

# <web-grundlagen />

Gerrit Burkert

Version 1.0.1 • 18.12.2021

# Web-Grundlagen

Gerrit Burkert

Version vom 18.12.2021

# Inhaltsverzeichnis

<b>Web-Grundlagen</b>	<b>6</b>
Einführung und Ziele . . . . .	6
Worum geht es in WBE? . . . . .	7
Webdesign und Webentwicklung . . . . .	7
Ziele und Inhalte von WBE . . . . .	7
Ziele und Inhalte des Vorkurses . . . . .	8
Hinweis zu diesem Skript . . . . .	9
Abgrenzung . . . . .	9
<b>Grundlagen</b>	<b>11</b>
Einführung und Ziele . . . . .	11
Ziele . . . . .	11
Relevanz für WBE . . . . .	11
Webseiten: HTML und CSS . . . . .	12
Terminologie . . . . .	16
HTML-Grundlagen . . . . .	17
Aufgaben . . . . .	20
Quellen und Verweise . . . . .	21
Quellenangaben . . . . .	21
<b>Das Web als Plattform</b>	<b>22</b>
Einführung und Ziele . . . . .	22
Ziele . . . . .	22
Relevanz für WBE . . . . .	22
Client und Server im Web . . . . .	23
Internet . . . . .	23
World Wide Web . . . . .	25
Grundelemente des Web . . . . .	26
Client: Web-Browser / User Agent . . . . .	26
Server: Web-Server . . . . .	29
Rolle von HTML, CSS und JavaScript . . . . .	30
Adressierung: URI/URL . . . . .	33
URL . . . . .	33
IP-Adresse . . . . .	34

Domain-Name . . . . .	34
Protokoll: HTTP . . . . .	35
Web-Entwicklung . . . . .	36
Web-Editoren . . . . .	37
Zeichencodierung . . . . .	42
Entwickler-Tools im Browser . . . . .	45
Verzeichnisstruktur . . . . .	46
Website bereitstellen . . . . .	48
Dateitransfer, Secure Shell . . . . .	48
Content Management Systeme . . . . .	50
Quellen und Verweise . . . . .	51
Weiterführendes Material . . . . .	51
Quellenangaben . . . . .	51
<b>Markup und HTML (Teil 1)</b>	<b>52</b>
Einführung und Ziele . . . . .	52
Ziele . . . . .	52
Relevanz für WBE . . . . .	52
Markup-Varianten . . . . .	53
HTML-Entwicklung . . . . .	57
HTML 4.01 . . . . .	59
XML und XHTML . . . . .	62
HTML - Living Standard . . . . .	64
Aufbau eines HTML-Dokuments . . . . .	65
Kopf eines HTML-Dokuments . . . . .	65
Grobstruktur des Dokuments . . . . .	68
Überschriften . . . . .	71
Semantic Markup . . . . .	72
Absätze und Listen . . . . .	72
Zitate und Tabellen . . . . .	74
Kontrolle und Fehlersuche . . . . .	76
Quellen und Verweise . . . . .	79
Material zur Vertiefung . . . . .	79
Spezifikationen und Standards: . . . . .	79
Quellenangaben . . . . .	79
<b>Markup und HTML (Teil 2)</b>	<b>80</b>
Einführung und Ziele . . . . .	80
Ziele . . . . .	80
Relevanz für WBE . . . . .	80
Verweise und Navigation . . . . .	81
Verweise . . . . .	81
Navigationsbereich . . . . .	82
Bilder und Videos . . . . .	83
Bilder einfügen . . . . .	83
Pixelbasierte Bild- und Grafikformate . . . . .	85
Vektorgrafiken . . . . .	87

Audio und Video . . . . .	88
Formulare . . . . .	88
Quellen und Verweise . . . . .	91
Material zur Vertiefung . . . . .	91
Referenzen . . . . .	91
Quellenangaben . . . . .	91
Weitere Tipps . . . . .	91
<b>Markup und HTML (Teil 3)</b>	<b>92</b>
Einführung und Ziele . . . . .	92
Ziele . . . . .	92
Relevanz für WBE . . . . .	92
Design von HTML5 . . . . .	93
Überschriften und Abschnitte in HTML5 . . . . .	94
Mehr zu HTML5 . . . . .	96
Neue Inline-Elemente . . . . .	96
Microformats . . . . .	97
Attribut contenteditable . . . . .	98
Selbst definierte data-Attribute . . . . .	99
HTML5 APIs . . . . .	99
Was funktioniert bereits? . . . . .	99
Vektorgrafiken und Formeln . . . . .	101
SVG . . . . .	101
MathML . . . . .	103
Quellen und Verweise . . . . .	106
Material zur Vertiefung . . . . .	106
Spezifikationen und Standards: . . . . .	106
Quellen . . . . .	106
<b>Darstellung mit CSS (Teil 1)</b>	<b>107</b>
Einführung und Ziele . . . . .	107
Ziele . . . . .	107
Relevanz für WBE . . . . .	107
CSS im Überblick . . . . .	108
Wozu CSS? . . . . .	108
Verbinden von CSS und HTML . . . . .	110
CSS-Regeln . . . . .	111
CSS-Versionen . . . . .	112
Selektoren in CSS 2.1 . . . . .	113
Typselektor . . . . .	113
Schreibweise der Regeln . . . . .	113
Vererbung . . . . .	113
Universalselektor . . . . .	114
class- und id-Selektor . . . . .	115
Kontext . . . . .	117
Gruppierung . . . . .	118
Pseudoklassen . . . . .	119

Pseudoelemente . . . . .	119
Nachfolger und Geschwister . . . . .	120
Attributselektoren . . . . .	120
Zusammenfassung . . . . .	121
Selektoren ab CSS3 . . . . .	121
Attributselektoren . . . . .	122
CSS3-Geschwister-Selektor . . . . .	123
CSS3-Pseudoklassen . . . . .	123
Eigenschaften . . . . .	123
Werte und Masseneinheiten . . . . .	123
Textdarstellung . . . . .	125
Schriftarten laden . . . . .	127
Listendarstellung . . . . .	128
Kaskade, Konflikte, Reset . . . . .	129
Browser-Stylesheet . . . . .	129
CSS-Reset . . . . .	130
Benutzer-Stylesheet . . . . .	131
Autoren-Stylesheet . . . . .	132
Kaskade und Konfliktauflösung . . . . .	132
Spezielle CSS-Regeln . . . . .	133
Kontrolle und Fehlersuche . . . . .	133
Quellen und Verweise . . . . .	136
Zur Vertiefung und zum Ausprobieren . . . . .	136
Referenzen . . . . .	136
Spezifikationen und Standards: . . . . .	136
Quellenangaben . . . . .	136
<b>Darstellung mit CSS (Teil 2)</b> . . . . .	<b>137</b>
Einführung und Ziele . . . . .	137
Ziele . . . . .	137
Relevanz für WBE . . . . .	137
Box-Modell . . . . .	138
Breite und Höhe . . . . .	138
Overflow . . . . .	140
Abstände und Rahmen . . . . .	140
Box-Modell im Überblick . . . . .	143
Horizontal zentrieren . . . . .	143
Fehlender Abstand? . . . . .	145
Hintergrund . . . . .	146
Minimale und maximale Dimensionen . . . . .	147
Positionierung . . . . .	147
position: static . . . . .	147
position: relative . . . . .	148
position: absolute . . . . .	149
position: fixed . . . . .	151
z-index . . . . .	151
Positionierung insgesamt . . . . .	152

Fliessende Boxen . . . . .	152
Layout mit CSS . . . . .	154
Layout-Varianten . . . . .	154
Mehrspaltiges Layout . . . . .	156
Horizontale Navigation . . . . .	156
Front-End-Framework: Bootstrap . . . . .	159
CSS Flexbox und CSS Grid . . . . .	161
Kontrolle und Fehlersuche . . . . .	163
Quellen und Verweise . . . . .	164
Material zur Vertiefung . . . . .	164
Referenzen . . . . .	164
Spezifikationen und Standards: . . . . .	164
Quellenangaben . . . . .	164
<b>Darstellung mit CSS (Teil 3)</b>	<b>165</b>
Einführung und Ziele . . . . .	165
Ziele . . . . .	165
Relevanz für WBE . . . . .	165
CSS-Techniken . . . . .	166
Hintergrundbilder . . . . .	166
CSS-Sprites . . . . .	168
Mehr zu CSS . . . . .	170
Boxen und Text verschönern . . . . .	170
Transitionen und Transformationen . . . . .	175
CSS-Variablen . . . . .	179
Drucken von Dokumenten . . . . .	179
CSS-Dateien, Werkzeuge . . . . .	180
CSS-Dateiaufbau . . . . .	180
CSS-Werkzeuge . . . . .	181
CSS optimieren . . . . .	182
Validieren, Fehlersuche . . . . .	183
LESS und Sass . . . . .	183
Quellen und Verweise . . . . .	185
Material zur Vertiefung . . . . .	185
Standards . . . . .	185
Quellenangaben . . . . .	185
<b>Anhang: Internet-Protokoll-Stack</b>	<b>186</b>
Ziele . . . . .	186
Relevanz für WBE . . . . .	186
Schichtenmodell . . . . .	187
Ablauf und Routing . . . . .	190
Unix-Befehle . . . . .	192
DNS . . . . .	193
Quellen und Verweise . . . . .	195
Material zur Vertiefung . . . . .	195
Quellenangaben . . . . .	195

# Web-Grundlagen

## Einführung und Ziele

Das *Erstellen von Webseiten und Webapplikationen* ist ein ausserordentlich umfangreiches Gebiet. Einerseits geht es um die Struktur von Dokumenten, welche mit der Auszeichnungssprache HTML umgesetzt ist, ausserdem aber auch um die Darstellung der Dokumente, welche mit CSS beschrieben werden kann. Schliesslich spielt die Programmierung eine immer wichtigere Rolle. Auf der Client-Seite (im Browser) geht es dabei vor allem um die Programmiersprache JavaScript, auf dem Server gibt es neben JavaScript auch eine Reihe weiterer Optionen.

Insgesamt könnte man rund um das Thema Webtechnologien problemlos Kurse über mehrere Semester anbieten und hätte dann immer noch nur einen Teil des Gebiets abgedeckt. Für den einsemestrigen Kurs WBE musste daher eine Auswahl getroffen werden, welche Themen aus dem riesigen Gebiet der Webtechnologien behandelt werden sollen.

Im Laufe der Jahre hat sich herausgestellt, dass Grundkenntnisse in HTML und CSS bei vielen Informatik-Studentinnen und -Studenten bereits vorhanden sind. Daher wurde entschieden, das Programmieren in JavaScript zum Schwerpunkt des Kurses WBE (Web-Entwicklung) zu machen.

Grundkenntnisse in HTML und CSS werden als Voraussetzung für den Kurs WBE angenommen, ebenso wie ein Grundverständnis der Web-Plattform: Browser, Webserver, URLs, DNS, HTTP-Protokoll und der für die Web-Entwicklung verwendeten Werkzeuge. Für den Fall, dass Sie über diese Vorkenntnisse nicht verfügen, wird ein Vorkurs mit einer Reihe von Übungsaufgaben angeboten.

Dieses Manuskript liefert Begleitmaterial zum Vorkurs. *Es enthält mehr als nur den für WBE zwingend vorausgesetzten Stoff.* Mit diesem Skript wird das Ziel verfolgt, die wichtigsten Grundlagen für die Beschäftigung mit Web-Themen zu vermitteln. Wenn Sie sich ausschliesslich in die WBE vorausgesetzten Themen einarbeiten möchten, können Sie sich an den Hinweisen und Zielen orientieren, welche am Anfang der einzelnen Kapitel stehen.

In diesem ersten Kapitel wird ein Überblick über den Kurs WBE sowie dieses Skript gegeben.

# Worum geht es in WBE?

## Webdesign und Webentwicklung

Das **Web** (kurz für: **World Wide Web**) ist heute eine wichtige Applikations- und Informationsplattform, sowohl auf Mobilgeräten als auch auf Desktop-Computern. Mittlerweile gibt es im Web-Umfeld unzählige Technologien und Spezifikationen. Einen kleinen Teil davon werden wir uns in WBE vornehmen.

Zunächst ist es sinnvoll, zwischen *Webdesign* und *Webentwicklung* zu unterscheiden.

Zentrales Thema im *Webdesign* ist das Erstellen von Webseiten mit **HTML** und **CSS**. Mit HTML (Hyper Text Markup Language) wird die Struktur und der Inhalt der Dokumente beschrieben und mit CSS (Cascading Stylesheets) deren Darstellung.

In der *Webentwicklung* geht es neben dem Erstellen von Webseiten auch um den Einsatz von Programmlogik im Web, um den Bau von *Webapplikationen*. Solche Applikationen können clientseitig oder serverseitig umgesetzt werden. Auch eine Kombination von beiden Ansätzen ist möglich. Auf dem Client kommt dabei vor allem die Programmiersprache **JavaScript zum Einsatz**, die im Browser direkt ausgeführt werden kann. Andere Sprachen (etwa TypeScript<sup>1</sup>, ReScript<sup>2</sup>, ClojureScript<sup>3</sup>) müssen zunächst mit einem **Transpiler zu JavaScript** übersetzt werden. Serverseitig können verschiedene Programmiersprachen, Frameworks und andere Applikationen wie beispielsweise Datenbanken eingesetzt werden.

Im Zusammenhang mit dem Erstellen von Webseiten oder Webapplikationen muss man sich neben HTML, CSS und JavaScript auch mit zahlreichen Werkzeugen, Protokollen, Frameworks und Bibliotheken beschäftigen. Zu den Werkzeugen gehören *Web-Browser und ihre Entwicklerwerkzeuge*, Code-Editoren mit passenden Erweiterungen, Programme zum Dateitransfer, Webserver und einige andere.

## Ziele und Inhalte von WBE

Webtechnologien spielen heute in der Informatik eine zentrale Rolle. Viele **IT-Systeme verwenden Web-Interfaces als plattformübergreifende Benutzerschnittstellen**, wobei von einfachen Web-Frontends bis zu komplexen Single Page Applications ein ganzes Spektrum von Möglichkeiten zur Verfügung steht. Auch Mobil- und Desktopapplikationen werden in Form von hybriden Applikationen häufig auf der Basis von Webtechnologien umgesetzt. Im Zentrum dieser Entwicklung steht die Programmiersprache JavaScript, die in den letzten Jahren komplett überarbeitet und umfassend erweitert wurde.

Ziele von WBE:

- Die Studierenden haben fundierte Kenntnisse über die Programmiersprache JavaScript und sind in der Lage, mit Hilfe von JavaScript client- und serverseitige Webapplikationen aufzubauen.
- Sie verstehen die Grundlagen der Webarchitektur bestehend aus Sprachen zur Auszeichnung und Darstellung von Dokumenten, sowie zum Umgang mit dem **DOM**

---

<sup>1</sup><https://www.typescriptlang.org>

<sup>2</sup><https://rescript-lang.org>

<sup>3</sup><https://clojurescript.org>

(Document Object Model) und zur asynchronen Client-Server Kommunikation (Ajax).

- Sie verstehen den Aufbau eines Web-Frameworks für die Entwicklung clientseitiger Single Page Applications und können ein solches Framework nutzen, um eigene Applikationen zu entwickeln.

Um diese Ziele zu erreichen, werden die WBE-Themen in drei Blocks aufgeteilt. Am Anfang des Semesters nimmt die Einführung in die Programmiersprache JavaScript einen grösseren Umfang ein. Anschliessend geht es um den Einsatz von JavaScript-Programmen im Browser und um den Bau eines kleinen Web-Frameworks. Hier der Aufbau von WBE mit ungefähren Angaben zur Anzahl der Lektionen:

Einführung in JavaScript mit Node.js (ca. 12 Lektionen)

- JS Engines, ECMAScript- und JavaScript-Versionen und Alternativen, Transpiler
- Grundlagen: Variablen, Datentypen, Arrays, Funktionen
- Objektmodell: Objekte, Konstruktoren, Prototypen, Klassen
- Asynchrone Ausführung von Funktionen, Callbacks, Event Queue, Promises
- Webserver mit Node.js: Modulsystem, JSON, RESTful APIs

JavaScript im Browser (ca. 8 Lektionen)

- Document Object Model
- Ereignisbehandlung im Browser
- Asynchrone Client-Server-Kommunikation (Ajax, Fetch-API)
- Zustand (Cookies, Sessions) und Authentisierung

Web Framework (ca. 8 Lektionen)

- Bau eines eigenen Frameworks für Single Page Applications
- Komponentenarchitektur und render-Methode
- Zustand von Komponenten, Properties, Komponententypen
- Lebenszyklus von Komponenten
- Ereignisbehandlung und Routenig
- Zustandscontainer

## Ziele und Inhalte des Vorkurses

Wie oben ausgeführt geht es in WBE im Wesentlichen um *Webentwicklung*. Wir starten dabei direkt mit einer Einführung in die Programmierung mit JavaScript. Grundkenntnisse im Webdesign, speziell HTML und CSS, werden dabei vorausgesetzt.

Wenn Sie sich bereits mit HTML, CSS und allgemein der Web-Plattform auskennen, hält der Vorkurs wenig Neues für Sie bereit. In diesem Fall können Sie den Vorkurs schnell durchsehen, ob der eine oder andere Abschnitt noch interessant für Sie sein könnte. In jedem Fall wird empfohlen, den Abschluss test des Vorkurses zu absolvieren, um Ihren Wissensstand einschätzen zu können.

Wenn Sie bereits Grundkenntnisse in HTML, CSS und der Web-Plattform haben, können Sie den Vorkurs nutzen, um mögliche Lücken zu schliessen.

Wenn Sie über keinerlei Vorkenntnisse in den genannten Bereichen verfügen, ist es empfehlenswert, den Vorkurs gründlich durchzuarbeiten und dieses Skript als Begleitmaterial zu verwenden. Anschliessend sollten Sie den Abschlusstest zur Selbsteinschätzung durchführen.

Aufbau des Vorkurses und dieses Manuskripts:

Thema	Inhalt
1	Einführung: WBE, Vorkurs, Abgrenzung
2	Grundlagen: Überblick HTML und CSS
3	Web-Plattform: Architektur, Werkzeuge
4	Markup, HTML (Teil 1: Grundlagen)
5	Markup, HTML (Teil 2): Formulare und mehr
6	Markup, HTML (Teil 3): HTML5, SVG, MathML
7	Darstellung mit CSS (Teil 1): Selektoren
8	Darstellung mit CSS (Teil 2): Positionierung, Layout
9	Darstellung mit CSS (Teil 3): CSS-Techniken
Anhang	Internet-Protokoll-Stack

Der Kurs stellt verschiedene Arten von Materialien zur Verfügung:

- Dieses Manuskript mit einer Einführung in die betreffenden Themen und Verweisen zu weiterführendem Material
- Verschiedene Tutorials zur Einführung in HTML und CSS
- Ein Abschlusstest zur Selbsteinschätzung

## Hinweis zu diesem Skript

Zu Beginn der einzelnen Kapitel dieses Skripts ist jeweils eine Liste der Ziele des Kapitels aufgeführt. Ziele, welche für WBE nicht zwingend Voraussetzung sind, werden jeweils in kursiver Schrift angegeben. Ein kurzer Absatz nach den Zielen beschreibt zudem die Relevanz des Kapitels für WBE.

## Abgrenzung

Es wurde bereits erwähnt, dass das Gebiet der Webtechnologien sehr gross ist. Damit ist klar, dass auch der WBE-Vorkurs und WBE selbst eine Reihe von Themen ausklammern müssen. Hier eine Auswahl von Web-Themen, die weder im Vorkurs noch in WBE behandelt werden können:

- Responsives Webdesign: Wie muss man Webseiten konzipieren, dass sie auf verschiedenen Gerätetypen einsetzbar sind? Dazu gehören unter anderem: Computerbildschirme in verschiedenen Größen, Smartphones, Tablet-Computer und TV-Geräte.
- Gestaltung und Usability: Wie muss eine Website gestaltet und umgesetzt werden, damit die Besucher ihre Ziele – zum Beispiel Informationen zu finden – schnell und auf angenehme Weise erreichen, und die Website gerne immer wieder besuchen.

- Barrierefreiheit: Auch die Bedürfnisse von Nutzern mit gewissen körperlichen oder geistigen Einschränkungen sollen bei der Gestaltung und den Bedienungsmöglichkeiten berücksichtigt werden.

Um gute Webangebote erstellen zu können, sollte man sich in verschiedenen Gebieten auskennen. In der Regel wird es aber so sein, dass man mit Experten verschiedener Disziplinen zusammenarbeitet, um gute Resultate zu erzielen.

In WBE legen wir den Schwerpunkt auf die Technologien. Ziel ist, dass Sie ein solides Wissen über die dem Web zugrunde liegenden Technologien erwerben.

Als kleine Einführung in die Bereiche *Gestaltung und Benutzbarkeit im Web* können Sie das *fakultative* Skript mit diesem Titel verwenden. Es ist nicht mehr auf dem allerneuesten Stand, aber die meisten behandelten Themen sind nach wie vor relevant. Es ist aber nicht Voraussetzung oder Bestandteil des Kurses WBE.

Das Thema *Responsives Webdesign* wird im Wahlfach *Mobile Applications 1 (MOBA1)* behandelt.

# Grundlagen

## Einführung und Ziele

Dieses Kapitel gibt einen ersten Überblick über HTML und CSS.

### Ziele

- Sie verstehen den Aufbau einer einfachen HTML-Datei.
- Sie erkennen, auf welche Weise CSS die Darstellung des HTML-Codes beschreibt.
- Sie verstehen HTML als Auszeichnungssprache, die es erlaubt, die hierarchische Struktur von Dokumenten mittels *Tags* zu beschreiben.

### Relevanz für WBE

Diese Themen gehören zum Grundwissen für Web-Entwickler. Sie werden auch in WBE vorausgesetzt.

# Webseiten: HTML und CSS

Die Grundidee ist ziemlich einfach:

Das **Web** (kurz für: **World Wide Web**) basiert auf einer *Client-Server-Architektur*. Der Client ist meist ein grafischer *Browser*. Er zeigt die Webseiten an, die über das Internet von einem *Webserver* geladen werden. Zu diesem Zweck wird im Browser eine Adresse eingegeben, die eine bestimmte Ressource im Web anfordert. Das angezeigte Dokument kann Verweise enthalten, die zu weiteren Ressourcen im Web führen.

Im Jahr 1992 waren die Webseiten noch relativ einfach aufgebaut. Das Bild zeigt eine Seite aus der Anfangszeit des World Wide Web.

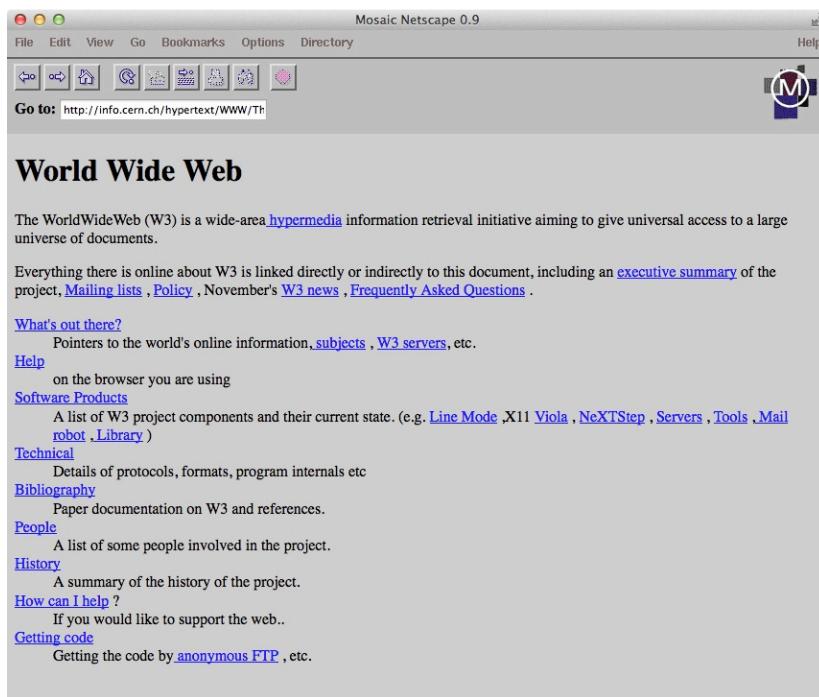


Abbildung 1: World Wide Web

Diese Seite kann unter der folgenden Adresse abgerufen werden:  
<http://info.cern.ch/hypertext/WWW/TheProject.html>.

Das funktioniert auch heute noch mit allen Browsern, obwohl das Dokument bald dreissig Jahre alt ist. Versuchen Sie einmal, ein dreissig Jahre altes Textverarbeitungsdokument zu öffnen (das war die Zeit von WordStar und den frühen Word-Versionen für 16-Bit-Windows).

Webseiten basieren auf der Auszeichnungssprache **HTML** (*Hypertext Markup Language*). Auf die Entwicklung von HTML und die verschiedenen Versionen werden wir noch eingehen. Hier nur so viel: Mit HTML kann *Inhalt und Struktur* von Dokumenten beschrieben werden. Inhalte werden mit sogenannten *Tags* ausgezeichnet, das sind eine Reihe von

vordefinierten Namen in spitzen Klammern, zum Beispiel <section>, teilweise erweitert durch Attribute.

Beispiel für ein HTML-Dokument:

```
1  <!DOCTYPE html>
2  <html lang="de">
3      <head>
4          <title>Mein erstes HTML-Dokument</title>
5          <meta charset="utf-8">
6      </head>
7
8      <body>
9          <p>Hallo Welt!</p>
10     </body>
11 </html>
```

Abbildung 2 zeigt, wie diese Seite im Browser dargestellt wird.

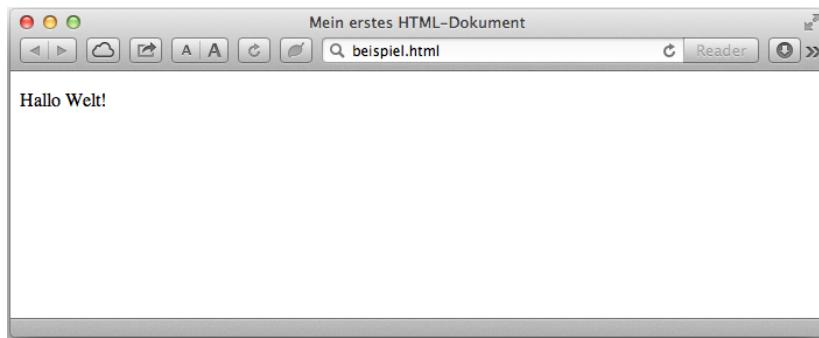


Abbildung 2: Unser erstes HTML-Dokument

Hier handelt es sich um ein einfaches HTML-Dokument, das in allen gängigen Browsern problemlos verarbeitet wird. In diesem ersten Beispiel besteht die komplette Seite aus einer HTML-Datei, das heisst es müssen keine weiteren Dateien geladen werden, um die Seite wie gewünscht im Browser anzuzeigen. Das ist nicht der Normalfall. In der Regel enthalten Webseiten auch Bilder, manchmal auch Video- oder Audiodateien. Diese Elemente sind in der Regeln in separaten Bild-, Video- oder Audio-Dateien untergebracht. Im HTML-Dokument wird auf diese Dateien verwiesen. Wenn der Browser die HTML-Datei lädt, werden auch diese Multimedia-Elemente vom Server geladen und auf der Seite angezeigt.

Oben wurde beschrieben, dass HTML für den Inhalt und die Struktur einer Webseite zuständig ist. Wenn eine HTML-Seite in den Browser geladen wird, zeigt dieser die Seite in einer Standard-Darstellung an: Überschriften werden grösser dargestellt, Absätze mit Abständen zueinander usw. Diese normalerweise nicht ausreichende Standard-Darstellung kann mit **CSS** (*Cascading Style Sheets*) angepasst werden.

Hier ist ein Beispiel einer etwas umfangreicherem HTML-Datei, die sowohl (eine Referenz auf) ein Bild enthält als auch auf eine CSS-Datei für die Darstellung verweist:

```
1  <!DOCTYPE html>
2  <html lang="de">
3      <head>
4          <title>Mein zweites HTML-Dokument</title>
5          <meta charset="utf-8">
6          <link rel="stylesheet" href="styles/screen.css">
7      </head>
8
9      <body>
10         <section>
11             <header>
12                 
13             </header>
14
15             <main>
16                 <article>
17                     <h1>Webseiten</h1>
18
19                     <p>Die Grundidee ist ziemlich einfach:</p>
20
21                     <p>Das <strong>World Wide Web</strong> (heute meist kurz: das Web)
22                         basiert auf einer <em>Client-Server-Architektur</em>. Der Client
23                         ist meist ein grafischer <em>Browser</em>. Er zeigt die Webseiten
24                         an, die über das Internet von einem <em>Webserver</em> geladen
25                         werden. Zu diesem Zweck wird im Browser eine Adresse eingegeben,
26                         die eine bestimmte Ressource im Web anfordert. Das angezeigte
27                         Dokument kann Verweise enthalten, die zu weiteren Ressourcen im
28                         Web führen.</p>
29
30                 </article>
31             </main>
32         </section>
33     </body>
34 </html>
```

Solange wir noch keine CSS-Datei haben, wird die Standard-Darstellung von HTML im Browser verwendet (Abbildung 3).

Bei CSS handelt es sich wie bei HTML um ein Textformat. CSS-Regeln können sowohl im HTML-Dokument selbst als auch in einer separaten Datei untergebracht werden. Im Sinne einer Trennung der verschiedenen Aspekte sollte praktisch immer die Variante mit der separaten Datei gewählt werden. Das hat auch den Vorteil, dass die in einer CSS-Datei beschriebene Gestaltung in mehreren HTML-Dateien verwendet werden kann.

Nehmen wir im obigen Beispiel ein paar CSS-Regeln hinzu, die wir in einer Datei `screen.css` notieren:

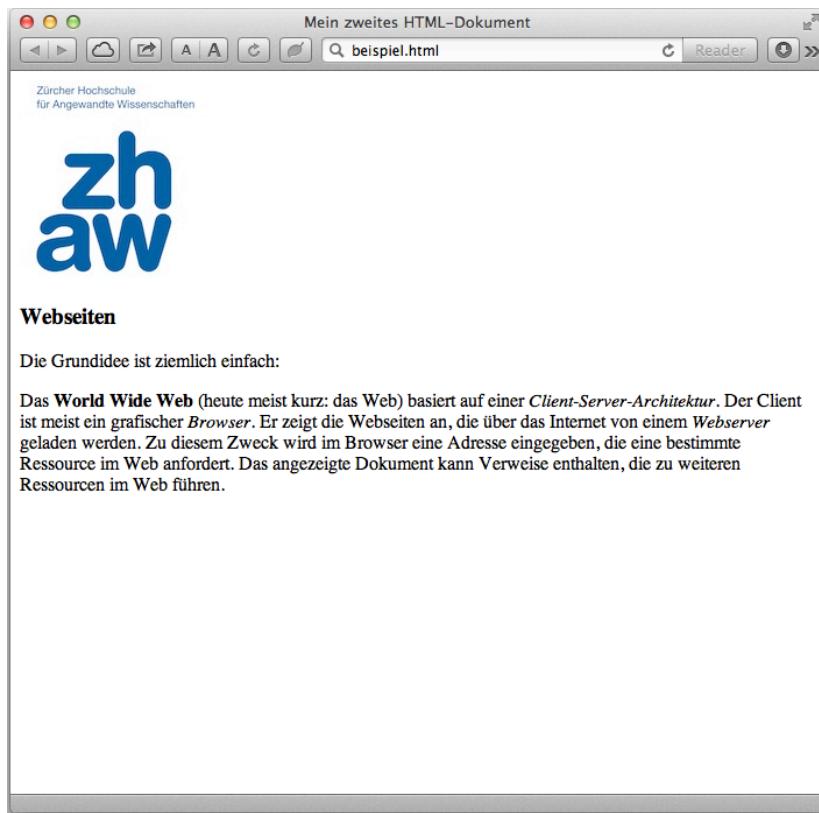


Abbildung 3: HTML-Dokument ohne CSS

```

1  @charset "UTF-8";
2
3  /*
4   * Stylesheet für das Einführungsbeispiel
5   */
6
7  body {
8      background-color: #999;
9  }
10
11 section {
12     min-height: 30em;
13     min-width: 20em;
14     max-width: 50em;
15     margin: 1em auto;
16     padding: 1em;
17     border-radius: 1em;
18     background-color: white;
19 }
20
21 header {
22     float: left;
23 }
24
25 article {
26     margin-left: 200px;
27     font-family: 'Georgia Pro', georgia, serif;
28     line-height: 140%;
29 }
30
31 h1 {
32     margin-top: 64px;
33     font-size: 2em;
34 }

```

Im Browser wird die Seite nun etwas ansprechender angezeigt (Abbildung 4).

## Terminologie

Im vorangehenden Abschnitt haben wir mehrfach den Begriff *Webseite* verwendet. Da die diese und andere Bezeichnungen rund um Webseiten nicht immer ganz einheitlich verwendet werden, sollen die Begriffe hier folgendermassen voneinander abgegrenzt werden:

- Eine **Webseite** ist ein Dokument im Web, „das mit einem Browser unter Angabe eines Uniform Resource Locators (URL) abgerufen und von einem Webserver angeboten werden kann“ (Quelle: Wikipedia). Technisch besteht eine Webseite (normalerweise) aus einer HTML-Datei und einer Reihe weiterer Dateien, die vom Browser

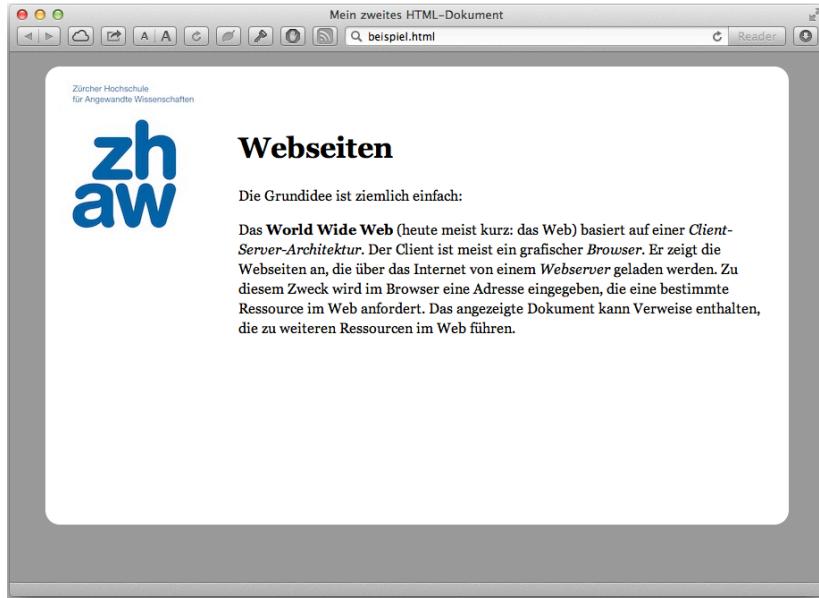


Abbildung 4: HTML-Dokument mit CSS

dazugeladen werden (CSS-Dateien, Bilder, ...). Man spricht daher auch von einer **HTML-Seite** oder einem **HTML-Dokument**.

- Als **Website** bezeichnet man eine Sammlung von Webseiten und anderen Ressourcen von einem bestimmten Anbieter und/oder zu einem bestimmten Thema, die im Web meist unter *einem* Domainnamen abgerufen werden können. Alternative Bezeichnungen: **Webauftritt**, **Webpräsenz**, **Webangebot**. Hier wurde die englische Bezeichnung *Website* nach vorne gestellt, da sie sehr gebräuchlich ist.
- Als **Startseite** (oder **Homepage**) bezeichnet man die Einstiegsseite einer *Website*, eine Seite, die meist über den Zweck der Website informiert und Möglichkeiten bietet, die Inhalte der Website zu erschliessen, zum Beispiel über Verweise zu weiteren Seiten.
- Von einer **Webapplikation** spricht man, wenn ein Webangebot eher Anwendungs- als Informationscharakter hat, also mehr interaktive Elemente enthält. Die Grenzen sind aber fliessend. Aus technischer Sicht enthält eine Webapplikation auch client- oder serverseitige Programmlogik. Webapplikationen können auch auf der Basis *einer* Webseite umgesetzt werden, indem ihre Inhalte dynamisch angepasst werden (**Single Page Applications**, **SPAs**).

## HTML-Grundlagen

Wie bereits erwähnt handelt es sich bei HTML um eine *Auszeichnungssprache (Markup Language)*: Bestimmte Bereiche in einem Dokument werden mit Hilfe von *Tags* ausge-

zeichnet, das heisst es wird spezifiziert, welche Rolle ein bestimmtes Inhaltselement im Dokument spielt.

Beispiel: Mit `<h1>...</h1>` wird eine Überschrift (heading) angegeben, mit `<p>...</p>` ein Absatz (paragraph). Der mit Hilfe der Tags markierte Bereich wird wie eine Klammer von den Tags umschlossen, beginnend mit dem *Start-Tag*, dann kommt der zu beschreibende Inhalt, schliesslich das *End-Tag*:

```
<h1>Webseiten</h1>
<p>Die Grundidee ist ziemlich einfach:</p>
```

Die Tags können nicht nur reinen Text umschließen, sondern auch ganze Bereiche mit weiteren Tags. Zum Beispiel umschliesst das `<html>`-Tag das ganze Dokument. Mit Tags markierte Bereiche dürfen also weitere solche Bereiche enthalten, sie dürfen sich aber nicht überlappen:

```
<!-- FALSCH: -->
<h1>Webseiten <p>Die Grundidee ist</h1> ziemlich einfach:</p>
```

Ein korrektes HTML-Dokument kann also als hierarchische Struktur von *Elementen* aufgefasst werden. Abbildung 5 hebt die hierarchische Struktur des Dokuments hervor.

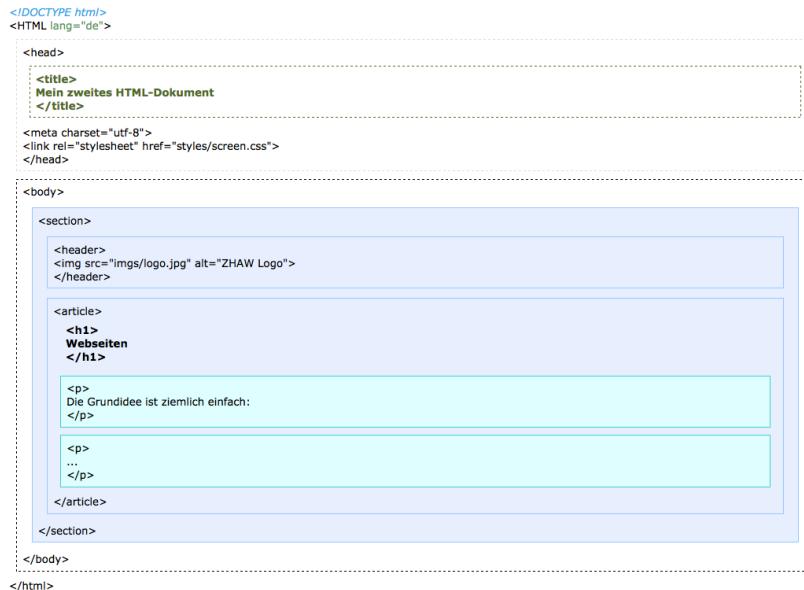


Abbildung 5: Hierarchische Struktur

Es gibt auch Elemente ohne Inhalt. In diesem Fall entfällt das End-Tag. Im Beispiel sind `img` und `meta` solche *leeren Elemente*:

```
<meta charset="utf-8">

```

Leere Elemente können mit einem Schrägstrich am Ende geschrieben werden:

```
<meta charset="utf-8" />

```

Vor einigen Jahren wurde versucht, HTML als Anwendung von **XML** (Extensible Markup Language) festzulegen. In XML ist der Schrägstrich bei leeren Elementen obligatorisch. Im aktuellen HTML-Standard ist der Schrägstrich erlaubt aber nicht vorgeschrieben, weshalb wir ihn meistens weglassen.

Die hierarchische Struktur eines HTML-Dokuments kann auch als Baumstruktur dargestellt werden (Abbildung 6).

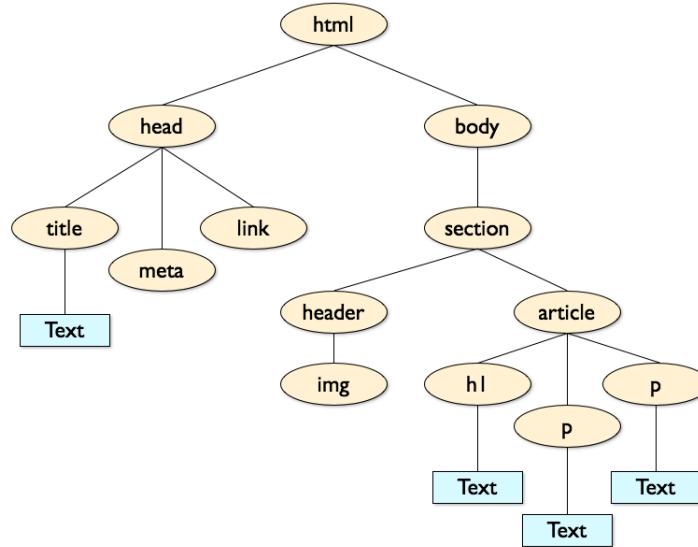


Abbildung 6: Hierarchische Struktur

Tags können *Attribute* enthalten. Diese stehen im Start-Tag (bzw. dem einzigen Tag bei einem leeren Element) und beschreiben das Element näher:

```

```

Im Beispiel hat das `img`-Element, das für das Einfügen des Bilds zuständig ist, zwei Attribute: Das `src`-Attribut gibt den Pfad zur Bilddatei an. Dies kann ein *relativer oder absoluter Pfad* oder eine Webadresse sein. Das `alt`-Attribut gibt einen Alternativtext an. Dieser Text wird angezeigt, wenn der Browser das Bild nicht anzeigt, was verschiedene Ursachen haben kann: Zum Beispiel weil das Laden von Bildern abgeschaltet ist oder unter dem angegebenen Pfad keine Bilddatei gefunden wird.

## Aufgaben

Überlegen Sie Antworten auf die folgenden Fragen:

- Die HTML-Datei besteht aus zwei Hauptteilen: Dem `head` und seinen Unterelementen, sowie dem `body` und seinen Unterelementen. Was meinen Sie, was ist die Rolle dieser beiden Teile?
- Wo erscheint im Browser der Inhalt des `title`-Elements?
- Das Beispiel besteht aus drei Dateien: `beispiel.html`, `screen.css` und `logo.jpg`. Wie sieht die Verzeichnisstruktur aus, damit die Seite korrekt geladen wird?
- UTF-8 ist eine Zeichencodierung. Vergleichen Sie die Angabe der Zeichencodierung in der HTML- und in der CSS-Datei.
- Die CSS-Datei ist ganz anders aufgebaut als die HTML-Datei. Die Überschrift `h1` wird mit der Schriftgrösse `2em` (`font-size`, mehr dazu in einer späteren Lektion) dargestellt. Welche Schriftart wird für `h1` verwendet? Hinweis: Für die Überschrift selbst ist zwar keine Schriftart angegeben, daher müssen Sie die Elemente weiter oben im Strukturbau ansehen, in denen sich die Überschrift befindet.

Hier sind noch einige Fragen für weitere Recherchen. Sie können die unten angegebenen Quellen verwenden, um die Fragen zu beantworten.

- Welche Rollen spielen HTML und CSS bei der Anzeige eines Dokuments im Browser genau?
- Wie werden Kommentare in einem HTML-Dokument geschrieben, die nicht auf der angezeigten Seite erscheinen sollen?
- Wozu dienen die HTML-Tags `html`, `p`, `h1`, `h2`?
- Kann man mit dem Notepad unter Windows eine Webseite bestehend aus HTML und CSS erstellen?
- Im Beispiel oben haben wir eine CSS-Datei mit einem `link`-Tag eingebunden. Welche Möglichkeit gibt es noch, CSS-Regeln in ein HTML-Dokument einzufügen?
- Mit welcher CSS-Eigenschaft wird die Hintergrundfarbe eines Elements festgelegt?
- Was bedeutet in diesem Zusammenhang die Farbangabe `#d2b48c`?

## Quellen und Verweise

- Kapitel 1 von: **Head First HTML and CSS**, Elisabeth Robson, Eric Freeman, Second Edition, 2012. Nicht mehr auf dem aktuellen Stand, aber als Einführung noch geeignet. Das Buch ist auch in deutscher Sprache verfügbar: *HTML und CSS von Kopf bis Fuss*.
- **A Beginner's Guide to HTML & CSS, Lesson 1: Building Your First Web Page.** Diese Einführung geht bereits etwas tiefer in das Thema HTML und CSS, als unsere Abhandlung oben. Ein Teil dieser Seite wäre also bereits Vorbereitung für spätere Lektionen. Adresse:  
<http://learn.shayhowe.com/html-css/building-your-first-web-page/>
- **HTML & CSS Crash Course Tutorial #1 - Introduction.** Diese Serie von Videos ist ebenfalls ein guter Einstieg in die Grundlagen von HTML und CSS. In der ersten Lektion gibt es eine Einführung in HTML und CSS sowie einen Überblick über wichtige Entwicklerwerkzeuge. Adresse:  
<https://www.youtube.com/watch?v=hu-q2zYwEYs>

## Quellenangaben

Ein Teil der Bilder und Inhalte stammt aus früheren Folien der Fächer WBD (Webdesign), INSY (Informationssysteme) und WEBT (Webtechnologien).

# Das Web als Plattform

## Einführung und Ziele

Die technischen Grundlagen des Internet und des World Wide Web sollten jedem Webentwickler bekannt sein, denn sie sind Voraussetzung für das Verstehen vieler Vorgänge beim Laden und Aufbauen von Webseiten und daher oft nötig, um aufgetretene Fehler einzukreisen und zu beheben.

Diese Einführung ist recht ausführlich. Einige Teile gehören eher zum Informatik-Grundwissen, etwa die Zeichencodierungen, speziell UTF-8, oder der Unterschied zwischen Internet und World Wide Web. In diesem Kapitel wird versucht, eine ganze Reihe von Themen zu behandeln, die rund um das Web als Plattform und die Entwicklung von Web-Angeboten wichtig sind. Beim Lesen können Sie problemlos die Teile überspringen, die Ihnen bereits bekannt sind.

### Ziele

- Sie verstehen das World Wide Web (WWW, Web) als Internet-Dienst.
- Sie wissen, aus welchen Grundelementen das Web aufgebaut ist.
- Sie wissen, welche Rolle HTML, CSS und JavaScript im Web spielen.
- Sie wissen, wie Ressourcen im Web adressiert werden.
- Sie kennen das Protokoll, mit dem Ressourcen im Web zwischen Server und Client übertragen werden.
- Sie kennen verschiedene Web-Editoren und die Developer Tools in Browsern.
- Sie wissen, wie eine Website im Internet publiziert werden kann.
- Sie wissen, wie eine Webadresse (URL) auf eine bestimmte Datei im Dateisystem des Webservers abgebildet wird und was man unter *Web-Root* versteht.
- Sie wissen, wozu SFTP und SSH gut sind und können sowohl auf der Kommandozeile als auch mit grafischen Clients Dateien zwischen Client und Server übertragen.
- Sie können sich mit SSH auf einem Server einloggen und dort einfache Aufgaben erledigen (Dateien kopieren, verschieben, umbenennen, löschen, bearbeiten, Verzeichnisse anlegen und löschen).

### Relevanz für WBE

Diese Themen gehören zum Grundwissen für Web-Entwickler. Sie werden auch in WBE vorausgesetzt.

## Client und Server im Web

Was geschieht beim Öffnen einer Webseite? Wir betrachten zunächst den häufigen Fall, dass das Client-Programm ein grafischer Browser ist:

1. Der Browser stellt eine Verbindung zum Server (mit Adresse/URL) und fordert ein Dokument (zunächst eine HTML-Datei) an.
2. Der Server schickt die Datei und unterbricht die Verbindung.
3. Der Browser analysiert das Dokument und beginnt mit dem Aufbau der Darstellung.
4. Falls in der HTML-Datei weitere benötigte Ressourcen referenziert werden, werden diese auf die gleiche Weise geladen.
5. Der Browser schliesst die Darstellung des Dokuments ab.

Dieser Ablauf wiederholt sich nach dem Klicken auf einen Link, wobei möglicherweise mit einem anderen Server verbunden wird. Ein im Browser angezeigtes Dokument kann dann durchaus dynamisch verändert werden oder asynchron weitere Informationen nachladen (*Ajax, Asynchronous JavaScript and XML*). Letzteres wird Thema im Kurs WBE sein.

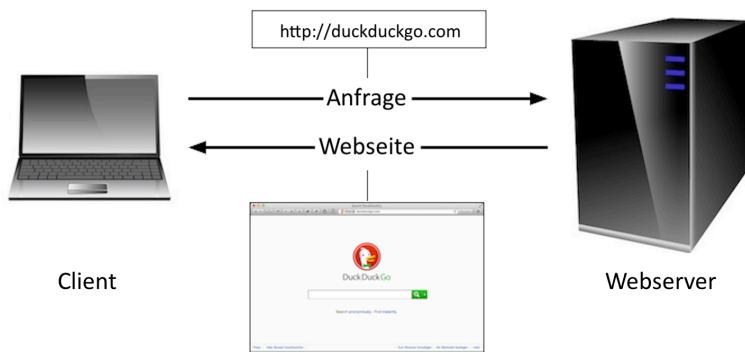


Abbildung 7: Client und Server im Web

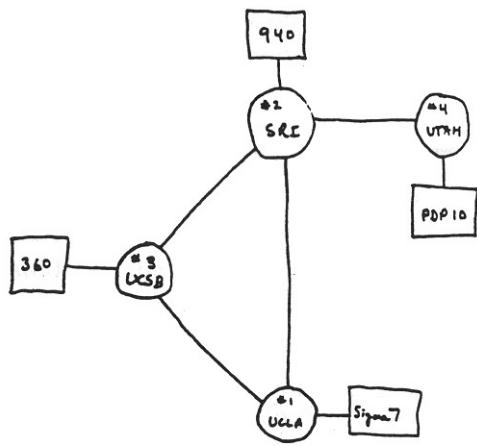
Zunächst zur Klärung einiger Begriffe:

Das **World Wide Web** und das **Internet** sind zwei verschiedene Dinge.

### Internet

Das **Internet** ist ein weltweites Netzwerk, bestehend aus vielen Rechnernetzwerken, durch die Daten ausgetauscht werden. Es basiert auf dem **ARPANET**, das 1969 mit vier Knoten in Betrieb ging (Abbildung 8; Stanford Research Institute, University of California Santa Barbara, University of California Los Angeles, University of Utah).

Als **Internet** wurde dieses Netzwerk erst ab 1987 bezeichnet, zu einem Zeitpunkt als es bereits etwa 27'000 Knoten umfasste. Verschiedene *Services* verwenden das Internet als Infrastruktur: E-Mail, Telnet, Usenet, Dateiübertragung, World Wide Web und zunehmend auch Telefonie, Radio und Fernsehen.



