsregeln sind relativ kompliziert (s.u.), aber wenn es sonst keinen Unterschied gibt, gewinnt die spätere Angabe.

  Die Priorisierung von Angaben verschiedener Stylesheets ist die "Kaskade", die CSS den Namen gegeben hat. Sie wird im letzten Abschnitt der Folien genauer besprochen.

Inhalt

1. Allgemeines
2. Einbindung in HTML
3. Auswahl von betroffenen Elementen
4. Eigenschaften der Darstellung
5. Konfliktlösung, Bedingungen

Syntax (1)

- Abgesehen von `@import`-Anweisungen ist ein Style Sheet eine Menge von Regeln.
- Eine Regel besteht aus
  - einem "Selector" (zur Auswahl der betroffenen Elemente) und
  - einer "Declaration" (Festlegung von Eigenschaften der Darstellung) in geschweiften Klammern.

$$\underbrace{\texttt{h1}}_{\text{Selector}}\ \texttt{\{}\ \underbrace{\texttt{color: green}}_{\text{Declaration}}\ \texttt{\}}$$

Syntax (2)

- Eine Deklaration besteht wiederum aus zwei Teilen (getrennt durch einen Doppelpunkt):
  - einer Eigenschaft ("Property") und
  - einem Wert ("Value").

$$\underbrace{\texttt{h1}}_{\text{Selector}}\ \texttt{\{}\ \underbrace{\texttt{color}}_{\text{Property}}\ \texttt{:}\ \underbrace{\texttt{green}}_{\text{Value}}\ \texttt{\}}$$

- Ein Wert ist nicht notwendigerweise ein einzelnes Wort, z.B. ist auch folgendes möglich:

```
body { background: url(texture.gif) white }
```

Wenn das Bild nicht verfügbar ist, wird der Hintergrund weiß gemacht.

Syntax (3)

- Im "Declaration"-Teil können mehrere Eigenschaften gesetzt werden. Sie müssen durch Semikolon getrennt werden. Ein Semikolon am Ende ist erlaubt (optional):

```
h1 { font-weight: bold; color: green }
```

- Man kann Leerplatz, Zeilenumbrüche, Kommentare beliebig einfügen, z.B.

```
h1 { /* Dies ist ein Kommentar */
    font-weight: bold;
    color: green;
}
```

Syntax (4)

- Mehrere Regeln mit der gleichen Eigenschaftsliste können zusammengefasst werden, dazu werden die Selektoren durch Kommata getrennt:

```
h1, h2 { color: green }
```

ist äquivalent zu

```
h1 { color: green }
h2 { color: green }
```


Auswahl von Elementen (1)

- Oben waren die Selektoren zur Auswahl der betroffenen Elemente immer Element-Namen, z.B.

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

- Wenn das Stylesheet für HTML verwendet wird, ist die Groß-/Kleinschreibung dabei egal.
- Man kann eine bestimmte Schachtelung verlangen, z.B. `li`-Elemente innerhalb eines `ul`-Elementes:

```
ul li { color: red; font-size: 14pt}
```


Auswahl von Elementen (2)

- Man kann aber auch Elemente im HTML Quelltext mit dem Attribut `class` markieren:

```
<h1 class="wichtig">Kapitel 1</h1>
```

- Im Stylesheet sind die Klassen-Angaben durch einen vorangestellten Punkt gekennzeichnet:

```
.wichtig { color: red; font-size: 14pt}
```

- Typ und Klasse können auch kombiniert werden:

```
h1.wichtig { color: red; font-size: 14pt}
```


Auswahl von Elementen (3)

- Entsprechend kann man ein bestimmtes Element mit einer ID markieren:

```
<h1 id="kap1">Kapitel 1</h1>
```

- IDs werden im Stylesheet gekennzeichnet, indem das Symbol “`#`” vorangestellt wird:

```
#kap1 { color: red; font-size: 14pt}
```

- Bei Links gibt es auch “pseudo classes” zur Klassifizierung der Links: `visited` (schon besucht), `link` (noch nicht besucht), `hover` (Maus darüber)

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

Auswahl von Elementen (4)

- Viele Eigenschaften (z.B. Farben, Zeichensätze) vererben sich im Baum.

  Sie gelten dann auch für geschachtelte Elemente, sofern sie nicht überschrieben werden. Hintergrund-Farben und Bilder werden nicht vererbt, sind aber als transparent voreingestellt (sonst würden Bilder bei jedem kleinen Stück für ein Element neu anfangen).
- Setzungen für `body` gelten also für das ganze Dokument, wenn nicht explizit eine andere Auswahl getroffen wird.
- Man kann für jede Eigenschaft auch den speziellen Wert "`inherit`" angeben.