THE ARPA NETWORK

DEC 1969

4 NODES

Abbildung 8: ARPANET 1969

## World Wide Web

Das **World Wide Web**, kurz **Web**, ist somit ein auf dem Internet basierender Dienst. Es wurde Anfang der 1990er Jahre am CERN in Genf von **Tim Berners-Lee** auf einem NeXT-Computer entwickelt. Hier ist die Ankündigung, übrigens in *Usenet News*, einem anderen Internet-Dienst:

```
From: ti...@nxoc01.cern.ch (Tim Berners-Lee)
Newsgroups: comp.sys.next.announce
Subject: WorldWideWeb wide-area hypertext app available
Keywords: hypertext application wide area hypermedia source
Message-ID: <1991Aug20.015441.913@news.media.mit.edu>
Date: 20 Aug 91 01:54:41 GMT
Sender: ne...@news.media.mit.edu (USENET News System)
Followup-To: poster
Organization: MIT Media Laboratory
Lines: 43
Approved: lac...@plethora.media.mit.edu
```

The WorldWideWeb application is now available as an alpha release in source and binary form from info.cern.ch.

WorldWideWeb is a hypertext browser/editor which allows one to read information from local files and remote servers. It allows hypertext links to be made and traversed, and also remote indexes to be interrogated for lists of useful documents. Local files may be edited, and links made from areas of text to other files, remote files, remote indexes, remote index searches, internet news groups and articles. All these sources of information are presented in a consistent way to the reader. For example, an index search returns a hypertext document with pointers to documents matching the query. Internet news articles are displayed with hypertext links to other referenced articles and groups.

The code is not strictly public domain: it is copyright CERN (see copyright notice is in the .tar), but is free to collaborating institutes.

Also available is a portable line mode browser which allows hypertext to be browsed by anyone with a dumb ascii terminal emulator. Hypertext may be made public by putting on an anonymous FTP server, or by using a HTTP daemon. A skeleton HTTP daemon is also available in source form. A server may be written to make other existing data readable by WWW browsers. Files are

/pub/WWWNeXTStepEditor_0.12.tar.Z	NeXT application + sources
/pub/WWWLineMode_0.11.tar.Z	Portable Line Mode Browser
/pub/WWWDaemon_0.1.tar.Z	Simple server

Basic documentation is enclosed. Details about our project and about hypertext in general are available in hypertext form on our servers, as are lists of

known bugs and features.

This project is experimental and of course comes without any warranty whatsoever. However, it could start a revolution in information access. We are currently using WWW for user support at CERN. We would be very interested in comments from anyone trying WWW, and especially those making other data available, as part of a truly world-wide web.

Tim BL

Tim Berners-Lee ti...@info.cern.ch  
World Wide Web project Tel: +41(22)767 3755  
CERN Fax: +41(22)767 7155  
1211 Geneva 23, Switzerland

## Grundelemente des Web

Das Web basiert auf drei Grundprinzipien:

1. Sprache  
**Wie ist ein Dokument aufgebaut?**
  2. Adressierung  
**Wie adressiere ich ein Dokument?**
  3. Protokoll  
**Wie komme ich an ein Dokument heran?**

Auf diese drei Grundelemente gehen wir in den folgenden Abschnitten noch näher ein.

Das **World Wide Web Consortium** (kurz **W3C**) ist das Gremium zur Standardisierung der Techniken rund um das World Wide Web. Es wurde am 1. Oktober 1994 am MIT Laboratory for Computer Science in Cambridge (Massachusetts) gegründet. Gründer und Vorsitzender des W3C ist Tim Berners-Lee.

## Client: Web-Browser / User Agent

Als Client-Programme im Web werden unter anderem grafische Web-Browser auf verschiedenen Gerätetypen verwendet.

Web-Browser haben in den letzten Jahren grosse Fortschritte gemacht. Während sie in den vergangenen Jahren neben der Anzeige von mehr oder weniger statischen Seiten hauptsächlich als *Thin Clients* für Webanwendungen eingesetzt wurden, hat sich nun zunehmend mehr Anwendungslogik auf den Browser verlagert. Dies geht einher mit grossen Verbesserungen in den JavaScript-Engines heutiger Browser (als Programmiersprache im Browser wird JavaScript verwendet) sowie zahlreichen neuen Möglichkeiten, die im Rahmen der HTML-Spezifikation entwickelt und nach und nach in die Browser eingebaut werden.



Abbildung 9: Web-Browser

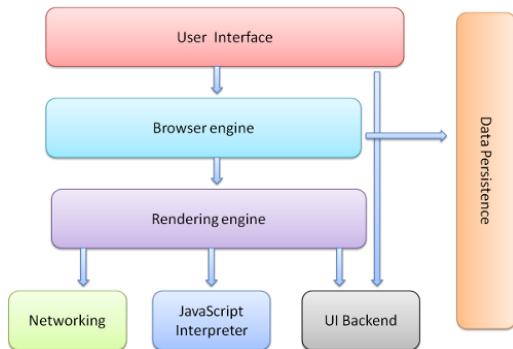


Abbildung 10: Browser main components

Abbildung 10 zeigt die wichtigsten Komponenten eines Web-Browsers. Es ist aus dem Artikel *How Browsers Work: Behind the scenes of modern web browsers*<sup>4</sup> entnommen.

Als Webdesigner sollte man eine Sammlung verschiedener gängiger Browser zur Verfügung haben, wenn möglich sogar in verschiedenen Versionen. Dabei ist es sinnvoll, von Zeit zu Zeit einen Blick in Nutzungsstatistiken zu werfen, um zu entscheiden, mit welchen Browerversionen man die eigenen Arbeiten mit Vorteil testet. Solche Nutzungsstatistiken geben aber in der Regel nur einen groben Hinweis über die Marktanteile der einzelnen Browser, da die Ergebnisse sehr stark von regionalen Gegebenheiten und den eingesetzten Messmethoden abhängig sind.

Die folgende Tabelle zeigt die Browser-Marktanteile im April 2021.<sup>5</sup>

Browser	Marktanteil Welt	Marktanteil Schweiz
Chrome	64.47%	41.05%
Safari	18.69%	34.56%
Firefox	3.59%	8.36%
Edge	3.39%	7.91%
Samsung Internet	3.3%	4.39%
Opera	2.22%	1.61%

Es ist klar zu sehen, welche drei bis vier Browser beim Testen berücksichtigt werden sollten. Welchen Browser man für den täglichen Gebrauch ansonsten benutzt, kann man dann immer noch entscheiden, und zum Beispiel überlegen, ob es von Vorteil ist, den Browser eines Konzerns zu verwenden, der seine Einnahmen in erster Linie aus Werbung generiert. Anstatt *Chrome* kann man übrigens auch den *Chromium* benutzen, der im Vergleich zum Chrome in Bezug auf Audio- und Videoformate ein paar Einschränkungen hat. Kurz: Chrome ist der durch Google leicht erweiterte und angepasste Build des Open Source Projekts Chromium.

Webseiten werden zwar häufig auf solchen grafischen Browsern auf Desktop-Computern mit Monitoren von zum Beispiel 24 oder 27 Zoll genutzt, aber nicht ausschliesslich. Beim Entwickeln von Webseiten sollte man daher auch an die verschiedenen anderen Gerätetypen denken, zum Beispiel:

- Auf einem Notebook-Monitor steht normalerweise weniger Platz zur Verfügung. Auf neuen hochauflösenden Notebook-Bildschirmen muss man zudem darauf achten, dass Texte nicht so klein werden, dass sie nicht mehr lesbar sind.
- Smartphones und Tablet-Computer haben im Laufe der Zeit immer mehr an Bedeutung gewonnen und sind heute ebenso wichtig wenn nicht wichtiger als Desktop- und Notebook-Computer. Neben dem eingeschränkten Platz ist hier auch auf andere Bedienkonzepte (Touch) zu achten. Ältere Handys sind dagegen kaum noch verbreitet. Sie haben meist sehr eingeschränkte Browser.

<sup>4</sup><http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/>

<sup>5</sup>Quelle: <https://gs.statcounter.com/>

- Es gibt auch textbasierte Browser wie den Lynx (Abbildung 11). Solche Browser werden zwar nur in bestimmten eher seltenen Situationen eingesetzt, sie können aber auch gute Hinweise geben, wie sich eine Webseite einer Suchmaschine präsentiert.
- Webseiten werden auch mit Sprachausgabe oder Braille-Geräten zum Fühlen der Inhalte genutzt. Vor allem Blinde und Sehbehinderte sind auf diese Möglichkeiten angewiesen.
- Zunehmend sollten Webseiten auch auf TV-Geräten nutzbar sein. Hier sind oft die Bedienungsmöglichkeiten eingeschränkt (Fernbedienung mit Pfeiltasten...) und auch der grösste Abstand zwischen Anzeige und Betrachter ist zu berücksichtigen.
- Webseiten werden auch ausgedruckt. Auch das sollte nicht vergessen werden.

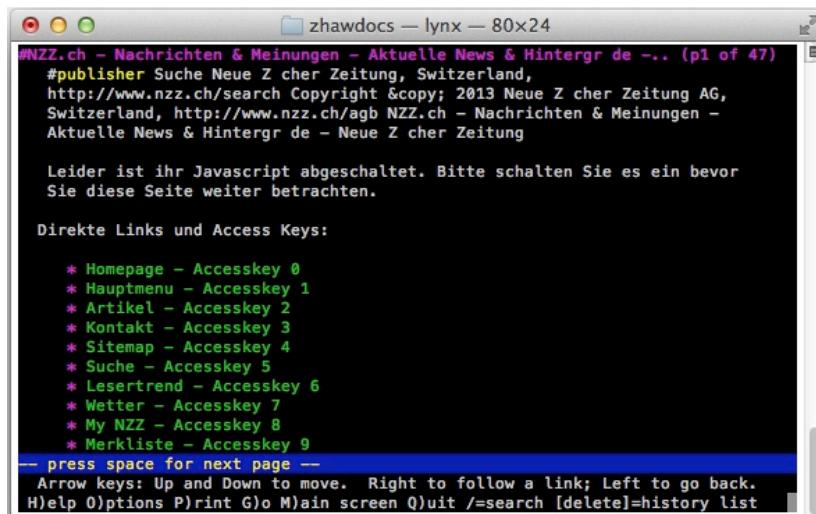


Abbildung 11: Lynx

Diese Auflistung zeigt, dass es nicht sinnvoll ist, Webseiten für einen Browser-Typ und eine Anzeigegröße zu „optimieren“. Mit Hilfe von *responsivem Webdesign* können Webseiten so flexibel konzipiert werden, dass sie auf verschiedenen Geräten nutzbar sind. Das ist aber Thema in einem anderen Kurs: *Mobile Applications (MOBA)*.

Da die Bezeichnung **Web-Browser** meist für grafische Browser wie den Chrome oder den Firefox verwendet wird, bevorzugt das W3C die Bezeichnung **User Agent**, um daran zu erinnern, dass Webseiten mit ganz unterschiedlichen Programmen auf verschiedenen Gerätetypen genutzt werden.

## Server: Web-Server

Als Server wird in der Client-Server-Anwendung *World Wide Web* eine Software verwendet, welche Anfragen im Protokoll HTTP (dazu gleich mehr) eines User Agent beantworten und die gewünschten Ressourcen (unter anderem HTML- und CSS-Dateien) an den User Agent sendet. Beispiele für häufig eingesetzte Web-Server sind:

- Apache HTTP Server
- Microsoft Internet Information Services (IIS)
- Nginx (ausgesprochen “engine-ex”)
- Apache Tomcat (Java)

## Rolle von HTML, CSS und JavaScript

Hier noch einmal die Grundelemente des Web:

1. **Sprache: Wie ist ein Dokument aufgebaut?**
2. Adressierung: Wie adressiere ich ein Dokument?
3. Protokoll: Wie komme ich an ein Dokument heran?

Wenden wir uns zunächst dem ersten Punkt zu.

Wir wissen bereits, dass zu diesem Zweck **HTML** (Hypertext Markup Language) die zentrale Rolle spielt. HTML, ist eine textbasierte Auszeichnungssprache zur Strukturierung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten.

```

1  <!DOCTYPE html>
2  <html lang="de">
3    <head>
4      <title>Mein zweites HTML-Dokument</title>
5      <meta charset="utf-8">
6      <link rel="stylesheet" href="styles/screen.css">
7    </head>
8
9    <body>
10   ...
11   </body>
12 </html>
```

HTML ist nicht nur eine *Markup Language*, es ist auch *Hypertext*, das heisst man kann über Verweise von einer Stelle zu einer anderen springen, auch zu anderen Dokumenten.

Ein HTML-Dokument kann Inhalte auch in weiteren Text- und Binärformaten dazuladen. Dazu gehören Bilder in den Formaten PNG, JPEG oder GIF (alles Pixelgrafiken), SVG (Vektorgrafik), MathML (mathematische Formeln), sowie verschiedene Binärformate für Audio und Video.

Für die Seitengestaltung wird **CSS** (Cascading Stylesheets) verwendet:

```

1 article {
2   margin-left: 200px;
3   font-family: 'Georgia Pro', georgia, serif;
4   line-height: 140%;
5 }
```

Im Beispiel-Code wird beschrieben, wie das **article**-Element dargestellt werden soll: Abstand auf der linken Seite des Artikels sowie Schriftart und Zeilenabstand des Texts.

Ausser der *logischen Struktur* und der *Darstellung* spielt noch eine dritte Komponente eine Rolle, das *Verhalten*. Mit Hilfe von **JavaScript** kann eine Webseite um Programmlogik ergänzt werden, zum Beispiel um Elemente, die auf der Seite angezeigt werden, in bestimmten Situationen dynamisch anzupassen, oder Eingaben in Formularen zu validieren.

Die verschiedenen Inhaltstypen werden durch den **Content-Type** charakterisiert, der aus einem Typ und einem Subtyp besteht. Hier ein paar Beispiele:

```
text/html
text/css
application/javascript
application/pdf
image/jpeg
image/svg+xml
video/mp4
audio/mpeg
```

Die *Content-Types* entstammen eigentlich dem E-Mail-Dienst. Dort werden sie verwendet, um eine E-Mail-Nachricht aus verschiedenen Komponenten unterschiedlichen Typs zusammensetzen zu können, zum Beispiel Text und ein Word-Dokument als Anhang. Die Spezifikation dafür heisst MIME (Multipurpose Internet Mail Extensions). Daher werden die *Content-Types* auch als *Mime-Types* bezeichnet.

Wenn wir von Bildern und anderen Spezialformaten einmal absehen, haben wir im Fall von statischen Webseiten die in Abbildung 12 gezeigte Situation.

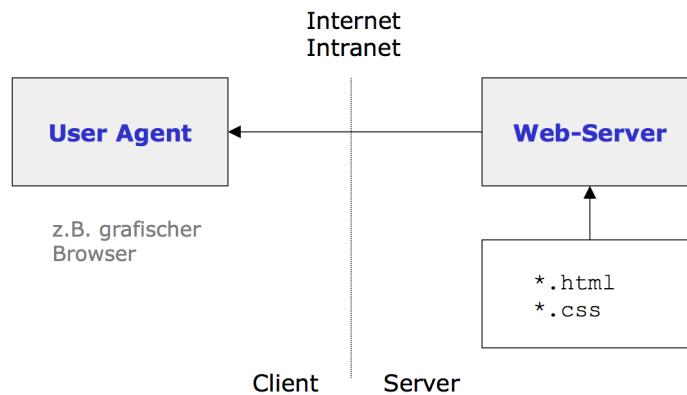


Abbildung 12: Statische Webseiten

Mit clientseitiger Programmlogik wird zusätzlich ein Script-Interpreter auf der Clientseite benötigt. Als Programmiersprache wird JavaScript verwendet. Die Situation ist dann wie in Abbildung 13 gezeigt.

Programmlogik muss nicht immer auf dem Client angesiedelt sein. Es ist nämlich nicht unbedingt nötig, dass die zum Client geschickten Daten in statischen Dateien auf dem Server abgelegt sind. Sie können auf dem Server auch dynamisch nach Bedarf generiert

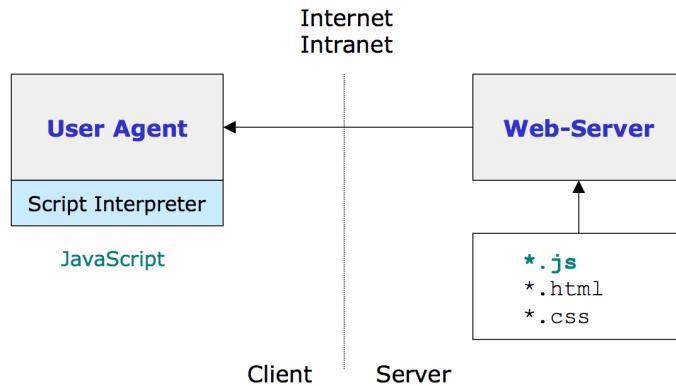


Abbildung 13: Client-Programme

werden, um zum Beispiel Daten aus einer Datenbank zu integrieren oder zusätzliche Informationen von anderen Servern zu holen. Während man im Browser auf JavaScript beschränkt ist (was nicht unbedingt schlecht ist, denn JavaScript ist eine ausdrucksstarke und flexible Programmiersprache), kann man auf dem Server aus einer ganzen Reihe möglicher Technologien wählen (Abbildung 14).

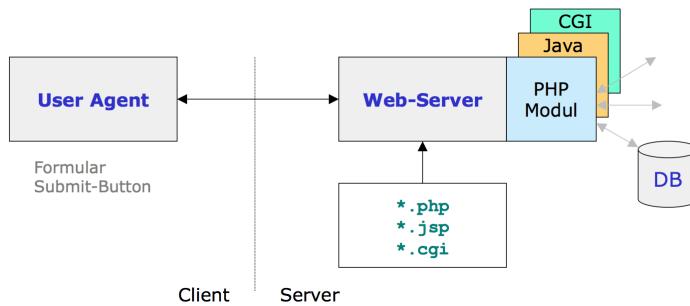


Abbildung 14: Server-Programme

**CGI** steht übrigens für *Common Gateway Interface*, eine Schnittstelle zum Datenaustausch zwischen Webserver und Anwendungsprogramm. Serverseitige Programme können daher in fast beliebigen Programmiersprachen geschrieben (wie C, Python, Ruby, Lisp) und mittels CGI an den Webserver angebunden werden.

## Adressierung: URI/URL

Nun zur Frage 2 im Zusammenhang mit den Grundelementen des Web:

1. Sprache: Wie ist ein Dokument aufgebaut?
2. **Adressierung: Wie adressiere ich ein Dokument?**
3. Protokoll: Wie komme ich an ein Dokument heran?

Zum Zweck der Adressierung wird im Web die sogenannte **URI** (*Universal Resource Identifier*, statt Universal auch: Uniform) verwendet. Der Aufbau ist folgendermassen:

`URI ::= scheme ":" [authority] path [ "?" query ] [ "#" fragment ]`

Diese Notation nennt man übrigens **BNF** (*Backus-Naur-Form*). Vereinfacht ausgedrückt: Elemente in eckigen Klammern [...] sind optional, Elemente in Anführungszeichen stehen genau so im resultierenden Ausdruck und Namen ohne Anführungszeichen sind Variablen, die an anderer Stelle genauer definiert sind.

## URL

Für das Web interessiert uns vor allem die **URL** (*Uniform Resource Locator*). URLs sind eine Subklasse von URIs. Ursprünglich war geplant, URIs als Verallgemeinerung von zwei Arten von Adressangaben zu spezifizieren, den URLs und den **URNs** (*Uniform Resource Name*). Die Trennung ist allerdings nicht so klar und es gibt Adressen, die eigentlich keiner den beiden Klassen angehören, zum Beispiel *mailto*-Angaben (Abbildung 15).

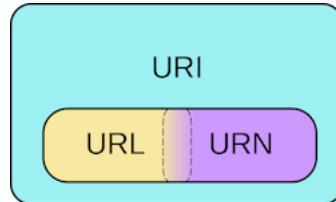


Abbildung 15: URI, URL, URN (Bild: Wikipedia)

Der Aufbau einer URL hängt vom verwendeten Protokoll ab. Für HTTP ist eine URL wie folgt aufgebaut:

`"http://" [user[:password]@] host [:port] path [ "?" query ] [ "#" fragment ]`

Hier ist ein Beispiel (Quelle: Wikipedia):

```
scheme-specific-part → → → |  
|  
http://hans:geheim@example.org:80/demo/example.cgi?land=de&stadt=aa#geschichte  
| | | | | | | | | |  
| | | | host url-path query fragment  
| | | password port  
| | user  
scheme (hier gleich Netzwerkprotokoll)
```

User und Passwort werden eher selten angegeben. Die Portnummer kann man weglassen, wenn der Standard-Port verwendet wird. Dieser ist für HTTP 80 und für HTTPS 443. Der Hostname kann als Domainname oder als IP-Adresse angegeben werden.

Noch eine Bemerkung zum Genus der Abkürzungen **URI** und **URL**: Gelegentlich werden diese auch als Maskulinum aufgefasst und mit dem Artikel *der* verwendet, da die Abkürzungen für *Identifier* und *Locator* stehen. Verbreiteter sind aber mittlerweile die Bezeichnungen *die URI* und *die URL*, angelehnt an das Wort *Internetadresse*.

## IP-Adresse

Jeder Computer im Internet hat eine eindeutige numerische Adresse, die **IP-Adresse** (das *eindeutig* stimmt hier nicht ganz, s.u.). Am meisten verbreitet ist heute noch **IPv4**, bei dem die Rechneradresse aus 32 Bit besteht, die normalerweise als vier dezimale Zahlen (je ein Byte), durch Punkte getrennt, dargestellt werden. Beispiel:

**173.194.40.95**

32 Bit heisst: Es gibt etwa 4.2 Milliarden Adressen. Das reicht heute nicht mehr aus. Es gibt zwar ein paar Möglichkeiten, sparsam mit den Adressen umzugehen. Eine Möglichkeit ist, die Adressen dynamisch zuzuteilen, wenn ein Computer an ein Netzwerk angeschlossen wird. Das Protokoll dazu heisst **DHCP** (*Dynamic Host Configuration Protocol*). Eine weitere Variante ist, ein Subnetz unter einer einzigen IP-Adresse ans Internet anzuschliessen und im Subnetz private IP-Adressen zu verwenden. Ein *Router* am Übergabepunkt setzt die Adressen dann mittels **NAT** (*Network Address Translation*) um.

Da für den Host-Teil der URL auch eine IP-Adresse möglich ist, ist auch dies eine gültige URL:

**<http://17.251.224.35/projects/goals.html>**

Die eigentliche Lösung für das Problem mit der Adressknappheit ist **IPv6**, das zunehmend eingesetzt wird. IPv6 verwendet 128-Bit-Rechneradressen, dargestellt als 8 hexadezimale 16-Bit-Zahlen. Beispiel:

**243f:6a88:85a3:08d3:1319:8a2e:0370:7344**

Damit stehen etwa  $3.4 \times 10^{38}$  Adressen zur Verfügung, das sollte für eine Weile reichen...

## Domain-Name

IP-Adressen sind schlecht zu merken, Abhilfe bietet das *Domain Name System (DNS)*. Anstelle der numerischen Adressen können Namen verwendet werden, die vom DNS in die IP-Adressen übersetzt werden. Für den Domain-Namen **www.zhaw.ch** wird zum Beispiel die IP-Adresse **160.85.104.111** geliefert.

Analogie: Die IP-Adresse kann man mit einer Telefonnummer vergleichen und das DNS mit einem Telefonbuch, das dazu dient, die Nummer nachzuschlagen.

Domain-Namen setzen sich aus Domains zusammen. Zum Beispiel für **www.zhaw.ch**:

- **ch**: Top-Level-Domain
- **zhaw**: Second-Level-Domain
- **www**: Third-Level-Domain

Die einzelnen Namen werden durch Punkte voneinander getrennt. Am Ende steht eigentlich ebenfalls ein Punkt, wie beispielsweise in `www.zhaw.ch.`, der letzte Punkt wird aber üblicherweise weggelassen. Das DNS bildet also eine hierarchische Struktur und innerhalb der einzelnen Bereiche werden die Namen autonom verwaltet (Abbildung 16).

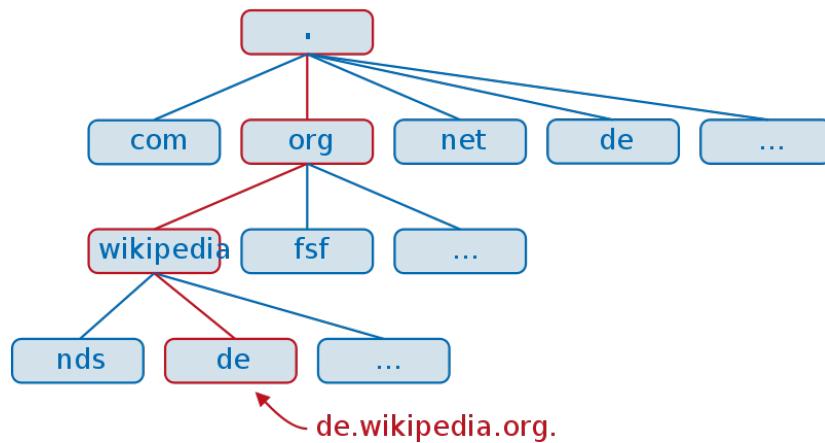


Abbildung 16: DNS-Raum (Bild: Wikipedia)

## Protokoll: HTTP

Es bleibt noch die Frage 3 nach dem Protokoll:

1. Sprache: Wie ist ein Dokument aufgebaut?
2. Adressierung: Wie adressiere ich ein Dokument?
3. **Protokoll: Wie komme ich an ein Dokument heran?**

Zum Transport der Webseiten-Bestandteile wird das HTTP-Protokoll (*Hypertext Transfer Protocol*) verwendet. Es definiert eine Reihe von Befehlen, über die Ressourcen vom Webserver angefragt oder modifiziert werden können. Am wichtigsten ist der GET-Befehl.

Beispiel: Es soll die Webseite `http://dublin.zhaw.ch/~bkrt/hallo.html` in den Browser geladen werden. Dazu stellt der Browser zunächst eine Verbindung zum Server `dublin.zhaw.ch` her und sendet dann ein GET-Kommando mit der gewünschten Datei und ein paar zusätzlichen Informationen an den Server:

```

GET /~bkrt/hallo.html HTTP/1.1
Host: dublin.zhaw.ch
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:22.0) Gecko/20100101 Firefox
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive

```

Ist die Datei vorhanden, sendet der Server zunächst einen Header mit Informationen und nach einer Leerzeile den Datei-Inhalt:

```

HTTP/1.1 200 OK
Date: Mon, 15 Jul 2013 17:10:56 GMT
Server: Apache/2.2.15 (CentOS)
Last-Modified: Wed, 17 Oct 2012 08:10:22 GMT
ETag: "5b018a-af-4cc3cccd575780"
Accept-Ranges: bytes
Content-Length: 175
Connection: close
Content-Type: text/html; charset=UTF-8

```

```

<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Hallo</title>
  </head>
  <body>
    <h1>Hallo</h1>
    <p>Ich bin eine Webseite</p>
  </body>
</html>

```

Anhand des Content-Type kann der Browser entscheiden, ob und wie der Datei-Inhalt angezeigt werden soll. HTTP geht davon aus, dass Textdaten korrekt von einem Computer zu einem anderen übertragen werden können. Dazu setzt es auf weiteren Protokollen auf, speziell TCP/IP. Der Internet-Protokoll-Stack wird in anderen Vorlesungen des Informatik-Studiums behandelt. Im Anhang finden Sie eine Einführung.

## Web-Entwicklung

Für das Entwickeln von Web-Angeboten werden eine Reihe von Werkzeugen eingesetzt:

- Das wichtigste Werkzeug ist ein geeigneter Editor. Gleich mehr dazu.
- Das Web besteht nicht nur aus HTML-, CSS- und JavaScript-Dateien. Es werden auch Bilder (pixel- oder vektorbasiert), Videos in verschiedenen Formaten, Animationen (Flash oder andere), Audio-Dateien, oder andere Seiterelemente eingesetzt.

Für diese Formate werden in der Regel ebenfalls geeignete Autorenwerkzeuge benötigt.

- Die Website muss schliesslich auf einen Server geladen werden. Dazu benötigt man Programme zum Dateitransfer. Ausserdem sollte man sich auch auf einem Server anmelden und dort direkt Dateien anlegen, kopieren, verschieben, löschen, oder deren Berechtigungen anpassen können.
- Zu den Werkzeugen zum Überprüfen der Ergebnisse gehören Lint-Tools, Validatoren und Test-Werkzeuge.
- Die tatsächlich auf der Website eingesetzten Formate werden oft aus anderen Formaten generiert. So werden teilweise flexiblere Stylesheet-Sprachen wie Sass oder LESS verwendet, die dann aber nach CSS konvertiert werden müssen (mehr in einem späteren Kapitel). Oder TypeScript als Scriptsprache, die für den Einsatz im Web nach JavaScript übersetzt werden muss.
- Auch wenn eine Website bereits “funktioniert”, gibt es oft noch Optimierungspotential. Bilder können optimiert werden, ebenso CSS und JavaScript. Bei letzteren können zum Beispiel Kommentare und unnötiger Whitespace entfernt werden. Ausserdem können mehrere CSS-Dateien zu *einer* CSS-Datei zusammengefügt werden, damit beim Laden der Seite nur ein HTTP-Request zum Server für die Styles nötig ist. Das gleiche gilt für JavaScript-Dateien.

Bei der Webentwicklung sind also eine ganze Reihe von Werkzeugen involviert. Es ist klar, dass man – um produktiv arbeiten zu können – einen guten Überblick über die verfügbaren Tools haben und die wichtigsten davon gut beherrschen sollte.

## Web-Editoren

HTML-Dateien sind Textdateien, sie können also im Prinzip mit jedem Texteditor erstellt und bearbeitet werden, zum Beispiel mit dem Notepad (Windows). Es ist aber ein grosser Vorteil, wenn ein Editor zusätzliche Unterstützung für das Bearbeiten von HTML-, CSS- oder JavaScript-Dateien mitbringt, zum Beispiel:

- Syntax Coloring: Farbiges Hervorheben bestimmter Teile entsprechend der Sprachsyntax (bei HTML zum Beispiel Tags, Attribute, Inhalt in verschiedenen Farben)
- Automatisches Vervollständigen von Elementen und Attributen
- Verwalten von Code-Bibliotheken
- Prüfen auf Einhalten der korrekten Syntax
- Rechtschreibprüfung und -korrektur
- Browser-Vorschau

Die Auswahl und Konfiguration eines geeigneten Editors und das Beherrschung dieses Werkzeugs ist ein wichtiger Faktor erfolgreicher Web-Projekte. Auf den Developer-Seiten von Google gab es unter *Web Fundamentals* ein Kapitel *Choose a Good Editor*, in dem unter anderem stand:

Code editors are the quintessential development tool. But what makes an editor more than a simple word processor are its shortcuts that get you working faster.

You know you've truly learned to use your editor when you are able to quickly move around your code using a range of keyboard short-cuts and editor commands. It's always good to keep a cheatsheet handy for editor commands. Top editors also let you customize commands and keyboard shortcuts so that you only have to remember your own implementations.

Ein weit verbreiteter Code-Editor ist **Visual Studio Code** (kurz: **VSCODE**) von Microsoft (Abbildung 17).<sup>6</sup> Dieser Editor eignet sich sehr gut für die Web-Entwicklung und kann durch Erweiterungen um zahlreiche Funktionen ergänzt werden. Eine solche Erweiterung ist **Live Server**. Damit kann ein kleiner Webserver gestartet werden, so dass die Dateien über das HTTP-Protokoll zur Vorschau in den Browser geladen und bei Änderungen dort direkt aktualisiert werden.

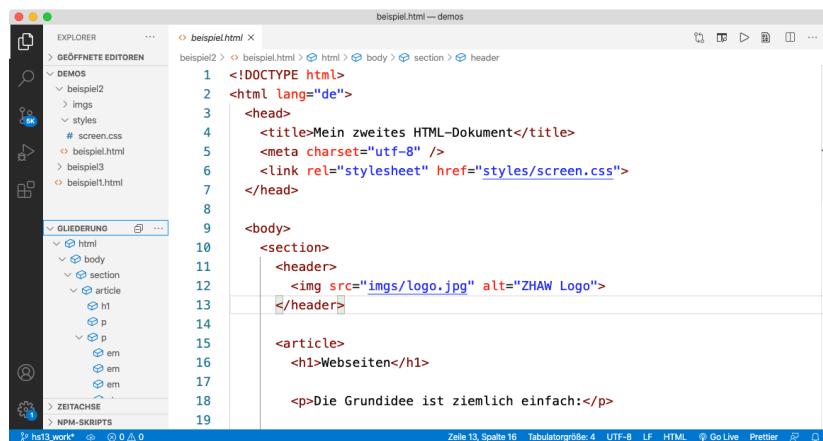


Abbildung 17: Visual Studio Code

Der Visual Studio Code basiert selbst auf Web-Technologien, genauer auf einem Werkzeug namens Electron<sup>7</sup>, das es erlaubt, aus einer Webapplikation eine plattformübergreifende Anwendung zu erstellen. Dabei setzt es auf dem Chromium-Projekt auf.

Es gibt noch zahlreiche weitere Editoren und Entwicklungsumgebungen für die Web-Entwicklung. Hier eine kleine Auswahl:

- Wie VSCODE ebenfalls auf Web-Technologien basieren **Atom**<sup>8</sup> von GitHub (mittlerweile zu Microsoft gehörend) und **Brackets**<sup>9</sup> von Adobe. Beide sind Open Source und flexibel konfigurierbar.
- Ziemlich verbreitet in der Webentwicklung ist **Sublime Text**.<sup>10</sup> Für diesen Editor, der für Windows, Linux und Mac angeboten wird, existieren zahlreiche Plugins und Konfigurationsmöglichkeiten. Er ist nicht kostenlos, kann aber ohne Bezahlung getestet werden.

<sup>6</sup><https://code.visualstudio.com>

<sup>7</sup><https://www.electronjs.org>

<sup>8</sup><https://atom.io>

<sup>9</sup><http://brackets.io>

<sup>10</sup><http://www.sublimetext.com>

- Eine leistungsfähige Entwicklungsumgebung für HTML, CSS und JavaScript ist **WebStorm**.<sup>11</sup> Es ist ein kommerzielles Werkzeug der Firma JetBrains, für Ausbildungszwecke werden kostenlose Lizenzen vergeben.
- **Komodo Edit**<sup>12</sup> läuft unter Windows, Linux und Mac und bietet gute Unterstützung für das Bearbeiten von Web-Dokumenten (u.a. einen speziellen Modus für HTML5), ist erweiterbar und gratis verfügbar.
- Auch Code-Editoren wie **Notepad++**<sup>13</sup> (Windows), **PSPad**<sup>14</sup> (Windows), **Bluefish**<sup>15</sup> (Linux), sind Kandidaten für die Webentwicklung und kosten nichts.
- Gute Entwicklungswerkzeuge nicht nur für die Web-Entwicklung sind sicher auch **Vim**<sup>16</sup> und **Emacs**<sup>17</sup>. Beide benötigen einige Einarbeitungsaufwand, bis man produktiv mit ihnen arbeiten kann. Wenn Sie einen davon bereits einsetzen, können Sie diesen auch für die Webentwicklung verwenden. Emacs ist mittlerweile etwas in die Jahre gekommen und deutlich mehr als Vim eine Welt für sich. Vi ist der *VI*sual Editor, Vim (Vi IMproved) eine Weiterentwicklung davon. Jeder Entwickler sollte wenigstens so viel **vi** beherrschen, um auf der Kommandozeile – zum Beispiel auf einem Server – damit Dateien öffnen, bearbeiten, speichern und wieder schliessen zu können.

Weniger geeignet für die Web-Entwicklung sind WYSIWYG-Editoren. WYSIWYG steht für *What you see is what you get*. Wie wir bereits wissen, funktioniert das im Web aber nicht: *Die einzige und korrekte Darstellung* einer Webseite gibt es nicht. Wie die Seite letztendlich aussieht, ist von einer Reihe verschiedener Faktoren abhängig.

Es gibt also zahlreiche für die Webentwicklung geeignete Programme und viele davon können kostenlos eingesetzt werden. Nehmen Sie sich die Zeit, verschiedene Programme zu testen. Prüfen Sie dabei auch, ob sich der Editor auch erweitern lässt. Von Vorteil ist, wenn Plugins zum Beispiel für folgende Funktionen verfügbar sind:

- Auto-Vervollständigen von Tags, Attributen, CSS-Eigenschaften
- Emmet für einfachere Code-Eingabe (gleich mehr dazu)
- Formatieren von HTML-, CSS- und JavaScript-Code
- Lint-Tools: Hinweis auf mögliche Fehler

Wer beim Schreiben von HTML-Code noch etwas Zeit sparen möchte, sollte sich **Emmet**<sup>18</sup> ansehen. Das früher unter dem Namen *Zen Coding* bekannte Tool ist eine Erweiterung für verschiedene Web-Editoren. Im VSCode ist es bereits integriert. Unter anderem erlaubt es eine sehr kompakte Schreibweise:

```
div#page>div.logo+ul#navigation>li*5>a
```

kann durch Tastendruck expandiert werden in

---

<sup>11</sup><https://www.jetbrains.com/webstorm/>

<sup>12</sup><http://www.activestate.com/komodo-edit>

<sup>13</sup><http://notepad-plus-plus.org>

<sup>14</sup><http://www.pspad.com/de/>

<sup>15</sup><http://bluefish.openoffice.nl/index.html>

<sup>16</sup><http://www.vim.org>

<sup>17</sup><http://www.gnu.org/software/emacs/>

<sup>18</sup><http://emmet.io>

```

1 <div id="page">
2   <div class="logo"></div>
3   <ul id="navigation">
4     <li><a href=""></a></li>
5     <li><a href=""></a></li>
6     <li><a href=""></a></li>
7     <li><a href=""></a></li>
8     <li><a href=""></a></li>
9   </ul>
10 </div>

```

Wenn Sie mit einem neuen HTML5-Dokument beginnen möchten, genügt es, `html:5` auf einer leeren Seite einzugeben und das Tastatursymbol zum Expandieren zu drücken. Der Text wird dann ersetzt durch den Rahmen eines HTML5-Dokuments:

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Document</title>
8   </head>
9   <body>
10
11 </body>
12 </html>

```

Die Zeile `<meta http-equiv="...">` wird wegen den verschiedenen Microsoft-Browsern eingefügt, welche verschiedene Kompatibilitätsmodi unterstützen. Wenn Ihre Website auch ältere Internet Explorer unterstützen muss, ist es sinnvoll, diese Meta-Angabe einzufügen und den Kompatibilitätsmodus anzugeben. Wenn nur neuere Browser (IE11 oder Edge) unterstützt werden müssen, kann die Zeile auch gelöscht werden.<sup>19</sup>

Wenn Sie im Team an einer Website arbeiten, kann außerdem die Editor-Erweiterung **EditorConfig** nützlich sein.<sup>20</sup> Diese Erweiterung unterstützt Sie dabei, einheitliche Codierstile und Einstellungen zu verwenden, auch wenn verschiedene Web-Editoren und Entwicklungsumgebungen zum Einsatz kommen. Sobald Entwickler mit ihrem Editor eine Datei des Projekts öffnen, werden die Einstellungen des Editors angepasst, so dass projektspezifische Einstellungen für Tabulatoren, Zeilenendezeichen, etc. verwendet werden.

Um das zu erreichen ist ein Dateiformat für die Einstellungen definiert. Editoren suchen dann beim Öffnen von Dateien nach einer Datei `.editorconfig` im gleichen oder einem übergeordneten Verzeichnis, um die Einstellungen zu laden. Hier ist eine Beispiel-Konfiguration:

---

<sup>19</sup><https://stackoverflow.com/questions/6771258>

<sup>20</sup><https://editorconfig.org>

```

1  # EditorConfig is awesome: http://EditorConfig.org
2
3  # top-most EditorConfig file
4  root = true
5
6  # Unix-style newlines with a newline ending every file
7  [*]
8  end_of_line = lf
9  insert_final_newline = true
10
11 # 4 space indentation
12 [*.py]
13 indent_style = space
14 indent_size = 4
15
16 # Tab indentation (no size specified)
17 [*.*js]
18 indent_style = tab

```

Für einige Editoren gibt es Plugins, um EditorConfig zu unterstützen. Zu diesen gehören VSCode, Atom, Brackets, Emacs, jEdit, Notepad++, Sublime Text, Vim und andere.

**Lint Tools** sind Werkzeuge zur Code-Analyse, die für verschiedene Programmiersprachen verfügbar sind. Auch für CSS gibt es ein solches Tool: *csslint*. Die Überprüfung kann dabei live während der Code-Eingabe im Editor stattfinden oder explizit gestartet werden, zum Beispiel im Zusammenhang mit einem Build-Prozess. Es ist empfehlenswert, Lint-Tools für CSS (*csslint*) und JavaScript (*jslint*, *jshint*) im Editor zu installieren, um möglichst früh auf Fehler oder problematische Code-Teile hingewiesen zu werden.

Abbildung 18 zeigt den Atom-Editor mit installierten Paketen *Linter* und *Linter CSSLint* und eine fehlerhafte CSS-Regel.

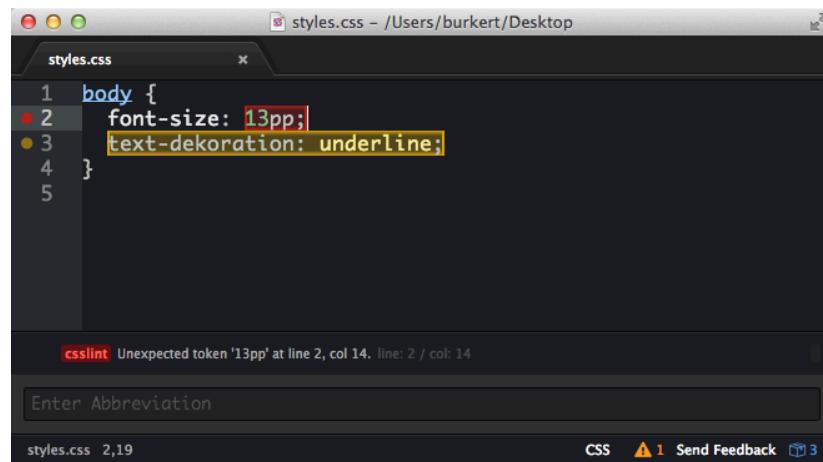


Abbildung 18: Atom Editor mit CSSLint

## Zeichencodierung

Was gelegentlich übersehen wird ist, dass Textdatei nicht gleich Textdatei ist. Das Problem beginnt, sobald der Zeichenvorrat von ASCII nicht mehr ausreicht, also zum Beispiel Umlaute oder andere Sonderzeichen benötigt werden. Ist dies der Fall, kann man nicht mehr von einer einfachen Austauschbarkeit von Textdateien ausgehen – man muss dann auch die Zeichencodierung berücksichtigen.

Wir gehen gleich noch näher auf die verschiedenen Zeichencodierungen ein. Das Ergebnis sei hier schon vorweg genommen: **Verwenden Sie UTF-8**.

- Stellen Sie Ihren (HTML-, CSS-) Editor auf die Zeichencodierung UTF-8 ein.
- Fügen Sie im **head** der HTML-Datei ebenfalls UTF-8 als Zeichencodierung ein:

```
1 <meta charset="utf-8">
```

## ASCII und HTML-Entitäten

**ASCII** steht für *American Standard Code for Information Interchange*, eine Zeichencodierung, die schon ziemlich lang im Einsatz und weit verbreitet ist. ASCII nutzt nur die niederen sieben Bits eines Bytes: Für jedes Zeichen wird somit ein Byte reserviert, aber nur sieben Bit für die Zeichencodierung verwendet. Damit stehen 128 verschiedene Zeichencodes zur Verfügung, von denen einige für Steuerbefehle verwendet werden und nicht für darstellbare Zeichen verfügbar sind (Abbildung 19).

An den Steuerbefehlen kann man teilweise noch die lange Geschichte von ASCII erkennen: BEL diente dazu, die Glocke des Fernschreibers anzuschlagen, und der Zeilenwechsel ist aufgeteilt in CR (Carriage Return, Druckwagen nach links fahren) und LF (Line Feed, Papierwalze um eine Zeile drehen).

Code	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1...	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2...	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3...	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4...	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	^	_	
6...	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Abbildung 19: ASCII Tabelle (Bild: Wikipedia)

Das Problem ist, dass der Zeichenvorrat von ASCII für Webseiten nicht ausreicht, insbesondere wenn die Sprache der Seite nicht englisch ist. Aber neben landesspezifischen Zeichen wird meistens auch noch das eine oder andere Sonderzeichen benötigt.

Zunächst behalf man sich im Webdesign mit dem Einsatz von *HTML-Entitäten*, das sind Zeichensequenzen, die mit einem & beginnen und einem ; enden. Hier ein paar Beispiele:

Entität	Zeichen
&auml;	ä
&euro;	€
&copy;	©

Auch wenn einige Webeditoren die Umsetzung automatisch machen ist der Umgang mit den Entitäten umständlich. Daher verwendet man besser eine andere Zeichencodierung.

### ISO-Codierungen

Da bei ASCII pro Byte noch ein Bit frei wäre, liegt es nahe, dieses achte Bit (Bit 7, wir zählen ja meistens ab 0...) auch noch zu verwenden und somit weitere 128 Zeichencodes zur Verfügung zu haben. Genau das machen die ISO-Codierungen. Wenn Bit 7 auf 0 steht, handelt es sich um ein ASCII-Zeichen, andernfalls um ein Zeichen aus einer erweiterten Zeichentabelle.

Leider ist das Problem damit nicht gelöst, denn die zusätzlichen Zeichencodes reichen wieder nicht aus. Daher musste man verschiedene ISO-Codierungen einführen:

Bezeichnung	Beschreibung
ISO 8859 -1	Latin-1, Westeuropäisch
ISO 8859 -2	Latin-2, Osteuropäisch
ISO 8859 -3	Latin-3, Südeuropäisch
ISO 8859 -4	Latin-4, Baltisch
ISO 8859 -5	Kyrillisch
...	...

Bei Texten, die eine dieser ISO-Codierungen verwenden, muss immer klar sein, welche der Codierungen verwendet wird und das Anzeigeprogramm oder der Texteditor muss korrekt eingestellt sein, um die Texte richtig anzuzeigen.

Bei E-Mails wird daher im Header ebenfalls die Zeichencodierung angegeben, zum Beispiel:

```
Content-Type: text/plain; charset=iso-8859-1
```

Und wenn die Betreff-Zeile (Subject) bereits Zeichen ausserhalb des ASCII-Zeichensatzes verwendet, wird dies direkt in der Zeile angegeben:

```
Subject: =?iso-8859-1?Q?film_=FCber_wiesensteig_und_geislingen?=
```

Ein Nachteil der ISO-Codierungen ist, dass in einem reinen Textdokument nicht Zeichen verschiedener ISO-Codierungen verwendet werden können. Aus diesem Grund werden ISO-Codierungen heute im Web nicht mehr häufig eingesetzt. Abhilfe bieten verschiedene Unicode-Varianten, die zur Repräsentation eines Zeichens mehr als 8 Bit verwenden. Vorherrschend ist heute die Unicode-Variante UTF-8.

## UTF-8

Bei **UTF-8** wird jedem Zeichen eine Bytekette variabler Länge zwischen einem und vier Bytes zugeordnet. Es ist abwärtskompatibel zu ASCII, das heißt Bytes, bei denen Bit 7 auf 0 gesetzt ist, entsprechen der ASCII-Codierung. Je nach Länge der Bytefolge für ein Zeichen stehen bis zu 21 Bit für den Zeichencode zur Verfügung. Damit ergibt sich ein riesiger Zeichenvorrat.

Unicode code points	UTF-8 encoding (binary)
00-7F (7 bits)	0uvwxyz
0080-07FF (11 bits)	110pqrst 10uvwxyz
0800-FFFF (16 bits)	1110jklm 10npqrst 10uvwxyz
010000-10FFFF (21 bits)	11110efg 10hijklm 10npqrst 10uvwxyz

Abbildung 20: UTF-8: Aufbau der Bytefolge

Hier sind einige Beispiele für UTF-8-Zeichen. Das A ist ein ASCII-Zeichen, kann also auch in UTF-8 mit einem Byte dargestellt werden. Hier entspricht UTF-8 also ASCII. Das ä benötigt in UTF-8 zwei Bytes. Das geschwungene Anführungszeichen benötigt bereits drei Bytes.

Zeichen	Hexcode	Binärkode
A	41	01000001
ä	C3 A4	11000011 10100100
„	E2 80 9E	11100010 10000000 10011110

UTF-8 ist natürlich etwas schwieriger zu verarbeiten, da das n-te Byte in einem Dokument nicht mehr automatisch auch das n-te Zeichen ist. Das ist aber kein grosses Problem.

Da heute sowohl Betriebssysteme als auch Texteditoren, Entwicklungsumgebungen und Browser mit UTF-8 umgehen können (was nicht heißt, dass sie alle möglichen Zeichen darstellen können), wird diese Zeichencodierung immer häufiger verwendet.

Um UTF-8 verwenden zu können, müssen Sie auf zwei Dinge achten:

1. Ihr Web-Editor muss so eingestellt sein, dass er die UTF-8-Codierung verwendet.
2. Der Browser muss wissen, dass ein Dokument die UTF-8-Codierung verwendet.

Bei dem Web-Editor können Sie das in den Einstellungen konfigurieren. Für den Browser wird die Zeichencodierung im HTTP-Header mitgeschickt:

`Content-Type: text/html; charset=UTF-8`

Zusätzlich sollten Sie die Zeichencodierung im HTML-Dokument mit Hilfe eines Meta-Elements spezifizieren:

```

1  <!-- aktuelles HTML: -->
2  <meta charset="UTF-8">
3
4  <!-- früher verwendet: -->
5  <meta http-equiv="Content-type" content="text/html; charset=UTF-8">
```

Abschliessend noch zwei Hinweise zu den Zeichencodierungen:

- Welches (ASCII-) Zeichen den Zeilenwechsel repräsentiert, ist nicht immer gleich definiert. Windows verwendet hier die Sequenz aus den zwei Zeichen CR und LF. Unix-Systeme (Linux, MacOS) verwenden nur das LF-Zeichen. Das kann bei manchen Texteditoren zu Problemen führen, wenn Textdokumente zwischen den Plattformen ausgetauscht werden sollen. Aus diesem Grund verfügen Texteditoren meist über eine Möglichkeit, das gewünschte Zeilenendezeichen einzustellen.
- UTF-8-Dateien werden gelegentlich drei Bytes vorangestellt, die als *Byte Order Mark (BOM)* die verwendete Byte-Reihenfolge angeben, was in UTF-8 eigentlich nicht nötig wäre. Diese Bytes führen manchmal zu Problemen, zum Beispiel in PHP-Scripts. Wenn der Texteditor also eine Möglichkeit bietet, BOM zu deaktivieren, sollten Sie das tun.

## Entwickler-Tools im Browser

Sehr hilfreich sind die Entwicklerwerkzeuge, die mittlerweile in fast alle Browser eingebaut sind. Mit diesen sollten Sie sich als Web-Entwickler unbedingt vertraut machen, denn sie sind bei der Web-Entwicklung und der Fehlersuche sehr nützlich. Die Möglichkeiten in den verschiedenen Browsern sind ähnlich. Üblicherweise stehen mindestens folgende Werkzeuge zur Verfügung:

- Struktur des Dokuments analysieren
- CSS-Regeln für ein bestimmtes Element anzeigen/ändern
- Alle für ein Element gültigen CSS-Deklarationen anzeigen
- Abmessungen eines Elements anzeigen
- Gespeicherte Daten für die Seite (DBs, Cookies) analysieren/bearbeiten
- Zugriffszeiten auf die Ressourcen einer Seite analysieren
- JavaScript-Debugger und -Konsole

In diesem Vorkurs und in WBE kommen die Browser-Entwicklerwerkzeuge immer wieder zum Einsatz. Dabei beziehen wir uns normalerweise auf den Firefox-Browser (Abbildung 21). Als Webentwickler sollten Sie sich angewöhnen, diese Tools in die Arbeit zu integrieren, da sie speziell für die Fehlersuche sehr hilfreich sind.

Neben den eingebauten Entwicklerwerkzeugen gibt es für den Firefox eine grosse Sammlung an Erweiterungen, von denen einige für die Webentwicklung geeignet sind. Dazu gehört *Web Developer*<sup>21</sup>, eine Erweiterung, welche zahlreiche nützliche Funktionen zur Verfügung stellt.

---

<sup>21</sup><https://chrispederick.com/work/web-developer/>

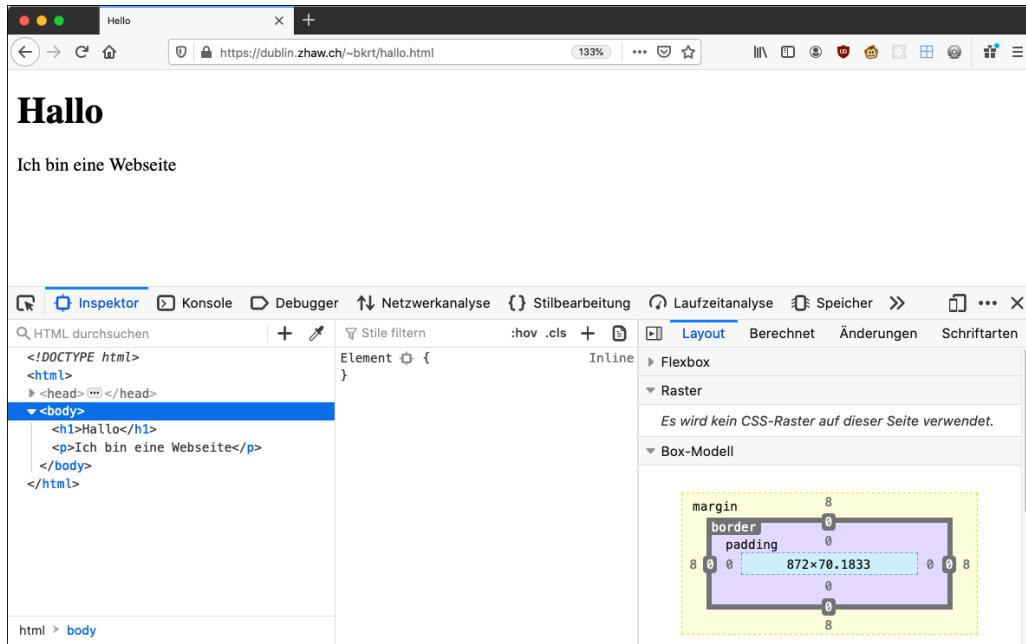


Abbildung 21: Firefox Developer Tools

## Verzeichnisstruktur

Normalerweise werden Sie Ihre Website lokal entwickeln und wenn alles im gewünschten Zustand ist, die benötigten Dateien auf den Server laden (*Deployment*). Bei einer (serverseitig) statischen Website ist das ziemlich einfach: Alle HTML-, CSS- und JavaScript-Dateien müssen zusammen mit den weiteren Teilen (Bilder- und Videodateien zum Beispiel) auf den Server geladen werden. Wenn der Webserver richtig konfiguriert ist und Dateinamen/Verzeichnisstruktur stimmen, sind die Seiten anschliessend im Web sichtbar.

Etwas komplizierter wird es, wenn auch serverseitige Programme und Datenbanken verwendet werden. In diesem Fall müssen die Komponenten auf dem Server konfiguriert und zum Beispiel Datenbanken mit den benötigten Tabellen und initialen Daten eingerichtet werden. Zudem benötigt man auch in der lokalen Testumgebung Webserver und Datenbank. Häufig werden zu diesem Zweck einfach zu installierende Tools wie XAMPP oder MAMP verwendet. Das Deployment wird in diesem Fall aufwändiger. So ist es durchaus möglich, dass in der lokalen Entwicklungsumgebung andere Zugangsinformationen für die Datenbank verwendet werden als auf dem Server oder lokal spezielle Testdaten verwendet werden, die nicht auf den Server übertragen werden sollen.

In diesem Vorkurs werden wir uns auf die einfachere statische Variante beschränken.

Zunächst muss die Verzeichnisstruktur angelegt werden. Sobald Ihre Website mehr als eine Handvoll Dateien umfasst, sollten die einzelnen Dateien in einer übersichtlichen Verzeichnisstruktur angeordnet werden: Bilder in einem eigenen Verzeichnis, alles was nur der

Darstellung dient (CSS-Dateien und spezielle Bilder für die Gestaltung) in einem anderen Verzeichnis, ebenso Scripts (Abbildung 22).

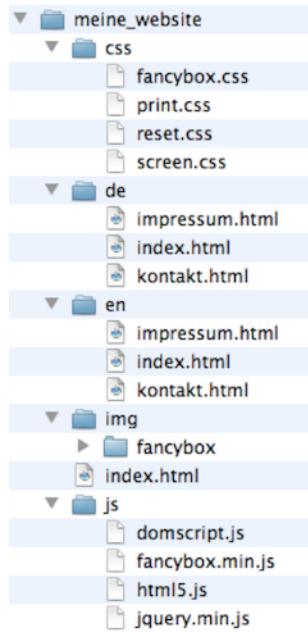


Abbildung 22: Verzeichnisstruktur

Ein Hinweis zu den Dateinamen: Auch wenn Betriebssysteme heute problemlos mit Gross-/Kleinschreibung, Umlauten und anderen Sonderzeichen, sowie Leerzeichen in Dateinamen umgehen können: Die Erfahrung zeigt, dass irgendwann Probleme auftauchen, wenn man diese Möglichkeiten nutzt: Sei es, dass der Upload nicht richtig funktioniert, Sonderzeichen auf dem Server plötzlich verändert sind, oder irgendwelche Scripts Probleme machen, zum Beispiel beim Backup. Befolgen Sie daher den Rat:

**Verwenden Sie nur Kleinbuchstaben oder Ziffern in Dateinamen,  
keine Leerzeichen, keine Sonderzeichen ausser dem Underscore,  
auch keine Umlaute.**

Noch ein weiterer Hinweis zu den Dateinamen. Oben wurde der *Content-Type* eingeführt. Dieser wird neben anderen Angaben im HTTP-Header vom Web-Server zum Browser geschickt. Damit der Webserver den richtigen *Content-Type* liefert, sollten Sie Dateien mit den üblichen Erweiterungen schreiben:

- .html - text/html
- .css - text/css
- .js - application/javascript

Die Startseite heisst normalerweise `index.html`. Das heisst: Wenn in einer URL kein Dateinamen enthalten ist, liefert der Server eine Datei dieses Namens aus, wenn in dem betreffenden Verzeichnis eine solche gefunden wird.

Für grössere Projekte gibt es bereits fertige Vorlagen für Webprojekte, die entweder komplett heruntergeladen oder zunächst nach Bedarf konfiguriert werden können. Einige davon enthalten bereits häufig benötigte CSS- und JavaScript-Bibliotheken, zum Beispiel HTML5 Boilerplate.<sup>22</sup>

## Website bereitstellen

Sie wissen bereits, wie eine Website lokal angelegt und entwickelt wird. Damit sind die Seiten natürlich noch nicht im Internet verfügbar. Nehmen wir an, die Website liegt nun fertig entwickelt auf Ihrem PC oder Notebook vor und Sie haben die Tipps bezüglich Verzeichnisstruktur und Dateinamen berücksichtigt. Nun möchten Sie die Seiten im Internet verfügbar machen. Man nennt diesen Vorgang auch *Deployment*.

Ja nach eingesetzten Technologien haben Sie eine Reihe von Optionen:

- Häufig wird man die Website über einen *Web-Hoster* – auch als *Provider* bezeichnet – ins Internet bringen. Dort steht in der Regel eine ganze Palette von Angeboten zur Verfügung: vom eigenen (virtuellen) Server bis zu virtuellen Hosts, welche man mit anderen Kunden teilt. Auch einen Domainnamen kann man üblicherweise über den Provider registrieren. Je nach Anforderungen wählt man den benötigten Speicherplatz sowie die bereitzustellenden Technologien wie PHP-Unterstützung oder Datenbanken.
- An der ZHAW können Sie Ihre Website auf dem virtuellen Laborserver `dublin.zhaw.ch` bereitstellen. Ihre Website laden Sie einfach ins `www`-Verzeichnis unterhalb Ihres Home-Verzeichnisses. Die Datei `index.html` ist dann über `https://dublin.zhaw.ch/~<kurzzeichen>` verfügbar. Neben PHP können auch CGI-Shell-Scripts und Datenbanken für serverseitige Programme eingesetzt werden.

Wenn Sie selbst einen Webserver installieren, legen Sie unter anderem in der Konfiguration des Servers das Wurzelverzeichnis (*Web-Root*) fest. Unter welcher Adresse die einzelnen Seiten auf dem Server erreichbar sind hängt dann von dieser Web-Root-Angabe ab. Das ist nicht die Wurzel des Dateisystems auf dem Server, sondern das Verzeichnis ab dem die Pfadangabe in der URL beginnt (Abbildung 23).

## Dateitransfer, Secure Shell

Um Dateien auf den Server zu übertragen, können Sie **SFTP** (*SSH File Transfer Protocol*) verwenden, und zwar sowohl auf der Kommandozeile als auch mit grafischen Programmen. Das Kommandozeilenprogramm heisst wenig überraschend `sftp`. Beispiele für Programme mit grafischer Oberfläche sind **FileZilla**<sup>23</sup> (Win, Mac, Linux) und **Cyberduck**<sup>24</sup> (Win, Mac). FileZilla erscheint zunächst etwas gewöhnungsbedürftig, erweist sich aber als robustes Werkzeug für Dateiübertragungen (Abbildung 24).

Auch wenn die grafischen Tools komfortabler zu bedienen sind, sollten Sie sich auch mit dem Kommandozeilenprogramm anfreunden. Dies gilt nicht nur für die Dateiübertragung. Es gibt durchaus Situationen, in denen es von Vorteil ist, Aufgaben direkt auf

---

<sup>22</sup><http://html5boilerplate.com>

<sup>23</sup><https://filezilla-project.org> (FileZilla Client herunterladen)

<sup>24</sup><https://cyberduck.ch>

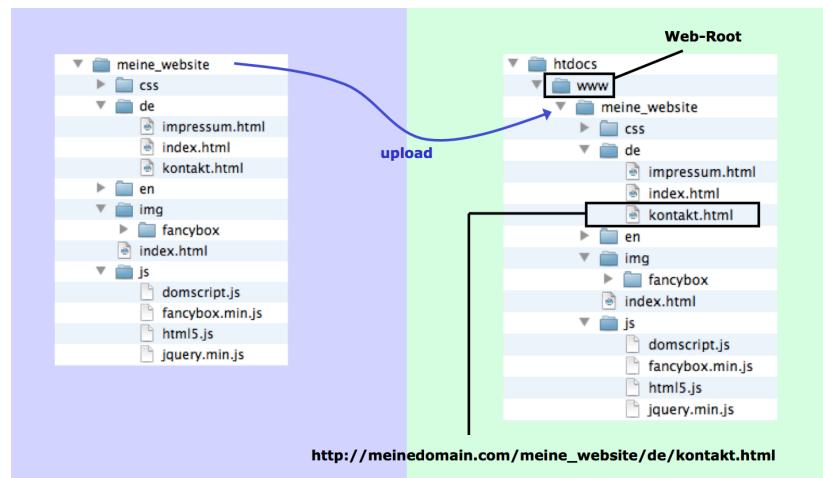


Abbildung 23: Verzeichnisstruktur und Web-Root

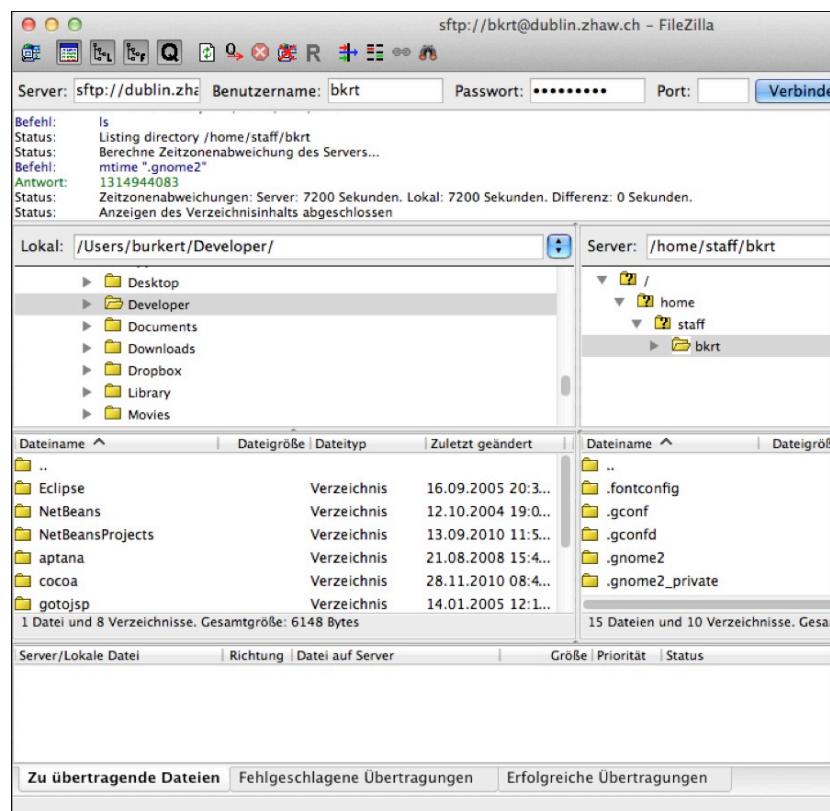


Abbildung 24: FileZilla

dem Server zu erledigen. Sei es, um dort ein paar Dateien zu verschieben, schnell eine Konfigurationsdatei zu bearbeiten, oder die Auslastung des Servers zu überprüfen. Die meisten Webserver laufen auf Unix-Systemen, daher sollte jeder Webdesigner die wichtigsten Kommandozeilen-Befehle von Unix kennen. Dazu gehören:

- Dateien auflisten, verschieben und kopieren (ls, mv, cp)
- Berechtigungen und Eigentümer anpassen (chown, chmod)
- Prozesse und Festplattenbelastung ansehen (ps, top, df)
- Dateitransfer und Verbinden mit anderem Host (sftp, scp, ssh)
- Netzwerkbefehle (ping, traceroute, host, dig)
- Dateien bearbeiten (vi)

Der **vi** gilt als komplizierter Editor. Ein paar wichtige Befehle hat man sich aber schnell angeeignet. Mit etwa einer Handvoll Befehlen kann man bereits eine Datei öffnen, bearbeiten und wieder speichern.

Mit **ssh** lässt sich eine sichere Verbindung zum Server herstellen und dort auf der Kommandozeile arbeiten. Unter Unix und OS-X geben Sie dazu in einem Terminal **ssh** ein, unter Windows können Sie ein Programm wie *Putty* verwenden. Hier ist ein Beispiel, wie mit **ssh** eine Verbindung zum Server *dublin.zhaw.ch* hergestellt werden kann:

```
$ ssh bkrt@dublin.zhaw.ch
bkrt@dublin.zhaw.ch's password:
Last login: Tue Jul 16 13:47:05 2020 from ...
$ ls
ggt.py  ggt.pyc  index.html  ine1  private  public  www
$ cd www
$ mv index.html old.html
$ exit
```

## Content Management Systeme

Content Management Systeme (**CMS**) sind Applikation auf dem Server zum Aufbau und zur Pflege einer Website über den Browser. Anstatt statische Webseiten zu erstellen, werden Seitenvorlagen erstellt, die dann online mit Inhalt gefüllt werden können. Das ist natürlich eine stark vereinfachte Beschreibung eines CMS. Mittlerweile gibt zahlreiche Open Source Systeme (unterschiedlicher Qualität). Weit verbreitet ist **WordPress**<sup>25</sup>, welches ursprünglich für Blogs entwickelt wurde.

---

<sup>25</sup><https://wordpress.org>

## Quellen und Verweise

### Weiterführendes Material

- **HTTP 101** (Weblog *Curiosity is bliss*)  
<http://blog.monstuff.com/archives/000149.html>
- **Architecture of the World Wide Web, Volume One** (W3C)  
<http://www.w3.org/TR/webarch/>
- **The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)**  
(Joel Spolsky)  
<http://www.joelonsoftware.com/articles/Unicode.html>

### Quellenangaben

Ein Teil der Bilder und Inhalte stammt aus früheren Folien der Fächer WBD (Webdesign), INSY (Informationssysteme) und WEBT (Webtechnologien).

# Markup und HTML (Teil 1)

## Einführung und Ziele

In dieser Lektion werden die Grundlagen von HTML behandelt. Neben einem kurzen Überblick über verschiedene Markup-Ansätze wird auf die Entwicklung von HTML – inklusive HTML 4.01 und XHTML – bis hin zur aktuellen HTML-Version eingegangen. Anschliessend wird erklärt, wie ein HTML-Dokument aufgebaut ist und welche Elemente zur Beschreibung von Überschriften, Absätzen, Listen, Zitaten und Tabellen zur Verfügung stehen.

Der wichtigste Abschnitt in diesem Kapitel ist *Aufbau eines HTML-Dokuments*. Die anderen Abschnitte, speziell *Markup-Varianten* und *HTML-Entwicklung* können Sie als fakultativ betrachten, für ein gutes Verständnis des heutigen Stands von HTML sind sie jedoch nützlich.

## Ziele

- Sie verstehen HTML als eine von verschiedenen Auszeichnungssprachen.
- Sie kennen einige wichtige Eckpunkte der Entwicklung von der ersten HTML-Version bis zum aktuellen Stand.
- Sie kennen den grundsätzlichen Aufbau eines HTML-Dokuments.
- Sie können die Elemente im Kopf eines HTML-Dokuments sinnvoll einsetzen.
- Sie kennen die neueren HTML-Elemente zur inhaltlichen Auszeichnung der Grundbestandteile eines Dokuments (`article`, `section`, `nav`, usw.).
- Sie kennen die wichtigsten Elemente zur Auszeichnung von Überschriften, Absätzen, Listen, Zitaten und Tabellen.
- Sie kennen den Unterschied zwischen Block-level- und Inline-Elementen.
- Sie wissen, wie ein HTML-Dokument auf Konformität mit der Spezifikation getestet werden kann.

## Relevanz für WBE

Der Vergleich verschiedener Auszeichnungssprachen sowie die Entwicklung von HTML sind für WBE nicht zwingend erforderlich. Wichtig sind die Abschnitte ab *Aufbau eines HTML-Dokuments*.

## Markup-Varianten

Es gibt zahlreiche verschiedene Dokumentenformate, proprietäre und standardisierte, Binärformate und Textformate, solche, die einfach in einem Editor bearbeitet werden können und solche, die nur von bestimmten Programmen verarbeitet werden.

Proprietäre *Binärformate* sind zum Beispiel Photoshop- (Dateien enden üblicherweise mit .psd) und Word-Dokumente (in der älteren Variante mit der Erweiterung .doc). Ein standardisiertes Binärformat verwenden etwa Open-Office-Dokumente (Erweiterung .odt für Dateien, die von der Open Office Textverarbeitung angelegt wurden). Open-Office-Dokumente sind jedoch nur Binärdokumente, weil sie aus mehreren Dateien bestehen, die als ZIP-Archiv zusammengefasst und komprimiert werden. Intern enthalten sie unter anderem XML-Dokumente (mehr zu XML s.u.).

Zu den *Textformaten* gehört **RTF** (Rich Text Format), allerdings ist es nicht dazu geeignet, in einem beliebigen Texteditor bearbeitet zu werden. Vielmehr ist es ein (relativ beschränktes) Austauschformat zwischen Textverarbeitungsprogrammen. Es beschreibt im wesentlichen die Darstellung der einzelnen Textteile. RTF wurde von 1987 von Microsoft eingeführt.

```
\rtf1\ansi\ansicpg1252\cocoartf1138
{\fonttbl\f0\fnil\fcharset0 Verdana;}
{\colortbl;\red255\green255\blue255;}
\paperw11900\paperh16840\margl1440\margr1440\vieww10800\viewh8400\viewkind0
\deftab720
\pard\pardeftab720\li540\fi-540

\f0\fs40 \cf0 Fr\'fche elektronische Dokumentenformate legten vor
allem die Darstellung fest\
\pard\pardeftab720\li1170\fi-450
\cf0 T
\fs56 \sub E
\fs40 \nosupersub X, troff\
\pard\pardeftab720\li540\fi-540
\cf0 Generic Coding: Beschreibende Tags anstelle von Formatanweisungen\
\pard\pardeftab720\li1170\fi-450
\cf0 LaTeX\
Inhalt und Struktur eines Dokuments werden von der Darstellung getrennt
\fs36 \
}
```

Ebenfalls zur Markierung bestimmter Textteile für eine bestimmte Darstellung dienen die Auszeichnungssprachen **TeX** (Donald E. Knuth, ab 1977 entwickelt) und **Troff**. Beide verfolgen das Ziel, qualitativ hochwertigen Textsatz zu ermöglichen. Troff wird in Unix-systemen für die Formatierung der Online-Dokumentationen verwendet (man-Pages).

Was versteht man unter einer *Auszeichnungssprache (Markup Language)*? Hier die Wikipedia-Definition:

A (document) markup language is a modern system for annotating a document in a way that is syntactically distinguishable from the text. The idea and terminology evolved from the “marking up” of paper manuscripts, i.e., the revision instructions by editors, traditionally written with a blue pencil on authors’ manuscripts. In digital media this “blue pencil instruction text” was replaced by tags, that is, instructions are expressed directly by tags or “instruction text encapsulated by tags”.

Mit der Zeit stellte man fest, dass das Auszeichnen der Texte mit Formatanweisungen für viele Zwecke mit verschiedenen Nachteilen verbunden ist. Es werden zum Beispiel immer wieder die gleichen Formate benötigt, etwa um Überschriften auszuzeichnen, die ja möglichst einheitlich aussehen sollen. Wenn nun jede Überschrift mit einer bestimmten Schriftgrösse und Schriftart ausgezeichnet wird, ist immer noch nicht bekannt, dass es sich um eine Überschrift handelt. Genau diese Information könnte jedoch ebenfalls genutzt werden, zum Beispiel beim Generieren von Inhaltsverzeichnissen.

Es ist daher besser, mit Hilfe der Auszeichnungssprache zu beschreiben, was mit den Textteilen gemeint ist und das Problem der Darstellung separat zu lösen (*Generic Coding*). Also: Die Auszeichnungssprache beschreibt Inhalt und Struktur eines Dokuments und die Darstellung wird separat beschrieben. Zum Darstellungsaspekt gehören Dinge wie:

- Art der Kapitel- und Abschnittnummierung
- Schriftarten, Ränder, Abstände, ...
- Automatisches Erzeugen eines Inhaltsverzeichnisses
- Sollen Kapitel jeweils auf neuen Seiten beginnen?
- Wo kommen die Fussnoten hin?

Ein Beispiel für eine solche auf die Bedeutung der Elemente fokussierende Sprache ist **LaTeX** (Leslie Lamport, Anfang der 1980er Jahre). Es basiert auf TeX und verwendet Kommandos mit beginnendem Backslash für die Auszeichnung.

```

1 \documentclass[12pt]{article}
2 \usepackage{amsmath}
3 \title{\LaTeX}
4 \date{}
5 \begin{document}
6   \maketitle
7   \LaTeX{} is a document preparation system for the \TeX{}
8   typesetting program. It offers programmable desktop publishing
9   features and extensive facilities for automating most aspects of
10  typesetting and desktop publishing, including numbering and
11  cross-referencing, tables and figures, page layout, bibliographies,
12  and much more. \LaTeX{} was originally written in 1984 by Leslie
13  Lamport and has become the dominant method for using \TeX; few
14  people write in plain \TeX{} anymore. The current version is
15  \LaTeXe.
16
17  % This is a comment; it will not be shown in the final output.

```

```

18  % The following shows a little of the typesetting power of LaTeX:
19  \begin{align}
20    E &= mc^2 \\ 
21    m &= \frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}}
22  \end{align}
23  \end{document}

```

Abbildung 25 zeigt die für dieses Dokument generierte Ausgabe:



ETEX is a document preparation system for the TeX typesetting program. It offers programmable desktop publishing features and extensive facilities for automating most aspects of typesetting and desktop publishing, including numbering and cross-referencing, tables and figures, page layout, bibliographies, and much more. L<sup>E</sup>T<sub>E</sub>X was originally written in 1984 by Leslie Lamport and has become the dominant method for using TeX; few people write in plain TeX anymore. The current version is L<sup>E</sup>T<sub>E</sub>X<sup>2ε</sup>.

$$E = mc^2 \tag{1}$$

$$m = \frac{m_0}{\sqrt{1 - \frac{v^2}{c^2}}} \tag{2}$$

Abbildung 25: LaTeX-Ausgabe

LaTeX produziert sauber gesetzte Seiten, bei Bedarf in Postscript oder PDF. Es eignet sich auch für sehr grosse Dokumente und erzeugt schön gestaltete Formeln. Außerdem ist es erweiterbar und für verschiedenste Zwecke anpassbar.

LaTeX hätte sich vielleicht als Grundlage für das Web geeignet, aber Tim Berners-Lee hat SGML als Grundlage vorgezogen. SGML steht für *Standard Generalized Markup Language* und basiert auf GML, das bereits in den 1960er Jahren entwickelt wurde (IBM).

SGML ist eine Markup-Variante, die die Bedeutung der Textteile mittels *Tags* und *Attributen* beschreibt, wie Sie das bereits von HTML kennen. SGML bestimmt aber keinen bestimmten Satz von Tags, sondern nur, wie Tags und Attribute aussehen und dass sie hierarchisch angeordnet sein müssen. SGML wird daher auch als *Metasprache* bezeichnet.

```

1  <book>
2      <author>Erik Wilde<author>
3      <heading>Wilde's WWW</heading>
4      <chapter> ...
5          <chapter>
6              <heading>SGML</heading>
7              <paragraph> ...
8              <section>
9                  <heading>Content and Presentation</heading>
10                 <paragraph> ...
11                 <figure> ...
12             </chapter>
13         </book>

```

Zu einer Sprache wird es, wenn noch die Menge der erlaubten Tags, deren Bedeutung, mögliche Attribute und die Kombinationsmöglichkeiten festgelegt werden. Diese Vorgaben werden formal in einer **DTD** (*Document Type Definition*) beschrieben. Na ja, ausser der Bedeutung, die kann auch in der DTD nur in Form von Kommentaren angegeben werden.

**HTML** ist also **SGML** mit einer **DTD**, zumindest bis HTML 4.01, für das es sogar drei DTDs gibt. Mehr dazu im nächsten Abschnitt.

Zum Abschluss der Einführung ins Markup sehen wir noch kurz eine andere Art von Auszeichnung an, die mit relativ wenig Markup auskommt und **Markdown** genannt wird. Ein Markdown-Dokument besteht aus reinem Text, der nach ein paar Regeln formatiert ist. Außerdem werden nur wenige Auszeichnungselemente verwendet. Markdown-Dokumente sind also im Texteditor sehr gut lesbar, können aber auch leicht in schön formatierte HTML- oder PDF-Dokumente konvertiert werden.

A First Level Header

=====

A Second Level Header

-----

Now is the time for all good men to come to  
the aid of their country. This is just a  
regular paragraph.

The quick brown fox jumped over the lazy  
dog's back.

### Header 3

```

> This is a blockquote.
>
> This is the second paragraph in the
> blockquote.

```

Some of these words \*are emphasized\*.

- \* Candy.
- \* Gum.
- \* Booze.

Markdown wurde von John Gruber und Aaron Swartz entworfen, basierend auf den Ideen zu Setext (Ian Feldman, 1992). Mittlerweile gibt es zahlreiche Tools (auch Konverter in JavaScript) und Editoren für Markdown. Auch gibt es eine Reihe von Vorschlägen für die Erweiterung von Markdown. Die Beschreibung der Syntax finden Sie hier:

<http://daringfireball.net/projects/markdown/>

Im Webdesign spielt Markdown eine Rolle, weil es als übersichtliche Eingabemöglichkeit in Content Management Systemen oder Blogs verwendet werden kann. Auch dieses Dokument ist übrigens in Markdown geschrieben, ebenso wie die in WBE verwendeten Folien. Bereits seit einigen Jahren bin ich ein Anhänger dieser Notation. Als es noch Setext hiess und die Auszeichnungselemente ein wenig anders ausgesehen haben, habe ich als eines meiner ersten C-Programme zur Übung einen Setext-nach-HTML-Konverter geschrieben, da war HTML noch ziemlich neu. Auf jeden Fall dürfte der Konverter kein Meisterwerk der Software-Entwicklung gewesen sein :-)

```
-----  
// Datei:      etx2html.c  
// Beschreibung: Setext nach HTML Konverter  
// Autor:       Gerrit Burkert / inbe / CH/INSE  
// Umgebung:   Entwickelt auf: HPUX 9.0  
//  
// Bemerkungen:  
//  
// Modifikationsgeschichte:  
// 00.10 22.09.1995 Gerrit Burkert / inbe / CH/INSE  
// ...
```

## HTML-Entwicklung

Hier ist etwas Geschichte unerlässlich: Um den aktuellen Stand im Bereich der Web-Technologien zu verstehen, ist es hilfreich, die bisherige Entwicklung zu kennen.

Wie bereits bekannt wurde die erste Version von **HTML** von Tim Berners-Lee am CERN in Genf entwickelt. Das war 1989. Zum Stand dieser ersten HTML-Version:

- An SGML angelehnt
- Sehr einfach: Absatz, Überschriften, Listen
- Hypertext Links
- Noch kein `img`-Element

Die Sprache hiess damals einfach HTML, ohne Versionsnummer. Erst später wurde diese Version als **HTML 1** bezeichnet. Dann folgten einige HTML-Versionen mit Versionsnummer. Erstaunlicherweise hat die heutige Version von HTML wieder keine Versionsnummer mehr, doch dazu später mehr.

Zwischen 1993 und 1997 wurde HTML über verschiedene Versionen weiterentwickelt: **HTML+**, **HTML 2.0**, **HTML 3.0**, **HTML 3.2**. Es gab aber keinen wirklichen Standard in dieser Phase, denn die Browser entwickelten sich schnell weiter, vor allem Netscape (mit dem gleichnamigen Browser) und Microsoft (mit dem Internet Explorer) lieferten sich ein Rennen, wer schneller neue Features in den Browser einbauen konnte. HTML war, was die Browser verarbeiten konnten.

Erst mit **HTML 3.2** wurde wieder ein etwas stabilerer Zustand erreicht. Denn 1995 wurde das World Wide Web Consortium (W3C) zur Standardisierung von HTML und anderer Web-Techniken gegründet. HTML 3.2 war die erste Version, die als *W3C Recommendation* veröffentlicht wurde (1997).

Noch im gleichen Jahr (1997) wurde **HTML 4.0** veröffentlicht: Darstellung ausgelagert in Stylesheets (CSS), Tabellen mit mehr Möglichkeiten, Frames. Dies war, abgesehen von einer kleinen Aktualisierung 1999 zu **HTML 4.01**, für einige Jahre Stand der Technik im Webdesign. HTML 4 hat eine formale Grundlage in SGML und drei DTDs spezifizieren präzise, wie ein HTML-4-Dokument aufgebaut sein muss. Auf HTML 4 gehen wir in einem eigenen Abschnitt noch etwas ausführlicher ein.

Da SGML schon relativ alt und in vieler Hinsicht nicht mehr zeitgemäß war, wurde **XML** entwickelt, die *Extensible Markup Language*, eine modernere Weiterentwicklung von SGML, die besser an den Einsatz im Web angepasst ist. Merkmale: Strengere Syntax, sowie ein paar Einschränkungen in den Möglichkeiten der DTD. XML ist damit einfacher zu verarbeiten als SGML. Aber auch XML ist zunächst nur eine Meta-Sprache, die erst durch Definition der Tags, Attribute und Kombinationsmöglichkeiten zu einer richtigen Sprache wird. XML erlaubt auch Namespaces, um verschiedene solcher Sprachen kombinieren zu können.

Das W3C hat nun versucht, XML als neue Basis für HTML zu etablieren. Noch vor HTML 4.01 wurde **XHTML 1.0** veröffentlicht, das vom Sprachumfang weitgehend HTML 4 entsprach, aber auf XML aufbaute. Webseiten wurden in der Folge häufig mit XHTML-1.0-Syntax geschrieben, für die Verarbeitung im Browser wurde aber der fehlertolerante HTML-4-Parser verwendet.

In **XHTML 1.1** hat das W3C dann veraltete Elemente entfernt und der Inhaltstyp `application/xhtml+xml` wurde obligatorisch. Damit müssen die Dokumente korrektweise als XML-Dokumente verarbeitet werden und mit der Fehlertoleranz war es vorbei. Ein Markupfehler führt dann im Browser zu einer Fehlermeldung und nicht zur Anzeige des gewünschten Dokuments. Konsequenz: XHTML 1.1 wurde praktisch nicht benutzt, die Webdesigner setzten weiterhin auf HTML 4 oder als HTML verarbeitetes XHTML.

Das W3C begann, sich von den Bedürfnissen der Web-Entwickler und Browser-Hersteller zu entfernen. Es war bereits die nächste Version in Arbeit, **XHTML 2.0**, das grösstere Änderungen mit sich bringen sollte. Die Kompatibilität mit Vorgängerversionen wurde

dafür aufgegeben. XHTML 2.0 sollte eine sauber definierte Auszeichnungssprache sein, um Dokumente für die verschiedensten Zwecke im Web zu repräsentieren.

Nach und nach nahm die Kritik am W3C zu: Ihre aktuellen Arbeiten seien zu abgehoben, zu wenig an der Realität orientiert. Ebenso führte der Fokus auf Dokumente zu Kritik: Die Arbeiten des W3C konzentrierten sich vor allem auf die Beschreibung von Web-Dokumenten, obwohl Web-Applikationen bereits grosse Bedeutung hatten. Gefordert wurde ein pragmatischer und weniger theoretischer Ansatz.

Aus diesen Gründen riefen Vertreter der Browser-Hersteller Mozilla, Opera und Apple die *Web Hypertext Application Technology Working Group (WHATWG)* ins Leben. Später stiess auch Microsoft dazu. Diese Gruppe begann, eigene Spezifikationen zu entwickeln, zunächst mit der Bezeichnung:

- Web Applications 1.0
- Web Forms 2.0

Die Arbeiten fanden grosses Interesse bei den Web-Entwicklern und es war klar, dass die Browserhersteller nach und nach die beschriebenen Funktionen in ihre Browser integrieren würden. Das W3C lenkte schliesslich ein, beendete die Arbeit an XHTML 2.0 und akzeptierte die Entwürfe der WHATWG als Basis einer neuen HTML-Version: **HTML5**. HTML5 ist deutlich mehr als nur eine Auszeichnungssprache. Es ist eher eine Standardfamilie mit Ergänzungen zu HTML und CSS, sowie verschiedenen JavaScript APIs.

Die WHATWG hat die Versionsnummer mittlerweile aus dem Namen entfernt und nennt die Spezifikation **HTML – Living Standard**. Zumindest was die Versionsnummern angeht sind wir damit wieder am Anfang.

## HTML 4.01

**HTML 4** basierte auf **SGML**, einem etwas betagten aber sehr flexiblen Auszeichnungsformalismus. SGML erlaubt im Prinzip sogar, das Aussehen der Tags nach Bedarf anzupassen und statt `<title>` also zum Beispiel `<<title>>` zu schreiben. Wie Tags auszusehen haben, wird in der *SGML Declaration* beschrieben. Standard (genannt: *Reference Concrete Syntax*) sind aber Tags, wie wir sie von HTML kennen:

```
1  <!-- Element mit Inhalt: -->
2  <element attr1="value1" attr2="value2">
3      Inhalt
4  </element>
5
6  <!-- Leeres Element -->
7  <element attr1="value1" attr2="value2">
```

In der SGML Declaration wird auch angegeben, ob End-Tags weggelassen werden können, wenn aus dem Kontext klar ist, wo sie stehen, ob Attribute in abgekürzter Form geschrieben werden können, und ähnliche Dinge.

In HTML wie auch in SGML wird Text, der zwischen `<!--` und `-->` auftaucht, als **Kommentar** aufgefasst. Kommentare sind nur im Quellcode sichtbar, sie dienen daher zur Erklärung des Quellcodes.

Versuchen Sie, die Bezeichnungen **Tag** und **Element** auseinanderzuhalten. Ein *Tag* hat vorne und hinten spitze Klammern. Es dient dazu, ein *Element* in ein Dokument einzuführen, das *Element* umfasst den Bereich von *Start-Tag*, Inhalt und *End-Tag*. Man spricht aber auch allgemein von einem *Element*, wenn alle Elemente mit einem bestimmten Namen gemeint sind (müsste vielleicht genauer als *Element-Typ* bezeichnet werden): Ein **img**-Element kann **width-** und **height-Attribute** haben.

Ein **Attribut** besteht aus *Attributname* und *Attributwert*. Natürlich kann ein Element mehr oder weniger als die oben gezeigten zwei Attribute haben. Attribute dienen zur näheren Beschreibung eines Elements.

SGML lässt einige Freiheitsgrade: Attributwerte müssen nicht unbedingt in Anführungszeichen stehen; boolesche Attribute brauchen gar keinen Wert: Es zählt dann einfach das Vorhandensein – oder Nichtvorhandensein – des Attributnamens. Das ist natürlich eine problematische Regelung, denn Attribute deren Defaultwert *true* ist, müssen dann anders behandelt werden.

```

1  <!-- in HTML möglich, in XML/XHTML nicht: -->
2  <li id="eins">Erste Zeile
3  <li id=zwei>Zweite Zeile</li>
4  <input checked>
```

Es wurde bereits erwähnt, dass SGML eine *Meta-Sprache* ist. Eine konkrete Sprache wie HTML wird es erst, wenn die erlaubten Elemente, Attribute und Kombinationsmöglichkeiten beschrieben werden. Das wird mit einer **DTD** (*Document Type Definition*) gemacht. DTDs haben eine spezielle Syntax:

```

1  <!-- %head.misc; defined earlier on as "SCRIPT/STYLE/META/LINK/OBJECT" -->
2  <!ENTITY % head.content "TITLE & BASE?">
3
4  <!ELEMENT HEAD 0 0 (%head.content;) +(%head.misc;) -- document head -->
5  <!ATTLIST HEAD
6      %i18n;                      -- lang, dir --
7      profile        %URI;          #IMPLIED -- named dictionary of meta info --
8      >
```

**ENTITY** beschreibt Abkürzungen, die im weiteren Verlauf der DTD genutzt werden können. Eine andere Art von Entitäten kann auch im SGML-/HTML-Dokument genutzt werden. **ELEMENT** beschreibt die erlaubten Elemente, ob Start- und/oder End-Tag obligatorisch sind, und welchen Inhalt sie haben können (und welchen nicht). **ATTLIST** schliesslich spezifiziert die möglichen Attribute eines Elements, ihre Wertebereiche, mögliche Defaultwerte und ob ein Attribut obligatorisch ist oder nicht.

Für HTML 4 wurden vom W3C drei DTDs angegeben:

- **strict**: Gewünschte Form von HTML
- **transitional**: Zur Kompatibilität mit früheren Versionen, enthält als veraltet angelehnte Elemente
- **frameset**: Für Frames

Welche DTD verwendet wird, wird zu Beginn eines Dokuments in der Doctype-Deklaration angegeben, zum Beispiel:

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
2   "http://www.w3.org/TR/html4/strict.dtd">
```

Anhand der DTD kann von einem Programm überprüft werden, ob ein Dokument ein korrektes HTML-4-Dokument ist. Ein solches Programm nennt man **Validator**. Das W3C stellt einen Validator als Online-Service zur Verfügung. Ein Dokument, das die elementaren Syntaxregeln von SGML erfüllt, in dem also etwa Tags korrekt notiert und Elemente korrekt verschachtelt sind, nennt man **wohlgeformt** (*well-formed*). Ein wohlgeformtes Dokument, das auch die Spezifikation der DTD erfüllt, nennt man **gültig** (*valid*).

Hier noch ein paar Beispiele aus der HTML-4.01-strict-DTD:

```
1 <!ELEMENT OL -- (LI)*          -- ordered list --
2 <!ELEMENT LI - O (%flow;)*    -- list item --
3 <!ELEMENT A -- (%inline;)* -(A) -- anchor --
4 <!ELEMENT TITLE -- (#PCDATA) -(%head.misc;) -- title --
5 <!ELEMENT META - O EMPTY      -- generic metainformation -->
```

Ein **ol**-Element (geordnete Liste) kann ein oder mehrere **li**-Elemente (list items) enthalten. Start- und Endtag von **ol** sind obligatorisch (ausgedrückt durch die beiden **-**). Beim **li**-Element ist das End-Tag optional. Das **a**-Element (Verweis) enthält Inline-Content, also Text oder Elemente, die im Textfluss vorkommen können, wie Bilder. Ferner ist angegeben, dass **a**-Elemente nicht weitere **a**-Elemente enthalten können (durch die Exclusion **-(A)**). Das **title**-Element enthält einfach Text (parsed character data). Und **meta** ist ein leeres Element, das durch seine Attribute genauer charakterisiert wird.

Die Attribute eines Elements werden mit einem ATTLIST-Eintrag spezifiziert:

```
1 <!ATTLIST FORM
2   method      (GET|POST)      GET      -- HTTP method ... --
3   name       CDATA          #IMPLIED -- name of form ... --
4 ...
5 >
```

Das Element **form** kann ein Attribut **method** haben mit zwei möglichen Werten: **GET** oder **POST**. Wird das Attribut nicht angegeben, gilt der Default-Wert **GET**. Es kann ferner ein **name**-Attribut haben, das Text enthält (**CDATA**). Dieses Attribut muss nicht angegeben werden (**#IMPLIED**, sonst stünde hier **#REQUIRED**).

**Muss man als Webdesigner eine SGML-DTD lesen können?** Bis vor kurzem, also so lange man HTML 4 eingesetzt hat, hätte man die Frage mit einem klaren *Ja* beantwortet. Anders war es kaum möglich, korrekte HTML-Dokumente zu schreiben und die Fehlermeldungen des Validators zu verstehen. Ausserdem kann in einer DTD sehr viel schneller nachgeschlagen werden, wie ein Element eingesetzt werden kann, als dies in einer Dokumentation in natürlicher Sprache möglich wäre.

Im Zuge der Entwicklung von HTML5 verlor die DTD an Bedeutung. Allerdings werden DTDs auch in XML eingesetzt und XML ist nach wie vor wichtig. Grundkenntnisse zu

DTDs sind also vorteilhaft, insbesondere wenn man die Entwicklung von HTML verstehen möchte.

Schön beschrieben ist der Einsatz von SGML zur Beschreibung von HTML im Kapitel 3 *Über SGML und HTML* in der deutschsprachigen Übersetzung der HTML-4.01-Spezifikation.<sup>26</sup>

## XML und XHTML

**XML**, die *Extensible Markup Language* wurde 1996 in einem Working Draft veröffentlicht und zwei Jahre später eine W3C Recommendation. XML ist eine Vereinfachung von SGML im Hinblick auf den Einsatz im Internet. Wie SGML ist es eine Metasprache, das heißt Grundlage für anwendungsspezifische Auszeichnungssprachen, *XML-Anwendungen* genannt. Beispiele für solche Anwendungen sind **XHTML**, **SVG** (Scalable Vector Graphics) oder **MathML** (Mathematical Markup Language). XML wird heute aber in den verschiedensten Bereichen verwendet, etwa als Datenaustauschformat, für Konfigurationsdateien, als Dateiformat für Textverarbeitungsdateien oder andere Anwendungen.

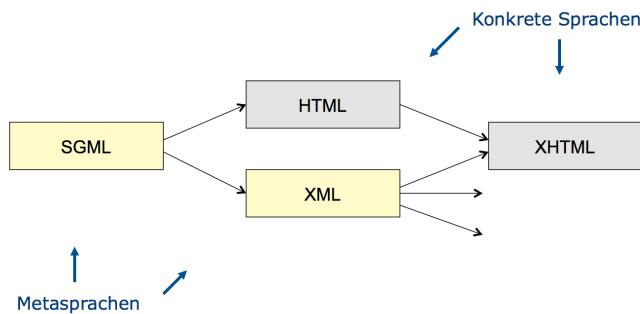


Abbildung 26: SGML, HTML, XML und XHTML

XML enthält im Vergleich zu SGML weniger Freiheitsgrade: in XML dürfen keine Tags weggelassen oder Attribute ohne Anführungszeichen geschrieben werden. Auch ist die XML-DTD etwas einfacher aufgebaut als eine SGML-DTD. XML ist daher einfacher zu verarbeiten als SGML.

Auf der Basis von XML lassen sich spezifische Sprachen definieren. Die Grammatik der Sprache lässt sich mit Hilfe einer DTD spezifizieren. Die *Bedeutung* der Sprachelemente kann aber nur in natürlicher Sprache angeben werden.

Hier ist eine XML-Datei für eine Liste von Personen (`peoplelist`):

<sup>26</sup><http://www.edition-w3.de/TR/1999/REC-html401-19991224/intro/sgmltut.html>

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE peoplelist SYSTEM "peoplelist.dtd">
3
4  <peoplelist>
5      <person member="false">
6          <name>Fred Bloggs</name>
7          <birthdate>2008-11-27</birthdate>
8          <gender>Male</gender>
9      </person>
10     <person person member="true">
11         <name>Maria Muster</name>
12         <birthdate>2009-3-14</birthdate>
13         <gender>Female</gender>
14     </person>
15 </peoplelist>
```

XML-Dateien können am Anfang eine XML-Deklaration enthalten, die Version und Zeichencodierung angibt. Die DTD zu dieser XML-Datei könnte folgendermassen aufgebaut sein:

```

1  <!ELEMENT peoplelist (person)*>
2  <!ELEMENT person (name, birthdate?, gender?, socialsecuritynumber?)>
3  <!ELEMENT name (#PCDATA)>
4  <!ELEMENT birthdate (#PCDATA)>
5  <!ELEMENT gender (#PCDATA)>
6  <!ELEMENT socialsecuritynumber (#PCDATA)>
7
8  <!ATTLIST person
9      member          (true|false)      #REQUIRED
10     active           CDATA            #IMPLIED
11 >
```

In XML-DTDs gibt es keine Angabe, ob einzelne Tags optional sind und auch keine Inclusions oder Exclusions.

Neben DTDs gibt es für XML noch weitere Möglichkeiten, die Grammatik einer Sprache zu definieren, zum Beispiel **XML-Schema** (genauer: *XML Schema Definition*), in Dateien mit der Endung .xsd abgelegt. Solche Schema-Definitionen sind selbst XML-Dateien und verfügen über mehr Ausdrucksmöglichkeiten als DTDs.

Die *Darstellung* von XML-Dateien kann wie bei HTML mit Hilfe von CSS angegeben werden. CSS ist jedoch stark auf HTML zugeschnitten und zu eingeschränkt für beliebige XML-Dateien. Daher wurde als leistungsfähigere Variante **XSL** (Extensible Style-sheet Language) entwickelt, eine Familie von Sprachen zur Beschreibung der Darstellung von XML-Dokumenten. Dazu gehören **XSL-FO** (XSL-Formatting Objects) für XML-Dokumente, die Formatierungsanweisungen und Stilangaben enthalten, und **XSLT** (XSL Transformations) für die Transformation eines XML-Dokuments in eine andere XML-Sprache (zum Beispiel SVG oder XSL-FO).

**XHTML 1.0** war eine Neuformulierung von HTML 4 in XML, auch hier gab es die Varianten *strict*, *transitional*, *frameset*. Im Gegensatz zu HTML 4 waren die Regeln für wohlgeformte Dokumente aber strikter, außerdem mussten Elemente und Attribute klein geschrieben werden (in HTML 4 war Gross- oder Kleinschreibung möglich) und leere Elemente waren in XML-Notation zu schreiben, also etwa <br /> statt <br>.

Hier ein Beispiel für ein XHTML-1.0-Dokument:

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <html xmlns="http://www.w3.org/1999/xhtml"
6   xml:lang="de" lang="de">
7   <head>
8     <title>Mein erstes XHTML-Dokument</title>
9   </head>
10
11  <body>
12    <p>Hallo Welt!</p>
13  </body>
14 </html>
```

XHTML hatte durchaus einige Vorteile gegenüber HTML. So gab es einen definierten Mechanismus, es mit anderen Dokumenttypen wie SVG oder MathML zu kombinieren. Mit der weiteren Entwicklung in Richtung XHTML 1.1 und 2.0 verlor das W3C aber die Bodenhaftung und damit auch die Unterstützung der Entwicklergemeinde, was zur bereits beschriebenen Gründung der WHATWG führte. Das Pendel schlug nun in der anderen Richtung aus (viele Webentwickler meinen: zu weit) und statt dem formalen aber etwas komplizierten wurde ein pragmatischer Ansatz gewählt. Die HTML-Spezifikation ist dadurch aber nicht einfacher geworden, im Gegenteil.

## HTML - Living Standard

In den vorangehenden Abschnitten haben wir gesehen, dass es eine Reihe von verschiedenen HTML- und XHTML-Versionen gab und die Bezeichnungen nicht immer ganz eindeutig waren. So verwendete die WHATWG zunächst die Bezeichnung **HTML5** (ohne Leerzeichen) und spricht nun von **HTML - Living Standard**. Das W3C lehnt an diese Spezifikation ihre Drafts an, bezeichnet als **HTML5** und später **HTML 5.1** bis **HTML 5.3** (mit Leerzeichen).

Mittlerweile hat das W3C die eigenen HTML-Spezifikationen aufgegeben und verweist auf das Dokument **HTML Living Standard** der WHATWG:  
<https://html.spec.whatwg.org/multipage/>

In den folgenden Abschnitten und Kapiteln verwenden wir die Bezeichnungen **HTML4** für die W3C-Spezifikation 4.01, **HTML5** zur expliziten Bezeichnung der Weiterentwicklung von HTML nach **HTML4**, und einfach **HTML** für die aktuelle Version des **Living Standard**.

## Aufbau eines HTML-Dokuments

HTML ist nicht sehr kompliziert aufgebaut. Etwas problematisch ist vielleicht, die Elemente und Attribute entsprechend der eigentlich beabsichtigten Bedeutung einzusetzen. Hier ist noch einmal das bereits gezeigte Beispiel:

```
1  <!DOCTYPE html>
2  <html lang="de">
3      <head>
4          <title>Mein zweites HTML-Dokument</title>
5          <meta charset="utf-8" />
6          <link rel="stylesheet" href="styles/screen.css">
7      </head>
8
9      <body>
10         <section>
11             <header>
12                 
13             </header>
14
15             <article>
16                 <h1>Webseiten</h1>
17
18                 <p>Die Grundidee ist ziemlich einfach:</p>
19
20                 <p>Das <strong>World Wide Web</strong> (heute meist kurz: das Web)
21                     basiert auf einer <em>Client-Server-Architektur</em>. Der Client
22                     ist meist ein grafischer <em>Browser</em>. Er zeigt die Webseiten
23                     an, die über das Internet von einem <em>Webserver</em> geladen
24                     werden. Zu diesem Zweck wird im Browser eine Adresse eingegeben,
25                     die eine bestimmte Ressource im Web anfordert. Das angezeigte
26                     Dokument kann Verweise enthalten, die zu weiteren Ressourcen im
27                     Web führen.</p>
28
29             </article>
30         </section>
31     </body>
32 </html>
```

## Kopf eines HTML-Dokuments

Die Deklaration des Dokumenttyps `<!DOCTYPE html>` gibt an, dass es sich um ein HTML-Dokument nach der aktuellen Spezifikation handelt. Die Doctype-Angaben von HTML4 und XHTML waren wesentlich komplizierter.