Inhalt

1. Allgemeines
2. Einbindung in HTML
3. Auswahl von betroffenen Elementen
4. Eigenschaften der Darstellung
5. Konfliktlösung, Bedingungen

Farben (1)

- Beispiel: Rote Schrift auf schwarzem Hintergrund:

```
body {
    color: red;
    background-color: black;
}
```

- In CSS kann man 16 Farben über Namen ansprechen: `black`, `silver`, `gray`, `white`, `maroon`, `red`, `purple`, `fuchsia`, `green`, `lime`, `olive`, `yellow`, `navy`, `blue`, `teal`, `aqua`.
- Alternativ kann man Farben auch als RGB-Werte spezifizieren (siehe nächste Folie).

Farben (2)

- Farben können als Mischung der Grundfarben Rot, Grün, Blau angegeben werden, und zwar
  - ◇ als Prozent-Werte: `rgb(100%, 0%, 12.75%)`
  - ◇ als Zahlen im Bereich 0..255: `rgb(255, 0, 20)`
  - ◇ hexadezimal mit zwei Stellen pro Farbe: `#FF0014`
  - ◇ hexadezimal mit einer Stelle pro Farbe: `#F01`

    Die Ziffern werden intern verdoppelt, also äquivalent zu `#FF0011`.
- In CSS2 wurden "system colors" eingeführt, z.B. wählt "`background: Menu; color: MenuText`" die im Betriebssystem für Menüs verwendeten Farben.

Hintergrund-Bilder (1)

- Man kann ein Hintergrund-Bild für das ganze Dokument oder auch für einzelne Elemente festlegen:

```
body {
    background-image: url(myimg.png);
    background-repeat: no-repeat;
    background-color: black;
    background-attachment: fixed;
    background-position: top center;
}
```

- `background-image` enthält die URL der Bilddatei.

Hintergrund-Bilder (2)

- `background-repeat` steuert, ob das Bild wie Fliesen den ganzen Hintergrund ausfüllt (durch entsprechende Wiederholung), oder nur einmal gezeigt wird.

  Mögliche Werte: `repeat` (Default), `repeat-x` (nur horizontal wiederholen), `repeat-y` (nur vertikal widerholen), und `no-repeat`.

- `background-color` ist wichtig, wenn
  - ◇ das Bild auch transparente Anteile hat oder nicht den ganzen Hintergrund ausfüllt,
  - ◇ während der Ladezeit der Bilddatei,
  - ◇ wenn die Bilddatei nicht gefunden wird.

Hintergrund-Bilder (3)

- `background-attachment` hat zwei mögliche Werte:
  - ◇ `scroll` (Default): beim Scrollen wird das Bild mit bewegt (Text ist direkt auf das Bild gedruckt),
  - ◇ `fixed`: das Bild steht fest, nur der Text wird darüber bewegt (Text auf transparenter Schicht).
- `background-position` spezifiziert, wie das Bild im Browser-Fenster positioniert ist.

  Man kann hier Prozentwerte, absolute Positionen, oder Schlüsselworte verwenden. Default: Die linke obere Ecke des Bildes wird in linke obere Ecke des Fensters positioniert (entspricht "`top left`" oder "`0% 0%`"). Allgemein: Punkt x% von Breite, y% von Höhe des Bildes wird auf x% von Breite, y% von Höhe des Browser-Fensters gelegt.

Hintergrund-Bilder (4)

- Es gibt auch eine Eigenschaft `background`, die eine Zusammenfassung aller Eigenschaften `background-*` ist. Man kann die einzelnen Werte in beliebiger Reihenfolge angeben, z.B.

```
background: url(myimg.png) black no-repeat
            top center fixed;
```

- Dies ist äquivalent zu den einzeln gesetzten Werten auf Folie 9-45.
- Man muss bei `background` nicht für alle Eigenschaften Werte angeben, die übrigen werden dann auf den jeweiligen Default-Wert gesetzt.

Hintergrund-Bilder (5)

- Die Default-Farbe für alle Elemente ist `transparent`, daher scheint das Hintergrundbild durch, wenn man nicht explizit eine andere `background-color` wählt.

  Ganz unten ist ein weißer Hintergrund, wenn man kein Bild und keine Farbe gewählt hat.

- Selbstverständlich kann man mit der Eigenschaft `background` auch nur die Hintergrund-Farbe setzen:

```
background: black;
```

  Damit werden auch die übrigen Hintergrund-Eigenschaften auf ihre Default-Werte gesetzt. Das kann bei der Kombination unterschiedlicher Stylesheets wichtig sein.