Das Wurzelement des Dokuments heisst `html`, die Angabe der Sprache mit Hilfe des `lang`-Attributs wird empfohlen. In XHTML sah das `html`-Element so aus:

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
```

Die Namespace-Angabe fällt in HTML5 weg und es bleibt:

```
<html lang="de">
```

Das `html`-Element enthält zunächst ein `head`- und dann ein `body`-Element. Diese Tatsache wurde in HTML4 mit Hilfe der DTD ausgedrückt:

```
<!ENTITY % html.content "HEAD, BODY">
<!ELEMENT HTML O O (%html.content;) -- document root element -->
```

In HTML5 gibt es keine DTD mehr, die Beschreibung steht im Text, s. Abbildung 27.

The screenshot shows the '4.1.1 The `html` element' section of the WHATWG Living Standard. It includes sections for Categories (None), Contexts (As document's `document` element, Wherever a subdocument fragment is allowed in a compound document), Content model (A `head` element followed by a `body` element), Tag omission in text/html (An `html` element's `start tag` can be omitted if the first thing inside the `html` element is not a `comment`. An `html` element's `end tag` can be omitted if the `html` element is not immediately followed by a `comment`), Content attributes (Global attributes), Accessibility considerations (For authors, For implementers), and DOM interface (IDL code: `[Exposed=Window] interface HTMLHtmlElement : HTMLElement { [HTMLConstructor] constructor(); // also has obsolete members };`). A note at the bottom states: 'The `html` element represents the root of an HTML document.'

Abbildung 27: HTML Element

Das `head`-Element enthält Angaben *über* das Dokument, also nicht den eigentlichen Inhalt. Dazu gehört der Titel des Dokument, Meta-Angaben, oder Verbindungen zu anderen Dokumenten. Auch Scripts können im `head` einer HTML-Datei untergebracht werden.

Ein wichtiges Element ist `title`. Der Titel des Dokuments sollte aussagen, um was es in dem Dokument geht. Nicht mehr und nicht weniger. Der Titel spielt an verschiedenen Stellen eine Rolle:

- Erscheint in der Titelzeile des Anzeigefensters
- Lesezeichen bekommen den Titel als Namen
- In der Liste der bereits besuchten Seiten erscheint der Titel
- Suchmaschinen gewichten den Titel relativ hoch

Man sollte also auch beim Einsatz eines CMS oder einer Blog-Software darauf achten, dass diese so eingestellt sind, dass sie aussagekräftige Titel erzeugen. Ein Titel sollte auch den Kontext liefern, also zum Beispiel nicht nur **Unsere Philosophie**. Ein solcher Eintrag in den Lesezeichen nützt wenig. Daher besser: **Restaurant Wildhof - Unsere**

**Philosophie.** Und was dabei auch berücksichtigt werden sollte: Potentielle Gäste werden mehr an der Speisekarte und den Öffnungszeiten als an der Philosophie des Restaurants interessiert sein :)

Mit Hilfe von Meta-Angaben sind weitere Informationen über das Dokument möglich. Sinnvoll ist es, zumindest die Zeichencodierung anzugeben, heute meist UTF-8. Bis HTML4 sah diese Angabe so aus:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

Neu ist sie zum Glück einfacher zu merken:

```
<meta charset="UTF-8">
```

Weitere Meta-Angaben können Hinweise auf Autoren des Dokuments, eine Zusammenfassung oder Stichwörter enthalten. Stichwörter waren früher wichtig für **Suchmaschinen**. Da die eingetragenen Stichwörter immer weniger mit dem Inhalt der Dokumente zu tun hatten, berücksichtigen Suchmaschinen diese heute kaum noch. Viel wichtiger sind der Titel (natürlich nur, wenn er zum Inhalt des Dokuments passt), die Überschriften, und ob das Dokument tatsächlich relevante Inhalte aufweist.

Das **link**-Element stellt Beziehungen zu anderen Dokumenten her. Seit wichtigster Einsatzzweck ist es, CSS-Dateien einzubinden:

```
<link rel="stylesheet" href="styles/screen.css">
```

Die bis HTML4 noch nötige Angabe `type="text/css"` ist nicht mehr erforderlich. Es können auch mehrere CSS-Dateien eingefügt werden, indem mehrere **link**-Elemente verwendet werden. Das **media**-Attribut dient dazu, das betreffende Stylesheet nur dann zu laden, wenn die HTML-Seite gerade mit einem Gerät genutzt wird, das zum Attributwert passt:

```
1 <!-- Stylesheet nur verwendet, wenn die Seite gedruckt wird: -->
2 <link rel="stylesheet" type="print" href="styles/print.css">
3
4 <!-- Stylesheet nur verwendet, wenn die Seite mir einem Beamer wird: -->
5 <link rel="stylesheet" type="projection" href="styles/beamer.css">
```

Die möglichen Angaben für den Wert des **type**-Attributs variieren etwas in den einzelnen Spezifikationen. Relativ neu ist die Möglichkeit, die Eigenschaften des Geräts (wie Bildschirmgrösse, Auflösung, Orientierung) genau zu beschreiben. Dazu gibt es die W3C Recommendation *Media Queries*.<sup>27</sup> Dort steht zu den Gerätetypen:

The ‘print’ and ‘screen’ media types are defined in HTML4. The complete list of media types in HTML4 is: ‘aural’, ‘braille’, ‘handheld’, ‘print’, ‘projection’, ‘screen’, ‘tty’, ‘tv’. CSS2 defines the same list, deprecates ‘aural’ and adds ‘embossed’ and ‘speech’. Also, ‘all’ is used to indicate that the style sheet applies to all media types.

Das **link**-Element dient nicht nur zum Laden von Stylesheets. Die Angabe des *Beziehungstyps* erfolgt mit dem **rel**-Attribut. Neben **Stylesheet** sind noch zahlreiche andere

---

<sup>27</sup><http://www.w3.org/TR/css3-mediaqueries/>

Typen definiert: **Alternate**, **Start**, **Next**, **Prev**, **Contents**, **Index**, **Glossary**, **Copyright**, **Chapter**, **Section**, **Subsection**, **Appendix**, **Help**, **Bookmark**. Diese Beziehungstypen sind nicht nur beim **link**-Element sondern auch beim **a**-Element, also normalen Verweisen, möglich.

Obwohl die Beziehungstypen schon über mehrere Versionen Bestandteil von HTML sind, ist die Browser-Unterstützung mager. Daher werden sie auch eher selten eingesetzt. Der **Seamonkey**-Browser kann eine zusätzliche Navigationsleiste anzeigen, wenn ein Dokument solche **link**-Elemente verwendet. Beim Seamonkey handelt es sich um ein Mozilla-Projekt, das dieselbe Rendering Engine wie der Firefox verwendet. Der Seamonkey verfügt neben dem Browser über einen Mail- und News-Client, einen Web-Editor, ein Adressbuch und ein Chat-Programm (IRC).

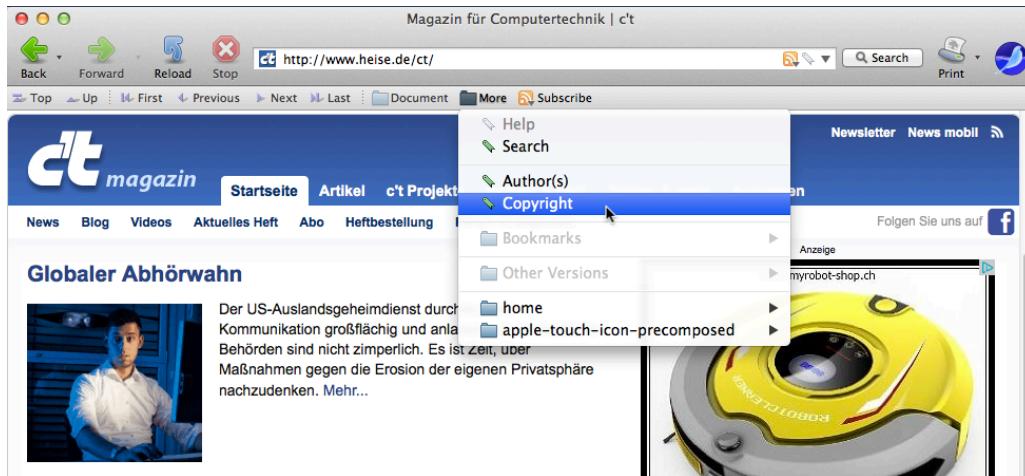


Abbildung 28: Link-Elemente im Seamonkey

Hier ist ein Ausschnitt aus dem **head** des angezeigten Dokuments:

```
1 <link rel="copyright" title="Copyright" href="/ct/c-t-Impressum-4780.html" />
2 <link rel="search" title="Suche" href="http://www.heise.de/ct/suche/" />
```

Die weiteren Elemente, die im **head** vorkommen können, sind:

- **base** für die Angabe einer Basis-URL zur Auflösung relativer Links
- **style**, um CSS-Regeln direkt ins HTML-Dokument schreiben zu können
- **script** für JavaScript-Anweisungen oder um eine JavaScript-Datei zu laden

## Grobstruktur des Dokuments

Nach dem **head**-Element wenden wir uns nun dem **body**-Element zu, das den *Inhalt* des Dokuments beschreiben soll.

In der Regel möchte man ein Dokument grob in verschiedene Blöcke aufteilen: Zum Beispiel einen Kopfbereich mit einem Banner-Bild, einen Inhaltsbereich und einen oder mehrere Navigationsbereiche. Dokumente können jedoch sehr unterschiedlich aufgebaut sein.

Ein Weblog kann aus einer Menge von Artikeln bestehen, eine Dokumentation aus vielen Kapiteln, ein Kontaktformular aus verschiedenen Bereichen mit Formularelementen.

Für die grobe Struktur eines Dokuments steht in HTML4 eigentlich nur das `div`-Element zur Verfügung, dessen Bedeutung nicht näher definiert ist, ausser dass es zur Gruppierung anderer Elemente eingesetzt werden kann. Allerdings erlauben **Attribute** eine nähere Beschreibung. Die folgenden vier Attribute können bei fast allen HTML-Elementen eingefügt werden:

- **class**: Weist dem Element eine bestimmte Klasse zu. Die Klasse kann mehrfach im Dokument vorkommen und in CSS oder JavaScript verwendet werden, um Elemente mit dieser Klasse anzusprechen.
- **id**: Weist dem Element eine bestimmte ID zu. Jede ID kann nur einmal im Dokument vorkommen und in CSS oder JavaScript verwendet werden, um genau dieses Element anzusprechen.
- **style**: Für CSS-Angaben, die genau dieses Element betreffen.
- **title**: Versieht ein Element mit einem erklärenden Text. Dieser wird normalerweise als Tooltip angezeigt, wenn der Mauszeiger auf das Element zeigt.

Um nun einem `div`-Element zusätzliche Bedeutung zu geben, können die beiden Attribute **class** oder **id** verwendet werden. Ein HTML4-Dokument ist also oft wie in Abbildung 29 gezeigt strukturiert (Bild aus: *A Preview of HTML 5*).<sup>28</sup>

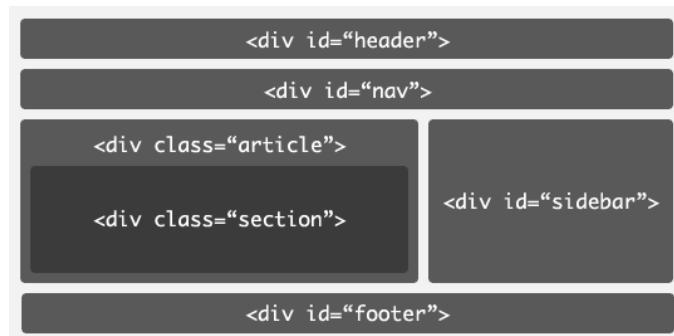


Abbildung 29: Struktur eines Dokuments in HTML4

Das Problem dabei ist: Es gibt keine allgemeine Konvention, wie diese Klassen und IDs genannt werden sollen. Häufig wäre es aber sinnvoll, wenn auch Suchmaschinen, Sprachausgabegeräte oder andere Programme, die den Inhalt einer Seite analysieren sollen, Navigations- und Inhaltsbereiche eindeutig identifizieren könnten. Beim Verwenden von `div`-Elementen mit Klassen muss aber immer mit verschiedenen Varianten für Blocks mit bestimmten Bedeutungen gerechnet werden:

```
<div id="nav">  
<div id="navi">  
<div id="navigation">
```

<sup>28</sup><http://alistapart.com/article/previewofhtml5>

Mit HTML5 hat man daher eine Reihe neuer Elemente für die Strukturierung eines Dokuments eingeführt. Für den Navigationsbereich ist das einfach:

### <nav>

Abbildung 30 zeigt, wie ein Dokument mit den HTML5-Elementen aufgebaut sein könnte (Bild ebenfalls aus: *A Preview of HTML 5*).

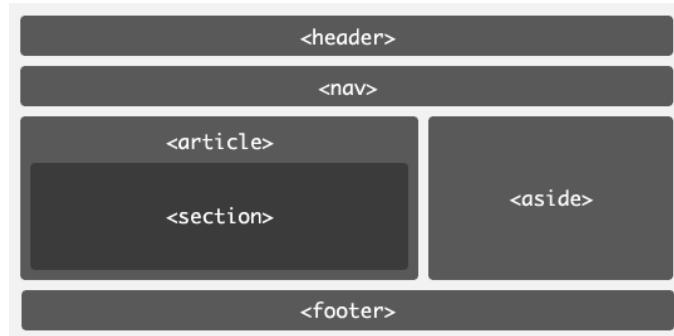


Abbildung 30: Struktur eines Dokuments in HTML5

Hier eine kurze Beschreibung der mit HTML5 eingeführten Elemente zum Seitenaufbau:

- **section**: Zum Gruppieren von thematisch zusammenhängendem Inhalt. Ähnlich dem div-Element, aber bei **section** sollte der Inhalt thematisch zusammenhängen.
- **header**: Dieses Element ist in der HTML5-Spezifikation so beschrieben: “*The header element represents a group of introductory or navigational aids.*” Also Teile, die zur Einführung oder zur Navigation dienen. Ein **header** kann mehrfach im Dokument auftreten, zum Beispiel kann auch eine Section ein **header**-Element enthalten.

Beispiel (aus: *HTML5 For Web Designers*<sup>29</sup>):

```
1 <section>
2   <header>
3     <h1>DOM Scripting</h1>
4   </header>
5   <p>The book is aimed at designers
6   rather than programmers.</p>
7   <p>By Jeremy Keith</p>
8 </section>
```

- **footer**: Auch wenn das Element nach einer Position im Dokument klingt, geht es doch um die Bedeutung. In diesem Fall: Es enthält Informationen zu dem Element, in dem es vorkommt. Also zum Beispiel: Autor, Copyright, Verweise zu ähnlichen Themen.
- **aside**: Für Inhalte, die nur am Rande mit den eigentlichen Inhalten zu tun haben (“*tangentially related*”). Natürlich ist etwas wie eine Sidebar damit gemeint,

<sup>29</sup><http://www.abookapart.com/products/html5-for-web-designers>

aber auch hier geht es um die Bedeutung und nicht die Position auf der Seite. Der Hauptinhalt der Seite sollte auch ohne den Inhalt des `aside`-Elements verständlich sein.

- `nav`: Enthält zentrale Navigationselemente für eine Seite, meist in Form einer Liste von Links. Das `nav`-Element kommt oft in einem `header`-Element vor. Achtung: Nicht jede Liste von Links gehört in ein `nav`.
- `article`: Für in sich abgeschlossenen, thematisch zusammenhängenden Inhalt. Blog-Posts wären ein typischer Fall für dieses Element. Es ist aber nicht immer einfach, `article` von `section` zu unterscheiden.
- `main`: Repräsentiert den Hauptinhalt des Dokuments. Es darf nur ein `main`-Element im Dokument geben und es soll nur als direktes Kindelement von `html`, `body`, `div`, oder `form` vorkommen.

Vermutlich erkennen Sie schon, was das Problem mit diesen Elementen ist: Es ist oft schwierig zu entscheiden, mit welchem Element ein bestimmter Teil eines Dokuments ausgezeichnet werden soll. Die Beschreibungen in der HTML-Spezifikation sind da auch nicht immer ganz eindeutig: `section` ist relativ allgemein, `div` noch etwas allgemeiner.

Was ist mit `div`? Ist das nicht HTML4? Nein, nicht nur: `div` ist weiterhin erlaubt. Wenn es möglich ist, sollten die Teile eines Dokuments aber mit den inhaltlich präziseren Elementen ausgezeichnet werden.

## Überschriften

Bis **HTML4** war das Thema Überschriften ziemlich einfach abzuhandeln: Es gibt sechs verschiedene Elemente `h1` bis `h6` für die verschiedenen Überschriftenebenen. `h1` ist die Hauptüberschrift, die weiteren Elemente können als untergeordnete Überschriften verwendet werden (Abbildung 31).



The screenshot shows a web browser window with the title "index.html". The left pane displays the HTML code:`<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Überschriften</title>
</head>
<body>
 <h1>Überschrift 1</h1>
 <h2>Überschrift 2</h2>
 <h3>Überschrift 3</h3>
</body>
</html>`The right pane shows the rendered content with three levels of headings:

# Überschrift 1

## Überschrift 2

### Überschrift 3

Abbildung 31: Überschriften

**Wichtig:** Die oberste Überschrift ist `h1`, die nächste `h2`, usw. Die Elemente dürfen nicht nach der Grösse der Standarddarstellung ausgewählt werden, es geht nur um die Bedeutung. Mehr dazu im Abschnitt *Semantic Markup*. Die Grösse, in der eine Überschrift im Browser erscheint, wird mit Hilfe von CSS festgelegt.

Das Problem mit den sechs möglichen Überschriften in HTML4 ist, dass diese Ebenen global für das ganze Dokument gelten, unabhängig davon, wie tief in der hierarchischen Struktur des Dokuments die Überschrift liegt. Das kann zu Schwierigkeiten führen, wenn Teile des Dokuments aus anderen Quellen eingebettet werden, welche eine eigene Überschriftenhierarchie haben, die möglicherweise selbst mit `h1` beginnt.

In **HTML5** hat man versucht, dieses Problem zu lösen. Leider ist die Lösung ziemlich kompliziert geraten und funktioniert bisher in den Browsern noch nicht vollständig.

Mehr dazu in einem späteren Kapitel.

## Semantic Markup

Bei der Auszeichnung eines Dokuments mit HTML soll die Darstellung möglichst komplett unberücksichtigt bleiben. Es geht allein um die Bedeutung, um den Aufbau einer nach inhaltlichen Kriterien definierten Dokumentenstruktur, nach der Rolle, die die einzelnen Teile zueinander und im Dokument insgesamt spielen. Die folgenden Aussagen dürfen also beim Markup *keine Rolle* spielen:

- Die Überschrift sollte etwas kleiner sein.
- Das Logo soll oben rechts stehen.
- Dieser Absatz soll einen grösseren Abstand zum Rest haben als die anderen.

Bei der letzten Aussage müsste man überlegen: Warum? Was unterscheidet diesen inhaltlich von den anderen. Dann kann man den Unterschied mit einem geeigneten Element oder einem passenden Wert für das Attribut `class` (oder `id`) ausdrücken.

Im Webdesign steht man oft vor der Aufgabe, ein vom Grafiker geliefertes Bild einer Webseite in HTML und CSS umzusetzen. Die Schwierigkeit dabei ist, sich die Gestaltung erst einmal wegzudenken und die logische Struktur herauszuarbeiten und in HTML abzubilden. Erst im nächsten Schritt kann dann die Gestaltung in Form von CSS-Regeln wieder dazu genommen werden.

## Absätze und Listen

Absätze werden mit dem `p`-Element notiert. Innerhalb eines Absatzes kann mit `br` ein Zeilenwechsel eingefügt werden. `br` ist ein Element ohne Inhalt, ein *leeres Element*.

```
1  <p>
2      Imagine there's no Heaven <br>
3      It's easy if you try <br>
4      No hell below us  <br>
5      Above us only sky <br>
6  </p>
```

Bei Listen gibt es verschiedene Varianten: Mit `ul` zeichnet man eine *unordered list* aus, eine Liste, deren Elemente meist mit Punkten oder Strichen dargestellt werden. Mit `ol` dagegen werden *ordered lists* eingeführt, die meist alphabetisch oder mit Zahlen durchnummieriert sind. Sowohl bei `ul` als auch bei `ol` werden die einzelnen Listenelemente mit `li`-Elementen (*list item*) versehen.

Eine andere Art von Liste ist `dl`, die *definition list*. Sie dient dazu, Begriffe und Definitionen (wie zum Beispiel in einem Glossar) einander zuzuordnen. Der zu definierende Begriff steht in einem `dt`-Element, die zugehörige Definition anschliessend in einem `dd`-Element.

Die Abbildung 32 zeigt einige Beispiele für Listen und die Standard-Darstellung im Browser (ohne CSS).

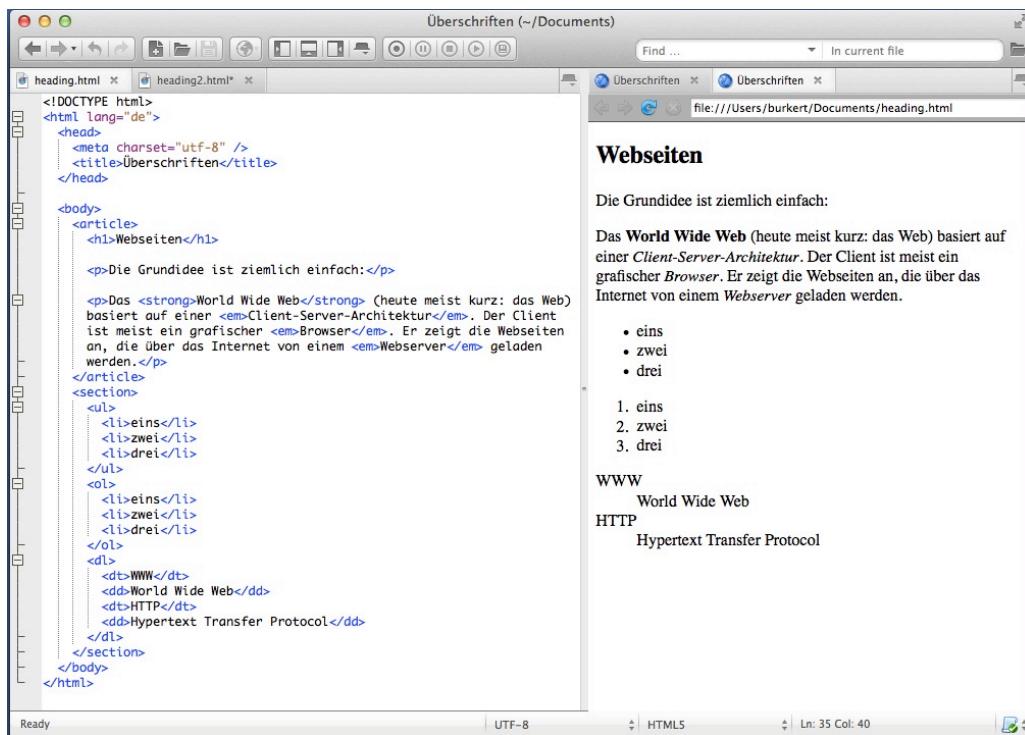


Abbildung 32: Absätze und Listen

Im Bild tauchen zwei weitere Elemente auf: `strong` und `em`. Mit diesen beiden können Textstellen hervorgehoben werden. Es gibt weitere, ähnliche Elemente. Die Elemente `b` und `i` zum Beispiel stehen ebenfalls für eine Hervorhebung. Früher stand `b` einfach für *bold*. Dies widerspricht aber dem Ziel eines *Semantic Markup* (s.o.), daher wurde es in der HTML5-Spezifikation inhaltlich definiert – wobei es schon nicht ganz einfach ist, die verschiedenen Hervorhebungen voneinander zu unterscheiden. Auszüge aus der HTML-Spezifikation (Living Standard, 24.05.2021):

- **strong:** The strong element represents strong importance, seriousness, or urgency for its contents.

- **em**: The em element represents stress emphasis of its contents.
- **b**: The b element represents a span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood, such as key words in a document abstract, product names in a review, actionable words in interactive text-driven software, or an article lede.
- **i**: The i element represents a span of text in an alternate voice or mood, or otherwise offset from the normal prose in a manner indicating a different quality of text, such as a taxonomic designation, a technical term, an idiomatic phrase from another language, transliteration, a thought, or a ship name in Western texts.
- **mark**: The mark element represents a run of text in one document marked or highlighted for reference purposes, due to its relevance in another context. [...]

Von dieser Liste wurde nur das **mark**-Element in HTML5 neu eingeführt. Aber auch bei den anderen Elementen hat sich die Bedeutung seit HTML4 etwas verändert. Die Aussage „*Jedes HTML4-Dokument ist auch ein HTML5-Dokument, wenn man nur den Doctype anpasst*“ ist also nicht ganz richtig, denn genau genommen kann sich dadurch auch die Bedeutung des Inhalts verändern.

Im Gegensatz zu den bisher eingeführten Elementen können diese Elemente innerhalb von Fließtext auftauchen. Während etwa **p**, **article**, **ol**, **h1** Blocks definieren, die den verfügbaren Platz in horizontaler Richtung einnehmen, werden Elemente wie **strong**, **em** auf Bereiche innerhalb von Texten angewendet. Wenn man die Darstellung dazu nimmt, ist das vergleichbar mit Absatzstilen und Textstilen in Textverarbeitungsprogrammen. Im Webdesign verwendet man dafür die Bezeichnungen **Block-level-Elemente** und **Inline-Elemente**. In der HTML5-Spezifikation werden die Inline-Elemente zwar unter der Bezeichnung **Text-level semantics** zusammengefasst, wir werden aber weiterhin die Bezeichnung **Inline-Elemente** verwenden.

Ein Inline-Element sollte an dieser Stelle noch erwähnt werden: **span**. Ähnlich wie **div** hat es keine eigene Bedeutung. Im Gegensatz zu **div** ist **span** ein Inline-Element. Es kann eingesetzt werden, wenn ein Textbereich ausgezeichnet werden soll, für den keines der anderen Elemente geeignet ist. Normalerweise erhält **span** dann zur näheren Beschreibung ein **class**-Attribut (**id** dürfte eher selten geeignet sein).

Weitere, vor allem mit HTML5 neu eingeführte, Inline-Elemente werden wir im nächsten Kapitel behandeln.

## Zitate und Tabellen

Um **Zitate** darzustellen gibt es einmal das Element **blockquote**. Das ist ein Block-Element, das einen Textabschnitt kennzeichnet, der von einer anderen Quelle übernommen ist. Ein **cite**-Attribut ist erlaubt, um die Quelle des Zitats in Form einer URL anzugeben. Weitere Quellenhinweise müssten ausserhalb des **blockquote**-Elements stehen.

Hier ein Beispiel aus der HTML-Spezifikation:

```
1 <blockquote>
2   <p>I contend that we are both atheists. I just believe in one fewer
3     god than you do. When you understand why you dismiss all the other
4     possible gods, you will understand why I dismiss yours.</p>
5 </blockquote>
6 <p>- Stephen Roberts</p>
```

Im Gegensatz zu `blockquote` ist `cite` ein Inline-Element. Es dient dazu, den Titel einer Arbeit auszuzeichnen, also zum Beispiel eines Buchs, eines Artikels oder eines Gedichts.

Auch hier ein Beispiel aus der HTML-Spezifikation:

```
1 <p>My favorite book is <code><cite>The Reality Dysfunction</cite></code> by
2 Peter F. Hamilton. My favorite comic is <code><cite>Pearls Before
3 Swine</cite></code> by Stephan Pastis. My favorite track is <code><cite>Jive
4 Samba</cite></code> by the Cannonball Adderley Sextet.</p>
```

Das `cite`-Element hat beim Übergang von HTML4 zu HTML5 eine Änderung erfahren. Hier ein Zitat dazu aus dem Buch *HTML 5 For Web Designers*:

The cite element has been redefined in HTML5. Where it previously meant “a reference to other sources,” it now means “the title of a work.” Quite often, a cited reference will be the title of a work, such as a book or a film, but the source could just as easily be a person. Before HTML5, you could mark up that person’s name using cite. Now that’s expressly forbidden—so much for backwards compatibility.

**Tabellen** dienen zur Darstellung tabellarischer Daten. Die Zeiten, in denen Tabellen mangels anderer Möglichkeiten zum Aufbau von Seitenlayouts verwendet wurden, sind lange vorbei. Das war nur eine Notlösung, solange die meisten Browser nicht gut mit CSS umgehen konnten. Tabellen werden mit dem `table`-Element eingeführt. Mit Hilfe einer Reihe von weiteren Elementen wird die Tabelle zusammengebaut:

Element	Bedeutung
<code>table</code>	Tabelle
<code>tr</code>	Zeile (table row)
<code>th</code>	Zelle der Kopfzeile (table header)
<code>td</code>	Zelle (table data)
<code>caption</code>	Tabellenbeschreibung
<code>thead</code>	Tabellenkopfzeile(n)
<code>tfoot</code>	Tabellenfusszeile(n)
<code>colgroup</code>	Definition und Zusammenfassen von Spalten
<code>col</code>	Definition von Spalten

Die Bedeutung der einzelnen Elemente und wie sie eingesetzt werden können Sie leicht der HTML-Spezifikation entnehmen. Hier soll ein Beispiel aus der HTML5-Spezifikation (leicht angepasst) genügen:

```

1  <table>
2    <thead>
3      <tr>
4        <th>
5        <th>2008</th>
6        <th>2007</th>
7        <th>2006</th>
8      </tr>
9    </thead>
10   <tbody>
11     <tr>
12       <th>Net sales</th>
13       <td>$ 32,479</td>
14       <td>$ 24,006</td>
15       <td>$ 19,315</td>
16     </tr>
17     <tr>
18       <th>Cost of sales</th>
19       <td> 21,334</td>
20       <td> 15,852</td>
21       <td> 13,717</td>
22     </tr>
23   </tbody>
24   <tbody>
25     <tr>
26       <th>Gross margin</th>
27       <td>$ 11,145</td>
28       <td>$  8,154</td>
29       <td>$  5,598</td>
30     </tr>
31   </tbody>
32   <tfoot>
33     <tr>
34       <th>Gross margin percentage</th>
35       <td>34.3%</td>
36       <td>34.0%</td>
37       <td>29.0%</td>
38     </tr>
39   </tfoot>
40 </table>
```

Mit geeigneten CSS-Angaben könnte die Tabelle aussehen wie in der Abbildung 33 gezeigt.

## Kontrolle und Fehlersuche

Ob ein HTML-Dokument Fehler enthält, kann mit einem so genannten **Validator** überprüft werden. Bis HTML 4.01 oder für die verschiedenen XHTML-Versionen hat ein

	2008	2007	2006
Net sales	\$ 32,479	\$ 24,006	\$ 19,315
Cost of sales	21,334	15,852	13,717
Gross margin	<u>\$ 11,145</u>	<u>\$ 8,154</u>	<u>\$ 5,598</u>
Gross margin percentage	34.3%	34.0%	29.0%

Abbildung 33: Beispiel-Tabelle

Validator in erster Linie die *Wohlgeformtheit* des Dokuments sowie das Einhalten der DTD überprüft. Seit HTML5 steht keine DTD mehr zur Verfügung, daher muss beim Validieren die Konformität zur Spezifikation geprüft werden.

Es gibt verschiedene Online-Validatoren. Am besten verwenden Sie direkt den Validator des W3C unter <http://validator.w3.org>. Viele Webeditoren stellen entweder eigene Validatoren zur Verfügung oder die Möglichkeit, ein Dokument auf einfache Weise mit einem Online-Validator zu prüfen. Wenn Sie im Firefox die *Web Developer Toolbar* installiert haben, können Sie das angezeigte Dokument einfach über das *Tools*-Menü zum Validator schicken. Im besten Fall liefert der Validator eine Meldung, dass keine Fehler gefunden wurden (s. Abbildung 34).

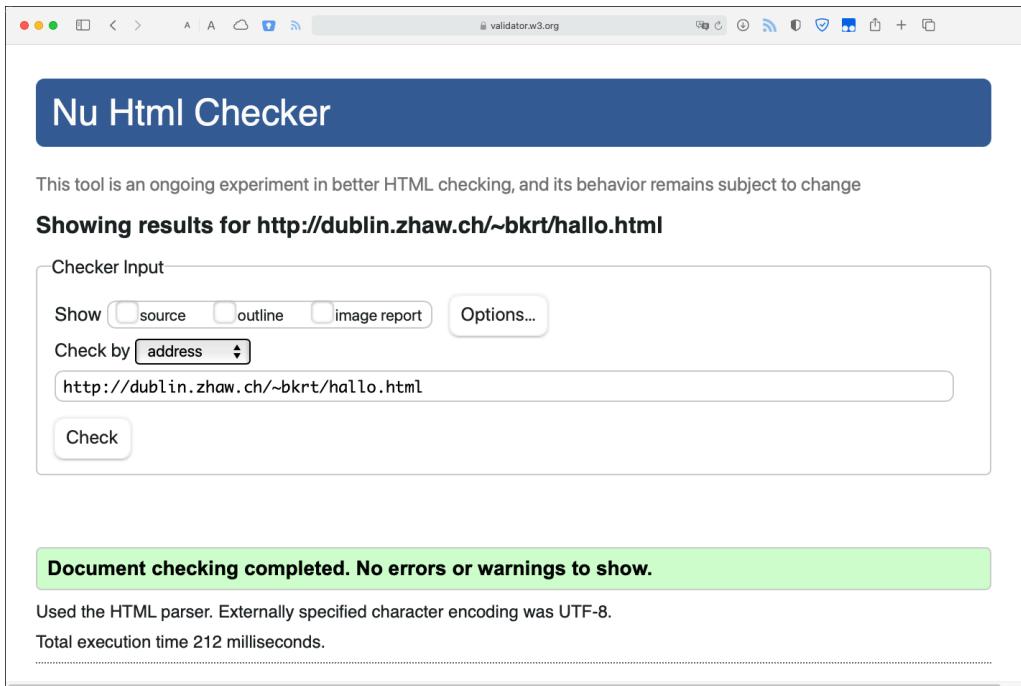


Abbildung 34: Validieren war erfolgreich

**Überprüfen Sie beim Erstellen einer Webseite diese regelmäßig mit dem Validator.** Studieren Sie auch die Warnungen. Je früher Sie Probleme und Fehler finden, desto

besser. Eine fehlerfreie Seite hat bessere Chancen, in vielen Browsern gut zu funktionieren. Auch ist die Gefahr von CSS- oder JavaScript-Problemen bei der weiteren Entwicklung der Seite geringer.

Mit den in der letzten Lektion behandelten Entwicklerwerkzeugen können Sie die HTML-Struktur des Dokuments näher untersuchen. Dazu wird der *Inspektor* ausgewählt, mit dessen Hilfe einfach festgestellt werden kann, welches Element in der HTML-Struktur auf der Seite wo dargestellt wird (Abbildung 35).

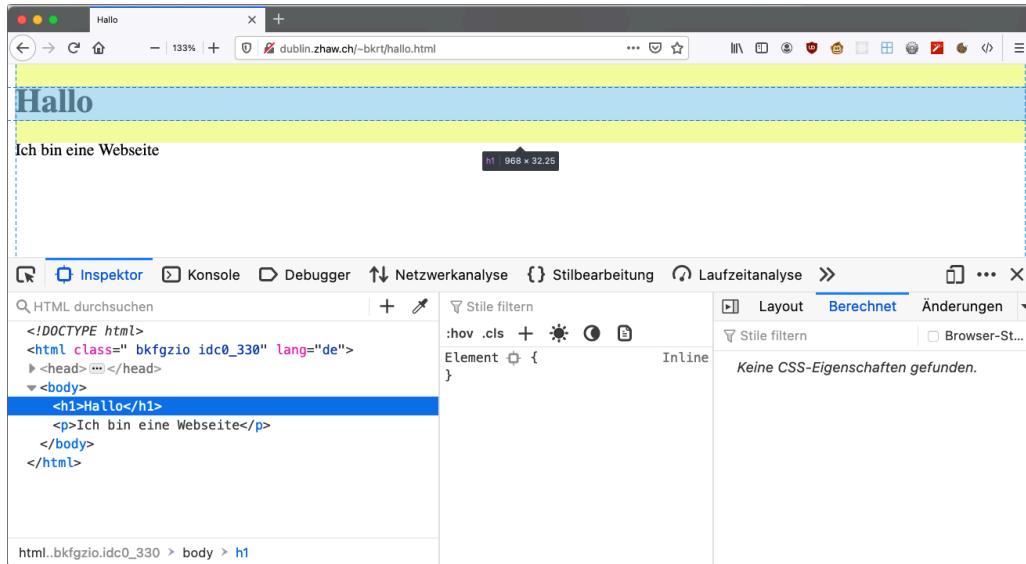


Abbildung 35: Darstellung der Dokumentstruktur

# Quellen und Verweise

## Material zur Vertiefung

- **Dive Into HTML5** (Mark Pilgrim)  
<http://diveintohtml5.info>
- **What Does It All Mean?** (Mark Pilgrim, Dive Into HTML5)  
<http://diveintohtml5.info/semantics.html>
- **How Did We Get Here?** A Quite Biased History of HTML5. (Mark Pilgrim, Dive Into HTML5)  
<http://diveintohtml5.info/past.html>
- **Über SGML und HTML** (W3C, Übersetzung der HTML-4.01-Spezifikation)  
<http://www.edition-w3.de/TR/1999/REC-html401-19991224/intro/sgmltut.html>

## Spezifikationen und Standards:

- **HTML 4.01 Recommendation** (W3C)  
<http://www.w3.org/TR/html401/>
- **HTML 5.1 Working Draft** (W3C)  
<http://www.w3.org/TR/html51/>
- **HTML Living Standard** (WHATWG)  
<https://html.spec.whatwg.org/multipage/>
- **Media Queries** (W3C Recommendation)  
<http://www.w3.org/TR/css3-mediaqueries/>

## Quellenangaben

- **A Preview of HTML 5** (Lachlan Hunt, A List Apart, 2007)  
<http://alistapart.com/article/previewofhtml5>
- **HTML5 For Web Designers** (Jeremy Keith, A Book Apart)  
<http://www.abookapart.com/products/html5-for-web-designers>

Ein Teil der Bilder und Inhalte stammt aus früheren Folien der Fächer WBD (Webdesign), INSY (Informationssysteme) und WEBT (Webtechnologien).

# Markup und HTML (Teil 2)

## Einführung und Ziele

Im zweiten Teil zum Thema *Markup und HTML* wird zunächst die Möglichkeit beschrieben, mittels Hyperlinks auf andere Webseiten und Stellen innerhalb von Seiten zu verweisen. Außerdem geht es um das Einfügen von Bildern, Videos und Audioelementen in Webseiten. Mit Formularen können Webangebote um Interaktionsmöglichkeiten erweitert werden. Wie Formulare in HTML aufgebaut werden und welche Erweiterungen HTML5 in diesem Zusammenhang bietet, ist Thema in WBE und wird daher hier nur kurz ange schnitten.

## Ziele

- Sie wissen, wie Verweise im HTML-Code angelegt werden.
- Sie kennen die Möglichkeiten absoluter und relativer Pfadangaben in Verweisen und wissen, wie auf Stellen innerhalb von Webseiten verwiesen wird.
- Sie wissen, wie Bilder in eine HTML-Seite eingefügt werden und kennen die drei Dateiformate für Pixelgrafiken im Web und deren wichtigste Eigenschaften.
- *Sie wissen, wie Audio- und Videoelemente in Webseiten eingefügt werden können.*
- *Sie sind in der Lage, ein HTML-Formular anzulegen, und kennen ein paar Formularelemente.*

## Relevanz für WBE

Wichtig sind das Einfügen von Bildern und Verweisen in HTML-Dokumente sowie die dabei möglichen Adress-Angaben. Nicht vorausgesetzt wird Wissen über das Einfügen von Audio- und Videoelementen in HTML-Dokumente. Formulare werden in WBE behandelt.

# Verweise und Navigation

## Verweise

Im ersten Teil zum Thema Markup haben wir uns mit einigen Elementen beschäftigt, um die inhaltliche Struktur eines Dokuments zu beschreiben, sowie einigen Elementen, welche die Bedeutung von Textausschnitten charakterisieren. Etwas Wesentliches fehlt noch: Das **HT** in **HTML**. Erst mit der Möglichkeit, mit Hilfe von Verweisen von einem Ort zu einem anderen zu „springen“, bezeichnen wir das Ganze als **Hypertext**.

Verweise (*Hyperlinks*, oder einfach: *Links*) werden mit dem **a**-Element angelegt. Das Verweisziel wird im **href**-Attribut angegeben (hypertext reference). Es kann eine ganze URL oder ein Pfad zu einer Datei auf dem gleichen Server sein. Zwischen dem Start- und dem End-Tag des **a**-Elements steht der Text, der zum Verweis wird. Hier ein Beispiel:

```
1 <p>Auf der <a href="http://www.zhaw.ch">Startseite</a>
2 finden Sie alle wesentlichen Informationen</p>
```

Normalerweise werden Verweise blau und unterstrichen dargestellt, wenn nicht mit CSS eine anderes Aussehen definiert wird:

Auf der [Startseite](http://www.zhaw.ch) finden Sie alle wesentlichen Informationen.

Abbildung 36: Hyperlink

Bei Verweiszielen innerhalb einer Website werden normalerweise statt einer kompletten URL nur relative oder (seltener) absolute Pfade angegeben:

```
1 <!-- gleiches Verzeichnis -->
2 <p>Auf der <a href="kontakt.html">Kontaktseite</a> können Sie ...</p>
3
4 <!-- relative Angabe -->
5 <p>Change to <a href="../en/index.html">english</a> language</p>
6
7 <!-- absolute Angabe ab Webroot -->
8 <p>Zurück zur <a href="/index.html">Hauptseite</a></p>
```

In früheren HTML-Versionen konnte man mit Hilfe des **a**-Elements auch ein Anker definieren, der ein Verweisziel innerhalb eines Dokuments bildet. Dazu hat man das **a**-Element mit einem **name**-Attribut verwendet. In HTML5 hat das **a**-Element kein **name**-Attribut. Das ist auch nicht mehr nötig, denn als Verweisziel innerhalb eines Dokuments kann jedes Element verwendet werden, das mit einem **id**-Attribut versehen ist.

Mit Hilfe des *Fragment Identifier* in der URL kann direkt zu einem Element mit einer bestimmten **id** gesprungen werden:

```

1  <!-- Überschrift als Verweisziel -->
2  <h1 id="foo">Foo Title</h1>
3
4  <!-- Verweis auf die Überschrift -->
5  <a href="#foo">Springe zu Foo Title</a>
6
7  <!-- Verweis auf die Überschrift in anderem Dokument -->
8  <a href="http://example.com/index.html#foo">Springe zu Foo Title</a>

```

## Navigationsbereich

Wenn eine Liste von Links die Navigation innerhalb der Website oder innerhalb einer einzelnen Seite ermöglicht, sollte diese mit einem `nav`-Element ausgezeichnet werden. In der HTML-Spezifikation ist dieses Element so beschrieben:

The `nav` element represents a section of a page that links to other pages or to parts within the page: a section with navigation links. Note: Not all groups of links on a page need to be in a `nav` element — the element is primarily intended for sections that consist of major navigation blocks. [...]

Oft kommt ein `nav`-Element in einem `header`-Element vor, denn das `header`-Element dient ja unter an derer zur Einführung und zu Navigationszwecken. Aus der HTML5-Spezifikation:

The `header` element represents a group of introductory or navigational aids.

Ein Ausschnitt aus dem Beispiel in der HTML-Spezifikation zeigt, wie das gehen kann:

```

1  <header>
2    <h1>Wake up sheeple!</h1>
3    <p><a href="news.html">News</a> -
4      <a href="blog.html">Blog</a> -
5      <a href="forums.html">Forums</a></p>
6    <p>Last Modified: <span itemprop="dateModified">2009-04-01</span></p>
7    <nav>
8      <h1>Navigation</h1>
9      <ul>
10        <li><a href="articles.html">Index of all articles</a></li>
11        <li><a href="today.html">Things sheeple need to wake up for today</a></li>
12        <li><a href="successes.html">Sheeple we have managed to wake</a></li>
13      </ul>
14    </nav>
15  </header>

```

# Bilder und Videos

## Bilder einfügen

Bilder werden mit dem `img`-Element eingefügt:

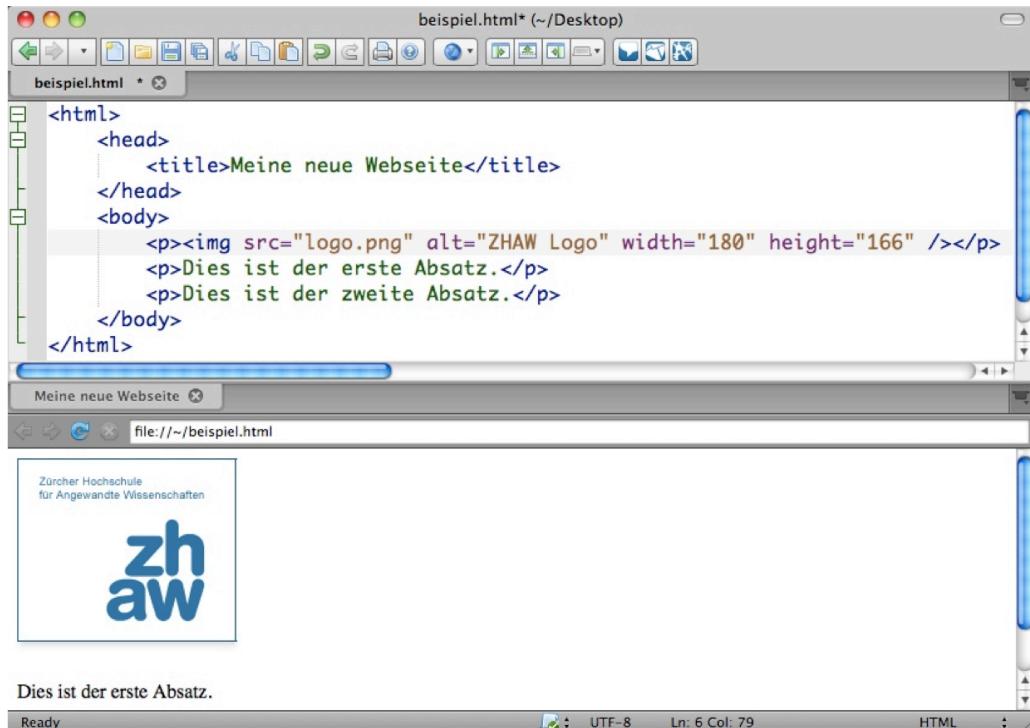


Abbildung 37: img-Element zum Einfügen von Bildern

Das `src`-Attribut gibt die URL oder den Pfad zur Bilddatei an. Ebenso wie bei den Verweisen sind auch hier relative Angaben möglich und üblich. Wenn die Datei `logo.png` relativ zur HTML-Datei in einem `images`-Verzeichnis liegt, wäre die Angabe also:

```
src="images/logo.png"
```

Die Angaben zur Breite und Höhe helfen dem Browser, das Seitenlayout bereits aufzubauen, bevor das Bild geladen ist. Auf diese Weise ist bereits nach dem Laden des HTML-Codes bekannt, wie viel Platz für das Bild reserviert werden muss. In einem fixen Layout war bisher darauf zu achten, dass `width` und `height` der tatsächlichen Grösse des Bildes entsprechen. Ist das tatsächliche Bild grösser, wird eine unnötig grosse Bilddatei vom Server geladen, was zu längeren Ladezeiten führt. Ist das Bild kleiner, wird es für die Anzeige hochskaliert, was fast immer zu schlechten Ergebnissen führt (das Bild wird "pixelig").

Für heutige hochauflösende Bildschirme muss diese Vorgabe relativiert werden, denn die einfache Regel, dass ein Pixel auf dem Bildschirm einem HTML- oder CSS-Pixel ent-

spricht, gilt hier nicht mehr. Um die Auflösung moderner Bildschirme zu nutzen, müssen die Bilder entweder generell in höherer Auflösung bereitgestellt werden, oder es wird eine Möglichkeit vorgesehen, ein für das jeweilige Gerät passendes Bild vom Server zu laden. Das geht zum Beispiel mit dem weiter unten besprochenen `srcset`-Attribut.

Das `alt`-Attribut gibt den Alternativtext an, da nicht in allen Situationen, in denen die Webseite genutzt werden könnte, Bilder angezeigt werden. Es ist in HTML4 ein obligatorisches Attribut. Bei dekorativen Bildern, die nichts zum Inhalt der Seite beitragen, behilft man sich meist so, dass dem `alt`-Attribut ein leerer String zugeordnet wird.

In der HTML5-Spezifikation wird auf zahlreichen Seiten beschrieben, welche Anforderungen an das `alt`-Attribut gelten. Fast immer muss ein `img`-Element auch gemäss HTML5 ein `alt`-Attribut haben. In bestimmten Fällen ist auch ein leeres `alt`-Attribut zulässig. Und in seltenen Fällen ist auch ein fehlendes `alt`-Attribut vom *HTML Conformance Checker* zu akzeptieren. Hier Ausschnitte aus der HTML5-Spezifikation (Stand September 2014):

Except where otherwise specified, the `alt` attribute must be specified and its value must not be empty; the value must be an appropriate functional replacement for the image. The specific requirements for the `alt` attribute content depend on the image's function in the page, as described in the following sections. (Abschnitt 4.7.1.1.2 General guidelines)

Mark up purely decorative images so they can be ignored by assistive technology by using an empty `alt` attribute (`alt=`). While it is not unacceptable to include decorative images inline, it is recommended if they are purely decorative to include the image using CSS. (Abschnitt 4.7.1.1.9 A purely decorative image that doesn't add any information)

A conformance checker must report the lack of an `alt` attribute as an error unless one of the conditions listed below applies: The `img` element is in a `figure` element that satisfies the conditions described above.. [...] (Abschnitt 4.7.1.1.22 Guidance for conformance checkers)

In HTML5 gibt es ein neues Attribut `srcset`, das es erlaubt es, mehrere Bilddateien anzugeben, kombiniert mit Angaben zu den Anzeigetypen, auf denen die einzelnen Varianten zum Einsatz kommen sollen. Das ist wichtig für *Responsive Webdesign*, ein Thema, das nicht zum Stoff dieses Vorkurses oder WBE gehört. Daher hier nur ein Beispiel (2x steht für doppelte Auflösung, 100w für ein Bild, das 100px breit ist):

```
1 <h1>
2   
5 </h1>
```

Eine weitere Neuerung in HTML5 sind die Elemente `figure` und `figcaption`. Damit hat man eine gute Möglichkeit, Bild und Bildunterschrift zusammenzufassen:

```

1 <figure>
2   
4   <figcaption>Bubbles at work</figcaption>
5 </figure>
```

Ein **figure**-Element muss nicht unbedingt ein **img** enthalten. Es könnte ebenso ein Video sein oder etwas, das eine Grafik erzeugt, zum Beispiel ein **canvas**-Element. Oder irgendwas anderes, das zusammen gehört und eine Beschriftung bekommen soll, wie zum Beispiel ein Gedicht.

## Pixelbasierte Bild- und Grafikformate

Pixelgrafiken (auch: Rastergrafiken) bestehen aus Bildpunkten (Pixeln), denen jeweils eine Farbe zugeordnet ist. Die im Web einsetzbaren pixelbasierten Formate sind: GIF, JPEG und PNG. Diese drei Formate werden von allen grafischen Browsern unterstützt.

**GIF** steht für *Graphics Interchange Format*. Die wichtigsten Merkmale:

- Content-type: `image/gif`
- Bis zu 256 Farben möglich, Farbtiefe aber auch kleiner wählbar
- Ein Wert kann als transparent definiert werden
- Verlustfreie Komprimierung (LZW)
- Animation möglich (*animated gif*)

GIF eignet sich schlecht für Fotos mit vielen Farben und Farbübergängen. Geeignet ist es dagegen für Grafiken mit wenig Farben (und “harten” Übergängen), Logos, Illustrationen, Cartoons. Da keine abgestufte Transparenz möglich ist, eignet sich GIF auch nicht, um ein freigestelltes Objekt vor einen Hintergrund zu stellen (Abbildung 38).

GIF wird heute nur noch selten verwendet. Unter anderem wegen Patentstreitigkeiten, die mittlerweile gelöst sind (alle fraglichen Patente sind ausgelaufen), wurde es nach und nach durch das freie und flexiblere **PNG** ersetzt.

**JPEG** ist die Abkürzung für *Joint Photographic Expert Group*. Merkmale von JPEG-Bildern:

- Content-type: `image/jpeg`
- 24 Bit Farbtiefe
- Verschiedene Komprimierungsstufen
- Verlustbehaftete Komprimierung

JPEG eignet sich sehr gut für Bilder mit vielen Farben (und “weichen” Übergängen). Im Vergleich zu verlustfreien Verfahren kann bei JPEG auch mit deutlich kleineren Dateien noch eine gute Qualität erreicht werden. JPEG eignet sich dagegen nicht für Grafiken mit scharfen Farbübergängen, da dort mit zunehmender Komprimierung vermehrt so genannte Kompressionsartefakte die Qualität verschlechtern.

Im Beispiel sieht man, dass die wenigen Farben eines GIF-Bilds nicht ausreichen, um den Farbverlauf des Himmels abzubilden. Das JPEG-Bild ist besser und die Datei kleiner (Abbildung 39).

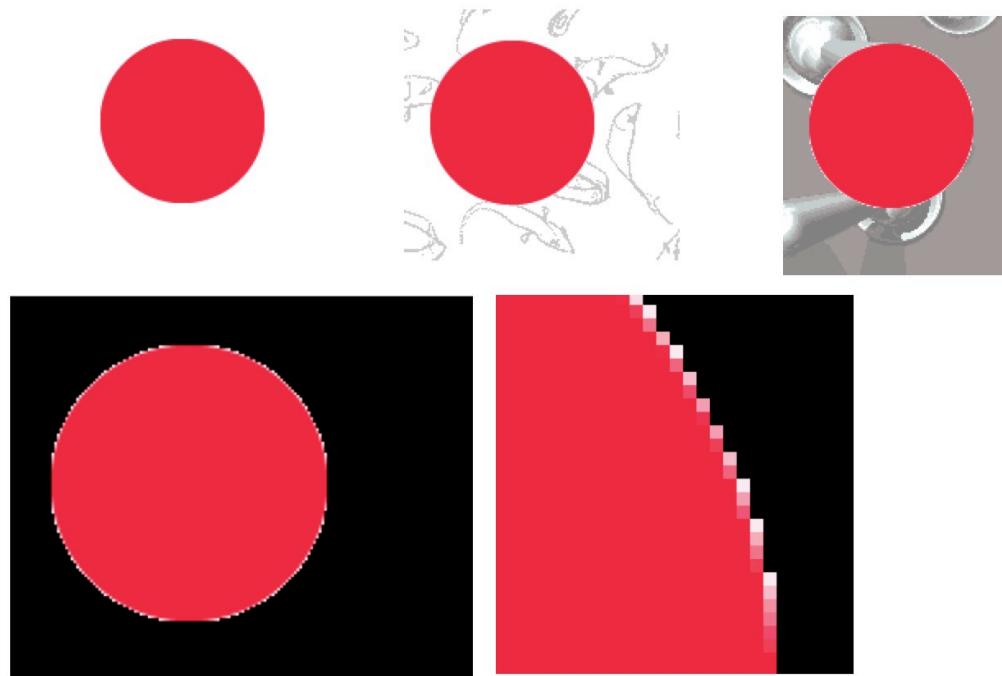


Abbildung 38: GIF mit Transparenz vor einem Hintergrund



Abbildung 39: GIF und JPEG im Vergleich

**PNG** steht für Portable Network Graphic. Seine Eigenschaften:

- Content-type: `image/png`
- Für den Einsatz im Web konzipiert
- Verlustfreie Komprimierung
- Verschiedene Farbtiefen (bis 24 Bit)
- Abgestufte Transparenz: 8 Bit Alpha-Kanal

Die angegebenen Grenzwerte sind die üblicherweise eingesetzten und in Programmen verfügbaren Werte. Theoretisch sind in PNG Farbtiefen bis  $3 \times 16$  Bit und ein Alphakanal mit 16 Bit möglich.

PNG ist ein guter Ersatz für GIF: Mit 8 Bit Farbtiefe sind beide fast identisch. Es ist also gut geeignet für Grafiken, Logos, oder Illustrationen. PNG eignet sich außerdem gut in Situationen, in denen ein Alpha-Kanal benötigt wird, um also zum Beispiel ein Bild vor einem Hintergrund zu platzieren.

Das PNG eine verlustfreie Komprimierung verwendet, führen Bilder mit vielen Farben und Farbverläufen zu deutlich grösseren Dateien als mit JPEG. Für diesen Zweck ist also fast immer JPEG vorzuziehen (Abbildung 40).



Abbildung 40: JPEG und PNG im Vergleich

## Vektorgrafiken

Im Gegensatz zu Pixelgrafiken sind Vektorgrafiken aus grafischen Elementen wie Linien, Kreisen, Polygonen oder Kurven zusammengesetzt. Vektorgrafiken sind beliebig skalierbar. Beispiele sind das Grafik- und Animationsformat von Flash (`.swf`), Encapsulated PostScript (`.eps`) und Scalable Vector Graphics (`.svg`).

**SVG** wurde speziell für den Einsatz im Web konzipiert. Eine kurze Beschreibung dieses Formats und wie solche Vektorgrafiken in die HTML-Datei eingebunden werden können folgt im nächsten Kapitel.

## Audio und Video

Auf die Themen Audio und Video gehen wir hier nur kurz ein. Bei Bedarf finden Sie Material zur Vertiefung in der Verweisliste am Ende des Kapitels.

Um Audio oder Video wiederzugeben, musste man früher auf Browser-Plugins zurückgreifen, da keine einheitliche Unterstützung von Formaten in die Browser eingebaut war. Meist verwendete man hierzu **Flash** (früher Macromedia, später Adobe), da das Flash-Plugin in vielen Browersn installiert war. Flash hat im Laufe der Zeit Jahren an Bedeutung verloren und wird heute kaum noch eingesetzt. Einerseits musste das Flash-Plugin wegen Sicherheitslücken häufig aktualisiert werden, andererseits waren viele Web-Nutzer verärgert über die vielen in Flash realisierten animierten Werbebanner. Vielleicht der Hauptgrund für das schwindende Interesse an Flash sind aber Probleme bei der Unterstützung von Mobilgeräten: IOS-Geräte unterstützen Flash überhaupt nicht und auf anderen Plattformen sorgte Flash immer wieder für Probleme, zum Beispiel verkürzte Akkulaufzeiten.

In HTML5 gibt es zum Einbetten von Audio und Video die neuen Elemente **audio** und **video**:

```
1 <video src="video.ogv" controls poster="poster.jpg" width="320" height="240">
2   <a href="video.ogv">Download movie</a>
3 </video>
4
5 <audio src="music.oga" controls>
6   <a href="music.oga">Download song</a>
7 </audio>
```

Das Problem ist: Bisher gibt es keine Einigung darüber, welche Video- und Audio-Formate unterstützt werden sollen. Daher müssen Videos weiterhin in verschiedenen Formaten angeboten werden.<sup>30</sup>

## Formulare

Formulare sind die Grundlage für Interaktion im Web: Eingabefelder können ausgefüllt werden, man kann aus einer Reihe von Menüeinträgen auswählen, Checkboxen aktivieren oder deaktivieren usw. Auf den Inhalt oder den Zustand der Formularelemente kann dann per JavaScript zugegriffen werden oder der aktuelle Formularinhalt wird durch ein **Submit**-Kommando an einen Server geschickt und dort verarbeitet.

Formulare werden in WBE behandelt, sie sollen daher hier nur kurz angesprochen werden.

Ein Formular besteht in HTML aus dem Element **form**, das als Inhalt eine Reihe von Formularelementen aufnehmen kann. Hier ist ein Beispiel:

---

<sup>30</sup>Ein Überblick zur Unterstützung des **video**-Tags finden Sie unter <https://caniuse.com/video>, unter *Sub-features* finden Sie Angaben zu verschiedenen Video-Formaten.

```

1 <form action="auswerten.php" method="post">
2   <p>
3     <label for="vornameid">Vorname:</label>
4     <input type="text" name="vorname" id="vornameid">
5     <label for="aboid">Newsletter abonnieren:</label>
6     <input type="checkbox" name="abo" id="aboid">
7     <input type="submit" value="Absenden">
8   </p>
9 </form>

```

Die Abbildung 41 zeigt die Standard-Darstellung dieses Formulars im Browser.

Abbildung 41: Beispiel für ein Formular

Das **action**-Attribut des **form**-Elements gibt die Seite auf dem Server an, an die der HTTP-Request mit den Formulardaten geschickt werden soll. Das ist normalerweise ein serverseitiges Programm oder Script, das die Formulardaten entgegennimmt und verarbeitet.

Das **method**-Attribut gibt das zu verwendende HTTP-Kommando an. Es kann hier zwei Werte annehmen:

- GET: Dient eigentlich zum Holen einer Seite vom Server; dazu wird die gewünschte URL an den Server geschickt, die Formulardaten werden an die URL angehängt, zum Beispiel:  
`http://www.google.com/search?client=safari&q=http`
- POST: Die Formulardaten werden im Body der HTTP-Anfrage geschickt (keine Einschränkung bzgl. Umfang und Art der Daten, nicht sichtbar in Adresszeile des Browsers).

GET kann in bestimmten Situationen sinnvoll sein, wenn die angefragte Seite plus die abgeschickten Formulardaten eine (virtuelle) Ressource im Web adressieren. Das ist zum Beispiel bei einer Suchanfrage der Fall. So kann die gesamte Anfrage auch als Lesezeichen im Browser abgespeichert werden.

In der Regel ist aber POST vorzuziehen.

In HTML4 gibt es eine Reihe von Formularelementen: **input**, **textarea**, **button**, **select** (in Kombination mit **optgroup**, **option**). Die meisten Varianten bietet das **input**-Element mit verschiedenen Werten des **type**-Attributs (Abbildung 42). Details zur Verwendung können Sie in der HTML-Spezifikation nachsehen.

**Passwortfeld**

**Mehrzeiliges Textfeld  
(textarea)**

**Button**

**Datei-Upload**

**Auswahlliste**

**Radio-Buttons**

**Checkboxes**

Abbildung 42: HTML4 Formularelemente

In HTML5 gibt es zahlreiche Erweiterungen bei den Formularen, zum Beispiel:

- placeholder-Attribut
- autofocus-Attribut
- required-Attribut
- autocomplete-Attribut
- list-Attribut und **datalist**-Element

Ausserdem neue Möglichkeiten für das type-Attribut des input-Elements: **search**, **email**, **url**, **tel**, **range**, **number**, **date**, **datetime**, **color**, **pattern**.

Mehr zu Formularen dann in WBE.

# Quellen und Verweise

## Material zur Vertiefung

- **Video On The Web:** Video in a Flash (Without That Other Thing) (Mark Pilgrim, Dive Into HTML5)  
<http://diveintohtml5.info/video.html>
- **A Form Of Madness** (Mark Pilgrim, Dive Into HTML5)  
<http://diveintohtml5.info/forms.html>

## Referenzen

- **DevDocs: HTML**  
<https://devdocs.io/html/>
- **htmlreference.io**  
<https://htmlreference.io>
- **HTML Living Standard** (WHATWG)  
<https://html.spec.whatwg.org/multipage/>

## Quellenangaben

- **HTML5 For Web Designers** (Jeremy Keith, A Book Apart)  
<http://www.abookapart.com/products/html5-for-web-designers>
- Ein Teil der Bilder und Inhalte stammt aus früheren Folien der Fächer WBD (Webdesign), INSY (Informationssysteme) und WEBT (Webtechnologien).

## Weitere Tipps

- **HTML Tutorial von W3Schools:** W3Schools hat nichts mit dem W3C zu tun, auch wenn der Name dies nahe legt. Auf der Website gibt es zahlreiche Tutorials, nicht nur zu HTML.  
<http://www.w3schools.com/html/>
- **Can I use...** - Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers  
<http://caniuse.com>

# Markup und HTML (Teil 3)

## Einführung und Ziele

In dieser Lektion werden wir uns noch etwas genauer mit den Neuerungen, welche mit HTML5 Einzug gehalten haben, beschäftigen. Auch wenn die Bezeichnung heute *HTML Living Standard* ist, verwenden wir hier meistens *HTML5*, da es besonders um die Unterschiede zu den früheren Versionen geht.

Den Aufbau eines HTML5-Dokuments kennen Sie ja bereits. Nach ein paar grundlegenden Überlegungen zum HTML5-Design wird auf die Strukturierung eines Dokuments mittels Überschriften und Abschnitten eingegangen. Anschliessend werden noch einige neue Elemente vorgestellt. Eine Beschreibung, wie Vektorgrafiken und mathematische Formeln in ein HTML5-Dokument eingebettet werden können, beschliesst die Lektion.

## Ziele

- Sie verstehen die unterschiedlichen Herangehensweisen, die der Spezifikation von HTML4 und HTML5 zugrunde liegen.
- Sie wissen, wie sich HTML4 und HTML5 in der Interpretation der Überschriftenebenen unterscheiden.
- Sie kennen *die neuen HTML5-Elemente mark, time, meter und progress*, sowie die Funktion des Attributs `contenteditable`.
- *Sie wissen, was Microformats sind und wie sie eingesetzt werden können.*
- *Sie kennen ausgewählte HTML5-APIs und wissen, wozu sie eingesetzt werden.*
- Die XML-Anwendungen SVG und MathML sind Ihnen bekannt. Sie wissen, wozu und wie diese Formate auf Webseiten eingesetzt werden können.

## Relevanz für WBE

Der Abschnitt *Design von HTML5* ist für WBE fakultativ, ebenso die folgenden Unterabschnitte zu HTML5: *Microformats* und *HTML5 APIs*. Wichtig sind hingegen die Möglichkeiten, Überschriften anzugeben, das Attribut `contenteditable`, sowie Grundwissen zu SVG (MathML ist weniger wichtig).

## Design von HTML5

Im Rahmen der Französischen Revolution wurde 1793 die Dezimalzeit per Dekret eingeführt. Der Tag wurde in zehn Stunden aufgeteilt, die Stunde in 100 Minuten, und diese wiederum in 100 Sekunden. Das war vielleicht gut durchdacht, aber es passte nicht zu bestehenden Gewohnheiten. Und daher konnte sich das Vorhaben auch nicht durchsetzen. Einige Uhren aus der Zeit zeigten zwar beide Uhrzeiten an, dennoch wurde die Dezimalzeit bereits 1795 als gültige Zeiteinheit wieder abgeschafft.

Ähnlich wie die Dezimalzeit wäre XHTML2 sicher ein gut durchdachter Standard gewesen, in vieler Hinsicht besser, als das zuvor verwendete HTML4 und die ersten XHTML-Versionen. Auch XHTML2 scheiterte, weil es nicht mit bestehenden Gewohnheiten vereinbar war und HTML – zumindest teilweise – neu erfinden wollte.<sup>31</sup>

Die Webentwickler, die sich in der WHATWG zusammengeschlossen haben, wählten einen anderen Weg. Ausgehend vom aktuellen Stand sollten *Erweiterungen* beschrieben werden. Das Design von HTML5 folgte (unter anderen) diesen Prinzipien:

- *Support existing content.*
- *Do not reinvent the wheel.*
- *Pave the cowpaths.*

So ist es zu verstehen, dass praktisch alle Sprachelemente aus HTML4 auch noch in HTML5 existieren. Außerdem sollten Lösungen, die im Webdesign weit verbreitet sind, von HTML5 unterstützt werden. Es wurde also ein pragmatischer Weg gesucht: Erst einmal beschreiben, was die Browser aktuell können, und neue Features hinzufügen, wenn sie nützlich sind. Diese Einstellung gibt auch das folgende Designprinzip wieder:

- *Priority of constituencies: In case of conflict, consider users over authors over implementers over specifiers over theoretical purity.*

Dieser Ansatz führt auch dazu, dass die HTML5-Spezifikation nicht nur beschreibt, wie korrektes HTML5 aufgebaut sein muss. Sie beschreibt auch, wie Browser mit fehlerhaftem Markup umgehen sollen. Das ist natürlich eine gewaltige Aufgabe, die zu einer umfangreichen Spezifikation führt. Und auch dazu, dass die Spezifikation nicht immer leicht zu lesen ist, da sie sich nicht nur an Webdesigner sondern auch an Browser-Entwickler richtet.

Nehmen wir den Doctype als Beispiel. Sie haben ja bereits die Doctypes von HTML4 und XHTML gesehen: Eine komplizierte Zeile, die man sich nicht merken konnte und daher jeweils aus anderen Dokumenten oder der Spezifikation zu kopieren hatte. Das war aber noch nicht das grösste Problem. Es hat sich gezeigt, dass die präzisen Doctype-Angaben gar nicht viel gebracht haben. So wurde zum Beispiel oft ein XHTML-Doctype verwendet, die Dokumente aber trotzdem von den Browsern als HTML-Dokumente verarbeitet, um Fehlermeldungen zu vermeiden. Außerdem mussten noch mit einem *Standards Mode* und teilweise zusätzlich mit einem *Almost Standards Mode* verschiedene Interpretationen bestimmter Webseitenelemente aktiviert werden.

---

<sup>31</sup>Der Vergleich mit der Dezimalzeit stammt aus dem Buch *HTML5 For Web Designers*: <https://abookapart.com/products/html5-for-web-designers>

Die pragmatische Lösung der WHATWG für die Doctype-Angabe:

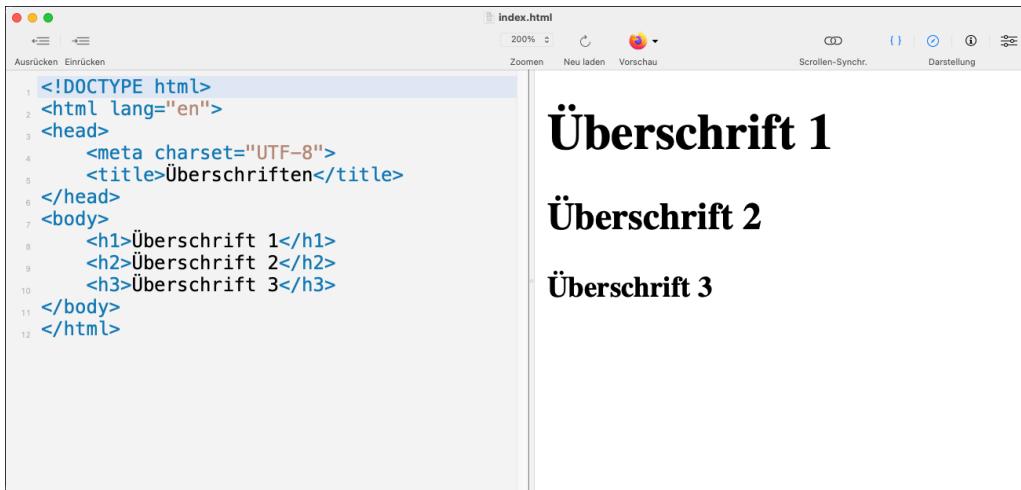
```
<!DOCTYPE html>
```

Damit ist gar keine Versionsnummer mehr enthalten. Konsequenterweise benutzt die WHATWG mittlerweile auch die Bezeichnung *HTML – Living Standard*, während das W3C die Versionszählung 5.0, 5.1 usw. noch einige Zeit weitergeführt und dann auch aufgegeben hat.

In früheren HTML-Spezifikationen wurden Elemente, die nicht mehr verwendet werden sollten, als *deprecated* (veraltet, überholt) bezeichnet. In HTML5 wird statt *deprecated* die Bezeichnung *obsolete* verwendet. Für Webdesigner spielt es keine Rolle, ob ein Element oder Attribut veraltet oder obsolet ist: Es soll nicht mehr verwendet werden. Browserhersteller müssen aber auch obsolete Elemente und Attribute unterstützen, da auch ältere Dokumente sich auf eine definierte Art und Weise verhalten sollen.

## Überschriften und Abschnitte in HTML5

Bis HTML4 war das Thema Überschriften ziemlich einfach abzuhandeln: Es gibt sechs verschiedene Elemente h1 bis h6 für die verschiedenen Überschriftenebenen. h1 ist die Hauptüberschrift, die weiteren Elemente können als untergeordnete Überschriften verwendet werden. Soweit ist es bereits aus einer früheren Lektion bekannt (Abbildung 43).



The screenshot shows a browser window with the title "index.html". The left pane displays the HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>Überschriften</title>
6   </head>
7   <body>
8     <h1>Überschrift 1</h1>
9     <h2>Überschrift 2</h2>
10    <h3>Überschrift 3</h3>
11  </body>
12 </html>
```

The right pane shows the rendered content with three levels of headings:

# Überschrift 1

## Überschrift 2

### Überschrift 3

Abbildung 43: Überschriften

**Wichtig:** Die oberste Überschrift in h1, die nächste h2, usw. Die Elemente dürfen nicht nach der Grösse der Standarddarstellung ausgewählt werden, es geht nur um die Bedeutung. Die Grösse, in der eine Überschrift im Browser erscheint, wird mit Hilfe von CSS festgelegt.

Das Problem mit den sechs möglichen Überschriften in HTML4 ist, dass diese Ebenen global für das ganze Dokument gelten, unabhängig davon, wie tief in der hierarchischen

Struktur des Dokuments die Überschrift liegt. Das kann zu Schwierigkeiten führen, wenn Teile des Dokuments aus anderen Quellen eingebettet werden, welche eine eigene Überschriftenhierarchie haben, die möglicherweise selbst mit `h1` beginnt.

In HTML5 hat man versucht, dieses Problem zu lösen. Leider ist die Lösung ziemlich kompliziert geraten und funktioniert bisher in den Browsern noch nicht vollständig.

Bestimmte Elemente definieren *Sectioning Content*, einen eigenen Geltungsbereich für Überschriften. Dies sind: `section`, `article`, `aside` und `nav`. Die oberste Überschrift eines solchen Elements ist jeweils der Ausgangspunkt einer eigenen Überschriftenhierarchie, und zwar auf einer Ebene, die von der Umgebung des Elements definiert wird. Dabei ist es unerheblich, welches Überschriftenelement verwendet wird. Es kann also überall `h1` verwendet werden. Die Ebene wird dann aus der Dokumentenhierarchie abgeleitet (Beispiel aus: *HTML 5 For Web Designers*):

```
1 <h1>An Event Apart</h1>
2 <section>
3   <header>
4     <h1>Cities</h1>
5   </header>
6   <p>Join us in these cities in 2010.</p>
7   <section>
8     <header>
9       <h1>Seattle</h1>
10    </header>
11    <p>Follow the yellow brick road.</p>
12  </section>
13  <section>
14    <header>
15      <h1>Boston</h1>
16    </header>
17    <p>That's Beantown to its friends.</p>
18  </section>
19  <section>
20    <header>
21      <h1>Minneapolis</h1>
22    </header>
23    <p>It's so <em>nice</em>.</p>
24  </section>
25 </section>
26 </section>
```

In HTML4 wären alle diese Überschriften auf einer Ebene:

- An Event Apart
- Cities
- Seattle
- Boston
- Minneapolis

In HTML5 müsste die Struktur so interpretiert werden:

- An Event Apart
  - Cities
    - \* Seattle
    - \* Boston
    - \* Minneapolis

Es ist aber noch etwas komplizierter: Die Überschriften `h1...h6`, `hgroup` definieren implizite Sections, wenn sie nicht explizit in einem *Sectioning-Content-Element* enthalten sind. Ausserdem gibt es sogenannte *Sectioning Roots*, die eine unabhängige Überschriftenhierarchie definieren. Das sind die Elemente: `blockquote`, `body`, `details`, `dialog`, `fieldset`, `figure`, `td`.

Dies alles soll hier nicht weiter vertieft werden, da es in den Browsern noch nicht komplett umgesetzt ist. Interessierte finden weitere Informationen in der Linkssammlung am Ende des Kapitels.

## Mehr zu HTML5

In diesem Abschnitt gehen wir noch auf einige weitere Möglichkeiten ein, die HTML5 im Gegensatz zu den Vorgängerversionen anbietet.

### Neue Inline-Elemente

Sie haben bereits einige neue HTML5-Elemente kennengelernt, mit denen sich ein Dokument nach inhaltlichen Kriterien strukturieren lässt: `section`, `article`, `header`, usw. Das sind so genannte *Block-level-Elemente*. Ausserdem haben wir ein paar *Inline-Elemente* (Elemente mit *Text-level semantics*) angesehen: `strong`, `em`, `b`, `i`, und `span`. Diese Inline-Elemente existierten bereits in HTML4. In HTML5 sind noch ein paar Elemente zur Auszeichnung von Textabschnitten dazu gekommen, unter anderem sind dies: `mark`, `time`, `meter` und `progress`.

#### **mark**

`mark` haben wir ja bereits erwähnt. Es dient dazu, Textabschnitte hervorzuheben. Im Gegensatz zu `strong` oder `em` soll es aber nicht die Wichtigkeit des Ausschnitts erhöhen. Es drückt nur die Bedeutung in einem bestimmten Kontext aus, ähnlich wie man mit einem Textmarker Bereiche eines gedruckten Dokuments markiert. Daher werden in der Standarddarstellung von mit `mark` ausgezeichneten Textstellen diese auch mit einem gelben Hintergrund versehen. Ein Beispiel für den Einsatz von `mark` ist das Markieren von Suchergebnissen im Suchresultat.

#### **time**

In HTML5 gibt es ein spezielles Element, um Zeitangaben zu definieren: `time`. Es enthält entweder eine Zeitangabe in standardisierter Form oder ein `datetime`-Attribut mit standardisierter Zeitangabe:

```

1 <time>2012-07-17</time>
2 <time datetime="17:00">5pm</time>
3 <time datetime="2012-07-17">July 17th</time>
4 <time datetime="2012-07-17T17:00">5pm on July 17th</time>

```

Das `time`-Element kann übrigens auch problemlos mit den Microformat-Angaben (s.u.) kombiniert werden.

### **meter und progress**

Das `meter`-Element dient dazu, Werte (zum Beispiel Messwerte) zu beschreiben. Dazu gibt es verschiedene Attribute wie `low`, `high`, `min`, `max`, `optimum` und `value`. Der eigentliche Wert kann im `value`-Attribut oder als Inhalt des `meter`-Elements stehen.

Das `progress`-Element zeichnet Werte aus, die sich ändern, zum Beispiel die Angabe, wie viel Prozent einer Datei bereits heruntergeladen sind. Auch hier gibt es `min`-, `max`- und `value`-Attribute. Da sich ein Wert in einem HTML-Dokument nicht einfach ändert, wird dieses Element normalerweise dynamisch mit JavaScript aktualisiert.

Die beiden Elemente werden in grafischen Browsern mit verschiedenen Balkengrafiken dargestellt:

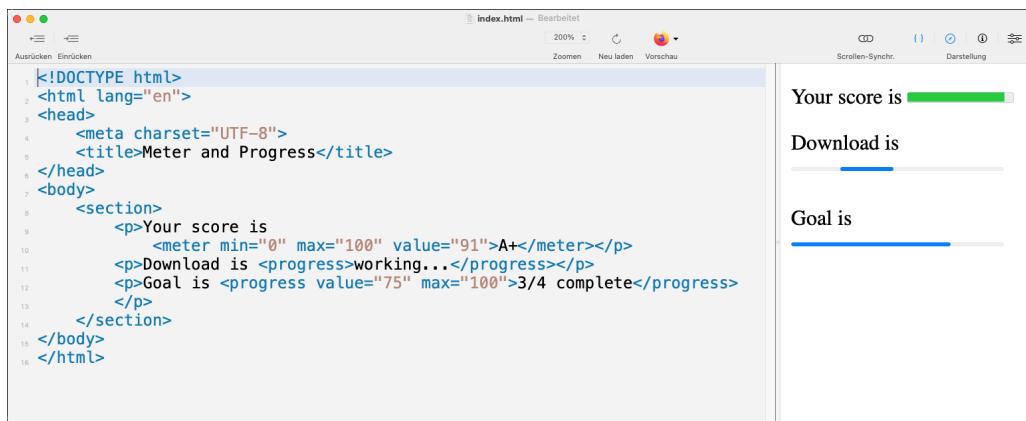


Abbildung 44: Werteangaben mit `meter` und `progress`

### **Microformats**

In HTML gab es vor HTML5 keine Möglichkeit, einen Text als Zeitangabe auszuzeichnen. Nun steht dafür das `time`-Element zur Verfügung. Häufig wünscht man sich aber weitere Ausdrucksmittel, um zum Beispiel Termine oder Kontaktdaten auszuzeichnen. Selbstverständlich lässt sich das jederzeit mit Hilfe etwa von `span`-Elementen mit bestimmten `class`-Attributen erreichen. Nur wenn diese aber einheitlich eingesetzt werden, können die Informationen auch automatisch aus den Seiten extrahiert und speziell ausgewertet werden.

Hier kommt eine als **Microformats** bezeichnete Technik ins Spiel: Man definiert eine Menge von Konventionen, wie man das `class`-Attribut zur näheren Beschreibung bestimmter Textabschnitte einsetzen kann. So entstanden diverse Microformats, zum Beispiel *hCard* für Kontaktangaben oder *hCalendar* für Termine.<sup>32</sup>

Beispiel für die Beschreibung eines Termins:

```
1 <span class="vevent">
2   <span class="summary">The microformats.org site was launched</span>
3   on <span class="dtstart">2005-06-20</span>
4   at the Supernova Conference
5   in <span class="location">San Francisco, CA, USA</span>.
6 </span>
```

Solche Microformats können dann automatisch verarbeitet werden. Vor einiger Zeit gab es eine Erweiterung namens *Operator Toolbar* für den Firefox, welche Microformats auf der Seite in einer Toolbar sichtbar gemacht und es unter anderem erlaubt hat, Adressen in die Kontaktliste und Termine in den Kalender zu übernehmen. Leider hat diese eigentlich gute Idee nicht viel Anklang gefunden.

Die aktuelle HTML-Spezifikation der WHATWG greift die Idee der Microformats in einem eigenen Kapitel *Microdata* auf und ergänzt HTML zu diesem Zweck um einige Attribute, unter anderem `itemprop`, eine spezifischere Variante für `class`. Das W3C stand dieser Entwicklung skeptisch gegenüber und hat dieses Kapitel nicht in die eigenen HTML5-Versionen übernommen. Ähnlich wie bei den Microformats sieht es nicht so aus, als würde Microdata auf Interesse der Webentwickler und Browser-Hersteller stossen (Abbildung 45).

## Microdata API

This API is **not supported** in any modern browser, nor are there any known plans to support it.

For more information see:

- [Specification](#)
- [Article: The Downward Spiral of Microdata](#)

Abbildung 45: Browser-Unterstützung von `microdata` (caniuse.com)

## Attribut `contenteditable`

Mit dem Attribut `contenteditable` kann man einen Bereich des Dokuments im Browser editierbar machen. Die Änderungen werden jedoch nur in der lokalen Seitenstruktur im Browser wirksam. Sollen diese dauerhaft gespeichert oder zum Server übertragen werden, muss dies mit JavaScript gelöst werden.

<sup>32</sup><http://microformats.org>

## Selbst definierte data-Attribute

In HTML5 können HTML-Elemente mit Hilfe von **data**-Attributen mit anwendungsspezifischen Daten angereichert werden. Dabei ist festgelegt, dass solche Attribute mit **data**-beginnen müssen:

```
<li data-length="2m11s">Beyond The Sea</li>
```

Auf die Werte der **data**-Attribute kann über JavaScript zugegriffen werden.

## HTML5 APIs

Bisher haben wir uns schwerpunktmaßig mit dem Thema *Markup* auseinandergesetzt. HTML5 wurde aber nicht nur als Markup-Sprache, sondern als Sammlung von Technologien rund um die moderne Web-Entwicklung verstanden. Je nach Interpretation werden zu HTML5 auch aktuelle Entwicklungen im Bereich der Stylesheets (CSS3) und verschiedene JavaScript-APIs<sup>33</sup> gezählt.

CSS wird in den folgenden Kapiteln noch ausführlich behandelt. JavaScript ist Hauptthema des Kurses WBE. Hier stellen wir nur einen kurzen Überblick über ein paar ausgewählte Schnittstellen für JavaScript-Programme zusammen, die im Zuge der Entwicklung von HTML5 eingeführt wurden. Nicht immer ist ganz klar, ob diese APIs eigentlicher Bestandteil von HTML5 oder eigenständige Spezifikationen sind (bzw. ändert sich das auch gelegentlich).

- **canvas-Element:** Dieses neue Element definiert eine Zeichenfläche für dynamische Bitmap-Grafiken, zum Beispiel für Grafiken, Diagramme, oder Spiele. Zum Erzeugen der Grafiken gibt es verschiedene JavaScript-Funktionen.
- **Web Storage:** Schlüssel-Wert-Paare werden in einer einfachen Datenbank auf dem Client gespeichert. Damit hat man deutlich mehr Möglichkeiten zur Verfügung, lokal Informationen zu speichern, als dies mit Cookies möglich wäre.
- **Application Cache:** Teile einer Applikation (spezifizierbar) werden lokal gespeichert, so dass die Anwendung auch *offline* funktioniert. Diese Spezifikation ist bereits wieder obsolet und mittlerweile durch das *Cache Interface* der *Service Worker API* ersetzt, eine Voraussetzung für *Progressive Web Apps (PWAs)*. Keine Sorge, wenn Ihnen dies alles im Moment nicht viel sagt...
- **Geolocation:** Zugriff auf den aktuellen Standort, wenn Benutzer dies akzeptieren.

Viele dieser APIs sind wichtig für Web-Applikationen auf Mobilgeräten. Da nicht immer mit einer Internet-Verbindung gerechnet werden kann, sollten diese Apps auch offline funktionieren und benötigen dabei natürlich in der Regel auch eine Möglichkeit, Daten lokal zu speichern.

## Was funktioniert bereits?

Eins sollte inzwischen klar geworden sein: Die Frage, ob eine bestimmte Browser-Version HTML5 unterstützt oder nicht, ist eine sinnlose Frage. Man beschäftigt sich also besser

---

<sup>33</sup>API = Application Programming Interface

mit der Frage, ob der Browser, auf dem eine Seite gerade geladen wird, ein bestimmtes Feature unterstützt oder nicht.

Wenn eine Website eine bestimmte Menge an Browsersversionen unterstützen soll, kann man Übersichten wie **Can I use...** konsultieren.<sup>34</sup> Allein die Menge der dort aufgeführten Features und Sub-Features zeigt, wie vielseitig und umfangreich die Themen rund um HTML5 und aktuelle Web-Technologien mittlerweile sind.

Abbildung 46 zeigt die Unterstützung der Web Storage API:

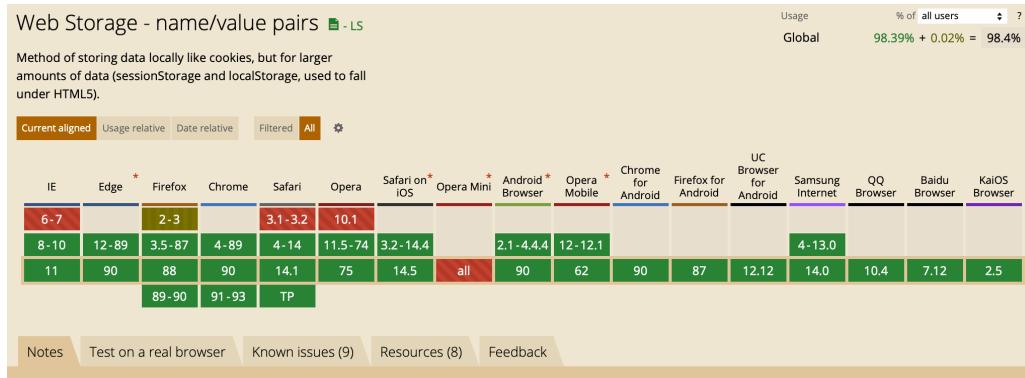


Abbildung 46: Browser-Unterstützung der Web Storage API

Anstatt nur bestimmte Browser-Versionen zu unterstützen ist es natürlich besser (wenn auch meist aufwändiger), dynamisch zu testen, ob ein Browser ein bestimmtes Feature unterstützt. Wenn ja kann es verwendet werden. Wenn nein wird auf eine Fallback-Lösung zurückgegriffen. Die Seite ist dann immer noch nutzbar, wenn auch mit kleinen Einschränkungen. Je besser ein Browser ist, um so besser (attraktiver, dynamischer, benutzerfreundlicher) wird die Seite. Man spricht von **Progressive Enhancement**.

Mit JavaScript werden wir uns in WBE beschäftigen. Hier nur als erster Eindruck, wie man mit Hilfe von JavaScript testen kann, ob ein Browser ein bestimmtes Feature unterstützt. Oder wie in diesem Beispiel: Ob ein Element ein bestimmtes Attribut unterstützt.<sup>35</sup>

```

1 function elementSupportsAttribute (element, attribute) {
2     var test = document.createElement(element);
3     if (attribute in test) {
4         return true;
5     } else {
6         return false;
7     }
8 }
```

Damit kann man testen, ob das `input`-Element das Attribut `placeholder` unterstützt:

<sup>34</sup><http://caniuse.com>

<sup>35</sup>Das Beispiel stammt aus dem Buch *HTML5 For Web Designers*.

```

1  if (!elementSupportsAttribute('input', 'placeholder')) {
2    // JavaScript fallback goes here.
3 }

```

## Vektorgrafiken und Formeln

### SVG

Es wurde bereits gezeigt, wie Bilder in eine Webseite eingebettet werden können. Zu diesem Zweck unterstützen Browser im Wesentlichen drei pixelbasierte Formate: JPEG, PNG und GIF.<sup>36</sup> Gemeinsam ist diesen Formaten, dass die Bilder nicht über die Originalgrösse hinaus skaliert werden können, ohne dass die Qualität leidet. Vektorgrafiken weisen dieses Problem nicht auf. Sie sind aus einfachen Elementen wie Linien, Kreisen, Polygonen oder Kurven aufgebaut. Diese Elemente werden durch ihre Merkmale beschrieben, ein Kreis etwa durch Mittelpunkt, Radius, Liniendicke und Farbe. Das hat zur Folge, dass solche Grafiken gut skalierbar sind, indem die anzuzeigenden Pixel einfach für den betreffenden Skalierungsfaktor neu berechnet werden.

Lange Zeit wurden Vektorgrafiken im Web vor allem durch das proprietäre Flash-Format unterstützt. Mittlerweile hat **SVG** (Scalable Vector Graphics) aber stark an Bedeutung zugelegt. Der Grund ist, dass seit einiger Zeit alle wichtigen Browser eine Unterstützung für SVG anbieten.

SVG ist ein auf XML basierendes Format zur Beschreibung zweidimensionaler Vektorgrafiken. Es ist beschrieben in *Scalable Vector Graphics (SVG) 2* vom 4. Oktober 2018 (W3C Candidate Recommendation). Hier ist ein Beispiel für eine SVG-Datei (Quelle: Wikipedia):

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3    xmlns:xlink="http://www.w3.org/1999/xlink"
4    xmlns:ev="http://www.w3.org/2001/xml-events"
5    version="1.1" baseProfile="full"
6    width="700px" height="400px" viewBox="0 0 700 400">
7
8  <!-- Anschlüsse links und rechts -->
9  <line x1="0" y1="200" x2="700" y2="200" stroke="black" stroke-width="20px" />
10 <!-- Das Rechteck -->
11 <rect x="100" y="100" width="500" height="200" fill="white" stroke="black"
12   stroke-width="20px" />
13 <!-- Der Schleifer -->
14 <line x1="180" y1="370" x2="500" y2="50" stroke="black" stroke-width="15px"/>
15 <!-- Die Pfeilspitze -->
16 <polygon points="585 0 525 25 585 50" transform="rotate(135 525 25)" />
17 </svg>

```

---

<sup>36</sup> Googles Format WebP wird zunehmend von unterstützt (allerdings nicht auf Internet Explorer und auf Safari erst ab MacOS 11 Big Sur). Es verfügt über eine bessere Komprimierung bei vergleichbarer Qualität als JPEG, PNG und GIF.

Es handelt sich um eine XML-Datei mit SVG-Namespace. Das Wurzelement ist `svg`. Die SVG-Elemente `line`, `rect`, `polygon` definieren einzelne Elemente der Grafik, die nähere Beschreibung erfolgt mittels Attributen. Kommentare werden wie in HTML geschrieben. Die Abbildung 47 zeigt die Anzeige der Datei im Safari-Browser.

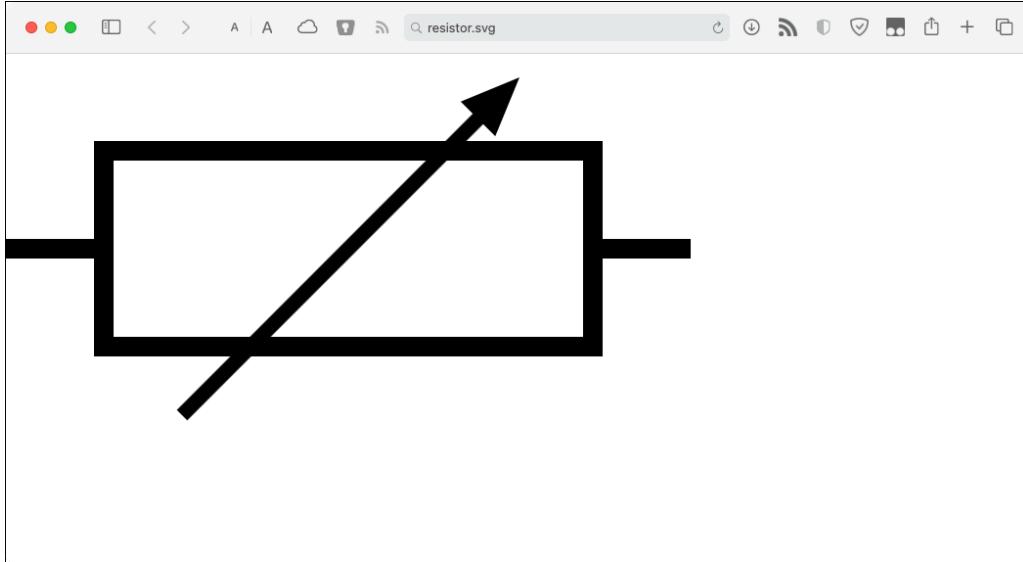


Abbildung 47: SVG-Beispiel im Safari

Zur Beschreibung der Darstellung werden in SVG teilweise Attributnamen verwendet, die CSS-Eigenschaften entsprechen, zum Beispiel `font-family`, `text-decoration` oder `color`. Tatsächlich ist es möglich, die Darstellung einer in SVG beschriebenen Grafik mit CSS-Regeln zu spezifizieren, indem die CSS-Regeln in die SVG-Datei eingefügt werden oder durch Verweis auf eine externe CSS-Datei nach der XML-Deklaration:

```
<?xml-stylesheet href="styles.css" type="text/css"?>
```

SVG fügt sich also gut in bestehende Webtechnologien ein. Es ist auch möglich, SVG-Grafiken dynamisch mit JavaScript anzupassen, und diese Anpassungen an Ereignisse an bestimmten Elementen zu binden.

Viele Vektorgrafikprogramme erlauben es, SVG-Dateien zu importieren und zu exportieren. Eine speziell auf SVG zugeschnittene Software ist Inkscape<sup>37</sup> (Open Source). Für einfache Zwecke reicht auch ein Online-Tool wie SVG-edit<sup>38</sup> aus (Abbildung 48).

Seit HTML5 kann SVG direkt als *Embedded Content* in HTML-Code eingebettet werden. Ein Einbinden über ein `img`- oder ein `object`-Element ist aber auch möglich. Außerdem kann SVG auch mit CSS als Hintergrundbild von Elementen spezifiziert werden.

---

<sup>37</sup><https://inkscape.org>

<sup>38</sup><https://github.com/SVG-Edit/svgedit>

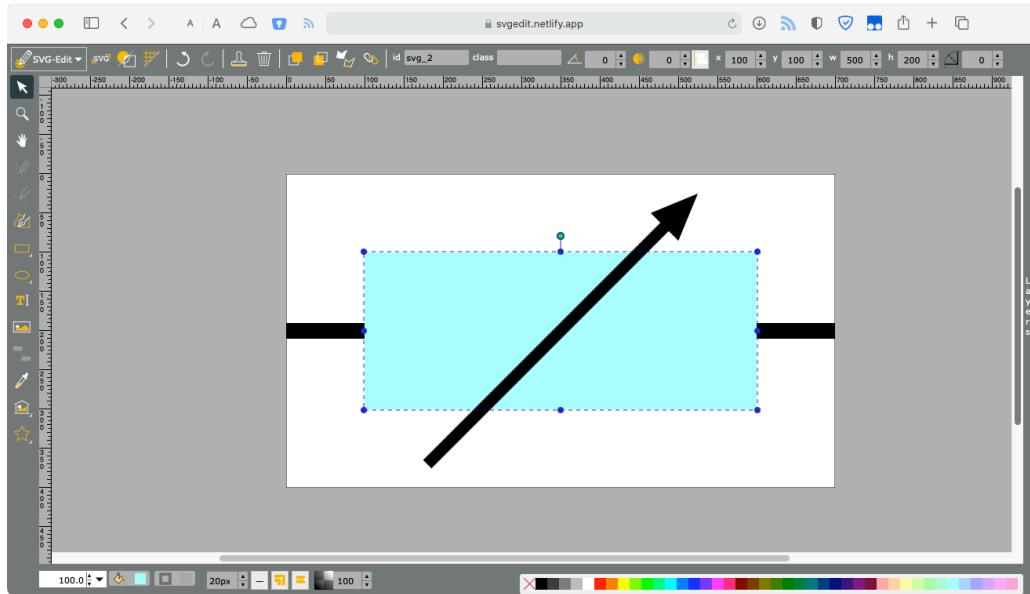


Abbildung 48: SVG-Beispiel in SVG-edit

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>SVG-Demo</title>
5      </head>
6      <body>
7          <h1>Beispiel-Grafik</h1>
8          <p>
9              <svg>
10                 <svg width="100" height="100">
11                     <circle cx="50" cy="50" r="40" stroke="blue" stroke-width="4" fill="yellow" />
12                 </svg>
13             </svg>
14         </p>
15     </body>
16 </html>
```

## MathML

Wie SVG ist auch MathML eine XML-Anwendung. MathML steht für *Mathematical Markup Language* und dient zur Darstellung mathematischer Formeln. Wie SVG können auch MathML-Formeln in HTML5 direkt in die HTML-Datei eingebunden werden. Hier ist ein Beispiel mit der Formel zum Lösen quadratischer Gleichungen:

```

1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <title>Quadratischer Gleichungen</title>
6  </head>
7
8  <body>
9      <h1>Lösung quadratischer Gleichungen</h1>
10     <p>
11         <math display='block'>
12             <mrow>
13                 <mi>x</mi>
14                 <mo>=</mo>
15                 <mfrac>
16                     <mrow>
17                         <mo>&#x2212;</mo><mi>b</mi><mo>&#x00B1;</mo>
18                         <msqrt>
19                             <mrow>
20                                 <msup><mi>b</mi><mn>2</mn></msup>
21                                 <mo>&#x2212;</mo>
22                                 <mn>4</mn><mi>a</mi><mi>c</mi>
23                             </mrow>
24                         </msqrt>
25                     </mrow>
26                     <mrow><mn>2</mn><mi>a</mi></mrow>
27                 </mfrac>
28             </mrow>
29         </math>
30     </p>
31 </body>
32
33 </html>

```

Abbildung 49 zeigt die Darstellung dieser Formel im Browser Safari.

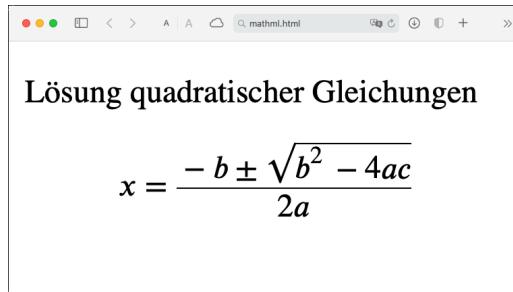


Abbildung 49: MathML im Safari

Im Gegensatz zu SVG kann bei MathML allerdings nicht von einer breiten Browser-Unterstützung ausgegangen werden. Aktuell (2021) wird MathML von Safari und Firefox unterstützt. Google hat die Arbeiten an einer MathML-Unterstützung im Chrome-Browser eingestellt mit der Begründung, dass die Darstellung von MathML-Formeln auch von JavaScript-Bibliotheken übernommen werden kann. Damit werden alle auf Chromium basierenden Browser (Edge, Opera, Vivaldi) keinen MathML-Support erhalten.

Tatsächlich existiert mit *MathJax*<sup>39</sup> eine Bibliothek, mit der eine browser-übergreifende Darstellung mathematischer Formeln möglich ist. *MathJax* unterstützt Formeln in verschiedenen Formaten (unter anderem MathML und LaTeX) und verwendet für die Darstellung wahlweise HTML und CSS oder SVG.

---

<sup>39</sup><https://www.mathjax.org>

# Quellen und Verweise

## Material zur Vertiefung

- **Headings and sections** (HTML5 Recommendation W3C)  
<http://www.w3.org/TR/html5/sections.html#headings-and-sections>
- **Sections and Outlines of an HTML5 Document** (MDN)  
[https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Sections\\_and\\_Outlines\\_of\\_an\\_HTML5\\_document](https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Sections_and_Outlines_of_an_HTML5_document)
- **Dive Into HTML5** (Mark Pilgrim)  
<http://diveintohtml5.info>

## Spezifikationen und Standards:

- **HTML 4.01 Recommendation** (W3C)  
<http://www.w3.org/TR/html401/>
- **HTML Living Standard** (WHATWG)  
<https://html.spec.whatwg.org/multipage/>
- **Scalable Vector Graphics** (SVG) 2  
<https://www.w3.org/TR/SVG2/>
- **Mathematical Markup Language** (MathML) Version 3.0 2nd Edition  
<http://www.w3.org/TR/MathML/>

## Quellen

- **HTML5 For Web Designers** (Jeremy Keith, A Book Apart)  
<http://www.abookapart.com/products/html5-for-web-designers>

# Darstellung mit CSS (Teil 1)

## Einführung und Ziele

Bisher haben wir uns vor allem um den Inhalt und die Struktur von Dokumenten gekümmert und wie die Bedeutung der Elemente mit Hilfe von HTML in den Dokumenten beschrieben werden kann. Nun werden wir uns auch um den Darstellungsaspekt kümmern: Mit Hilfe von CSS (Cascading Style Sheets) wird das Erscheinungsbild von Webseiten beschrieben. Gegenstand dieses Kapitels ist eine Einführung in CSS.