Zeichensatz Auswahl (1)

- Mit `font-family` kann man das generelle Aussehen der Zeichen wählen.
- Da die genaue Menge von Zeichensätzen, die der Browser unterstützt, nicht vorgeschrieben ist, kann man eine Prioritätsliste angeben:

```
body {font-family: Times, "Courier New", serif}
```

- Bei dieser Spezifikation wird
  - ◇ `Times` genommen, falls es zur Verfügung steht,
  - ◇ sonst `Courier New`, wenn es das gibt,
  - ◇ sonst ein beliebiger Zeichensatz der Klasse `serif`.

Zeichensatz Auswahl (2)

- Welche Fonts zur Verfügung stehen, ist abhängig von Browser, Betriebssystem und Installation.

  Beispiele für Zeichensatz-Namen: `Arial`, `Arial Black`, `Charcoal`, `Comic Sans MS`, `Courier`, `Courier New`, `Fixedsys`, `Georgia`, `Helvetica`, `Impact`, `MS Gothic`, `MS Sans Serif`, `MS Serif`, `Tahoma`, `Terminal`, `Times`, `Times New Roman`, `Trebuchet MS`, `Verdana`.

- Generische Zeichensatz-Namen: `serif`, `sans-serif`, `cursive`, `monospace`, `fantasy`.

  Um die generischen Zeichensatz-Namen darf man keine Anführungszeichen setzen, weil es Schlüsselworte in CSS sind. Da es die generischen Zeichensätze immer gibt, machen sie nur ganz am Ende der Prioritätenliste Sinn.

Zeichensatz Auswahl (3)

- `font-style`: Aufrechte oder schräge Schrift.

  Möglichen Werte sind: `normal`, `italic` und `oblique`. Für serifenlose Fonts sind `italic` und `oblique` meist gleich, bei Fonts mit Serifen ist `oblique` einfach die schräg gestellte Normalschrift, und `italic` eine neu entworfene Schrägschrift.

- `font-weight`: Normale Schrift oder Fettdruck.

  Mögliche Werte sind: `normal`, `bold`, `bolder`, `lighter`, `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, `900`. Dabei entspricht `normal` dem Wert `400`, und `bold` dem Wert `700`. Die Werte `bolder` and `lighter` sind relativ zum Wert im Elternelement. Natürlich muss nicht jeder Font in jeder Abstufung zur Verfügung stehen.

- `font-variant`: Für Schrift nur aus Großbuchstaben.

  Mögliche Werte: `normal`, `small-caps`. Nicht in alten Browsern.

Zeichensatz Auswahl (4)

- `font-size`: Schriftgröße. Auswahl über
  - ◇ Schlüsselworte, z.B. `large`.

    `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`.
  - ◇ Absolute Längen: `cm`, `mm`, `in`, `pt`, `pc`.
  - ◇ Pixel, z.B. `8px`.

    Die Einheit soll so wie auf einem typischen Bildschirm wirken. Beim Ausdruck auf einem hochauflösenden Laserdrucker kann `1px` mehr als ein Punkt sein.
  - ◇ Werte relativ zur Schriftgröße im Elternelement, z.B. `120%`, `1.2em`.

    `em` ist hier die Fontgröße des Elternelements.

Zeichensatz Auswahl (5)

- Man kann die Festlegungen für den Zeichensatz in der Eigenschaft `font` zusammenfassen, z.B.

```
body {font: italic large Helvetica, sans-serif}
```

- Die Reihenfolge ist:
  - ◇ Zuerst Festlegungen für `font-style`, `font-weight`, `font-variant` (beliebige Reihenfolge, optional),
  - ◇ dann die `font-size` (muss angegeben werden),

    optional danach ein “/” und die `line-height`,
  - ◇ anschließend die `font-family` (notwendig).

Zeichensatz Auswahl (6)

- Alternativ kann man Zeichensätze wählen, die das Betriebssystem verwendet, z.B.

```
body {font: message-box}
```

- Schlüsselworte für diese Art der Zeichensatzwahl: `caption` (z.B. auf Knöpfen), `icon`, `menu`, `message-box` (Text in Dialogboxen), `small-caption`, `status-bar`.

  Dies setzt alle Zeichensatz-Eigenschaften, aber man kann die Setzungen modifizieren, indem man zusätzlich bestimmte Eigenschaften einzeln angibt (z.B. `font-size`).

Zeichensatz Auswahl (7)

- Weitere Zeichensatz-Eigenschaften:
  - ◇ `font-stretch`

    `normal`, `wider`, `narrower`, `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `semi-expanded`, `expanded`, `extra-expanded`, `ultra-expanded`.
  - ◇ `text-decoration`

    `none`, `underline`, `overline`, `line-through`, `blink`.
  - ◇ `font-size-adjust`

    Für Zeichensatz-Ersetzung bei Zeichensätzen mit ungewöhnlichen Verhältnis der Größe von Klein- und Großbuchstaben.
  - ◇ `text-transform`

    `capitalize`, `uppercase`, `lowercase`, `none`.