### Ziele

- Sie wissen, welche Rolle CSS im Webdesign spielt.
- Sie kennen die verschiedenen Varianten, CSS mit HTML zu verbinden.
- Sie wissen, wie mit Hilfe von CSS-Regeln die Darstellung von Seitenelementen eines HTML-Dokuments beschrieben wird.
- Sie kennen die CSS2-Selektoren und wissen, wie diese sinnvoll eingesetzt und kombiniert werden können.
- *Sie kennen ausserdem einige in CSS3 neu eingeführte Selektoren.*
- Sie wissen, wie Farben und Größen in CSS angegeben werden können.
- Sie kennen grundlegende Angaben zur Textdarstellung und wissen, wie Listen formatiert werden können.
- *Sie wissen, welche Arten von Stylesheets einen Einfluss auf die Darstellung von Seitenelementen haben und welche Prioritäten dabei gelten.*
- Sie kennen den W3C-Validator für Stylesheets und die Möglichkeit, CSS-Regeln in den Entwicklertools der Browser anzeigen und untersuchen zu können.

### Relevanz für WBE

Die meisten Abschnitte in diesem einführenden Kapitel zu CSS sind fürs Verständnis von CSS wichtig und werden damit auch in WBE vorausgesetzt. Als fakultativ betrachten können Sie die CSS3-Selektoren (die kann man bei Bedarf nachschlagen) sowie das Zusammenwirken verschiedener Stylesheets (das ist fürs Webdesign wichtig, wird in WBE aber nicht benötigt).

# CSS im Überblick

## Wozu CSS?

Aus den vorangehenden Kapiteln wissen wir bereits: Mit HTML wird die logische Struktur und der Inhalt eines Dokuments unabhängig von der Darstellung beschrieben. HT steht für Hypertext, d.h. die Möglichkeit, Dokumente über Links miteinander zu vernetzen. ML steht für Markup Language (Auszeichnungssprache), eine Sprache, um Inhalte um eine logische Struktur zu ergänzen:

```
1 <!-- ohne Markup: -->
2 Das ist eine Überschrift
3 <!-- mit Markup: -->
4 <h1>Das ist eine Überschrift</h1>
```

CSS fügt nun den Darstellungsaspekt hinzu. Eine Website soll ja nicht nur ihren Inhalt und die logische Struktur zeigen, sondern auch noch schön aussehen: Ansprechend gestalteter Seitenaufbau, gut lesbare Texte, problemlos navigierbare Menüs usw. Diese Trennung von Inhalt und Darstellung hat eine Reihe von Vorteilen:

- Auf diese Weise ist es einfacher, ein breites Spektrum von User Agents zu unterstützen, die jeweils ganz unterschiedliche “Darstellungen” benötigen.
- Es ist einfacher, das Aussehen einer ganzen Website, bestehend aus vielen Einzelseiten, anzupassen, wenn die Gestaltung nicht über alle Seiten verteilt sondern an einer Stelle konzentriert ist.
- Seiten eines Webangebots haben in der Regel einen ähnlichen Seitenaufbau und enthalten ähnliche Gestaltungselemente. Die Beschreibung des Seitenlayouts und anderer Gestaltungsaspekte auf jeder einzelnen Seite anzugeben führt zu Redundanz. Vermeiden von Redundanz führt zu kleineren Dateien und somit schnelleren Downloads.
- Nutzer der Website haben mehr Kontrolle über die Darstellung: Wenn nötig kann CSS auch ganz deaktiviert oder durch eigene CSS-Regeln ergänzt werden.
- Wenn die HTML-Datei rein nach inhaltlichen Kriterien aufgebaut ist, können etwa Screenreader oder Suchmaschinen die Inhalte besser verarbeiten.

Abbildung 50 zeigt ein Beispiel für eine Seite ohne CSS. Das Beispiel stammt aus einem Tutorial des Mozilla Developer Network (MDN): *CSS basics*.<sup>40</sup> Mit Hilfe von ein paar CSS-Regeln kann die Darstellung der Seite angepasst werden (Abbildung 51). Mit Hilfe von CSS könnte problemlos noch eine Variante für einen kleinen Smartphone-Bildschirm oder zum Ausdrucken der Seite erstellt werden.

---

<sup>40</sup>[https://developer.mozilla.org/en-US/docs/Learn/Getting\\_started\\_with\\_the\\_web/CSS\\_basics](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics)



Abbildung 50: Seite ohne CSS

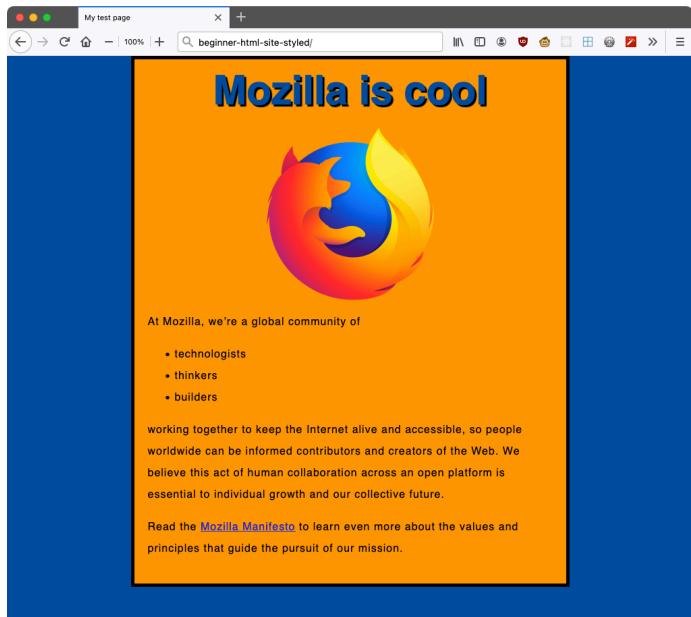


Abbildung 51: Seite mit CSS

## Verbinden von CSS und HTML

Es gibt mehrere Möglichkeiten, CSS-Angaben mit HTML-Code zu verbinden.

Die erste Variante ist die wichtigste: Die CSS-Angaben liegen in separaten Dateien (mit der Erweiterung `.css`), die mit Hilfe des `link`-Elements im Kopf der HTML-Datei eingebunden werden. Nur diese Variante liefert eine klare Trennung zwischen Inhalt und Darstellung:

```
1 <!DOCTYPE html>
2 <html lang="de">
3   <head>
4     <meta charset="utf-8" />
5     <title>Mein zweites HTML-Dokument</title>
6     <link rel="stylesheet" href="styles/screen.css">
7   </head>
8   ...
9 </html>
```

Dabei können zusätzlich auch noch ein oder mehrere Gerätetypen angegeben werden, für die das Stylesheet geladen werden soll. Auf diese Weise kann die Darstellung an den Gerätetyp angepasst werden:

```
1 <link rel="stylesheet" href="styles/screen.css" media="screen">
2 <link rel="stylesheet" href="styles/print.css" media="print">
```

In der zweiten Variante werden Style-Angaben im `head` der HTML-Datei in einem Element `style` direkt eingefügt. Die CSS-Regeln werden in diesem Fall gleich geschrieben wie in einer separaten CSS-Datei:

```
1 <style type="text/css">
2 p {
3   color: #f60;
4   font-size: 16px;
5 }
6 </style>
```

Es ist klar, dass diese Variante nicht die oben beschriebene gewünschte Trennung von Inhalt und Darstellung liefert. In manchen Situationen kann sie trotzdem nützlich sein:

- Beim Aufbau eines neuen Seitenlayouts ist es manchmal praktisch, alles in einer Datei zu haben und schnell bearbeiten zu können. Man vermeidet dadurch auch Probleme mit dem Caching der CSS-Datei. Wenn die Seite fertig ist, wird der CSS-Teil in eine separate Datei kopiert und das `style`-Element durch ein `link`-Element ersetzt.
- Ebenfalls beim Entwickeln einer Website: Um Regeln einer zentralen CSS-Datei lokal zu Testzwecken zu überschreiben. Selbst wenn eine solche “Anpassung” nur für eine einzelne HTML-Seite nötig ist, wird man sie normalerweise später in die zentrale oder eine zusätzliche CSS-Datei verschieben. Eine temporäre Anpassung von Stilen ist übrigens auch in den Entwickertools der Browser möglich.

In der dritten Variante werden die Stilangaben direkt in das HTML-Element geschrieben, dessen Darstellung spezifiziert werden soll. Dazu verwendet man das **style**-Attribut:

```
<p style="color: #f60; font-size: 16px;">Lorem ipsum dolor sit amet...</p>
```

In diesem Fall muss nicht angegeben werden, auf welche Elemente die Darstellung angewendet werden soll, denn die Beschreibung ist ja bereits im betreffenden Element enthalten. Das ist natürlich die stärkste Vermischung von Inhalt und Darstellung und daher fast immer zu vermeiden. Es gibt nur wenige Fälle, in denen diese Variante zum Einsatz kommen sollte. Hier ein Beispiel:

Wenn ein Content Management System (CMS) eine Webseite dynamisch erstellt, wird normalerweise HTML-Code generiert, auf den bestehende Stile aus separaten CSS-Dateien angewendet werden. Soll jedoch in dem generierten Teil auch ein Gestaltungselement mit angepasst werden, kann es sinnvoll sein, dies mit Hilfe eines **style**-Attributs einzufügen, zum Beispiel bei einem persönlichen Hintergrundbild:

```
<body style="background-image: url(bg_user44.jpg)">...</body>
```

## CSS-Regeln

CSS Regeln bestehen aus *Selektoren* und *Deklarationen*. Deklarationen bestehen aus *Eigenschaften* und *Werten* (Abbildung 52). Eine CSS-Datei enthält eine Sammlung solcher Regeln und weitere Angaben, die wir später ansehen werden.

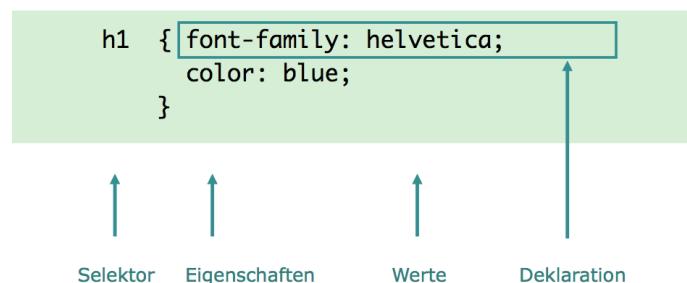


Abbildung 52: Aufbau einer CSS-Regel

Der *Selektor* bestimmt, welche Elemente in einem HTML-Dokument von einer CSS-Regel betroffen sind. In diesem Beispiel besteht der Selektor nur aus dem Element (eigentlich Elementtyp) `h1`. Das bedeutet, dass die Regel die Darstellung aller Überschriften der Ebene 1 beschreibt. Die Deklarationen, also Eigenschaften mit ihren Werten, folgen in geschweiften Klammern. Sie bestimmen die Darstellung und werden jeweils mit einem Semikolon abgeschlossen. Eine Regel kann mehrere Deklarationen enthalten, um verschiedene Gestaltungsaspekte (Farbe, Schriftart, Größe, ...) zu beschreiben.

In den beiden folgenden Abschnitten werden wir eine Auswahl von Selektoren und Eigenschaften ansehen. Und in weiteren Lektionen wird diese Liste von Selektoren und

Eigenschaften erweitert und beschrieben, wie man CSS zum Aufbau von Seitenlayouts einsetzen kann.

## CSS-Versionen

Erste Ansätze für Stylesheets erschienen bereits 1994, zum Beispiel der Vorschlag *Cascading HTML style sheets - a proposal* von Håkon W Lie.<sup>41</sup> Zwei Jahre später wurde *Cascading Style Sheets, level 1* vom W3C als Empfehlung herausgegeben.

Version	Zeit
CSS Level 1 (CSS 1)	Dez 1996
CSS Level 2 (CSS 2)	Mai 1998
CSS Level 2 Rev. 1 (CSS 2.1)	2002 begonnen, 2011 Recommendation
CSS Level 2 Rev. 2 (CSS 2.2)	2016 Public Working Draft
CSS Level 3 und folgende	2000 begonnen

CSS Level 3, genannt **CSS3** ist erheblich umfangreicher als die Versionen vorher. Aus diesem Grund wurde es vom W3C modular konzipiert. Es gibt also neu verschiedene CSS-Module, die ganz unterschiedliche Reifegrade aufweisen und teilweise als *Level 3* oder sogar mit einer höheren Versionsnummer versehen sind. Erst ein Teil davon hat bereits den Status einer **W3C Recommendation**, andere sind noch in einer frühen Entwurfsphase.

Aktuelle Browser unterstützen CSS 2.2 und eine Reihe aktueller CSS-Module bereits gut. Um mit einer Website eine gute Browser-Abdeckung zu erreichen, muss man sich jedoch mit dem Status der Browserunterstützung verschiedener CSS-Features beschäftigen.

Dan Cederholm empfiehlt (in *CSS3 For Web Designers*), beim Einsatz von CSS3 in Websites zwischen kritischen und nicht-kritischen Website-Funktionen zu unterscheiden:

Critical	Non-critical
Branding	Interaction
Usability	Visual Rewards
Accessibility	Feedback
Layout	Movement

Bei den kritischen Funktionen muss auf eine breite Browser-Unterstützung geachtet werden. Für die nicht-kritischen Funktionen können auch neue CSS-Features verwendet werden, die nicht von allen Browsern unterstützt werden. Wichtig ist dabei, dass die Website auch in den älteren Browsern (je nach angepeilter Browser-Abdeckung) grundsätzlich verwendbar und bedienbar ist. Die neuen Möglichkeiten sollen die Bedienung und das Erlebnis beim Besuchen der Website verbessern, aber nicht erst möglich machen (**Progressive Enhancement**).

<sup>41</sup> <https://www.w3.org/People/howcome/p/cascade.html>

## Selektoren in CSS 2.1

Wie erwähnt dient der *Selektor* einer CSS-Regel dazu, Elemente – oder allgemeiner: Stellen (es müssen nämlich nicht immer Elemente sein) – in einem HTML-Dokument auszuwählen, auf die die Deklarationen der Regel angewendet werden sollen.

### Type Selektor

Der einfachste Selektor ist der im Beispiel oben bereits verwendete *Type Selektor*, der zum Einsatz kommt, wenn eine Elementart (z.B. alle p-Elemente) immer gleich dargestellt werden soll. Als Selektor wird einfach der Elementname geschrieben.

```
1 h1 {  
2     font-size: 150%;  
3 }  
4  
5 h2 {  
6     font-size: 120%;  
7     color: blue;  
8 }
```

Die erste Regel wählt *alle* h1-Elemente, die zweite *alle* h2-Elemente aus.

### Schreibweise der Regeln

Wie die Whitespace-Zeichen (also Leerzeichen, Zeilenwechsel, Tabulatoren) verteilt werden, spielt in CSS keine Rolle. Man könnte also eine CSS-Regel komplett in einer Zeile schreiben. Wichtig ist, eine möglichst übersichtliche, schnell erfassbare Darstellung zu wählen. Die im Beispiel mit den h1- und h2-Selektoren verwendete Darstellung ist weit verbreitet. Für die definitive Website können die Stylesheets dann immer noch optimiert und komprimiert werden.

### Vererbung

Wenn ein Selektor ein bestimmtes Element im HTML-Dokument auswählt, sind in der Regel auch dessen Unterelemente betroffen, außer es wird für eine Eigenschaft eine spezifischere Darstellung für ein untergeordnetes Element explizit (in einer anderen CSS-Regel) oder implizit (Standard-Darstellung des Browsers) angegeben.

Beispiel: Die CSS-Regel

```
1 p {  
2     font-size: 3em;  
3     color: red;  
4 }
```

wird auf folgenden HTML-Ausschnitt angewendet:

```
1 <p>Text, in den ein <a href="verweis.html">Verweis</a>  
2 eingebaut ist.</p>
```

Die Schriftgrösse gilt sowohl für das p-Element als auch für das eingebettete a-Element. Der Absatz wird in roter Farbe angezeigt, nur der Verweis erscheint weiterhin blau, da für das a-Element in der Standard-Darstellung des Browsers vorgegeben ist, dass es blau angezeigt werden soll. Dies kann man auch ändern, man müsste dann aber direkt das a-Element mit dem Selektor adressieren.

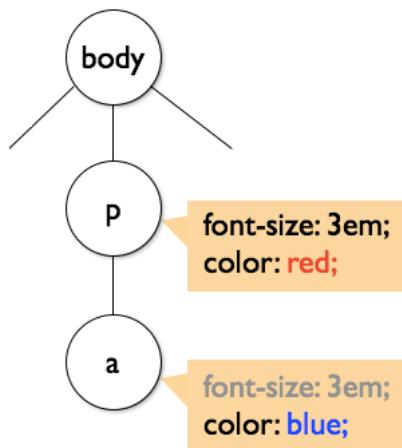


Abbildung 53: Vererbung von Eigenschaften

## Universalselektor

Der Universalselektor \* steht für alle Elemente eines Dokuments. Mit der folgenden Regel werden alle Aussen- und Innenabstände auf 0 gesetzt:

```

1  * {
2      margin: 0;
3      padding: 0;
4 }
  
```

Den Universalselektor sollte man mit Vorsicht einsetzen. Er versieht alle Elemente mit der entsprechenden Deklaration. Möchte man die Einstellung an einer bestimmten Stelle in der Hierarchie des Dokuments anpassen, funktioniert die Vererbung nicht wie erwartet: Die Unterelemente wurden bereits durch den Universalselektor ausgewählt und festgelegt.

Hier ein Ausschnitt aus einem früheren Beispiel:

```

1 <article>
2   <h1>Webseiten</h1>
3
4   <p>Die Grundidee ist ziemlich einfach:</p>
5
6   <p>Das <strong>World Wide Web</strong> (heute meist kurz: das Web)
7     basiert auf einer <em>Client-Server-Architektur</em>. Der Client
8     ist meist ein grafischer <em>Browser</em>. Er zeigt die Webseiten
9     an, die über das Internet von einem <em>Webserver</em> geladen
10    werden. Zu diesem Zweck wird im Browser eine Adresse eingegeben,
11    die eine bestimmte Ressource im Web anfordert. Das angezeigte
12    Dokument kann Verweise enthalten, die zu weiteren Ressourcen im
13    Web führen.</p>
14 </article>
```

Mit diesen beiden CSS-Regeln werden alle Elemente blau eingefärbt, ausser p-Elemente, die schwarz sein sollen:

```

1 * { color: blue; }
2 p { color: black; }
```

Der schwarze Text wird aber nicht auf die Unterelemente von p vererbt, da diesen bereits eine Farbe über die erste Regel zugewiesen wurde (Abbildung 54).

## Webseiten

Die Grundidee ist ziemlich einfach:

Das **World Wide Web** (heute meist kurz: das Web) basiert auf einer *Client-Server-Architektur*. Der Client ist meist ein grafischer *Browser*. Er zeigt die Webseiten an, die über das Internet von einem *Webserver* geladen werden. Zu diesem Zweck wird im Browser eine Adresse eingegeben, die eine bestimmte Ressource im Web anfordert. Das angezeigte Dokument kann Verweise enthalten, die zu weiteren Ressourcen im Web führen.

Abbildung 54: Universalselektor

### class- und id-Selektor

Oft werden mit dem Typselektor zu viele Elemente erfasst, d.h. der Selektor ist zu unspezifisch. Daher gibt es noch eine Reihe weiterer Selektortypen. Das Ziel ist, möglichst präzise einzelne Elemente des HTML-Dokuments auswählen zu können, ohne dass dafür viele Anpassungen im HTML-Code nötig sind.

Mit Hilfe des *class-Selektors* können alle Elemente im Dokument ausgewählt werden, die einen bestimmten Wert für das `class`-Attribut haben. Die beiden Elemente `p` und `blockquote` in folgendem Beispiel kann man mit dem Klassenselektor auswählen:

```

1  <!-- HTML-Ausschnitt -->
2  <p class="hinweis">...</p>
3  <blockquote class="hinweis">...</blockquote>

1  /* CSS-Regel */
2  .hinweis {
3      color: black;
4      background-color: #8c8c8c;
5      border: 3px solid #ecf7dd;
6  }

```

Wenn man nur alle p-Elemente mit einer bestimmten Klasse adressieren möchte, kann man diese als *Subklasse* eines Typs spezifizieren:

```

1  p.hinweis {
2      ...
3  }

```

In diesem Fall wäre das `blockquote`-Element nicht betroffen.

Auch mehrere Klassenangaben sind möglich:

```

1  p.info.error {
2      ...
3  }

```

Dieser seltener benötigte Selektor wählt alle p, die *sowohl* die Klasse `info` *als auch* die Klasse `error` haben. Im HTML-Code werden die verschiedenen Klassen mit Leerzeichen (und nicht etwa mit Kommas oder anderen Zeichen) getrennt:

```

1  <p class="info error">...</p>

```

Mit dem Klassenselektor wählt man normalerweise eine Gruppe von Elementen des HTML-Dokuments aus. Eine Möglichkeit, ein ganz bestimmtes Element im HTML-Dokument zu adressieren, ist der *id-Selektor*. Er setzt voraus, dass das Element im HTML mit einem `id`-Attribut versehen ist. So kann ein Element

```

1  <div id="kommentarbereich">...</div>

```

mit dem id-Selektor adressiert werden:

```

1  #kommentarbereich {
2      color: black;
3      background-color: #f3c600;
4  }

```

Zu Erinnerung: Jede `id` darf in einem Dokument nur einmal vorkommen. Mit dem id-Selektor wird also genau ein Element im Dokument angesprochen.

**class- und id-Bezeichnungen:** Erinnern Sie sich daran, dass ein HTML-Dokument so weit möglich nur *nach inhaltlichen Kriterien* aufgebaut sein soll? Das hat zur Konsequenz, dass mit den `id`- und `class`-Attributnen die *Bedeutung* der Elemente präzisiert werden soll.

Also zum Beispiel `info` oder `error`. Attributwerte, die die Darstellung beschreiben, haben im HTML eigentlich nichts verloren: `topright`, `green`, `column2`.

**Reichen diese Selektoren nicht aus?** Mit Typ-, id- und class-Selektor kann man ja jedes Element in der HTML-Datei auswählen (eigentlich würde sogar id allein reichen). Braucht man wirklich noch weitere Selektoren? Ja, denn man möchte vermeiden, dass man überall in der HTML-Datei künstlich `class`- und `id`-Attribute verteilen muss, nur um einzelne Elemente im CSS ansprechen zu können. Das widerspricht dem Ziel, Inhalt und Darstellung zu trennen.

## Kontext

Wenn in einem Selektor mehrere Angaben durch ein Leerzeichen getrennt werden, bedeutet das: Es wird ein Element ausgewählt, das die Beschreibung nach dem Leerzeichen erfüllt, und Unterelement eines Elements ist, das die Beschreibung vor dem Leerzeichen erfüllt. Ein Beispiel soll das verdeutlichen:

```
1 ol li {  
2     list-style-type: decimal;  
3 }
```

Hier werden alle `li`-Elemente (list items) ausgewählt, die Unterelemente eines `ol`-Elements (ordered list) sind. Andere `li`-Elemente, zum Beispiel solcher einer ungeordneten Liste `ul`, sind nicht betroffen.

Die Unterelement-Beziehung darf sich auch über mehrere Stufen erstrecken. Zum Beispiel wählt `header a` alle `a`-Elemente im `header` aus. Zwischen `header` und `a` können aber noch mehrere Ebenen von Elementen liegen.

Mehrere solcher Kontextangaben können miteinander verbunden werden:

```
1 ol ol li {  
2     list-style-type: lower-alpha;  
3 }
```

Das bedeutet: Alle `li`, die in einem `ol` vorkommen, die selbst wieder in einem `ol` vorkommen. Oder anders ausgedrückt: Die `li` auf der zweiten Ebene einer verschachtelten Liste.

**Konfliktauflösung:** Wenn die letzten beiden Regeln zusammen in einem Stylesheet auftauchen, haben wir scheinbar eine Mehrdeutigkeit. Betrachten Sie dieses HTML-Fragment:

```
1 <ol>  
2     <li>eins  
3         <ol>  
4             <li>zwei</li>  
5             <li>drei</li>  
6         </ol>  
7     </li>  
8     <li>vier</li>  
9 </ol>
```

Welche der beiden oben angegebenen Regeln passt auf <li>zwei</li>? Antwort: beide. Das ist normalerweise kein Problem, es werden einfach die Deklarationen aller passenden Regeln kombiniert. Hier haben wir aber zwei gegensätzliche Angaben für `list-style-type: decimal` und `lower-alpha`. In diesem Fall *gewinnt* die Regel mit dem *spezifischeren* Selektor. Damit wird das HTML-Fragment so ausgegeben:

1. eins
  - a. zwei
  - b. drei
2. vier

Wie sich allgemein feststellen lässt, welcher Selektor spezifischer ist als der andere, soll hier nicht weiter vertieft werden. In den meisten Fällen wird man das intuitiv richtig einschätzen. Wer es genauer wissen möchte, kann die Details in der CSS-Spezifikation nachlesen: Calculating a selector's specificity.<sup>42</sup>

Auch wenn die Beispiele anderes nahelegen: Kontextselektoren werden nicht nur aus Typselektoren zusammengesetzt. Es können auch beliebige andere Angaben verwendet werden. Sehr häufig werden sie mit dem id- oder class-Selektor kombiniert. Auf diese Weise können zum Beispiel sehr einfach alle Links im Kommentarbereich ausgewählt und speziell gestaltet werden:

```
1 #kommentarbereich a {  
2     ...  
3 }
```

## Gruppierung

Wenn mehrere CSS-Regeln gleiche Deklarationen haben, können diese auch zusammengefasst werden. Dadurch wird das Stylesheet kleiner und besser zu überblicken. Anstatt

```
1 h1 { font-family: Verdana, Arial, Helvetica, sans-serif; }  
2 h2 { font-family: Verdana, Arial, Helvetica, sans-serif; }  
  
schreibt man kürzer  
  
1 h1, h2 {  
2     font-family: Verdana, Arial, Helvetica, sans-serif;  
3 }
```

Auch hier können mehr als nur zwei Teile kombiniert werden, und die einzelnen Teile können beliebige Kombinationen der anderen Selektoren umfassen.

Kontextselektoren binden stärker als die Gruppierung mit einem Komma. Eine Klammerung ist nicht möglich. `h1 em`, `strong` heisst also *alle em in einem h1 oder alle strong* und nicht: *alle em oder strong in einem h1*.

---

<sup>42</sup><https://www.w3.org/TR/CSS22/cascade.html#specificity>

## Pseudoklassen

Pseudoklassen kann man als *virtuelle Klassen* verstehen: Elemente, die in einem bestimmten Zustand sind, dessen Bedeutung aber nicht durch ein explizites `class`-Attribut beschrieben sondern durch andere Einflüsse definiert ist. Zum Beispiel: Ein `a`-Element, auf dem gerade der Mauszeiger steht. Das folgende Beispiel zeigt verschiedene CSS-Regeln mit Pseudoklassen des `a`-Elements:

```
1 a { text-decoration: none; outline: none; }
2 a:link { color: #d90000; }
3 a:visited { color: #cc6666; }
4 a:hover, a:focus { text-decoration: underline; }
5 a:active { color: white; background-color: #d90000; }
```

Eine weitere Pseudoklasse ist `:first-child`. Sie bezieht sich auf einen Element in der HTML-Struktur, das erstes Unterelement eines anderen Elements ist. In diesem Beispiel wird für das erste `li` einer Liste im `div` mit der Klasse `results` ein anderer Abstand nach oben gewählt:

```
1 div.results li:first-child {
2     margin-top: 0.5em;
3 }
```

## Pseudoelemente

Pseudoelemente kann man als *virtuelle Elemente* verstehen: Sie beziehen sich auf Bereiche im HTML-Dokument, die bestimmte Eigenschaften haben, aber nicht explizit durch Elemente ausgezeichnet sind.

- `::first-line`: Die erste Zeile eines Elements.
- `::first-letter`: Das erste Zeichen eines Elements.
- `::before`: Eine Position vor dem Inhalt eines Elements.
- `::after`: Eine Position nach dem Inhalt eines Elements.

**Hinweis zur Schreibweise:** Pseudolemmata werden erst ab CSS3 mit zwei Doppelpunkten geschrieben. Wir verwenden hier gleich die neue Schreibweise. Die alte Schreibweise mit *einem* Doppelpunkt ist aber vorerst weiterhin zulässig.

Die ersten beiden der genannten Pseudoelemente werden normalerweise im Zusammenhang mit einem `p`-Element verwendet: `p::first-line` und `p::first-letter`. Man kann sich `::first-line` wie ein dynamisch die erste Zeile umfassendes `span`-Element vorstellen. `::first-letter` könnte man sogar explizit durch ein `span`-Element ersetzen.

Die beiden anderen Pseudoelemente `::before` und `::after` spielen eine besondere Rolle, da sie keinen Bereich sondern eine Position beschreiben, vorstellbar also durch ein leeres `span`-Element. Es ist natürlich nicht besonders sinnvoll, einem leeren Element eine bestimmte Gestaltung zu verpassen. Daher wird es praktisch ausschliesslich zusammen mit einer bestimmten Eigenschaft – nämlich `content` – verwendet. Damit kann man an der betreffenden Stelle Inhalt ins Dokument einfügen, anzugeben als String, Verweis auf eine Bilddatei, oder als Wert eines Attributs des ausgewählten Elements:

```
1 td.preis::before { content: "Preis: "; }
2 td.preis::after  { content: ".- EUR"; }
```

Das Element `td` beschreibt eine Zelle in einer Tabelle. Der HTML-Code

```
1 <td class="preis">35</td>
```

führt dann zur Ausgabe:

Preis: 35.- EUR

Ob es sinnvoll ist, mit Hilfe von CSS Inhalt in ein Dokument einzufügen, ist natürlich eine andere Frage. In vielen Fällen dürfte JavaScript zu diesem Zweck vorzuziehen sein. Eine mögliche Anwendung könnte aber sein, Verweise beim Ausdruck eines Dokuments um das Verweisziel zu ergänzen. Dazu kann folgende Regel in ein Stylesheet geschrieben werden, das nur beim Drucken geladen wird (`media="print"`):

```
1 a::after {
2   content: "[ " attr(href) " ]";
3 }
```

Eine andere Anwendung ist das automatische Einfügen von Kapitel- und Unterkapitelnummern vor Überschriften. Wenn Sie sich dafür interessieren, suchen Sie am besten in den Dokumentationen nach `counter`, `counter-increment` und `counter-reset`.

## Nachfolger und Geschwister

Der bereits erwähnte Kontextselektor gehört auch zu den Nachfolgerselektoren. Eine speziellere Art der Nachfolge wird mit dem *Child-Combinator* > ausgedrückt: Während `p em` alle `em` auswählt, die irgendwo unter einem `p` liegen, wählt `p > em` alle `em` aus, welche *direkt* unter einem `p` liegen.

Die Variante `p * em` ist eigentlich nur eine Kombination aus Kontext- und Universalselektor: Alle `em` unter `p`, aber es muss mindestens noch ein Element dazwischen liegen.

Schliesslich ist noch der *Adjacent Sibling Combinator* zu erwähnen: `h2 + p` wählt alle `p` aus, die unmittelbar auf ein Geschwisterelement `h2` folgen. Dieser Selektor ist oft praktisch, um mehrere Elemente aus einer Sequenz auszuwählen:

- `li + li`: Alle `li` ab dem zweiten `li` in der Liste
- `li + li + li`: Alle `li` ab dem dritten `li` in der Liste

Ab CSS3 gibt es zahlreiche weitere Möglichkeiten, einzelne Elemente zu adressieren.

## Attributselektoren

CSS 2.2 unterstützt noch eine Reihe von Selektoren, die Attribute der gesuchten Elemente beschreiben, die so genannten *Attributselektoren*. Hier mit Beispielen:

- `p[title]`: p-Elemente, die ein Attribut *title* haben
- `p[title=Einleitung]`: ... die ein Attribut *title* mit Wert *Einleitung* haben
- `p[title~="Einleitung"]`: ... die ein Attribut *title* haben, in dessen Wert das Wort *Einleitung* vorkommt (Attributwerte durch Leerzeichen getrennt)

- `p[lang|=en]`: ... die ein Attribut *lang* haben, das mit *en* beginnt, gefolgt von einem Bindestrich und dem Rest des Attributwerts

**Frage:** Wie kann man `p[class~=neu]` kürzer schreiben?

## Zusammenfassung

Hier ist die Liste der CSS2-Selektoren (Tabelle aus der CSS-Spezifikation):<sup>43</sup>

Pattern	Meaning
<code>*</code>	Matches any element.
<code>E</code>	Matches any E element (i.e., an element of type E).
<code>E F</code>	Matches any F element that is a descendant of an E element.
<code>E &gt; F</code>	Matches any F element that is a child of an element E.
<code>E:first-child</code>	Matches element E when E is the first child of its parent.
<code>E:link</code>	Matches element E if E is the source anchor of a hyperlink of which the target is not yet visited ( <code>:link</code> ) or already visited ( <code>:visited</code> ).
<code>E:visited</code>	
<code>E:active</code>	Matches E during certain user actions.
<code>E:hover</code>	
<code>E:focus</code>	
<code>E:lang(c)</code>	Matches element of type E if it is in (human) language c (the document language specifies how language is determined).
<code>E + F</code>	Matches any F element immediately preceded by a sibling element E.
<code>E[foo]</code>	Matches any E element with the foo attribute set (whatever the value).
<code>E[foo=warning]</code>	Matches any E element whose foo attribute value is exactly equal to warning.
<code>E[foo~=warning]</code>	Matches any E element whose foo attribute value is a list of space-separated values, one of which is exactly equal to warning.
<code>E[lang =en]</code>	Matches any E element whose lang attribute has a hyphen-separated list of values beginning (from the left) with en.
<code>DIV.warning</code>	Language specific. (In HTML, the same as <code>DIV[class~=warning]</code> .)
<code>E#myid</code>	Matches any E element with ID equal to myid.

## Selektoren ab CSS3

CSS3 unterstützt zahlreiche neue Selektoren, welche in der Spezifikation *Selectors Level 3* beschrieben sind.<sup>44</sup> Diese Spezifikation hat bereits den Status einer *W3C Recommendation*. Inzwischen wird bereits an der nächsten Version gearbeitet: *Selectors Level 4* ist ein *Working Draft*.<sup>45</sup>

<sup>43</sup><https://www.w3.org/TR/CSS22/selector.html#pattern-matching>

<sup>44</sup><https://www.w3.org/TR/selectors-3/>

<sup>45</sup><https://www.w3.org/TR/selectors-4/>

Hier soll nur eine kleine Auswahl der neuen Selektoren betrachtet werden.

## Attributselektoren

Die Attributselektoren in CSS2 waren ziemlich eingeschränkt. In CSS3 gibt es mehr Möglichkeiten:

- `^=` bedeutet: beginnt mit
- `$=` bedeutet: endet mit
- `*=` bedeutet: enthält (Frage: Was ist der Unterschied zu `~= ?`)

Angenommen wir haben in einem HTML-Dokument Verweise zu externen Quellen sowie Verweise zu PDF-Dateien:

```
1 <a href="http://www.sitepoint.com">SitePoint</a>
2 <a href="file.pdf">target link</a>
```

Dann können wir diese mit Attributselektoren adressieren und die Verweise passend kennzeichnen:

```
1 a[href^="http"] {
2   padding: 0 20px 0 0;
3   background: #ffff url(icon_external-site.gif) no-repeat right center;
4 }
5
6 a[href$=".pdf"] {
7   padding: 0 20px 0 0;
8   background: #ffff url(icon_pdf.gif) no-repeat right center;
9 }
```

So wird jeder Link zu einer externen Seite oder auf ein PDF durch eine kleine Grafik erweitert. Sollte ein Browser diese Selektoren noch nicht kennen (das ist heute kaum mehr der Fall), wird keine Grafik angezeigt, ansonsten entsteht kein grosser Schaden (Abbildungen 55 und 56). Das bezeichnet man als *Graceful Degradation* und es sollte immer das Ziel sein, wenn neue CSS- oder HTML-Features verwendet werden. Mit der oben bereits verwendeten Bezeichnung *Progressive Enhancement* ist übrigens fast das gleiche gemeint, nur aus einem anderen Blickwinkel.

Supported	Not Supported
Lorem ipsum <a href="http://www.sitepoint.com">SitePoint</a> at, consectetuer adipiscing elit. Etiam nisi. Duis eros	Lorem ipsum <a href="#">SitePoint</a> at, consectetuer adipiscing elit. Etiam nisi. Duis eros

Abbildung 55: CSS-Selektor: Attribut beginnt mit...

Supported	Not Supported
Lorem ipsum <a href="file.pdf">target link</a> at, consectetuer adipiscing elit. Etiam nisi. Duis eros	Lorem ipsum <a href="#">target link</a> consectetuer adipiscing elit. Etiam nisi. Duis eros

Abbildung 56: CSS-Selektor: Attribut endet mit...

## CSS3-Geschwister-Selektor

Zu dem `+`-Selektor gesellt sich ein weiterer Geschwisterselektor, `~` als *General Sibling Combinator*. `h1 ~ pre` bedeutet: Ein `pre`-Element, das als Geschwister (nicht unbedingt unmittelbar) auf ein `h1`-Element folgt.

## CSS3-Pseudoklassen

In CSS3 wurden einige Pseudoklassen hinzugefügt, die im Vergleich zum `:first-child` aus CSS2 mehr Möglichkeiten bieten, einzelne Elemente auszuwählen. Zum Beispiel:

- `:nth-child(n)`
- `:nth-last-child(n)`
- `:nth-of-type(n)`
- `:nth-last-of-type(n)`
- `:last-child`
- `:first-of-type`
- `:last-of-type`
- `:not(...)`

Die Bedeutung dieser Selektoren dürfte aufgrund der Bezeichnungen klar sein. Die komplette Liste inklusive Erklärungen finden Sie in *Selectors Level 3*.<sup>46</sup>

## Eigenschaften

Nachdem wir die Selektoren sehr ausführlich behandelt haben, soll noch ein kurzer Blick auf die Deklarationen geworfen werden. Bereits CSS2 stellt zahlreiche Eigenschaften zur Verfügung, etwa um

- Texte auf eine bestimmte Art darzustellen
- Listen zu formatieren
- Elemente zu positionieren
- Abstände zu definieren
- Hintergrundfarben und -bilder festzulegen

Diese Liste ist noch nicht einmal vollständig. In CSS3 kommen noch zahlreiche weitere Eigenschaften dazu. Es ist nicht das Ziel, in diesem Kurs möglichst viele davon zu behandeln. Wir werden uns auf eine Auswahl häufig benötigter Eigenschaften beschränken. In diesem Kapitel legen wir den Schwerpunkt auf die Darstellung von Texten und Listen. Weitere Eigenschaften werden wir in den kommenden Kapiteln im Zusammenhang mit dem Einsatz von CSS zum Aufbau von Seitenlayouts einführen.

## Werte und Masseneinheiten

Deklarationen bestehen aus Eigenschaften und Werten. Als Werte kommen Zahlenwerte in Frage, teils mit und teils ohne Einheit, Farbangaben, Verweise auf Dateien (URLs), Schlüsselwörter oder beliebige Strings.

<sup>46</sup><https://www.w3.org/TR/selectors-3/#selectors>

```

1  #beispiel {
2      margin: 0;
3      font-size: 1.5em;
4      font-family: "Times New Roman", serif;
5      color: black;
6      background-color: #ffcc00;
7 }

```

Hinweise:

- Zwischen Zahl und Einheit darf kein Leerzeichen stehen.
- 0 braucht keine Einheit, auch 0cm ist einfach nur 0.
- Werte werden ohne Anführungszeichen geschrieben, ausser sie bestehen aus mehreren Wörtern.

Für Größen- bzw. Längenangaben gibt es verschiedene Möglichkeiten. Zu den absoluten Größenangaben gehört neben den physikalischen Größen auch das Pixel. Hier sind die wichtigsten in CSS verwendeten absoluten Längeneinheiten:

- **in**: Inch, 1in ist etwa 2.54cm
- **cm**: Zentimeter
- **mm**: Millimeter
- **pt**: Punkt — 1pt entspricht 1/72 Inch.
- **px**: Pixel – 1px entspricht 0.75pt

Bis auf das Pixel handelt es sich hier klar um absolute Größenangaben. Aber ist ein Pixel, geschrieben mit der Einheit **px**, nicht einfach ein Pixel auf dem Bildschirm, das je nach Auflösung und Grösse der Anzeige sehr unterschiedlich gross ausfallen kann? Im Prinzip schon. Aber die CSS-Spezifikation unterscheidet hier nach Ausgabemedien:

- Für die Druckausgabe oder auf ähnlichen hochauflösenden Medien soll sich die Größenberechnung an den physikalischen Größen (etwa **in**, **cm**) orientieren. Ein Pixel ist dann tatsächlich 0.75pt gross.
- Für Bildschirme mit verschiedenen Auflösungen und Abständen zum Betrachter (Smartphone, Computer-Monitor, Reklametafel) orientiert sich die Größenberechnung an einem so genannten *Referenzpixel*, eine bestimmte Grösse, die den Betrachtungsabstand einbezieht. Bei Armlängendistanz (71cm angenommen) entspricht ein Pixel etwa 0.26mm. Das entspricht in der Regel natürlich nicht der Grösse der tatsächlichen Bildschirmpixel. Ein Pixel (**1px**) ist daher ein (meist ganzzahliges) Vielfaches der tatsächlichen Grösse eines Bildschirmpixels, so dass das Referenzpixel möglichst gut angenähert wird. Die anderen Einheiten (**in**, **cm**, etc.) werden dann von dieser Pixelgrösse abgeleitet.

Wenn man den Website-Besuchern möglichst viel Kontrolle darüber geben möchte, wie eine Website auf ihrem Ausgabegerät aussieht – und das sollte eigentlich das Ziel sein – ist zu überlegen, ob man nicht alle (Schrift-) Größen ausgehend von der im Browser eingestellten Standard-Schriftgrösse berechnen möchte. Dazu dient die Einheit **em**. Für ein bestimmtes Element ist **1em** die für dieses Element berechnete Schriftgrösse.

Das Problem mit `em`: die berechnete Schriftgrösse kann von der Position im Dokument abhängen, besonders wenn prozentuale Angaben im Spiel sind. Aus diesem Grund wurde mit CSS3 eine neue Grösse `rem` eingeführt: diese Angabe entspricht der für das Wurzel-element (`html`) berechneten Schriftgrösse und ist damit unabhängig von der Position im Dokument.

Ebenfalls neu in CSS3 sind die Grössenangaben `vw`, `vh`, `vmin`, `vmax`. Diese lehnen sich an die Viewport-Grösse an. Der **Viewport** ist der Bereich des Browsers, in dem die Webseite angezeigt wird. `1vw` entspricht einem Prozent der Viewport-Breite, `1vh` einem Prozent der Viewport-Höhe. `1vmin` (`1vmax`) ist ein Prozent der Höhe oder Breite, je nachdem welches kleiner (grösser) ist. Diese Grösseneinheiten sind nützlich, wenn Seiten für sehr unterschiedliche Bildschirmgrössen gestaltet werden, etwa für Smartphones und für Desktop-Monitore.

Eine *relative* Grössenangabe ist *Prozent*, angegeben mit der Einheit `%`. So kann zum Beispiel der Zeilenabstand mit der Deklaration `line-height: 120%` um 20% vergrössert werden. Aus Prozentangaben werden feste Grössen berechnet und diese dann an untergeordnete Elemente weitervererbt.

Für Grössenangaben auf Bildschirmen werden normalerweise `px` oder `em` verwendet. Beim Drucken von Dokumenten `pt` (vor allem für die für die Schriftgrösse) oder die anderen absoluten Grössenangaben.

Für Farbangaben gibt es eine Reihe von vordefinierten Farbnamen. In CSS2 sind diese Farbnamen definiert: *aqua*, *black*, *blue*, *fuchsia*, *gray*, *green*, *lime*, *maroon*, *navy*, *olive*, *orange*, *purple*, *red*, *silver*, *teal*, *white*, *yellow*.

Bei manchen Eigenschaften kann als Farbe auch `transparent` angegeben werden.

Farben können ausserdem durch ihren Rot-, Grün- und Blauanteil spezifiziert werden. Einmal, indem die hexadezimalen Anteile zwischen 00 und ff beginnend mit `#` aneinandergehängt werden. `#ffcc00` bedeutet: 100% rot, etwas weniger grün, kein blau. Statt `#ffcc00` kann auch kurz `#fc0` geschrieben werden. `#000` ist schwarz, `ffff` ist weiss.

Mit Hilfe von `rgb(r,g,b)` können die Werte auch dezimal oder mit Prozent angegeben werden. `#ffcc00` entspricht `rgb(255,204,0)` oder `rgb(100%,80%,0%)`.

In **CSS3** sind weitere Farbangaben möglich. Nützlich ist zum Beispiel `rgba`, wo zusätzlich die Transparenz (Alpha-Kanal) angegeben wird. `rgba(0,0,255,0.1)` ist ein ziemlich durchsichtiges Blau und `rgba(255,0,0,1)` ein undurchsichtiges Rot. Für Farben gibt es in CSS3 eine eigene Spezifikation: *CSS Color Module Level 3*.<sup>47</sup>

So viel zu den Grössen- und Farbangaben. Weitere – nur für bestimmte Eigenschaften mögliche – Werte werden wir zusammen mit diesen Eigenschaften behandeln.

## Textdarstellung

Wichtig für die Textdarstellung ist zunächst einmal die Schriftart (eigentlich: Schriftfamilie), spezifiziert mit der Eigenschaft `font-family`. Da man nicht weiss, welche Schriftarten

---

<sup>47</sup><https://www.w3.org/TR/css-color-3/>

auf den Computern der Website-Besucher installiert sind, sollte man hier mehrere Schriftarten, durch Kommas getrennt, angeben. Die erste passende Schriftart aus dieser Liste wird für den anzugezeigenden Text verwendet. Für den Fall, dass keine der angegebenen Schriftarten auf dem Client-Computer zur Verfügung steht, gibt man in der Deklaration als letztes einen generischen Schriftart-Typ an:

- **serif**: Schriften mit Serifen (Antiqua-Schriften)
- **sans-serif**: Serifenlose Schriften (Grotesk-Schriften)
- **cursive**: Schreibschriften oder kursiver Typ
- **fantasy**: Dekorative Schriften
- **monospace**: Schriften mit einheitlicher Zeichenbreite

In diesem Fall überlässt man die Auswahl einer zur generischen Angabe passenden Schriftfamilie dem Browser. Meistens versucht man als Wert dieser Eigenschaft eine Reihe von ähnlich aussehenden Schriftfamilien anzugeben, mit denen man die typischerweise verwendeten Plattformen und Systemversionen abdeckt. Hier zwei Beispiele:

```
1 font-family: Cambria, "Hoefler Text", Utopia, "Liberation Serif",
2      "Nimbus Roman No9 L Regular", Times, "Times New Roman", serif;
3 font-family: Constantia, "Lucida Bright", Lucidabright, "Lucida Serif",
4      Lucida, "DejaVu Serif", "Bitstream Vera Serif", "Liberation Serif",
5      Georgia, serif;
```

Auf die Möglichkeit, mit der Website eigene Schriftarten auszuliefern, kommen wir gleich zurück.

Die Schriftgrösse wird mit **font-size** angegeben, meist in **em** oder **px**, für den Druck in **pt**. Neben der Schriftgrösse kann auch der Zeilenabstand spezifiziert werden, und zwar mit der Eigenschaft **line-height**. Dies ist fast immer erforderlich, da der Default-Zeilenabstand im Browser für gute Lesbarkeit meistens zu klein ist. Werte können sein: **normal**, eine Zahl (z.B. 1.5), eine Größenangabe, eine Prozentangabe, oder **inherit** (vom Elternelement übernehmen).

Hier sind weitere Angaben zur Schriftart und ihre möglichen Werte:

- **font-style**: italic | oblique | normal
- **font-variant**: small-caps | normal
- **font-weight**: bold | bolder | lighter | 100,200,...,900 | normal

Es gibt außerdem eine Eigenschaft **font**, die die genannten Eigenschaften zusammenfasst. Die verschiedenen Möglichkeiten können Sie der CSS-Spezifikation entnehmen.

Zur Textdarstellung gehört natürlich auch die Textfarbe, angegeben mit der **color**-Eigenschaft. Die Hintergrundfarbe wird mit **background-color** spezifiziert.

Mit Hilfe von **text-decoration** kann festgelegt werden, ob ein Text unterstrichen (Wert: **underline**), durchgestrichen (Wert: **line-through**), oder ganz normal (Wert: **none**) dargestellt werden soll. Die anderen möglichen Werte werden selten benötigt. Mit **text-decoration: none** kann man zum Beispiel das Unterstreichen von Verweisen abschalten. Man muss dann aber darauf achten, dass die Verweise trotzdem gut erkennbar bleiben.

Weitere Text-Eigenschaften zusammengefasst:

- `text-indent`: Einrücken der ersten Zeile, Größen- oder Prozentangabe
- `text-align`: Text ausrichten: `left` | `right` | `center` | `justify`
- `letter-spacing`: Zeichenabstand: Größenangabe oder `normal`
- `word-spacing`: Wortabstand: Größenangabe oder `normal`
- `text-transform`: `capitalize` | `uppercase` | `lowercase` | `none`

Bei den meisten Eigenschaften ist zusätzlich noch `inherit` als Wert möglich. `text-align` ist übrigens nur zum Ausrichten von Text und nicht zum Ausrichten von ganzen Blocks geeignet.

Hier noch ein Beispiel zur Textdarstellung:

```
1 address {  
2     font-family: "Times New Roman", serif;  
3     text-align: center;  
4     font-size: 80%;  
5     font-style: normal;  
6     letter-spacing: 2px;  
7     line-height: 1.5em;  
8 }
```

Für ein `address`-Element im HTML-Code sieht die Ausgabe dann so aus:

ZHAW  
Zürcher Hochschule für Angewandte Wissenschaften  
Technikumstr. 9  
8401 Winterthur

Abbildung 57: CSS Textdarstellung

## Schriftarten laden

Die Möglichkeit, Schriftart-Daten zu laden, war bereits in CSS 2.0 vorgesehen, wurde aber in CSS 2.1 wegen mangelnder Browser-Unterstützung wieder entfernt. In CSS3 ist es nun wieder möglich, Schriftarten nachzuladen. Das Problem ist aber, dass es verschiedene Arten von Font-Dateien gibt:

- WOFF - Web Open Font Format
- TTF/OTF - TrueType and OpenType fonts
- EOT - Embedded OpenType fonts

WOFF dürfte die Zukunft für Schriftarten im Web sein. Für möglichst grosse Browserunterstützung sollten aber auch die anderen Formate, wenigstens TTF oder OTF bereitgestellt werden. EOT wurde nur vom Internet Explorer verwendet, seit Version 9 kann aber auch dieser Browser WOFF verarbeitet.

Hier ein Beispiel, wie eine Schriftart *GentiumTest* in zwei verschiedenen Formaten zum Laden bereitgehalten und verwendet werden kann:<sup>48</sup>

```
1 @font-face {  
2   font-family: GentiumTest;  
3   src: url(fonts/GenR102.woff) format("woff"),  
4       url(fonts/GenR102.ttf) format("truetype");  
5 }  
6  
7 body {  
8   font-family: GentiumTest, Times, Times New Roman, serif;  
9 }
```

Wichtig ist, dass die Lizenzbedingungen der Schriftart die Veröffentlichung im Web erlauben, sonst kann es teuer werden.

Beschrieben ist das Ganze in der Spezifikation *CSS Fonts Module Level 3*.<sup>49</sup> Welche Browser welche Font-Typen unterstützen, können Sie unter [caniuse.com](http://caniuse.com) nachschlagen.<sup>50</sup> Beachten Sie die Sub-Features für die verschiedenen Font-Typen (unter der Kompatibilitätstabelle).

## Listendarstellung

Bei der Darstellung von Listen gibt es eine Reihe von Voreinstellungen was die Abstände und das Listenzeichen angeht. Bei einer geordneten Liste `ol` wird die Liste mit arabischen Zahlen nummeriert, bei einer ungeordneten Liste `ul` erscheinen dicke Punkte als Listenzeichen.

Anpassungen sind über die folgenden Eigenschaften möglich:

- `list-style-type`
- `list-style-image`
- `list-style-position`

Für `list-style-type` sind in CSS 2.1 die folgenden Werte definiert: `disc` | `circle` | `square` | `decimal` | `decimal-leading-zero` | `lower-roman` | `upper-roman` | `lower-greek` | `lower-latin` | `upper-latin` | `armenian` | `georgian` | `lower-alpha` | `upper-alpha` | `none` | `inherit`.

Es kann auch mit `list-style-image` ein eigenes Listenzeichen in Form einer Bilddatei spezifiziert werden. Verweise auf Ressourcen im Web werden durch eine URL-Angabe gemacht, wie Sie das bereits im Abschnitt über das Laden von Schriftarten gesehen haben. Beispiel:

```
1 ul {  
2   list-style-image: url("http://meinegrafiken.com/ellipse.png");  
3 }
```

In der Vergangenheit gab es mit `list-style-image` immer wieder Probleme, da verschiedene Browser das Bild unterschiedlich vor dem Text positioniert haben. Es hat sich daher

<sup>48</sup>Beispiel aus: <http://hacks.mozilla.org/2009/10/woff/>

<sup>49</sup><https://www.w3.org/TR/css-fonts-3/>

<sup>50</sup><http://caniuse.com/#feat=fontface>

bewährt, auf `list-style-image` zu verzichten und lieber ein Hintergrundbild passend zu positionieren.

Mit `list-style-position` kann schliesslich noch angegeben werden, ob das Listenzeichen ausserhalb des Textblocks (`outside`, das ist die Voreinstellung) oder in den Textblock eingerückt (`inside`) erscheinen soll.

## Kaskade, Konflikte, Reset

Wie ein Element letztlich dargestellt wird, kann von verschiedenen Deklarationen aus mehreren CSS-Regeln abhängen, die von mehreren Stylesheets beigesteuert wurden:

- Da gibt es einmal ein **Browser-Stylesheet**, das die Standard-Darstellung von HTML definiert, wenn keine weiteren Stylesheets geladen werden: Überschriften werden grösser dargestellt, Absätze mit Abstand zueinander, Zitate eingerückt usw. Das im Firefox verwendete Stylesheet kann auf einer Entwicklerseite von Mozilla inspiert werden.<sup>51</sup>
- Unter einem **Benutzer-Stylesheet** versteht man ein Stylesheet, das man selber im Browser installiert hat, und das beim Laden von Webseiten grundsätzlich mit berücksichtigt wird. Die Regeln dieses Stylesheets können bestimmte Probleme auf häufig besuchten Seiten beheben oder ganz allgemein Einstellungen verändern. Zum Beispiel kann versucht werden, mit einer Regel Text generell grösser darzustellen, etwa wenn man sehbehindert ist. Das kann natürlich dazu führen, dass der Seitenaufbau auf manchen Seiten nicht wie vom Webdesigner beabsichtigt aussieht.
- Das **Autoren-Stylesheet** (oder **Designer-Stylesheet**) schliesslich enthält die CSS-Angaben des Webdesigners. Es kann mehrere solche Autoren-Stylesheets geben, deren Inhalte für die Darstellung der Seitenelemente kombiniert werden.

### Browser-Stylesheet

In der CSS-2.2-Spezifikation gibt es einen Vorschlag vom W3C, wie ein HTML-4-Dokument ohne weiteres Stylesheet dargestellt werden sollte: *Appendix D. Default style sheet for HTML 4*.<sup>52</sup> Hier ist ein Ausschnitt davon:

```
1  ...
2  li          { display: list-item }
3  head        { display: none }
4  table       { display: table }
5  tr          { display: table-row }
6  ...
7  body        { margin: 8px }
8  h1          { font-size: 2em; margin: .67em 0 }
9  h2          { font-size: 1.5em; margin: .75em 0 }
10 ...
```

---

<sup>51</sup><https://searchfox.org/mozilla-central/source/layout/style/res/html.css>

<sup>52</sup><https://www.w3.org/TR/CSS22/sample.html>

Die meisten Browser werden sich ungefähr an diese Vorgaben halten. Hier ist zum Beispiel der Abstand von 8px zu sehen (für das `body`-Element definiert), der zwischen dem Rand des Browser-Fensters und den Inhalten auftaucht. Möchte man diesen Abstand nicht haben, muss man ihn explizit mit einer eigenen CSS-Regel deaktivieren:

```
1 body {  
2     margin: 0;  
3 }
```

## CSS-Reset

Die Voreinstellungen durch das Browser-Stylseteet sind für Webdesigner manchmal lästig, da diese Vorgaben in der Kombination mit den eigenen Regeln zu unerwarteten Ergebnissen führen können. Leider gibt es keine Möglichkeit, das Browser-Stylesheet per CSS-Befehl einfach abzuschalten. Aus diesem Grund behelfen sich viele Webdesigner mit **CSS-Reset-Stylesheets**, das sind Stylesheets, welche die meisten Voreinstellungen aus dem Browser-Stylesheet zurücksetzen. Bekannt ist zum Beispiel das Reset-Stylesheet von Eric Meyer.<sup>53</sup> Hier ein Ausschnitt:

```
1 html, body, div, span, applet, object, iframe, h1, h2,  
2 /*... und noch viel mehr */ {  
3     margin: 0;  
4     padding: 0;  
5     border: 0;  
6     font-size: 100%;  
7     font: inherit;  
8     vertical-align: baseline;  
9 }  
10 body {  
11     line-height: 1;  
12 }  
13 ol, ul {  
14     list-style: none;  
15 }  
16 /* usw. ... */
```

Ein solches Reset-Stylesheet – meist `reset.css` genannt – wird als erstes Stylesheet in der HTML-Datei referenziert. Es setzt allen Text auf eine einheitliche Grösse und alle Abstände auf 0. Man kann also quasi bei Null beginnen, die eigene Gestaltung aufzubauen.

Natürlich erscheint es nicht gerade geschickt, erst für alle Elemente eine Standard-Darstellung zu definieren und dann in einem weiteren Schritt diese Einstellungen wieder zurückzusetzen, also Brower-Rechenzeit einzusetzen um schliesslich wieder am Ausgangspunkt zu sein. Aus diesem Grund sind Reset-Stylesheets auch umstritten. Vermutlich sollte man das einfach pragmatisch sehen: In manchen Fällen ist es sicher nützlich, komplett bei Null mit dem Aufbau der Seitendarstellung beginnen zu können. Dann setzt man eben ein Reset-Stylesheet ein. In anderen Fällen bringt das nicht so viel. Dann kann man die Voreinstellungen nutzen und geeignet anpassen oder erweitern.

---

<sup>53</sup><https://meyerweb.com/eric/tools/css/reset/>

Mit der Zeit ist eine weitere Variante unter dem Namen **normalize.css**<sup>54</sup> populär geworden. Im Gegensatz zu einem Reset-CSS setzt es nicht alle Eigenschaften zurück sondern sorgt für eine konsistente Darstellung über Browergrenzen. Hier der Abschnitt *What does it do?* aus der Dokumentation:

- Preserves useful defaults, unlike many CSS resets.
- Normalizes styles for a wide range of elements.
- Corrects bugs and common browser inconsistencies.
- Improves usability with subtle improvements.
- Explains what code does using detailed comments.

Selbstverständlich sind sowohl von *reset.css* als auch von *normalize.css* verschiedene Varianten mit spezifischen Vor- und Nachteilen im Web zu finden.

## Benutzer-Stylesheet

Möchte man die Darstellung auf allen oder einzelnen Websites im Browser anpassen, kann man ein *User-Stylesheet* erstellen und im Browser installieren. Die meisten Browser bieten die Möglichkeit, solche User-Sheets zu installieren, die beim Laden von Webseiten zusätzlich zu den anderen Sheets berücksichtigt werden. Hier die Einstellung im Safari:

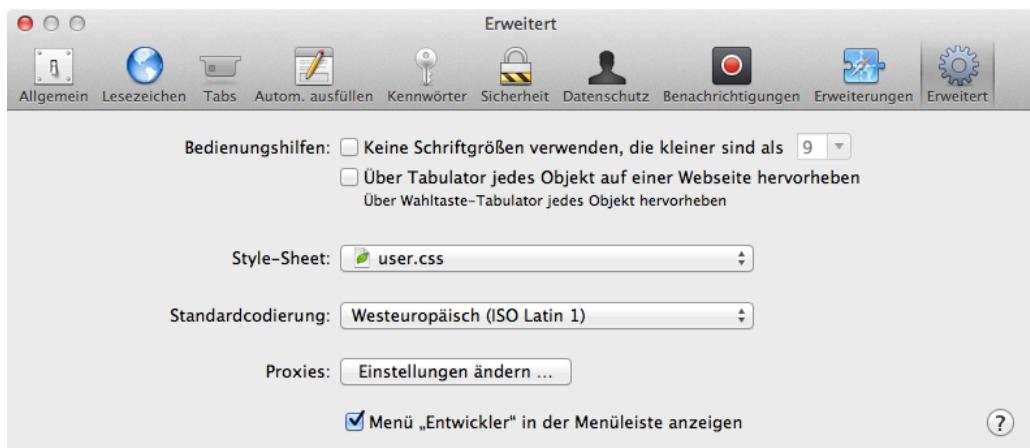


Abbildung 58: Benutzer-Stylesheet im Safari

Es ist alles andere als trivial, ein Benutzer-Stylesheet so zu konzipieren, dass genau der gewünschte Effekt eintritt und andere Seiten oder Seitenbereiche nicht zum Nachteil verändert werden.

Trotzdem sollten Webdesigner auch die Möglichkeit von Benutzer-Sheets nicht ganz vergessen: Was die Nutzer mit den über das Web gelieferten Seiten machen, bleibt ihnen überlassen. Sie können die *Gestaltung* mit Hilfe von Benutzer-Sheets (und übrigens auch das *Verhalten* mit Hilfe von Benutzer-Scripts) nach ihren Wünschen anpassen.

---

<sup>54</sup><https://github.com/necolas/normalize.css/>

Benutzer-Stylesheets eignen sich auch als *Werbeblocker*. Dazu erstellt man ein Stylesheet und verwendet Attributselektoren, die typische Werbebilder auswählen, zum Beispiel `img`-Elemente mit ganz bestimmten Größen. Für diese verwendet man dann eine dieser Deklarationen:

- `display: none` lässt das betreffende Element einfach verschwinden; der Platz, den das Element einnehmen würde, wird dadurch frei.
- `visibility: hidden` macht das betreffende Element unsichtbar; das Element nimmt seinen ursprünglichen Platz ein, ist aber nicht sichtbar.

## Autoren-Stylesheet

Dazu gehört alles, was mit der HTML-Seite geliefert wird:

- Mit `link`-Element verbundene CSS-Dateien
- Mit dem `style`-Element direkt in die HTML-Datei eingefügte Stile
- Mit dem `style`-Attribut direkt beim Element beschriebene Darstellung

## Kaskade und Konfliktauflösung

Um die Darstellung für ein Element zu bestimmen, werden die Eigenschaften aus allen CSS-Regeln mit passendem Selektor aus allen passenden Stylesheets zusammengetragen. Mit **Kaskade** sind die oben erwähnten Stylesheet-Typen gemeint: Autoren-, Benutzer- und Browser-Stylesheet.

Findet sich auf diese Weise kein Wert für eine bestimmte Eigenschaft, wird ein Default-Wert verwendet oder die Eigenschaft vom übergeordneten HTML-Element geerbt. Ob eine bestimmte Eigenschaft geerbt wird, können Sie der CSS-Spezifikation entnehmen.

Das geht soweit problemlos, ausser es treten Konflikte auf: Dies ist der Fall, wenn für eine bestimmte Eigenschaft aus verschiedenen Quellen unterschiedliche Werte geliefert werden. Dann stellt sich die Frage, welcher der Werte Priorität hat.

Ein Mittel, das zu steuern ist die `!important`-Angabe in einer Deklaration. Deklarationen mit `!important` haben Vorrang.

```
1 p {  
2     text-indent: 1em !important;  
3     font-style: italic !important;  
4     font-size: 18pt;  
5 }
```

Die Regeln verschiedener Stylesheets haben folgende Prioritäten:

1. Benutzer-Stylesheet mit `!important`
2. Autoren-Stylesheet mit `!important`
3. Autoren-Stylesheet
4. Benutzer-Stylesheet
5. Browser-Stylesheet

Sind dadurch Konflikte noch nicht aufgelöst, werden die Regeln danach sortiert, wie spezifisch ihr Selektor ist. Angaben in `style`-Attributen (also eigentlich ohne Selektor) haben

dabei höhere Priorität. Und ein id-Selektor hat eine höhere Priorität als ein class- oder Typselektor.<sup>55</sup>

Wenn dann immer noch Konflikte auftreten, überschreibt der zuletzt spezifizierte Wert den zuvor angegebenen Wert.

## Spezielle CSS-Regeln

Mit einer @charset-Angabe kann die verwendete Zeichencodierung angegeben werden. Die @charset-Regel muss als erstes in einer CSS-Datei stehen.

```
1 @charset "UTF-8";
```

Beim Einfügen eines Stylesheets mit Hilfe des link-Elements kann über das media-Attribut angegeben werden, für welche Art User Agents das Stylesheet geladen werden soll. Dies kann auch noch innerhalb eines Stylesheets geschehen. Mit einer @media-Angabe kann ein Ausschnitt der CSS-Datei auf bestimmte Gerätetypen eingeschränkt werden:

```
1 @media screen {
2     body {font-size:12px;}
3 }
4 @media print {
5     body {font-size:8pt;}
6 }
```

Solche @media-Angaben können noch wesentlich komplexer ausfallen. Mit Hilfe von *Media Queries* können CSS-Regeln anhand einer ganzen Reihe von Kriterien ausgewählt werden, unter anderem abhängig von der Bildschirmgrösse.<sup>56</sup> Damit können Webangebote realisiert werden, welche sich dynamisch an die Eigenschaften der verwendeten Geräte anpassen. Diese als *responsives Webdesign* bezeichnete Technik wird im Wahlmodul *Mobile Applications (MOBA)* behandelt.

Mit @import kann eine weitere CSS-Datei geladen werden. Die Adresse kann mit oder ohne url(...) geschrieben werden. Auch hier ist eine Medienangabe möglich:

```
1 @import "mystyle.css";
2 @import url("mystyle.css");
3 @import url("fineprint.css") print;
4 @import url("bluish.css") projection, tv;
```

## Kontrolle und Fehlersuche

Im Kapitel zum Thema *Markup* wurde der HTML-Validator vorgestellt. Auch für CSS gibt es einen Validator beim W3C.<sup>57</sup> Der frühere Ratschlag zum Validieren von HTML-Dateien gilt auch für CSS: **Testen Sie Ihre CSS-Dateien mit dem Validator** und korrigieren Sie die gemeldeten Fehler, bevor sie an unerwarteter Stelle viel Zeit kosten.

---

<sup>55</sup><https://www.w3.org/TR/selectors-3/#specificity>

<sup>56</sup><https://www.w3.org/TR/css3-mediaqueries/>, zudem bereits in Arbeit Level 4 und 5

<sup>57</sup><http://jigsaw.w3.org/css-validator/>

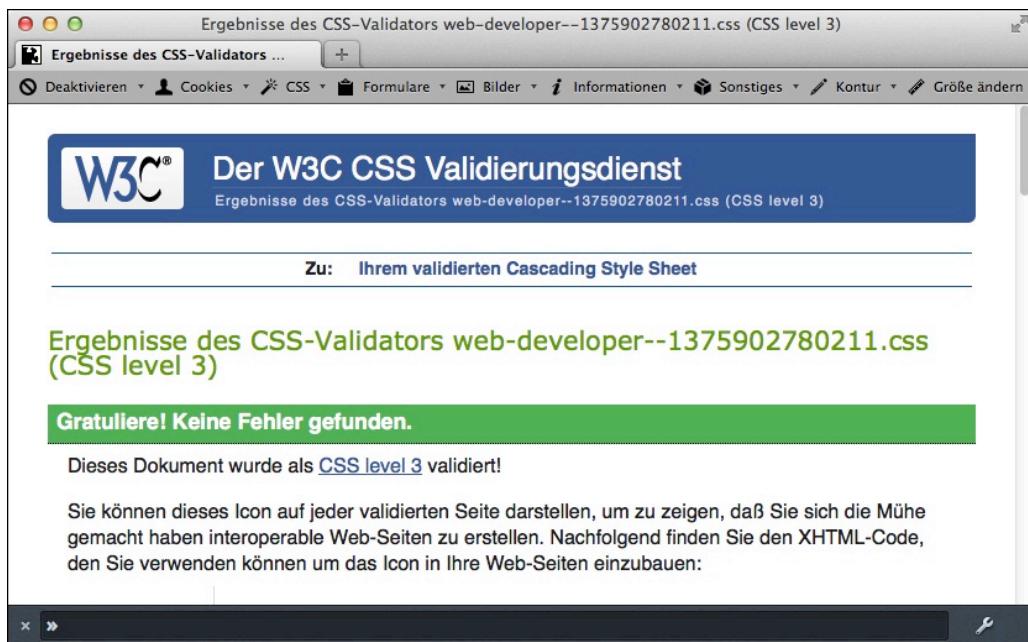


Abbildung 59: CSS-Validator

In den Entwicklertools (hier am Beispiel des Firefox) können Sie die für ein bestimmtes HTML-Element passenden CSS-Regeln anzeigen und übrigens auch testweise bearbeiten (Abbildung 60).

Nach einem Klick auf *Berechnet* werden alle Eigenschaften, die aus verschiedenen Regeln und CSS-Dateien kommen können, für das Element zusammengestellt. Durch Auswahl der Checkbox *Browser-Stile* werden auch die vom Browser vorgegebenen Eigenschaften mit angezeigt (Abbildung 61).

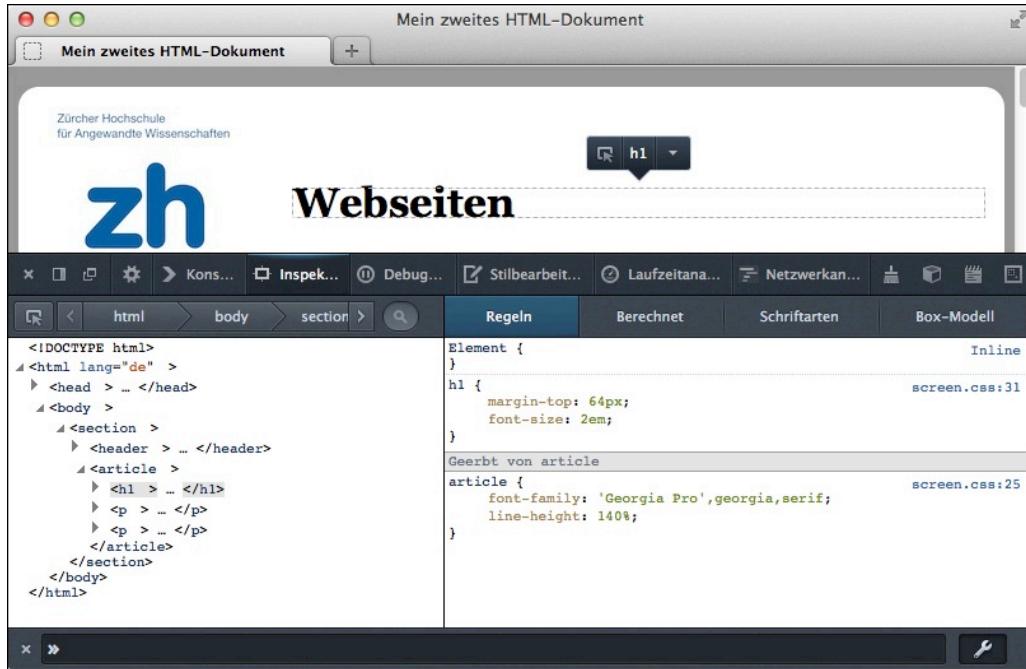


Abbildung 60: CSS-Regeln für ein bestimmtes Element

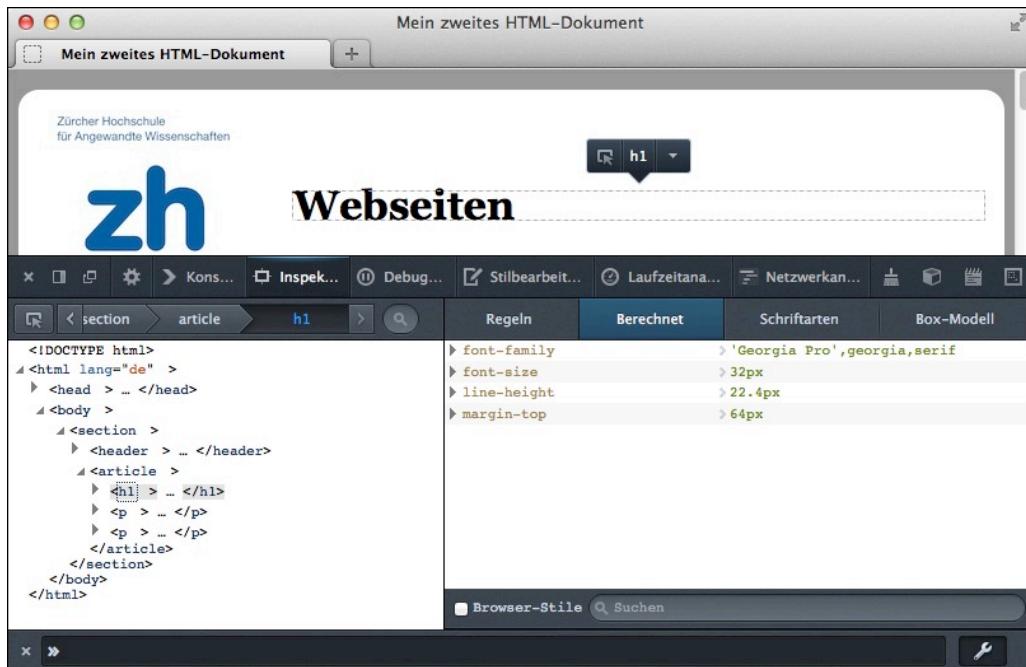


Abbildung 61: CSS-Eigenschaften für ein bestimmtes Element

## Quellen und Verweise

### Zur Vertiefung und zum Ausprobieren

- **Selectutorial - CSS selectors** (Online Tutorial)  
<http://css.maxdesign.com.au/selectutorial/index.htm>
- **Selector Demo** (Gerrit Burkert)  
<https://dublin.zhaw.ch/~bkrt/selectors/>

### Referenzen

- **DevDocs: CSS**  
<https://devdocs.io/css/>
- **CSS Reference**  
<https://cssreference.io>
- **Can I use...** - Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers  
<http://caniuse.com>

### Spezifikationen und Standards:

- **Cascading Style Sheets Level 2 Revision 2 (CSS 2.2)** Working Draft (W3C)  
<https://www.w3.org/TR/CSS22/>
- **Selectors Level 3** (W3C Recommendation)  
<http://www.w3.org/TR/css3-selectors/>
- **CSS current work & how to participate**  
<https://www.w3.org/Style/CSS/current-work.en.html>

### Quellenangaben

- **CSS3 For Web Designers** (Dan Cederholm, A Book Apart, 2010)  
<http://www.abookapart.com/products/css3-for-web-designers>

Ein Teil der Bilder und Inhalte stammt aus früheren Folien der Fächer WBD (Webdesign), INSY (Informationssysteme) und WEBT (Webtechnologien).

# Darstellung mit CSS (Teil 2)

## Einführung und Ziele

Nachdem die CSS-Grundlagen bekannt sind steht die Frage im Vordergrund, wie mit Hilfe von CSS ein Seitenlayout aufgebaut werden kann. Das wird Thema in diesem und dem folgenden Kapitel sein. Zunächst benötigen wir aber noch weitere Grundlagen:

- Das CSS-Box-Modell
- Die Möglichkeiten der CSS-Eigenschaften `position` und `float`

Mit diesen Grundlagen können wir erste Seitenlayouts aufbauen.

Die in diesem Kapitel behandelten CSS-Eigenschaften setzen einige Übung voraus, um richtig mit ihnen umgehen zu können. Wenn Sie damit noch nie gearbeitet haben, ist es sehr zu empfehlen, diese CSS-Möglichkeiten praktisch zu üben.

## Ziele

- Sie verstehen das *CSS-Box-Modell* und können die dafür benötigten CSS-Eigenschaften (Abstände, Rahmen, Größenangaben) einsetzen.
- Sie kennen die verschiedenen Möglichkeiten zur Positionierung von Elementen und die Auswirkungen der Positionierungsvarianten.
- Sie wissen, wie man die CSS-Eigenschaften `float` und `clear` einsetzt.
- Sie können mit den Mitteln von CSS2 ein mehrspaltiges CSS-Layout aufbauen.
- Sie kennen das *Front-End-Framework Bootstrap*.
- Sie verstehen *CSS Flexbox* und *CSS Grid* als Möglichkeiten zum Erstellen von Seitenlayouts und haben einen Eindruck von deren Einsatzgebieten.
- Sie kennen die Möglichkeiten der Entwicklertools im Browser zur Kontrolle der CSS-Eigenschaften für das Box-Modell und die Positionierung von Elementen.

## Relevanz für WBE

Das Box-Modell und die Möglichkeiten zur Positionierung sind für jede Art von Seitengestaltung mit CSS relevant. Traditionell erlaubt zudem die Eigenschaft `float`, Elemente nebeneinander darzustellen. Zumaldest rudimentär sollten Sie mit `float` umgehen können. Neuere Layout-Möglichkeiten wie *CSS Flexbox* und *CSS Grid* sind ungleich mächtiger, werden aber in WBE nicht vorausgesetzt. Das gilt auch für Frameworks wie *Bootstrap*.

## Box-Modell

In CSS ein Layout aufbauen heisst in erster Linie: Rechteckige Boxen auf einer Seite zu platzieren. Diese Boxen entsprechen Containerelementen im HTML-Dokument, wie `div`, `header`, `section` oder ähnlichen Elementen. Im einfachsten Fall haben die Boxen eine feste Grösse. In dynamischen bzw. flexiblen Layouts kann diese Grösse aber durchaus abhängig von verschiedenen Parametern variieren, zum Beispiel von der Grösse des Browser-Fensters oder der eingestellten Schriftgrösse.

### Breite und Höhe

Zunächst zur Breite und Höhe einer solchen Box. Diese wird mit den folgenden beiden Eigenschaften angegeben:

- `width` legt die Breite fest
- `height` definiert die Höhe

Tatsächlich ist das aber nur dann die Breite und Höhe der Box, wenn die Abstände null sind und kein Rahmen vorhanden ist. `width` und `height` definieren zunächst nämlich die Abmessungen des Boxinhals. Dazu gleich mehr.

Wenn `width` nicht angegeben wird, ist die Box so breit wie das umgebende Element. Falls keines der Elternelemente eine Breite definiert, legt das Browserfenster die Breite fest. Wenn `height` nicht angegeben ist, wird die Höhe durch den Inhalt der Box (in HTML-Ausdrucksweise: des Block-Elements) bestimmt.

Hier ist ein Ausschnitt aus einem einfachen HTML-Dokument, das nur aus einer `section` mit etwas Inhalt besteht:

```
1 <body>
2   <section>
3     Diese Box nimmt die ganze Breite ein. Überall dieselbe
4     alte Leier. ...
5   </section>
6 </body>
```

Wenn wir in CSS den Standard-Abstand des `body`-Elements zurücksetzen (zum Beispiel mit einem *Reset-Stylesheet*, s. letzte Lektion) und die `section` einfärben, um die Grösse sehen zu können, ist das Resultat wie in der Abbildung 62 gezeigt.

Die Box nimmt die gesamte verfügbare Breite und die benötigte Höhe ein. Nun ergänzen wir Breiten- und Höhenangaben:

```
1 section {
2   width: 300px;
3   height: 250px;
4 }
```

Die Box nimmt den vorgegebenen Platz ein (Abbildung 63).

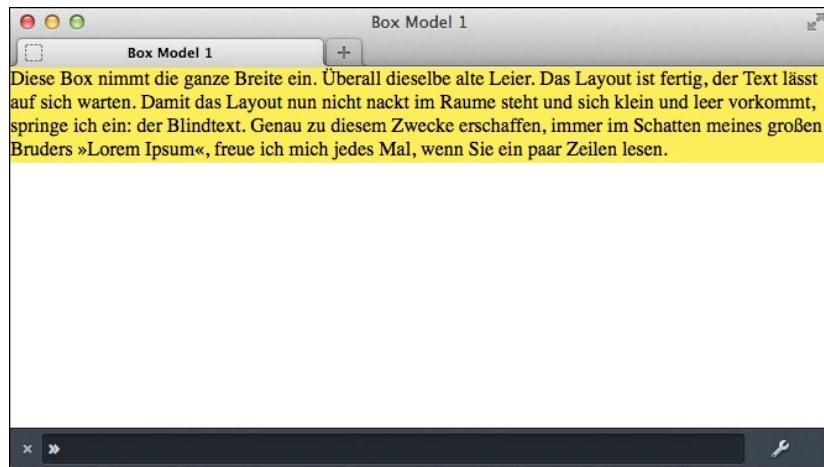


Abbildung 62: Box ohne Breiten- und Höhenangabe

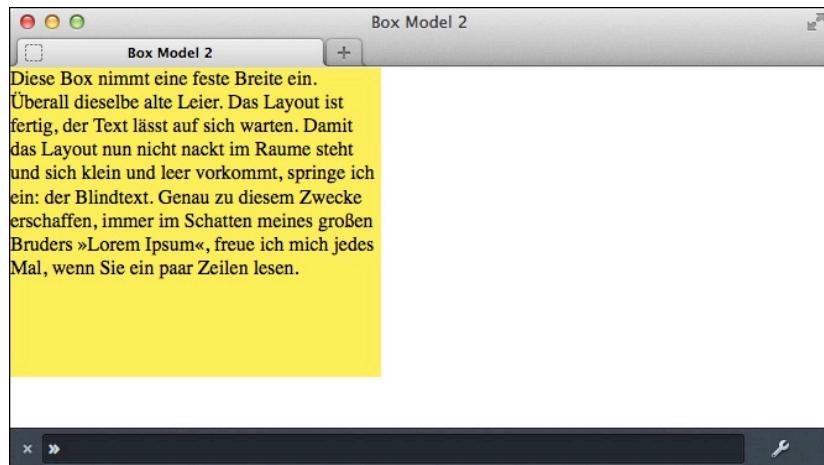


Abbildung 63: Box mit Breiten- und Höhenangabe

## Overflow

Der Textinhalt passt sich flexibel an eine vorgegebene Breite an: steht weniger Breite zur Verfügung, werden die Zeilen kürzer und der Absatz deht sich in vertikaler Richtung aus. Was geschieht aber, wenn die verfügbare Höhe für den Inhalt nicht ausreicht? Setzen wir die Höhe testweise auf nur 50px. Das wird kaum ausreichen. Das Ergebnis:

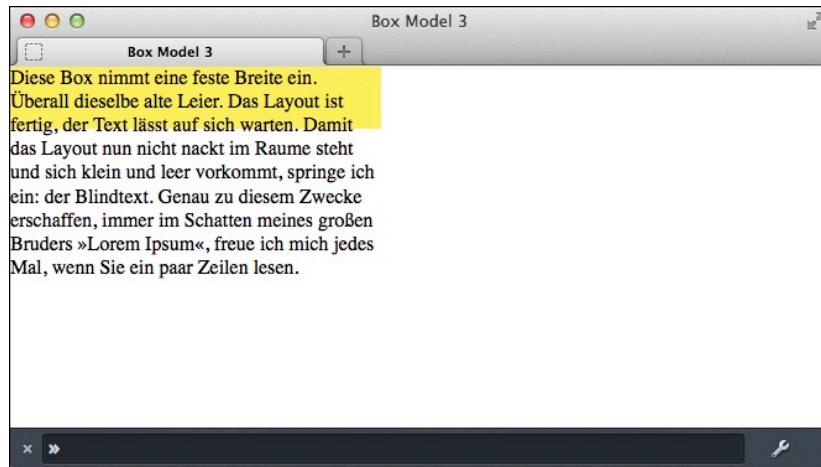


Abbildung 64: Box mit zu geringer Höhe

Das Ergebnis verwundert zunächst: Die Box hat die in der CSS-Regel spezifizierte Grösse, der Inhalt ragt aber über die Grenzen der Box hinaus. Das ist die Standardeinstellung der CSS-Eigenschaft `overflow`. Bei Bedarf kann man mit dieser Eigenschaft ein anderes Verhalten erreichen, indem man `overflow` einen der folgenden Werte gibt:

- `visible`: Inhalt ist sichtbar und kann aus der Box herausragen. Das ist wie erwähnt die Standard-Einstellung.
- `hidden`: Der Inhalt wird abgeschnitten, wenn er nicht in die Box passt.
- `scroll`: Eine Scrollbar soll angezeigt werden, um den Inhalt bei Bedarf verschieben zu können.
- `auto`: Verhalten bleibt dem User Agent überlassen. Wenn der Inhalt nicht in die Box passt, sollte aber ein Mechanismus zum Verschieben angeboten werden.

## Abstände und Rahmen

Mit den folgenden Eigenschaften kann Aussehen und Lage der Box weiter beschrieben werden:

- `padding`: Das ist der Innenabstand, zwischen Rahmen und Inhalt der Box.
- `border`: Damit lässt sich ein Rahmen um die Box zeichnen.
- `margin`: Aussenabstand der Box zu anderen Boxen oder dem übergeordneten Element.

Wir nehmen noch einmal das obige HTML-Dokument und verwenden diese CSS-Regel:

```

1  section {
2    width: 300px;
3    padding: 15px;
4    border: 10px solid #999;
5    margin: 25px;
6  }

```

Es ist keine Höhe angegeben. Diese soll sich nach dem Inhalt richten.

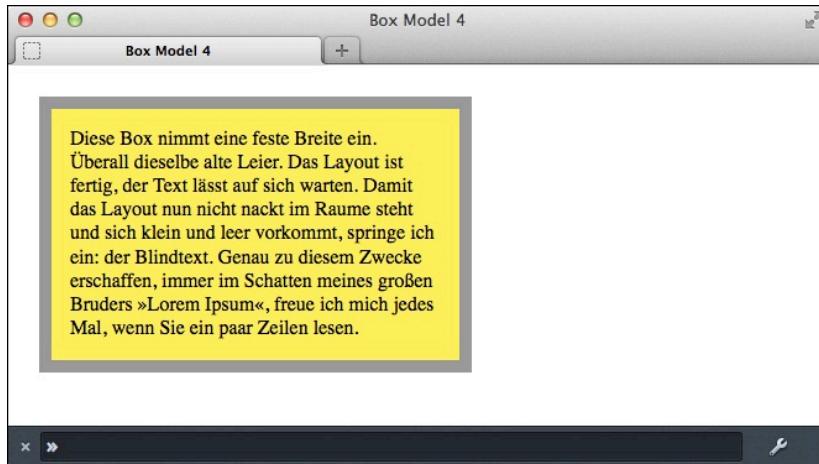


Abbildung 65: Box mit Abständen und Rahmen

Man sieht an diesem Beispiel gut den Innenabstand (`padding`) zwischen Rahmen und Inhalt. Der hier 10px breite graue Rahmen wird mit der `border`-Eigenschaft definiert. Der Außenabstand (`margin`) legt den Abstand zur Umgebung fest. Hier ist das der Abstand zum Browser-Fenster oben und links.

**Achtung:** Die Breitenangabe `width` bezieht sich nach wie vor nur auf den Inhalt der Box (Abbildung 66). Tatsächlich benötigt die Box aber mehr Platz, eine Tatsache, die oft zur Verwirrung beim Erlernen von CSS führt. Tatsächlich wurde es sogar in früheren Versionen des Internet Explorer falsch implementiert (der berühmte *Box-Model-Bug*).

Die Box selber besteht aus Inhalt, Innenabstand und Rahmen. Wird dem Element eine Hintergrundfarbe gegeben, dehnt sich diese hinter diesen drei Bereichen aus. Dass sie auch hinter dem Rahmen ist, sieht man natürlich nur, wenn dieser nicht durchgehend sondern zum Beispiel gestrichelt dargestellt wird. Das Gleiche gilt für ein Hintergrundbild.

Abstände und Rahmen können auch für die vier Seiten unterschiedlich sein. Die Angabe wird dann einfach mit `-top`, `-right`, `-bottom`, oder `-left` kombiniert, zum Beispiel:

- `padding-right: 30px;` Innenabstand rechts
- `border-left: 1px solid green;` Rahmen auf der linken Seite
- `margin-bottom: 1em;` Außenabstand unten

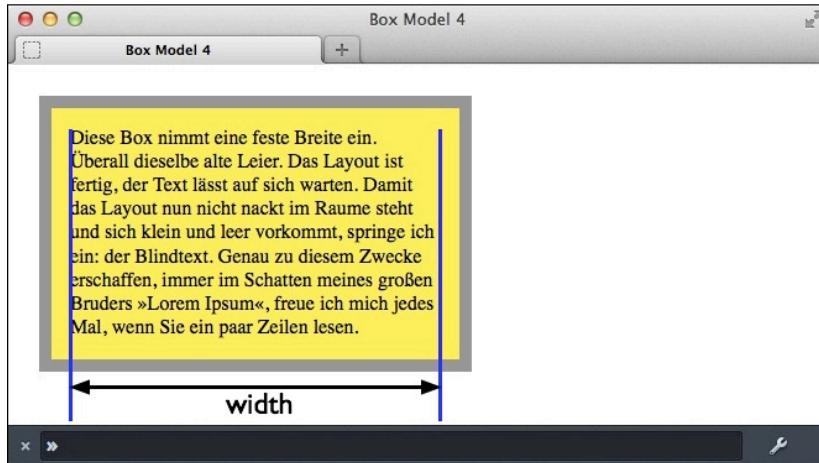


Abbildung 66: Box mit Abständen und Rahmen

Mit `margin` und `padding` können auch mehrere Zahlenangaben für die verschiedenen Seiten verwendet werden. Man sollte dann aber wissen, wie diese zu interpretieren sind. Bei vier Angaben beginnt es mit der Seite oben und dann weiter im Uhrzeigersinn:

- `margin: top right bottom left;`
- `margin: top right left bottom;`
- `margin: top bottom right left;`
- `margin: top bottom right left;`

Der Aussenabstand (`margin`) ist der Abstand zum Eltern- oder Nachbarelement: Beim ersten Element eines Container-Elements ist dies der Abstand zum Rand des Containers (Elternelement). Wenn der Container einen Innenabstand hat, summieren sich dessen Innenabstand und der Aussenabstand des Kindelements. Bei aufeinanderfolgenden Elementen definiert `margin` den Abstand zum vorangehenden Element. Aber Achtung: Wenn beide einen `margin` haben, wird nur der grösste der beiden Abstände berücksichtigt (Abbildung 67).

In den Eigenschaften `border`, `border-top`, etc. für die Definition von Rahmen sind bereits mehrere Eigenschaften zusammengefasst, die auch einzeln angegeben werden können:

- `border-width, border-top-width, ...:`  
Dicke des Rahmens. Angabe es Grösse oder einer der Werte `thin` / `medium` / `thick`
- `border-style, border-top-style, ...:`  
Art: `none` / `hidden` / `dotted` / `dashed` / `solid` / `double` / `groove` / `ridge` / `inset` / `outset`
- `border-color, border-top-color, ...:`  
Farbe des Rahmens: Farbangabe oder `transparent`

Eine weitere Eigenschaft `border-collapse` ist nur für Tabellen sinnvoll: Mit dem Wert `collapse` fallen die Rahmen benachbarter Zellen zusammen. Mit dem Wert `separate` (der Standardeinstellung) ist das nicht der Fall.

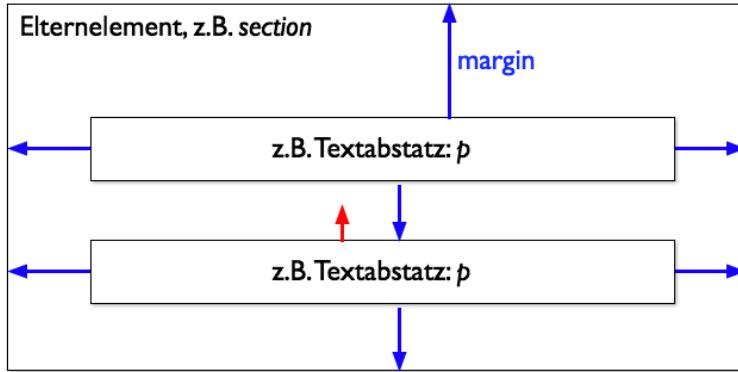


Abbildung 67: Aussenabstand

## Box-Modell im Überblick

Abbildung 68 zeigt die wichtigsten Größen des Box-Modells im Überblick.

Die Tatsache, dass `width` und `height` sich nur auf den Inhalt des Elements beziehen, hat immer wieder zu Problemen geführt. Aus diesem Grund wurde im *CSS Basic User Interface Module Level 3* eine neue Eigenschaft `box-sizing` eingeführt.<sup>58</sup> Sie erlaubt diese Werte:

- `content-box`: Größenangaben beziehen sich nur auf den Inhalt (Standard)
- `border-box`: auf Inhalt plus Innenabstand plus Rahmen bezogen

In vielen Fällen ist es sinnvoll, `box-sizing` auf `border-box` umzustellen, damit sich `width` und `height` auf die tatsächliche Größe des Elements mit Innenabstand und Rahmen bezieht.

## Horizontal zentrieren

Wird sowohl für den linken als auch für den rechten Außenabstand der Wert `auto` verwendet, wird die Box *horizontal zentriert*. Die `margin`-Angabe in der folgenden CSS-Regel macht genau das: Außenabstand oben und unten 10px, links und rechts `auto`:

```

1  section {
2    width: 300px;
3    padding: 15px;
4    border: 10px solid #999;
5    margin: 10px auto;
6  }

```

Im Browser erscheint die Box nun zentriert (Abbildung 69).

<sup>58</sup><https://www.w3.org/TR/css-ui-3/#box-model>

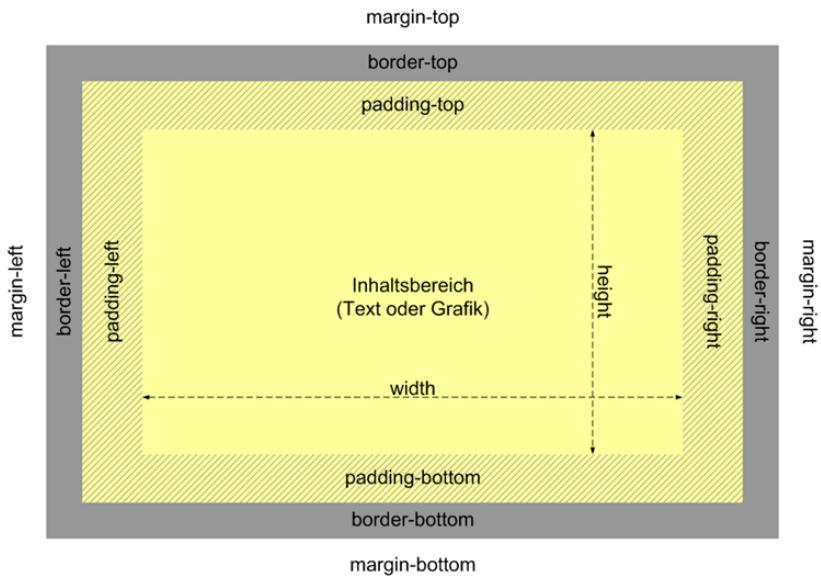


Abbildung 68: Box-Modell (Bild: little-boxes.de)

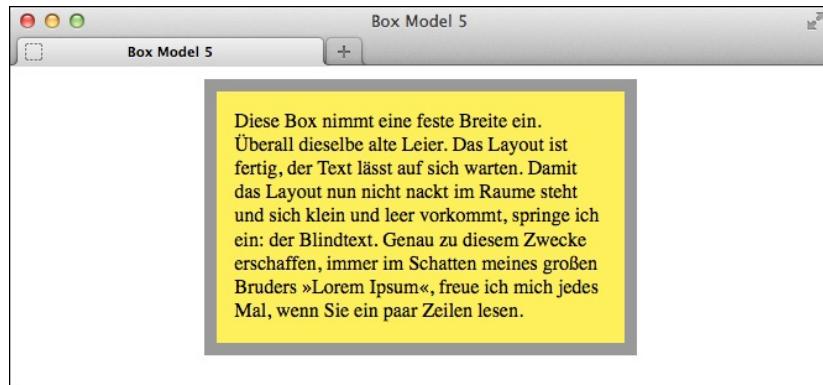


Abbildung 69: Horizontal zentrierte Box

## Fehlender Abstand?

In CSS gibt es schon die eine oder andere Ausnahme oder Abweichung vom eigentlich erwarteten Verhalten. Im folgenden Ausschnitt würde man um jedes der Elemente den Außenabstand von 20px erwarten:

```
1 <div id="eins">
2   <div id="zwei">
3     <div id="drei">aa</div>
4     <div id="vier">bb</div>
5   </div>
6 </div>

1 #eins, #zwei, #drei, #vier {
2   margin: 20px;
3 }
4 #eins {
5   height: 100px;
6 }
```

Tatsächlich fehlt bei den eingebetteten Elementen der Abstand nach oben und unten (Abbildung 70).

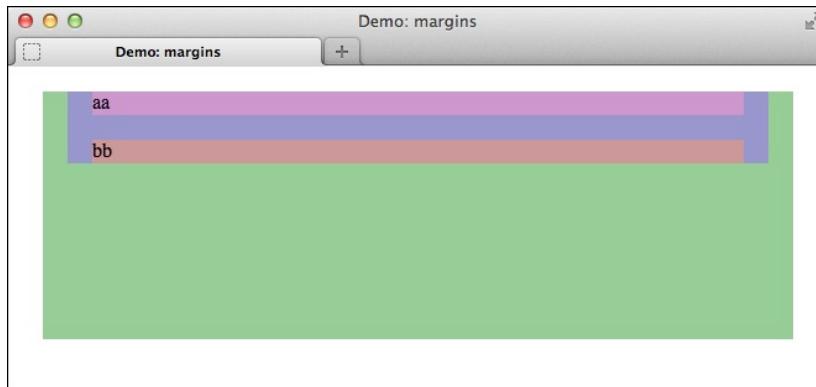


Abbildung 70: Fehlender Außenabstand

Eine Erklärung liefert die CSS-Spezifikation:<sup>59</sup>

“The top margin of an in-flow block-level element is adjoining to its first in-flow block-level child’s top margin if the element has no top border, no top padding, and the child has no clearance.”

---

<sup>59</sup><http://www.w3.org/TR/CSS22/box.html#collapsing-margins>

## Hintergrund

Eine Hintergrundfarbe kann mit `background-color` definiert werden.

Ein Hintergrundbild wird mit `background-image` festgelegt. Wenn es kleiner als die Box ist, kann es mehrfach aneinandergefügt werden. Das lässt sich mit `background-repeat` steuern, mögliche Werte sind `repeat` / `repeat-x` / `repeat-y` / `no-repeat` / `inherit`. Zwei weitere Eigenschaften `background-attachment` und `background-position` legen fest, ob der Hintergrund mit der Seite scrollen soll und wie das Hintergrundbild zu positionieren ist.

Beispiel:

```
1 body {  
2     background-color: #ccc;  
3     background-image: url("logo.png");  
4     background-attachment: fixed;  
5     background-position: 100% 100%;  
6     background-repeat: no-repeat;  
7 }
```

Ähnlich wie bei `border` gibt es auch hier eine Eigenschaft `background`, mit der die gerade beschriebenen Eigenschaften zusammengefasst werden können. Die obige Regel kann kürzer auch so geschrieben werden:

```
1 body {  
2     background: #ccc url("logo.png") no-repeat fixed 100% 100%;  
3 }
```

Das Hintergrundbild ist rechts unten fixiert:

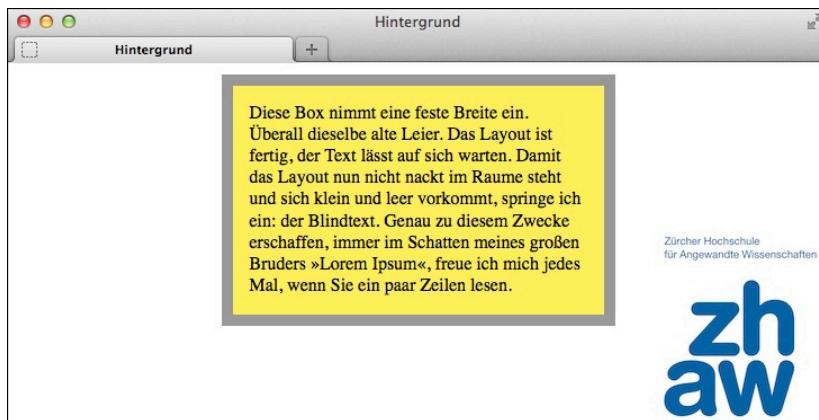


Abbildung 71: Hintergrund-Eigenschaften

## Minimale und maximale Dimensionen

Statt eine feste Breite oder Höhe anzugeben können auch erlaubte Bereiche angegeben werden:

- `min-width`: minimale Breite
- `max-width`: maximale Breite
- `min-height`: minimale Höhe
- `max-height`: maximale Höhe

Das ist sinnvoll beim Erstellen von dynamischen Layouts, die sich an die Fenstergrösse anpassen. Um gute Lesbarkeit zu erreichen, sollte die Textbreite (in der Typografie *Satzbreite* genannt) nicht zu klein und nicht zu gross sein.

## Positionierung

Die erste Grundlage für das Verstehen von CSS-Layouts ist das gerade behandelte Box-Model. Ebenso wichtig sind aber die verschiedenen Möglichkeiten, Elemente auf der Seite zu positionieren. Diesem Zweck dient die `position`-Eigenschaft. Je nach Art der Positionierung wird diese mit weiteren Angaben wie `top`, `left`, ... kombiniert.

### `position: static`

Das ist die Standardeinstellung. Aufeinanderfolgende statisch positionierte Block-Elemente werden einfach untereinander angeordnet. Um das zu demonstrieren, beginnen wir mit einem Stück HTML-Code, das drei solche Elemente enthält:

```
1 <body>
2   <section>Section 1</section>
3   <section id="section2">Section 2</section>
4   <section>Section 3</section>
5 </body>
```

Dazu wird folgender CSS-Code verwendet:

```
1 body {
2   margin: 0;
3   padding: 0;
4 }
5 section {
6   position: static;
7   height: 60px;
8   background-color: #fe6;
9   border: 1px solid #999;
10 }
11 #section2 {
12   width: 80%;
13   background-color: #cef;
14 }
```

Das `position: static` wäre gar nicht nötig, da es die Voreinstellung ist. Den mittleren Block gestalten wir etwas anders – der Selektor `#section2` ist spezifischer als `section`, deswegen “gewinnt” diese Regel. Das Resultat sieht wie erwartet aus (Abbildung 72).