Abstände im Text (1)

- `word-spacing`: Abstände zwischen Wörtern können vergrößert bzw. verkleinert werden.

  Mögliche Werte sind das Schlüsselwort `normal` oder Längenangaben, entweder absolut (z.B. `1mm`) oder relativ zur Fontgröße (z.B. `0.5em`). Die Längen werden auf den normalen Wortzwischenraum addiert. Sie können auch negativ sein, aber für negative Werte kann es implementierungsabhängige Grenzen geben.

- `letter-spacing`: Abstände zwischen den Buchstaben eines Wortes.

  Mögliche Werte wie bei `word-spacing`.

Abstände im Text (2)

- `line-height`: (Minimaler) Abstand der Zeilen.

  Mögliche Werte sind absolute und relative Längen, Prozentangaben (beziehen sich auf Fontgröße, `120%` ist äuquivalent zu `1.2em`), oder Zahlen (`1.2` bedeutet für dieses Element auch `1.2em`, aber der Zeilenabstand wird für Kindelemente mit veränderter Fontgröße neu berechnet, während bei `1.2em` der berechnete absolute Wert auch für die Kindelemente gilt, wenn dort nicht explizit ein neuer Wert gesetzt wird).

- `text-indent`: Einrückung der ersten Textzeile eines Block-Elementes (z.B. Paragraph).

  Mögliche Werte sind absolute und relative Längen, sowie Prozentangaben (beziehen sich auf die Breite des Paragraphen). Negative Werte sind möglich, aber nicht alle Browser können sie anzeigen.

Ausrichtung von Text

- `vertical-align`: Vertikale Ausrichtung von Inline-Elementen (z.B. für Exponenten und Indices).

  Die folgenden Werte beziehen sich auf das Eltern-Element: `baseline`, `middle`, `sub` (Index), `super` (Exponent), `text-top`, `text-bottom` (beziehen sich auf Font des Eltern-Elements). Zusätzlich gibt es die Werte `top` und `bottom`, die sich auf das tiefste bzw. höchste Element der Zeile beziehen. Prozentangaben sind auch möglich und beziehen sich auf die Eigenschaft `line-hight` des Elements.

- `text-align`: Horizontale Ausrichtung des Textes in Block-Elementen.

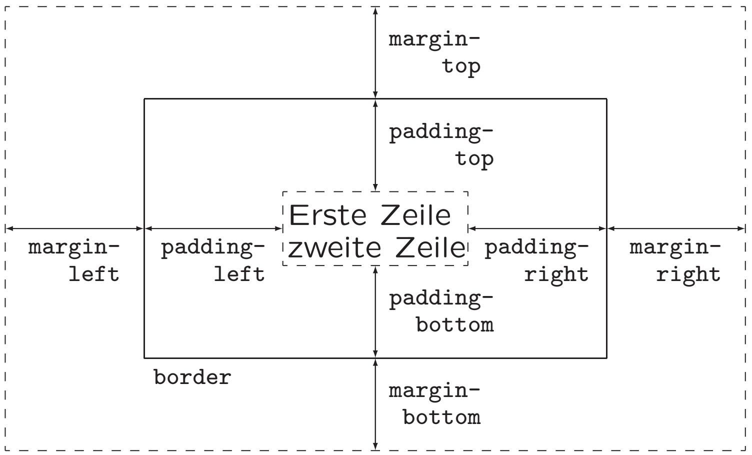
  Mögliche Werte sind `left`, `right`, `center`, `justify`. Der Wert `justify` wird eventuell nicht unterstützt. Die Ausrichtung bezieht sich auf die Breite des Elements, nicht auf das Browserfenster.

Abstände, Rand (1)

- Alle Elemente werden in einem (oder bei Zeilenumbrüchen ggf. in mehreren) rechteckigen Kästen gedruckt.
- Man kann den um die “bounding box” in einem gewissen Abstand einen Rand (“border”) zeichnen lassen, und darum nochmal Abstand lassen.
- Wenn man z.B. um Paragraphen nach oben und unten Platz lassen will, geht das so:

```
p { margin-top: 0.5em; margin-bottom: 0.5em; }
```

Abstände, Rand (2)



Abstände, Rand (3)

- `margin-top`, `margin-right`, `margin-bottom`, `margin-left` definieren die Abstände vom Rand (äußere Kante) zu benachbarten Kästen.
- `padding-top` u.s.w. definieren die Abstände vom eigentlichen Inhalt bis zum Rand (innere Kante).
- Auch wenn kein Rand gezeichnet wird, ist ein Unterschied, dass der "Margin"-Bereich transparent ist, während im "Padding"-Bereich die Hintergrundfarbe (Eigenschaft "`background`") benutzt wird.

  Transparent: Hintergrundfarbe übergeordneter Elemente ist zu sehen.

Abstände, Rand (4)

- Die Werte der Eigenschaften können in `px` (Pixeln), `cm`, `mm`, `in`, `pt`, `pc` `em`, `ex` und `%` angegeben werden.

  Dabei ist `1.0em` die Höhe des Zeichensatzes des Elementes, und `1.0ex` (normalerweise) die Höhe des Buchstabens "`x`". Prozentangaben beziehen sich auf die Breite des umgebenen Blocks (auch vertikale Prozentangaben beziehen sich auf die Breite).
- Es gibt auch Abkürzungen `margin` und `padding`, mit denen man die Abstände an den vier Seiten auf einmal setzen kann, z.B. `margin: 1em 0em 1em 2em`.

  Es können 1–4 Werte angegeben werden. Reihenfolge bei vier Werten: top, right, bottom, left. Wird nur ein Wert angegeben, werden alle vier Abstände auf diesen Wert gesetzt. Bei zwei oder drei Angaben werden fehlende Werte von den gegenüberliegen Seiten ergänzt.

Abstände, Rand (5)

- Benachbarte vertikale "Margin"-Werte werden nicht addiert, sondern es wird das Maximum genommen.

  Z.B. ist der Abstand zwischen zwei `li`-Elementen das Maximum des `margin-bottom` vom oberen Element und `margin-top` vom unteren Element. Dieses Verschmelzen ("collapse") von "Margin"-Rändern gilt auch für Abstände in der gleichen Richtung, wenn dazwischen kein Rahmen und nur leeres "Padding" ist. Unter diesen Bedingungen würde z.B. vom `margin-top` der `ul` und dem ersten `li` nur das Maximum genommen. Wenn man das `padding-top` der `ul` aber auch nur auf `1px` setzt, sind die beiden Ränder nicht mehr unmittelbar benachbart, und sie addieren sich. Die genauen Regeln sind relativ kompliziert, z.B. gilt das Verschmelzen nur für "block boxes in normal flow".
- Negative "Margin"-Werte sind möglich und werden dann abgezogen.

Abstände, Rand (6)

- `margin-top` und `margin-bottom` wirken sich für normale inline-Elemente nicht aus (nur Block-Elem.).
- Für horizontale Abstände gilt die Maximum-Regel nicht, hier wird immer einfach addiert.

  Dafür gibt es hier den interessanten Wert "`auto`". Für "block-level elements in normal flow" gilt, dass die Summe der Margin-Abstände (links/rechts), der Randdicken, der Padding-Abstände und der Inhaltsbreite (`width`) gleich der `width` des umgebenen Blocks ist. Oft hat `width` den Wert `auto`, dann kann dieser Wert aus der Breite der umgebenen Box berechnet werden. Wenn aber `width` einen festen Wert hat, kann man mit `auto`-Werten für `margin-left` und/oder `margin-right` linksbündige, rechtsbündige oder zentrierte Darstellung wählen (nicht IE bis 5.x). Falls kein Wert `auto` ist, wird der vorgegebene Wert von `margin-right` ignoriert und ein Wert gewählt, der die Gleichung erfüllt.

Abstände, Rand (7)

- Ob ein Rand dargestellt wird, entscheidet sich an den `border-style` Eigenschaften: `border-top-style`, `border-right-style`, u.s.w.
- Mögliche Werte sind: `none`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`.
- Es gibt auch eine Eigenschaft `border-style`, bei der man 1-4 Werte angibt.

  Bei einem Wert gilt dieser für alle Seiten. Bei zwei Werten: Erster für `top` und `bottom`, zweiter für `left` und `right`. Bei drei Werten: Erster für `top`, zweiter für `left` und `right`, dritter für `bottom`. Alle vier Werte werden in der Reihenfolge `top`, `right`, `bottom`, `left` angegeben. Alte Browser unterstützen `border-style` eher als einzelne Eigenschaften.

Abstände, Rand (8)

- `border-width` legt die Breite des Randes fest.

  Mögliche Werte sind `thin`, `medium`, `thick` und explizite Längen, wie z.B. `4px`. Wie oben für `border-style` erklärt, kann man bei `border-width` zwischen einem und vier Werten angeben, falls der Rand auf den verschiedenen Seiten unterschiedlich dick sein soll. Es gibt auch Eigenschaften für die einzelnen Seiten: `border-top-width` u.s.w. Die Breite des Randes wird zu Margin und Padding hinzu addiert.

- `border-color` bestimmt die Farbe des Randes.

  Wieder können 1–4 Werte angegeben werden, mit der gleichen Bedeutung wie oben. Es gibt auch den speziellen Wert `transparent`. Die Eigenschaften für die einzelnen Seiten heißen `border-top-color` u.s.w. (es gibt sie aber erst ab CSS2). Wenn keine Farbe festgelegt wird, wird die `color`-Eigenschaft des Elementes benutzt.

Abstände, Rand (9)

- Es gibt noch weitere Abkürzungen (Zusammenfassungen von Eigenschaften), z.B.

  ```
  border: thick solid red
  ```

  Man setzt also Breite, Stil und Farbe in einer Anweisung (die Reihenfolge ist egal). Man kann Teile weglassen, dann wird der "initial value" der Eigenschaft verwendet, das ist `medium` für die Breite, `none` für den Typ, und die `color`-Eigenschaft des Elementes für die Farbe des Randes. Es gibt entsprechend auch `border-top`, u.s.w.

- `width` schreibt die Breite des Inhalts vor.

  Z.B. wichtig für Bilder, auch für andere Bereiche, die von Text umflossen werden.

- `height`: Höhe des Inhalts.

Von Text umflossene Bereiche

- `float`: Mit dieser Eigenschaft kann man ein Element (z.B. ein Bild) an den linken oder rechten Rand seines Eltern-Elements (z.B. Paragraph) bewegen, und den weiteren Text darum fließen lassen.

  Mögliche Werte: `left`, `right`, `none`. Gibt es am entsprechenden Rand schon ein `float`-Element, kommt das neue Element daneben (darunter nur, falls der Platz nicht reicht). Das Umfließen gilt nicht nur für das Eltern-Element, sondern auch für Text aus folgenden Elementen. Die Breite des `float`-Elementes sollte bekannt sein (z.B. `width` setzen).

- `clear`: Hiermit kann man das Umfließen eines Elementes am linken oder rechten Rand beenden.

  Das aktuelle Element kommt dann darunter. Mögliche Werte: `none` (default), `left`, `right`, `both`.

Klassifizierungs Eigenschaften

- `display`: Bestimmt die Art der Formatierung, z.B. `block`, `inline`, `list-item`, `none`.

  Bei dem Wert `none` wird der Text nicht dargestellt (wenn man z.B. Navigationselemente in der Druckversion ausblenden will). In CSS2 gibt es außer den obigen vier Werten noch viele weitere. Es ist interessant, dass mit dieser Eigenschaft das Browserverhalten für einen Elementtyp definiert werden kann: So muss nichts "fest verdrahtet" im Browser sein, sondern es wird alles über Daten gesteuert (über ein im Browser enthaltenes Default-Stylesheet).

- `white-space`: Sollen Zeilenumbrüche oder Leerplatz in der Eingabe berücksichtigt werden?

  Mögliche Werte: `normal` (Leerplatz ist nur Worttrenner), `pre` (Leerplatz und Zeilenumbrüche sind wichtig), `nowrap` (keine automatischen Zeilenumbrüche, nur bei
).

Listen-Eigenschaften

- `list-style-type`: Markierung von Listenelementen.

  Mögliche Werte: `disc` (default), `circle`, `square`, `decimal`, `lower-roman`, `upper-roman`, `lower-alpha`, `upper-alpha`, `none`. Diese Eigenschaft wird üblicherweise im Element `UL` oder `OL` gesetzt. Sie wirkt sich aber für alle geschachtelten Element mit `display: list-item` aus.

- `list-style-image`: Festlegung eines kleinen Bildes als Listenmarkierung.

  Wert ist eine URL, z.B. `ul { list-style-image: url("bullet.gif") }`. Man kann `list-style-type` gleichzeitig setzen, für den Fall, dass die Bilddatei nicht verfügbar ist.

- `list-style-position`: `inside` oder `outside`.
- `list-style`: Zusammenfassung der Eigenschaften.

Hinweise zur Fehlersuche (1)

- Ein Browser muss Eigenschaften ignorieren, die er nicht kennt, oder deren Wert er nicht versteht.
- Wenn man Fehler macht, ist ein häufiger Effekt, dass die Setzungen nichts zu bewirken scheinen.
- Es empfiehlt sich, einen CSS Validator zu benutzen:

  [https://jigsaw.w3.org/css-validator/]

- Um zu testen, ob die Eigenschafts-Setzungen überhaupt angewendet werden, kann man z.B. auch eine ungewöhnliche Farbe wählen.

Hinweise zur Fehlersuche (2)

- Nützlich sind auch die Entwicklungswerkzeuge des Browsers, z.B. bei Firefox:
  - Die Browser-Konsole.

    Hier werden fehlerhafte und daher ignorierte Angaben aus dem Stylesheet angezeigt (wenn der CSS-Knopf/Tab oben farbig hinterlegt ist).
  - Der "Inspector" (ein Teil des Debuggers).

    Man kann links im Quellcode ein Element auswählen, und bekommt dann die verwendeten CSS-Regeln angezeigt (unter "Rules") und kann auch die Größe des Kastens und seiner Abstände sehen.

Inhalt

1. Allgemeines
2. Einbindung in HTML
3. Auswahl von betroffenen Elementen
4. Eigenschaften der Darstellung
5. Konfliktlösung, Bedingungen

Import Regeln

- Am Anfang eines Stylesheets (vor anderen Setzungen außer `@charset`) kann man Stylesheet-Daten aus weiteren Dateien laden:

```
@import "basic.css";
```

  Wenn man es nach normaler Stylesheet-Information macht, wird die `@import`-Anweisung ignoriert.

- Man kann auch verlangen, dass die Datei nur für einen bestimmten Medientyp geladen wird:

```
@import "print.css" print;
```


Prioritäten von Regeln (1)

- Die Setzungen aus den Stylesheets werden sortiert. Die letzte Setzung gewinnt.
- Sortierkriterium höchster Priorität ist die Quelle:
  - Default-Setzungen des Browsers
  - Normale Setzungen des Nutzers (Lesers)
  - Normale Setzungen des Autors
  - Mit "`!important`" markierte Regeln des Autors
  - Mit "`!important`" markierte Regeln des Nutzers

    Solche Regeln haben das höchste Gewicht.

Prioritäten von Regeln (2)

- Die folgenden Sortierkriterien bevorzugen spezifischere Selektoren.
- Zweites Kriterium: Setzung aus `style`-Attribut des Elementes nach Setzung aus Stylesheet.

  Das Attribut wird also bevorzugt. Dies sind dann auch immer Festsetzungen des Autors. Eine mit `!important` markierte Setzung des Autors aus einem externe Stylesheet hätte Vorrang, und natürlich auch eine `!important` Wahl des Lesers. In diesem Fällen kommt man nicht mehr zum Sortierkriterium zweiter Priorität, weil das erste die Reihenfolge bereits entscheidet. Ansonsten setzt sich das `style`-Attribut durch.

- Drittes Kriterium: Anzahl der ID-Attribute im Selektor (mehr als eins macht kaum Sinn).

Prioritäten von Regeln (3)

- Viertes Kriterium: Anzahl Bedingungen für Attribute (z.B. `class`) sowie Pseudo-Klassen wie `:hover`.

  Das `id`-Attribut wird nicht mitgezählt (schon im dritten Kriterium).

- Fünftes Kriterium: Anzahl der Element-Namen und Pseudo-Elemente wie `:first-letter`.
- Sechstes Kriterium: Gegebene Reihenfolge in der Spezifikation.

  Die spätere Angabe setzt sich durch, wenn die übrigen Sortierkriterien nicht schon eine Entscheidung geliefert haben (nach jedem Kriterium wird aufsteigend sortiert und am Ende die letzte Angabe genommen).

Prioritäten von Regeln (4)

- Nur wenn es gar keine anwendbare Regel gibt, wird der Wert vom Elternelement geerbt (falls die Eigenschaft vererbbar ist).

  Die obige "Kaskade" hat also Vorrang vor dem Vererben. Bei Bedarf kann man eine explizite Regel mit dem Wert "`inherit`" angeben.

- Das im Browser eingebaute Stylesheet enthält zumeist relative Setzungen, z.B. `h1 {font-size: 2em}`.

  Vorschlag aus [https://www.w3.org/TR/CSS2/sample.html].

- Obwohl hier keine Vererbung stattfindet (es ist ja ein Wert gesetzt), wirkt sich eine Setzung weiter oben im Baum doch aus.

Ausgabemedien (1)

- Es ist ein Ziel von CSS, dass der gleiche Inhalt (die gleiche HTML-Datei) auf unterschiedlichen Ausgabegeräten jeweils passend dargestellt werden kann.
- Z.B. kann zum Drucken ein anderes Stylesheet verwendet werden wie für die Darstellung im Browser.

  Z.B. `media="print"` im `link`-Element, das auf das Stylesheet verweist.

- Es können aber auch in einer `css`-Datei Teile auf bestimmte Ausgabemedien eingeschränkt werden:

```
@media print {
    .navigation { display: none; }
}
```

Ausgabemedien (2)

- Während es in CSS2 nur grobe Typen von Ausgabemedien wie `screen` und `print` gab, wurden in CSS3 "Media Queries" eingeführt, mit denen man die Fähigkeiten des Ausgabegrätes wesentlich genauer abfragen kann.
- "Media Queries" (ursprünglich "Media Queries Level 3") ist seit dem 19.06.2012 ein W3C Standard.

  [https://www.w3.org/TR/css3-mediaqueries/]
- Es ist eine Kerntechnologie für das "Responsive Webdesign" (Anpassung der Darstellung an Geräte).

Ausgabemedien (3)

- Wo bisher Medientypen wie "`screen`" oder "`print`" stehen konnten, können jetzt verschiedene Eigenschaften abgefragt werden:

```
@media (min-width: 50em) {
    p { left-margin: 5em; }
}
```

  Die Einheit `em` bezieht sich auf die Standard-Schriftgröße des Browsers. Es sind auch Angaben in Pixeln (`px`) möglich.
- Media Queries sind auch im `media`-Attribut der Elemente `link` und `style` möglich, sowie bei `@import`-Anweisungen im Stylesheet.

Ausgabemedien (4)

- Es sind auch Konjunktionen mehrerer Bedingungen möglich:

```
@media screen and (min-width: 50em)
              and (max-width: 100em) {
    ...
}
```

  Ein klassischer Medientype wie `screen` darf nur am Anfang stehen (eventuell nach `not` oder `only`, s.u.).
- Ebenso können mehrere alternative Bedingungen durch "`,`" verknüpft werden (Disjunktion).

  Wie in Logik und Programmiersprachen üblich, bindet die Disjunktion (hier das Komma) schwächer als die Konjunktion (Schlüsselwort `and`).

Ausgabemedien (5)

- Wenn man der Bedingung (Konjunktion) ein `not` voranstellt, wird sie negiert.

  `not` bindet also schwächer als `and`, aber stärker als "`,`".
- Alternativ kann man auch "`only`" voranstellen. Das bedeutet eigentlich nichts, bewirkt aber, dass ein alter Browser die Bedingung ignoriert.

  Er kennt nicht die Medientypen "`not`" und "`only`" und wertet die Bedingung deswegen zu `false` aus. Die Grammatik verlangt, dass nach `not` oder `only` zuerst ein klassischer Medientyp angegeben wird, weitere Abfragen von Geräteeigenschaften erst danach (mit `and` verknüpft). Schon in der HTML 4 Spezifikation wurden Abfragen von Geräteeigenschaften vorausgesehen und festgelegt, dass alle Medientypen, die keine Buchstabenfolge sind, ignoriert werden (deswegen die "`(...)`").

Ausgabemedien (6)

Eigenschaften:

- `width`, `min-width`, `max-width`: Breite des Viewports (Browser-Fenster bzw. Seite).

  Viele Eigenschaften erlauben den Präfix `min-` und `max-`, damit man nicht die Zeichen < und > verwenden muss, die in HTML schon eine andere wichtige Funktion haben. Die Bedingungen sind $\geq$ und $\leq$, also inklusive der Grenzen. Der Viewport enthält auch einen eventuellen Scrollbalken.

- `height`, `min-height`, `max-height`: Viewport-Höhe.
- `device-width`, `min-device-width`, `max-device-width`: Breite des Bildschirms.

Ausgabemedien (7)

Eigenschaften, Forts.:

- `device-height`, `min-device-height`, `max-...`: Höhe des Bildschirms.
- `orientation`: "`portrait`" oder "`landscape`".
- `aspect-ratio`, `min-aspect-ratio`, `max-aspect-ratio`: Verhältnis von `width` zu `height`.

  Der Wert muss als zwei Zahlwerte mit Schrägstrich angegeben werden, z.B. `(min-aspect-ratio: 16/9)`.

- `device-aspect-ratio`, `min-...`, `max-...`: Verhältnis von `device-width` zu `device-height`.

Ausgabemedien (8)

Eigenschaften, Forts.:

- `color`, `min-color`, `max-color`: Anzahl Bits pro Farbkomponente (z.B. R,G,B). 0 falls monochrom.

  Wenn man in die Klammern nur die Eigenschaft ("media feature") ohne Wert schreibt, testet das, dass der Wert verschieden von Null ist (geht nicht mit Präfix `min-` und `max-`). Daher kann man mit `(color)` testen, ob das Gerät Farbe darstellen kann.

- `color-index`, `min-...`, `max-...`: Anzahl Einträge in einer Farbtabelle.

  0, falls keine "Color Lookup Table" zur Verringerung der Anzahl Bits pro Pixel eingesetzt wird.

Ausgabemedien (9)

Eigenschaften, Forts.:

- `monochrome`, `min-...`, `max-...`: Anzahl Bits pro Pixel für Graustufen.

  0, falls kein monochromes Ausgabe-Gerät.

- `resolution`, `min-...`, `max-...`: Punkte pro Längeneinheit, z.B. `(min-resolution: 300dpi)`.
- `scan`: `progressive` oder `interlace` (für Fernseher)
- `grid`: 0 oder 1 (Terminal mit Font fester Breite)