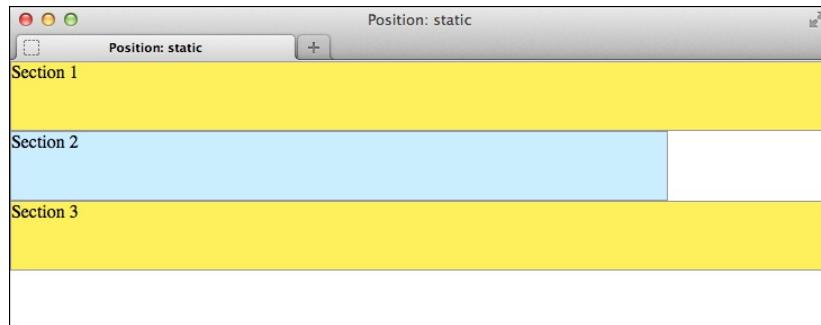


Abbildung 72: position: static

### position: relative

Ein relativ positioniertes Element kann von seiner Normalposition verschoben werden. Sein ursprünglicher Platz im Elementfluss bleibt trotzdem frei. Im Beispiel positionieren wir das mittlere Element relativ und verschieben es um 20px nach unten und um 80px nach rechts:

```
1 #section2 {  
2     position: relative;  
3     top: 20px; left: 80px;  
4     width: 80%;  
5     background-color: #cef;  
6 }
```

Abbildung 73 zeigt das Ergebnis.



Abbildung 73: position: relative

Neben `top` und `left` gibt es auch die Eigenschaften `bottom` und `right`, die die Verschiebung von unten bzw. rechts her definieren.

Es kann auch sinnvoll sein, ein Element relativ zu positionieren, ohne es zu verschieben. Es ist dann zwar an der gleichen Position, auf der es auch mit `position: static` wäre. Es verhält sich aber anders in Bezug auf die Positionierung von Kindelementen. Dazu gleich mehr.

### position: absolute

Elemente mit `position: absolute` werden aus dem Elementfluss herausgenommen und entsprechend der Angaben `top`, `left`, `bottom` or `right` positioniert. Ändern wir im Beispiel nur die Positionierung des mittleren Elements auf `absolute`:

```
1 #section2 {  
2     position: absolute;  
3     top: 20px; left: 80px;  
4     width: 80%;  
5     background-color: #cef;  
6 }
```

Dann rücken die beiden äusseren Sections zusammen und `#section2` ist absolut positioniert (Abbildung 74).



Abbildung 74: position: absolute

Eine Frage ist noch offen: Woran orientieren sich die Längenangaben der Position bei absoluter Positionierung? Antwort: Sie werden gemessen vom Rand des nächsten übergeordneten Elements, das nicht statisch positioniert ist. Falls es kein solches gibt: Dann am Rand des Wurzelelements `html`, also eigentlich der Rand des gesamten Dokuments. Wenn der Seiteninhalt nicht ins Browserfenster passt, scrollt das absolut positionierte Element mit dem Inhalt des Dokuments.

So *absolut* ist die *absolute Positionierung* also nicht. Sie kann sich auf ein anderes Element beziehen. Das sieht man etwas besser, wenn wir das Beispiel etwas anpassen. Jede der drei Sections enthält nun ein eingebettetes `div`-Element:

```

1 <body>
2   <section>Section 1 <div>1.1</div></section>
3   <section>Section 2 <div>2.1</div></section>
4   <section>Section 3 <div>3.1</div></section>
5 </body>

```

Das CSS wird ebenfalls etwas angepasst:

```

1 body {
2   margin: 0; padding: 0;
3 }
4 section {
5   position: static;
6   height: 60px;
7   background-color: #fe6;
8   border: 1px solid #999;
9 }
10 section:nth-child(2) {
11   position: relative;
12   width: 80%;
13   background-color: #cef;
14 }
15 section div {
16   position: absolute;
17   top: 10px; left: 100px;
18   width: 100px;
19   background-color: white;
20   border: 1px solid #999;
21 }

```

Das mittlere `section`-Element hat nun keine `id` mehr. Daher greifen wir über einen Pseudoklassen-Selektor auf dieses Element zu. Die Elemente sind so positioniert:

- Die äusseren Sections `static`
- Die mittlere Section `relative`
- Die eingebetteten Blocks `absolute`

Das Ergebnis sehen Sie in Abbildung 75.

Wo ist der Block mit dem Text “1.1” geblieben? Der ist unter dem “3.1” verschwunden. Der Grund ist, dass die absolut positionierten `div`-Elemente sich am Rand des Dokuments orientieren, da es kein übergeordnetes nicht-statisches Element gibt. Das heisst sie fallen auf die gleiche Position 10px von oben und 100px von links gemessen von der linken oberen Ecke der Seite. Die Ausnahme ist der Block mit dem Text “2.1”. Er hat ein relativ positioniertes Elternelement, das sein Koordinatensystem für die Positionierung festlegt.

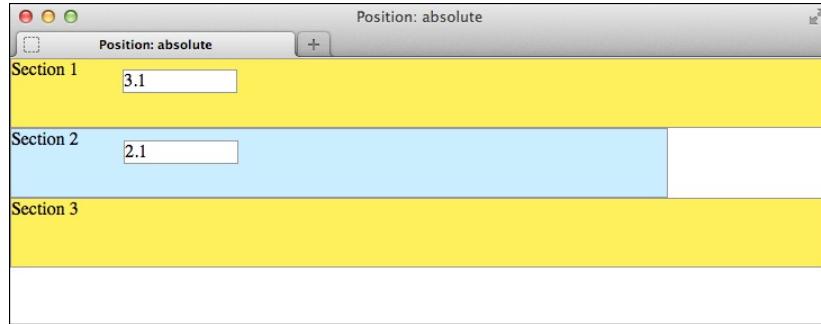


Abbildung 75: position: absolute

### position: fixed

In diesem Fall bezieht sich die Position auf den Rand des *Viewports* (Bereich des Browserfensters, in dem die Webseite angezeigt wird). Wenn das Dokument gescrollt wird, bleibt das Element mit `position: fixed` trotzdem stehen. Ändern wir zum Test in unserem Beispiel die Positionierung der inneren div- Blocks auf `fixed`:

```

1 section div {
2     position: fixed;
3     /*...*/
4 }
```

Dann sind alle eingebetteten Blocks übereinander und nur der letzte ist sichtbar. Beim Scrollen bleibt der kleine weisse Block an seiner Position stehen (Abbildung 76).

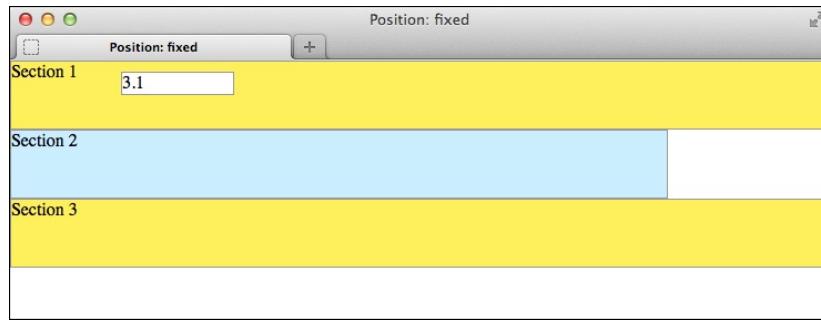


Abbildung 76: position: fixed

### z-index

Die Eigenschaft `z-index` ermöglicht es, die Reihenfolge, in der Elemente übereinander auf der Seite abgelegt (“gestapelt”) werden, zu beeinflussen. Das *z* steht hier für die dritte Koordinate zusätzlich zu *x* und *y*, die für die Position in der Ebene zuständig sind.

Fügen wir dem Beispiel oben noch diese Regel hinzu:

```

1  section:nth-child(2) div {
2      z-index: 1;
3 }

```

In diesem Fall hat das in der mittleren `section` eingebettete `div` einen höheren `z-index` als die anderen, deren Voreinstellung 0 ist. Statt “3.1” wird also “2.1” in der weissen Box angezeigt.

Es ist vor allem wichtig zu wissen, dass die Reihenfolge im Prinzip anpassbar ist. Im Detail kann das aber relativ kompliziert werden. Bei Bedarf können Sie die Zusammenhänge in der CSS-Spezifikation nachlesen.<sup>60</sup>

## Positionierung insgesamt

Wir haben hier nicht alle Details zur Positionierung und alle Abhängigkeiten berücksichtigt. Zum Beispiel: Welche Rolle spielt `margin` bei den verschiedenen Positionierungsarten? Wichtig ist, die verschiedenen Varianten zunächst einmal grundsätzlich zu verstehen. Um die Details kann man sich dann später kümmern, wenn das Ergebnis der Arbeit nicht wie gewünscht aussieht.

**Empfehlung:** Im Artikel *CSS Positioning 101* sind die verschiedenen Varianten mit Beispielen schön zusammengestellt.<sup>61</sup>

## Fliessende Boxen

Eine weitere wichtige Eigenschaft, mit der die Position von Elementen beeinflusst werden kann, ist `float`. Das ursprüngliche Einsatzgebiet für `float` war es, ein Bild von Text umfliessen zu lassen. Wenn für ein Element die `float`-Eigenschaft gesetzt wird, bedeutet dies: Das Element wird aus dem Elementfluss herausgehoben, nachfolgende Elemente rücken dadurch nach oben. Der Inhalt der nach oben rückenden nachfolgenden Elemente wird durch das “umflossene” Element dabei verdrängt. Die Eigenschaft `float` kann folgende Werte haben:

- `left`: Das Element rückt nach links und wird rechts vom Inhalt des nachfolgenden Elements umflossen.
- `right`: Das Element rückt nach rechts und wird links vom Inhalt des nachfolgenden Elements umflossen.
- `none`: Das Element wird nicht umflossen. Das ist der Normalfall und die Standardeinstellung.

Zunächst ein einfaches Beispiel: Der HTML-Code besteht nur aus einem `header` mit Bild und einer nachfolgenden `section` mit Text.

```

1 <body>
2     <header></header>
3     <section>Überall dieselbe alte Leier... </section>
4 </body>

```

---

<sup>60</sup><https://www.w3.org/TR/CSS22/visuren.html#z-index>

<sup>61</sup><https://alistapart.com/article/css-positioning-101/>

Normalerweise würde der Text unterhalb des Headers mit dem Bild positioniert. Setzt man für das Bild allerdings `float: left`, rückt der Text nach oben und fliest um das Bild. Hier das komplette CSS:

```

1 body {
2     margin: 8px; padding: 0;
3 }
4 header {
5     float: left;
6     width: 100px;
7 }
8 section {
9     outline: 2px dashed red;
10}

```

Mit `outline` wird hier ein gestrichelter Rahmen definiert. `outline` funktioniert ähnlich wie `border`, im Gegensatz zu `border` beeinflusst es aber das Box-Model nicht, verändert also nicht den Platzbedarf der Box. Daher kann `outline` problemlos zu Testzwecken temporär eingefügt werden.

Die gestrichelte Umrandung wurde hier nur eingefügt, um die Abmessungen des `section`-Elements zu zeigen. Dieses rückt komplett nach oben, sein Inhalt wird aber durch das umflossene Bild verdrängt (Abbildung 77).

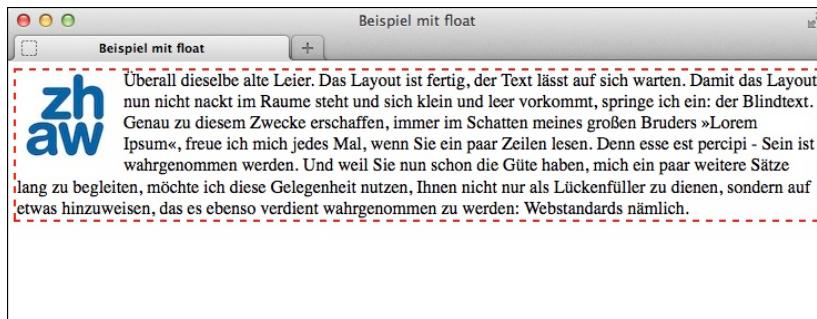


Abbildung 77: Positionierung mit float

Wenn wir den Abschnitt nun noch mit einem linken Außenabstand von 120px versehen, haben wir das Bild schön in einer eigenen "Spalte" untergebracht (Abbildung 78).

Es dürfte nun klar sein, dass man mit `float: right` erreichen kann, das Bild auf der rechten Seite zu positionieren.

Wenn das umflossene Element hoch ist, können mehrere nachfolgende Blocks nach oben geschoben und ihr Inhalt durch das umflossene Element verdrängt werden. Möchte man dies nicht, setzt man für das erste Nachfolgeelement, das das ursprüngliche Element nicht mehr umfliessen soll, die `clear`-Eigenschaft, die folgende Werte haben kann:

- `left`: Bricht das Umfliessen eines `float:left` ab
- `right`: Bricht das Umfliessen eines `float:right` ab

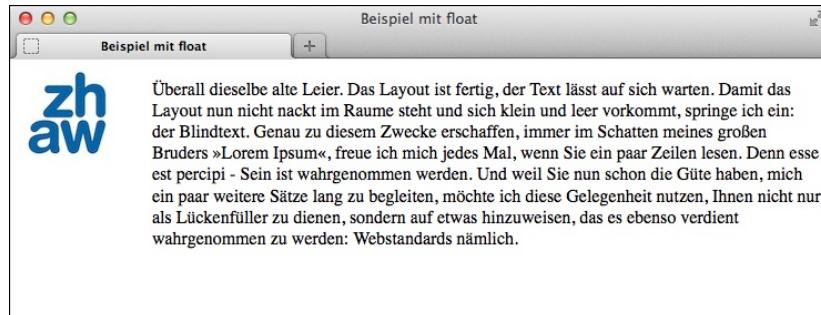


Abbildung 78: Positionierung mit float

- both: Bricht das Umfliessen auf beiden Seiten ab
- none: Bricht das Umfliessen nicht ab (Standard-Einstellung)

## Layout mit CSS

Mit den oben beschriebenen Mitteln lassen sich auch komplexe Layouts erstellen. Und zwar sowohl für heute meist hochauflösende Desktop- und Notebook-Bildschirme als auch für andere Geräte wie Smartphones oder Tablet-Computer. Im Sinne einer *Mobile-First*-Strategie müsste man sich zuerst mit der Mobilplattform beschäftigen, zumal diese stark an Bedeutung gewonnen hat. Aber unser vorrangiges Ziel im Moment ist es, CSS und seine Möglichkeiten zum Seitenlayout zu erlernen, und da bieten grössere Bildschirme mehr Möglichkeiten. Daher werden wir in dieser Lektion die speziellen Anforderungen kleiner Bildschirme von Mobilgeräten nicht berücksichtigen und uns entsprechend der Entwicklung im Web erst einmal mit üblichen Grössen von Desktop-Monitoren beschäftigen. Auf den *Mobile-First*-Ansatz kommen wir im Kurs *Mobile Applications (MOBA)* zurück. Dass wir uns vorerst mit Desktop-Geräten beschäftigen heisst nicht, dass unsere Seitenlayouts nicht auch auf Smartphones verwendbar sind. Smartphone-Browser sind recht gut darin, herkömmlich gestaltete Seiten auf dem kleinen Bildschirm nutzbar zu machen.

### Layout-Varianten

Häufig werden **Layouts mit fester Breite (fixed width)** eingesetzt. Der Grund ist, dass diese relativ einfach aufzubauen sind und weniger unerwartete Effekte auftreten, wenn zum Beispiel die Grösse des Browserfensters geändert wird. Ein anderer Grund könnte sein, dass die Gestaltung oft von Print-Designern entworfen wird, die feste Grössenverhältnisse bevorzugen.

Als Layout-Breite wird meist ein Wert im Bereich von etwa 950px bis 980px verwendet. Das ist ein Kompromiss, damit die Seite sowohl auf kleineren (1024px Auflösung in der Breite) als auch auf grösseren Bildschirmen gut verwendbar ist. Von einer solchen Breite gehen übrigens auch Smartphone-Browser aus, um die Seite auf die tatsächliche Bildschirmgrösse zu skalieren.

Merkmal eines solchen Layouts ist, dass dem Container-Element, das unter `body` angeordnet ist (oft ein `div`), eine feste Breite zugewiesen wird. Damit die Inhalte auf grossen Bildschirmen nicht auf der linken Seite „kleben“, wird das ganze Layout meistens mit `margin-left: auto; margin-right: auto;` horizontal zentriert (Abbildung 79).

```
1 div#main {
2     width: 960px;
3     margin-left: auto;
4     margin-right: auto;
5 }
```

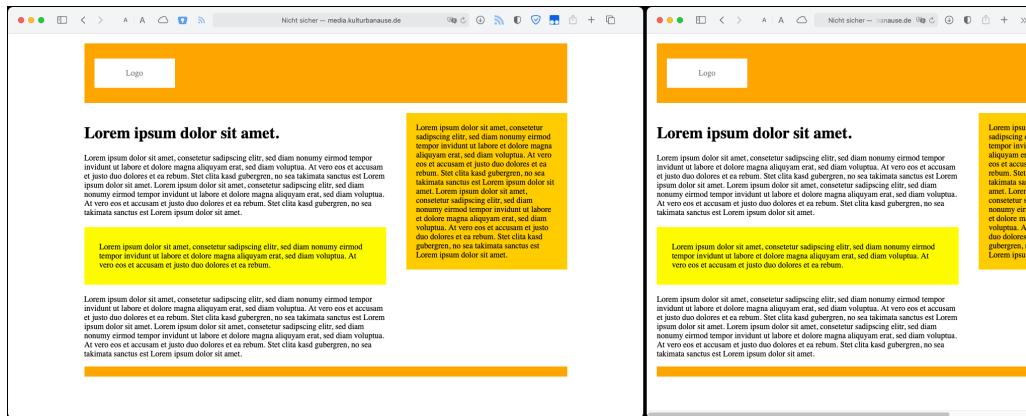


Abbildung 79: Fixed Layout, zentriert (Quelle: kulturbanause.de)

Eine andere Layoutvariante ist das **fliessende Layout (fluid, liquid)**. Es passt sich an die Grösse des Browserfensters an. Das gilt auch für die Inhalte, also auch die Breite der Texte. Es muss also darauf geachtet werden, dass die Texte weder zu schmal noch zu breit werden, um die Lesbarkeit nicht zu beeinträchtigen. Zu diesem Zweck können die Eigenschaften `max-width` und `min-width` verwendet werden (Abbildung 80).

Fluid Layouts sind etwas schwieriger umzusetzen als Layouts mit fester Breite. So muss zum Beispiel darauf geachtet werden, dass Bilder mit skaliert werden, wenn die Breite der umgebenden Box kleiner als die Breite des Bilds ist:

```
1 img {
2     max-width: 100%;
3 }
```

Ausserdem funktioniert ein fliessendes Layout nur in einem bestimmten Bereich von Viewport-Grössen gut. Wenn der Bildschirm sehr klein ist, kann es nötig werden, einzelne Elemente untereinander statt nebeneinander anzuordnen. Dann sind wir im Bereich der *responsiven Layouts* und benötigen zusätzlich *Media Queries*.

Eine dritte Layout-Variante wird gelegentlich mit der Bezeichnung **elastisches Layout (elastic layout)** versehen. Hier wird die Breite des Containers an die gewählte Schriftgrösse angepasst, die Breitenangabe im CSS erfolgt also in `em` oder `rem`. Auf diese Weise reagiert die Darstellung auf die Funktion *Text skalieren* des Browsers: Zusammen mit der

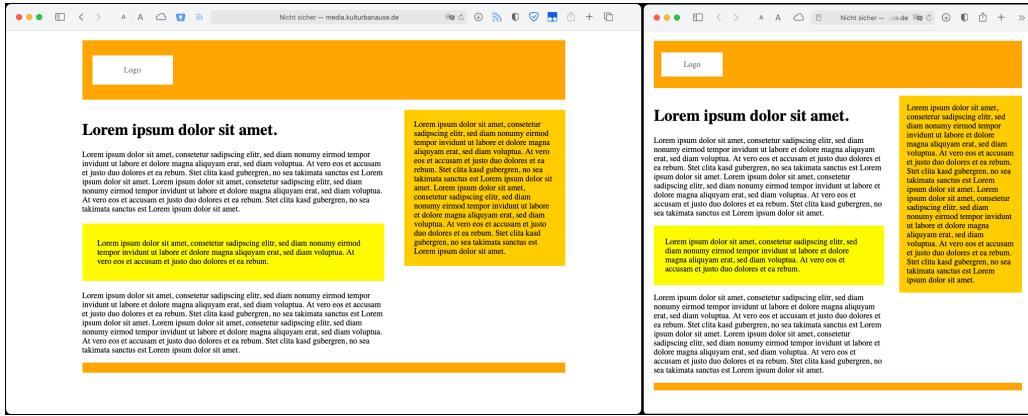


Abbildung 80: Fluid Layout, zentriert (Quelle: kulturbanause.de)

Schriftgrösse variiert die Breite der angezeigten Inhalte. Das Problem ist, dass heutige Browser in der Regel über eine Funktion zum Skalieren des ganzen Inhalts verfügen, so dass ein elastisches Layout keinen grossen Zusatznutzen bringt.

**Hinweis:** Die Bezeichnungen für die verschiedenen Layout-Typen (*fixed*, *fluid*, *liquid*, *elastic*) werden nicht immer ganz einheitlich verwendet.

## Mehrspaltiges Layout

Mit den Mitteln von CSS2 lassen sich mehrspaltige Layouts am besten mit **float** umsetzen. Das ist aber nicht ganz trivial und entsprechende Layouts funktionieren häufig nur unter bestimmten Randbedingungen, zum Beispiel dass der Inhalt der mittleren Spalte höher ist als der Inhalt der anderen Spalten. Abbildung 81 zeigt ein solches auf der Basis von **float** erstelltes Layout. Es ist unter *Tutorial 9. Liquid three column layout* auf einer nicht mehr ganz neuen Website mit CSS-Tutorials im Detail beschrieben.<sup>62</sup>

Sollen in den einzelnen Bereichen weitere Elemente positioniert werden, muss die **position**-Eigenschaft der Container geeignet gesetzt werden.

Für Seiten mit viel Text kann es sinnvoll sein, diesen in mehreren Spalten anzuzeigen. Zu diesem Zweck kann auch das *CSS Multi-column Layout Module Level 1* verwendet werden, das mittlerweile von den Browsern gut unterstützt wird (Abbildung 82).<sup>63</sup>

## Horizontale Navigation

Eine Navigationsleiste wird in HTML als geordnete oder ungeordnete Liste von Verweisen umgesetzt. Wenn es sich um die Hauptnavigation der Website handelt, wir sich die Liste in einem **nav**-Element befinden. Der HTML-Code könnte also folgendermassen aussehen:

<sup>62</sup><http://css.maxdesign.com.au/floottutorial/tutorial0901.htm>

<sup>63</sup><https://www.w3.org/TR/css-multicol-1/>

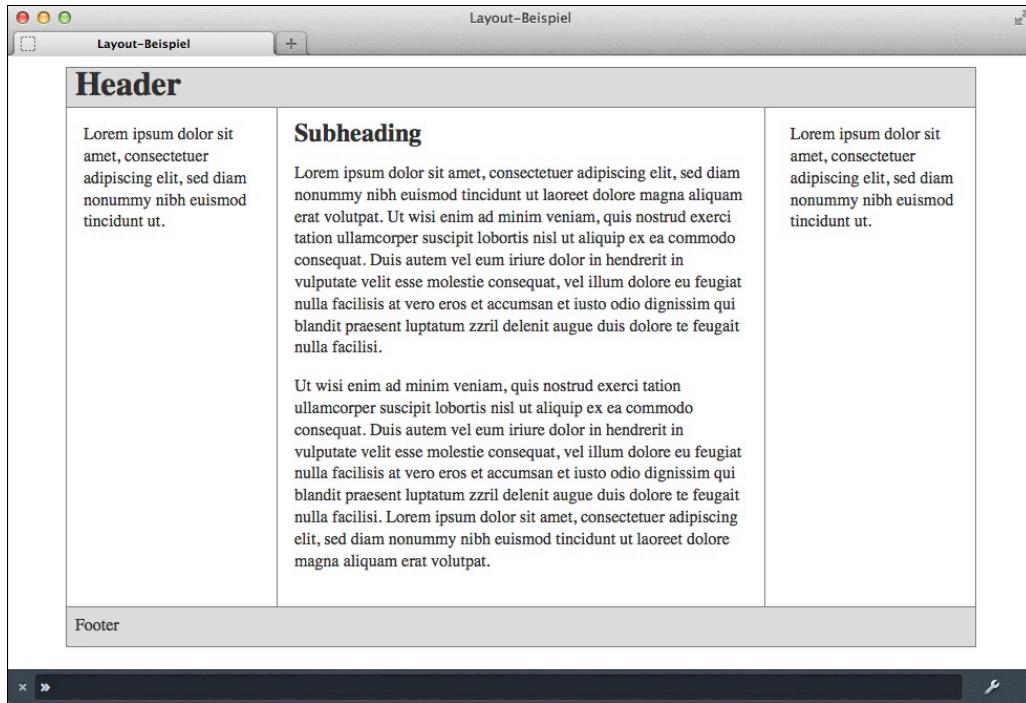


Abbildung 81: Layout-Beispiel

<h2>Example 2: Using Column-Count</h2> <p>In this example column-count has been used to set up the multi-columns, which makes the layout more flexible.</p> <p>Notice that if you resize your browser to make it narrower the width of the columns in this</p>	<p>example continues to adjust.</p> <p>The pseudo-algorithm calculates one based on the other and since the count is set, the width of the columns can adjust based on the available width.</p>	<p>The very simple css is seen below.</p> <pre>#example-2 {     column-count: 3;     column-rule: 1px solid #ccc }</pre> <p>The column-rule creates the vertical lines between columns in the center of the column-gap.</p>
--	---	---

Abbildung 82: Beispiel mit drei Spalten

```

1  <nav>
2      <ul>
3          <li><a href="#">Home</a></li>
4          <li><a href="#">About</a></li>
5          <li><a href="#">Services</a></li>
6          <li><a href="#">Contact us</a></li>
7      </ul>
8  </nav>

```

Die Attributwerte “#” als Verweisziel sind natürlich nicht sinnvoll. Es handelt sich hier nur um eine vorübergehende Lösung bis die definitiven Verweisziele bekannt sind und eingetragen werden können.

Die `li`-Elemente sind Block-Level-Elemente, das heisst normalerweise werden sie untereinander angeordnet. Angenommen, wir möchten die Navigation als horizontale Navigationsleiste auf der Seite platzieren, wie es die Abbildung 83 zeigt.

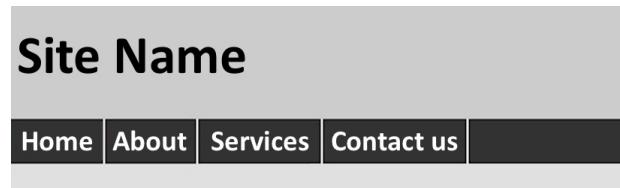


Abbildung 83: Horizontale Navigation

Dies lässt sich auf zwei Arten umsetzen:

- Mit `float:left` können die einzelnen `li`-Elemente nebeneinander statt untereinander angeordnet werden.
- Mit `display:inline` lässt sich dies ebenfalls erreichen.

Beide Varianten haben ihre Vor- und Nachteile. In den meisten Fällen dürfte `float:left` vorzuziehen sein. Nachdem `float` bereits ausführlich behandelt wurde, sind hier noch einige Bemerkungen zur `display`-Eigenschaft angebracht. Die `display`-Eigenschaft kann folgende Werte haben:

- `block`: Das Element wird zu einem Block-Level-Element. Es kann zum Beispiel sinnvoll sein, aus einem `a`-Element in einer Navigation ein Block-Element zu machen, das grösser ist als der eigentliche Text des Verweises. Die ganze Box reagiert dann auf Mausklicks.
- `inline`: Das Element wird zu einem Inline-Element. Damit können aus Block-Elementen wie `li` Inline-Elemente gemacht werden, die nebeneinander angeordnet sind. Damit sind aber auch die Nachteile von Inline-Elementen verbunden: da sie keine Block-Elemente enthalten können, sind die Möglichkeiten weiterer Strukturierung und Gestaltung begrenzt.
- `inline-block`: Ein Element mit dieser `display`-Eigenschaft verhält sich nach aussen hin wie ein Inline-Element, das heisst es kann im Textfluss vorkommen und mehrere dieser Elemente können nebeneinander angeordnet werden. Nach innen ist

es aber ein Block-Element, in dem weitere Elemente, auch Block-Elemente, auftauchen können.

- **none**: Das Element taucht in der Ausgabe nicht auf und belegt auch keinen Platz. Im Gegensatz dazu wird ein Element durch **visibility:hidden** nur unsichtbar, belegt aber weiterhin Platz.
- **list-item**: Macht aus dem Element ein Listenelement.
- Weitere Werte können Elementen das Verhalten von Tabellenelementen zuweisen: **table**, **inline-table**, **table-row-group**, **table-column**, **table-column-group**, **table-header-group**, **table-footer-group**, **table-row**, **table-cell**, **table-caption**. Es gab schon Ideen, diese Möglichkeit zu benutzen, um ohne Tabellenelemente nur mit CSS ein Tabellenlayout machen zu können. Das ist aber nur in Ausnahmefällen eine gute Idee.

Noch ein Hinweis zu **display: inline-block**: Eine neuere Spezifikation *CSS Display Module Level 3* schlägt verschiedene Erweiterungen der **display**-Eigenschaft vor.<sup>64</sup> So soll die Rolle eines Elements gemäss dieser Spezifikation nach aussen und nach innen separat beschrieben werden können:

```
1 display: inline-block;      /* legacy CSS2 display value */  
2 display: inline flow-root; /* equivalent level 3 specification */
```

Bisher ist diese Spezifikation aber nur im Firefox-Browser umgesetzt und es erscheint fraglich, ob sie sich durchsetzt.

Noch einmal zurück zum Einsatz von **display:inline** für die horizontale Navigation. Das Problem, dass bei der Änderung der Listenelemente zu einem Inline-Element diese keine Block-Elemente mehr enthalten können, kann wie beschrieben durch den Einsatz von **inline-block** behoben werden. Es gibt aber noch ein anderes Problem, das für beide Varianten zutrifft: Wenn zwischen den **li**-Elementen Leerzeichen oder Zeilenwechsel sind (das dürfte üblicherweise der Fall sein), taucht in der Navigationsleiste ein Leerzeichen zwischen den Verweisen auf. Wenn diese Lücken unerwünscht sind, kann man sie in vielen Fällen durch Tricks mit Hintergrundfarben unsichtbar machen.

## Front-End-Framework: Bootstrap

Das Erstellen von Webseiten mit HTML und CSS ist mit einigem Aufwand verbunden. Wenn Zeit und andere Ressourcen – zum Beispiel finanzielle oder personelle – begrenzt sind, wird man auf Hilfsmittel zurückgreifen müssen. In manchen Fällen genügt es, ein Content Management System (CMS) mit einem vorgefertigten Design einzusetzen. Häufig ist aber ein individuelles Design gefragt. Damit man auch in diesen Fällen nicht ganz bei Null anfangen muss, gibt es eine Reihe von Hilfsmitteln, zu denen auch die so genannten Front-End-Frameworks zählen.

---

<sup>64</sup>Die Spezifikation finden Sie hier: <https://www.w3.org/TR/css-display-3/>, Artikel zur Einführung: <https://hacks.mozilla.org/2019/10/the-two-value-syntax-of-the-css-display-property/>

Das bekannteste Front-End-Framework ist Bootstrap von Twitter. Auf der Website<sup>65</sup> wird es wie folgt beschrieben:

### Build fast, responsive sites with Bootstrap

Quickly design and customize responsive mobile-first sites with Bootstrap, the world's most popular front-end open source toolkit, featuring Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful JavaScript plugins.

Zum Aufbau von Seitenlayout unterstützt Bootstrap ein Grid-System. Das ist ein Gestaltungs raster, auf dem Elemente positioniert werden können. Bei Bootstrap besteht dieses Raster aus zwölf Spalten. Mit Hilfe von Klassen kann angegeben werden, über wie viele Spalten ein Element sich erstrecken soll, bei Bedarf auch abhängig von der verfügbaren Viewport-Breite:

```
1 <div class="container">
2   <div class="row">
3     <div class="col-12 col-md-8">.col-12 .col-md-8</div>
4     <div class="col-6 col-md-4">.col-6 .col-md-4</div>
5   </div>
6 </div>
```

Im Beispielcode wird eine Reihe von `div`-Elementen angegeben. Ab einer bestimmten Viewport-Breite (Default: 768px) werden die Angaben mit `-md` (Medium) verwendet, das heisst das erste `div` überdeckt acht und das zweite vier Spalten. Abbildung 84 zeigt diese Situation, wobei noch zwei weitere Reihen von Elementen abgebildet sind. Steht nicht genügend Platz zur Verfügung, bewirkt `col-12`, dass das erste `div` die ganze Breite einnimmt und das zweite `div` auf die nächste Zeile verschoben wird.



Abbildung 84: Bootstrap Grid

Weitere Möglichkeiten sind in der Dokumentation von Bootstrap beschrieben.<sup>66</sup> Ein solches Grid-System kann den Aufbau eines Seitenlayouts deutlich vereinfachen. Das ist der Grund, warum Bootstrap unter Web-Entwicklern sehr beliebt ist.

Neuere CSS-Standards, speziell *CSS Flexbox* und *CSS Grid*, erlauben ebenfalls den Aufbau komplexer Layouts (s.u.), so dass Werkzeuge wie Bootstrap mit der Zeit etwas an Bedeutung verlieren könnten. Bootstrap besteht allerdings nicht nur aus dem Grid-System.

<sup>65</sup><https://getbootstrap.com>

<sup>66</sup><https://getbootstrap.com/docs/5.0/layout/grid/>

Es enthält ausserdem:

- Typografie-Vorgaben: Darstellung für Überschriften, Absätze, Listen
- Gestaltungsvorlagen für Tabellen, Formulare, Navigationselemente, Buttons, ...
- Fortschrittsbalken, modale Dialoge, Alerts, u.a.
- Ein Satz von Icons
- Verschiedene optionale Erweiterungen

Sicher ist Ihnen aufgefallen, dass Klassen wie `col-12` nicht den Zielen der *semantischen Auszeichnung* entsprechen. Bootstrap lässt sich aber in vieler Hinsicht anpassen und es können problemlos passendere Klassenbezeichnungen wie `introtext` verwendet werden. Alternativ kann aber auch direkt eine fertige Distribution heruntergeladen werden, wenn keine oder weniger Anpassungen daran vorgenommen werden sollen.

## CSS Flexbox und CSS Grid

Das Erstellen von komplexen Layouts mit den Mitteln von CSS2 war sehr umständlich. Zwei neuere CSS-Standards sollen Abhilfe schaffen:

- *CSS Flexible Box Layout Module Level 1*, kurz: *Flexbox*<sup>67</sup>
- *CSS Grid Layout Module Level 1 und 2*<sup>68</sup>

Mit **Flexbox** ist es möglich, Seitenelemente flexibel in einem Container zu verteilen. Dieses Verteilen bezieht sich auf *eine Dimension*, horizontal oder vertikal. Dabei ist aber auch ein Umbruch auf die nächste Zeile oder Spalte möglich. Somit werden die Möglichkeiten beträchtlich erweitert.

Flexbox wird mit der `display`-Eigenschaft aktiviert:

```
1 .app-wrap {  
2   display: flex;  
3   flex-direction: column;  
4 }
```

Weitere CSS-Eigenschaften erlauben es, Anordnung, Grösse und Ausrichtung der enthaltenen Elemente zu steuern. Eine schöne Übersicht der Möglichkeiten liefert *A Complete Guide to Flexbox* von *CSS-Tricks*.<sup>69</sup>

Abbildung 85 zeigt ein Layout für Mobilgeräte, welches mit Flexbox umgesetzt wurde. Im Beispiel wurde Flexbox in vertikaler Richtung für den Hauptcontainer eingesetzt. Die Navigationsleiste in der Fusszeile verteilt die Buttons ebenfalls mit Flexbox, diesmal in horizontaler Richtung.

**CSS Grid** ist etwas neuer als Flexbox, mindestens Level 1 wird aber von aktuellen Browsern gut unterstützt. CSS Grid wird eingesetzt, wenn sich die Spezifikation eines Layouts über zwei Dimensionen erstreckt. Ein Layout wie in Abbildung 86 gezeigt lässt

<sup>67</sup><https://www.w3.org/TR/css-flexbox-1/>

<sup>68</sup><https://www.w3.org/TR/css-grid-1/>, <https://www.w3.org/TR/css-grid-2/>

<sup>69</sup><https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

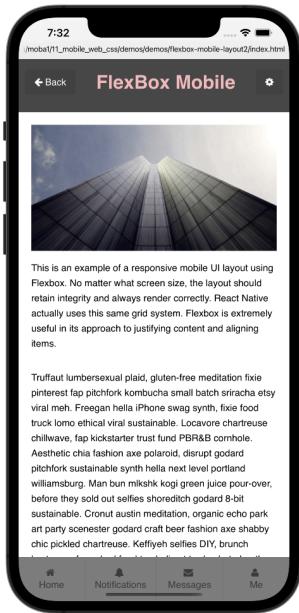


Abbildung 85: Flexbox Example

sich damit auf einfache Weise umsetzen. Der Artikel *A Complete Guide to Grid* von *CSS-Tricks* stellt die Möglichkeiten zusammen.<sup>70</sup>

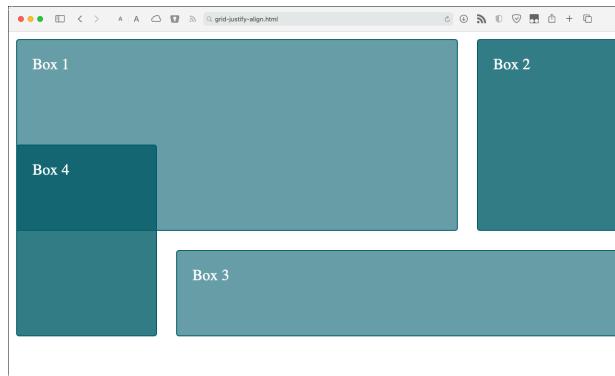


Abbildung 86: Grid Example

---

<sup>70</sup><https://css-tricks.com/snippets/css/complete-guide-grid/>

## Kontrolle und Fehlersuche

Layout-Probleme lassen sich am besten in den Entwickler-Tools der Browser analysieren. Man kann dort sehr einfach den Bereich auf der Seite ansehen, den jedes einzelne Element belegt. Neben den bereits in der letzten Lektion gezeigten Möglichkeiten zur Anzeige der CSS-Regeln und einer Liste aller angewendeten CSS-Eigenschaften lassen sich auch die wichtigsten Abmessungen und Abstände des Box-Modells anzeigen:

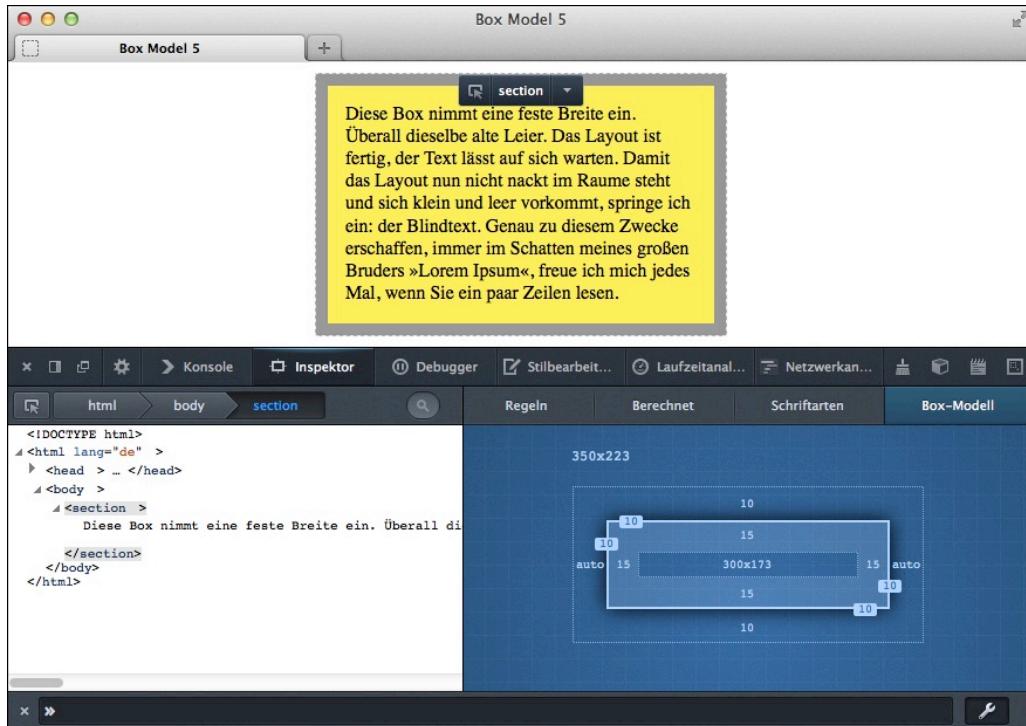


Abbildung 87: Box-Modell in den Browser-Entwicklertools

# Quellen und Verweise

## Material zur Vertiefung

- **Learn CSS Positioning in Ten Steps** (Online Tutorial)  
<http://www.barelyfitz.com/screencast/html-training/css/positioning/>
- **CSS Positioning 101** (A List Apart)  
<http://alistapart.com/article/css-positioning-101>
- **Floatutorial: Liquid three column layout** (Online Tutorial)  
<http://css.maxdesign.com.au/floatautorial/tutorial0901.htm>

## Referenzen

- **DevDocs: CSS**  
<https://devdocs.io/css/>
- **CSS Reference**  
<https://cssreference.io>
- **Can I use...** - Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers  
<http://caniuse.com>

## Spezifikationen und Standards:

- **Cascading Style Sheets Level 2 Revision 2 (CSS 2.2)** Working Draft (W3C)  
<https://www.w3.org/TR/CSS22/>
- **CSS current work & how to participate**  
<https://www.w3.org/Style/CSS/current-work.en.html>

## Quellenangaben

Ein Teil der Bilder und Inhalte stammt aus früheren Folien der Fächer WBD (Webdesign), INSY (Informationssysteme) und WEBT (Webtechnologien).

# Darstellung mit CSS (Teil 3)

## Einführung und Ziele

Im dritten Kapitel über CSS sehen wir uns weitere Möglichkeiten an, mit denen die Gestaltung und die Benutzung von Webseiten verbessert werden können. Zunächst werden die Möglichkeiten von Hintergrundbildern und CSS-Sprites beschrieben. Anschliessend steht eine kleine Auswahl neuerer CSS-Features auf dem Programm. Zum Schluss werden Ansätze zum Aufbau von CSS-Dateien diskutiert und Werkzeuge zur Arbeit mit CSS vorgestellt.

### Ziele

- *Sie kennen die Möglichkeiten, die CSS2 im Zusammenhang mit Hintergrundbildern bietet und wissen, dass CSS3 das Platzieren mehrerer Hintergrundbilder in einem Element erlaubt.*
- *Sie wissen, wozu und wie CSS-Sprites eingesetzt werden.*
- *Sie können Boxen mit abgerundeten Ecken und Schatten versehen und die Durchsichtigkeit von Farben und Elementen angeben.*
- *Sie kennen die wichtigsten Möglichkeiten von Transitionen, Transformationen und Animationen.*
- *Sie wissen, wie man mit CSS Variablen definieren und referenzieren kann.*
- *Sie kennen die CSS-Erweiterungen LESS und Sass.*

### Relevanz für WBE

Die in diesem Kapitel behandelten Themen runden das Wissen über CSS ab. Sie werden im Kurs WBE aber nicht vorausgesetzt.

# CSS-Techniken

## Hintergrundbilder

Elemente können mit einem Hintergrundbild versehen werden. Hier ist noch einmal das Beispiel aus der letzten Lektion, in dem die verschiedenen CSS-Eigenschaften für Hintergrundbilder gezeigt werden:

```
1 body {  
2     background-color: #ccc;  
3     background-image: url("logo.png");  
4     background-attachment: fixed;  
5     background-position: 100% 100%;  
6     background-repeat: no-repeat;  
7 }
```

Wir kennen nun zwei Möglichkeiten, Bilder auf einer Webseite zu platzieren: Einmal im HTML-Code über das `img`-Element und zum andern im CSS durch Hinzufügen eines Hintergrundbilds. Wann soll welche Variante gewählt werden? Die Antwort ist nicht so schwierig, wenn man überlegt, welche Rolle HTML und welche CSS spielen soll. Daher:

- Wenn ein Bild Teil des Inhalts eines Dokuments ist, sollte es im HTML in einem `img`-Element referenziert werden. Es ist dann auch bei ausgeschaltetem CSS als Teil des Dokuments sichtbar.
- Wenn ein Bild nur dazu dient, die Darstellung/Gestaltung eines Dokuments zu verbessern, also eine Art *Verzierung* ist, wird es über das Stylesheet hinzugefügt.

Sicher gibt es manchmal Fälle, in denen es nicht ganz klar ist, welche der beiden Rollen ein Bild spielt. Dann kann man sich entscheiden, welche Variante eher zutreffend ist. Die Unterscheidung sollte sich auch in der Verzeichnisstruktur der Website widerspiegeln, zum Beispiel so:

```
website  
|---- index.html  
|---- imgs (Bilder, die zum Inhalt gehören, im HTML referenziert)  
|---- styles  
|       |---- screen.css  
|       |---- imgs (Bilder, die zur Darstellung gehören, CSS)  
|       |
```

Gehen wir noch etwas auf die Möglichkeiten von Hintergrundbildern ein, inklusive einiger Erweiterungen, die CSS3 in diesem Bereich bietet. Zunächst zu den bereits erwähnten Eigenschaften:

- `background-image` legt ein Hintergrundbild fest. Die Bilddatei wird mit einer `url()`-Angabe spezifiziert. Seit der Beschäftigung mit dem Box-Modell wissen Sie, dass das Hintergrundbild den Bereich Inhalt, Innenabstand und Rahmen abdeckt. Das Hintergrundbild liegt über der Hintergrundfarbe. Wenn das Bild Transparenzen enthält, kann also die Hintergrundfarbe durchscheinen. Der Rahmen liegt über dem Hintergrundbild.

- `background-repeat` gibt an, ob das Hintergrundbild wiederholt werden soll, wenn es kleiner ist als die Box, die das Element repräsentiert. Mit dem Wert `repeat` wird es beliebig oft in beiden Richtungen wiederholt, mit `repeat-x` nur in horizontaler und mit `repeat-y` nur in vertikaler Richtung. Der Wert `no-repeat` legt fest, dass das Bild nicht wiederholt werden soll. Standardeinstellung ist `repeat`.
- `background-attachment` bietet eine Möglichkeit, die Position eines Hintergrundbilds relativ zum Viewport zu fixieren. Dies geschieht mit dem Wert `fixed`. Das Bild bewegt sich dann beim Scrollen nicht mit, ist aber nur hinter dem Element sichtbar, für das es mit `background-image` angelegt wurde. Standardwert für `background-attachment` ist aber `scroll`, das heißt das Hintergrundbild bewegt sich beim Scrollen mit dem Element.
- `background-position` gibt die Position des Hintergrundbilds an. Hier sind bis zu zwei Angaben möglich. Zuerst wird die horizontale, dann die vertikale Position angegeben:
  - Prozentangabe, 0% ist ganz links/oben und 100% ganz rechts/unten
  - Längenangabe in px oder ähnlich
  - Die Werte `left`, `center`, `right` bzw. `top`, `center`, `bottom`

Bei den Schlüsselwort-Werten kann die Reihenfolge auch vertauscht werden, also `top center` entspricht `50% 0%`. Standardwert ist links oben, also `0% 0%`. Die Positionsangaben beziehen sich normalerweise auf das Element, das den Hintergrund erhält. Anders ist es bei `background-attachment:fixed`. Dann bezieht sich die Position auf den Viewport.

Die genannten Eigenschaften können zusammen mit `background-color` in einer einzigen Eigenschaft `background` zusammengefasst werden:

```

1 body {
2   background: #ccc url("logo.png") no-repeat fixed 100% 100%;
3 }
```

Mit `background-attachment:fixed` lassen sich interessante Effekte erzielen. Zum Beispiel kann man verschiedene Elemente mit verschiedenen eingefärbten Versionen des gleichen Hintergrundbilds versehen. Dadurch entsteht der Eindruck, dass das Hintergrundbild durch den Hintergrund des Elements hindurchscheint. Das Scrollen über das Hintergrundbild verstärkt diesen Eindruck. Ein Beispiel ist die *Complexspiral Demo* von Eric Meyer (Abbildung 88).<sup>71</sup>

Eric Meyer weist darauf hin, dass diese Demo für die wesentlichen Teile nur CSS1-Features verwendet. Mit den Möglichkeiten von CSS3, insbesondere den Eigenschaften `opacity` und Farbwerten mit `rgba()` kann dieser Effekt auf einfachere Weise erzielt werden.

In **CSS3** wurden die Möglichkeiten für Hintergrundbilder erweitert. Nun ist es möglich, dass ein Element auch mehrere Hintergrundbilder hat. Jede der Eigenschaften hat dann mehrere, durch Kommas getrennte Werte. Hier ein Beispiel:<sup>72</sup>

---

<sup>71</sup><https://meyerweb.com/eric/css/edge/complexspiral/demo.html>

<sup>72</sup>Quelle: CSS3 for Web Designers, <http://www.abookapart.com/products/css3-for-web-designers>

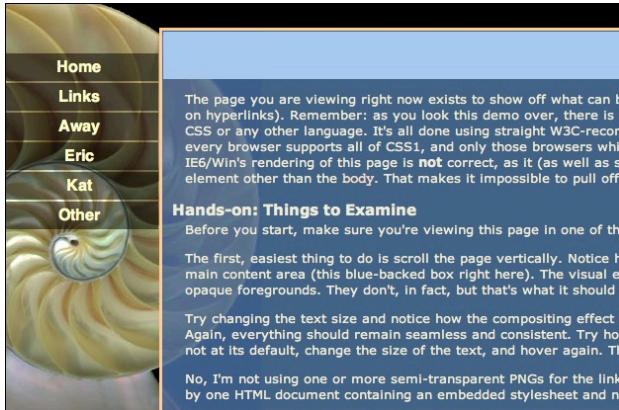


Abbildung 88: Complexspiral Demo

```

1 body {
2     background: url(..../img/space-bg.png) repeat-x fixed -80% 0;
3     background:
4         url(..../img/stars-1.png) repeat-x fixed -130% 0,
5         url(..../img/stars-2.png) repeat-x fixed 40% 0,
6         url(..../img/space-bg.png) repeat-x fixed -80% 0,
7         url(..../img/clouds.png) repeat-x fixed 100% 0;
8     background-color: #1a1a1a;
9 }

```

Mehrere Hintergrundbilder funktionieren in aktuellen Browsern problemlos. Für den Fall, dass ein Browser Probleme damit hat, wurde im Beispiel noch eine `background`-Eigenschaft mit nur einem Bild eingeführt. Neuere Browser überschreiben diese Deklaration mit der nächsten, die mehrere Hintergrundbilder definiert.

Ausser mehreren Hintergrundbildern stellt CSS3 noch weitere Möglichkeiten zur Verfügung. Für `background-repeat` gibt es noch die Werte `space` (Hintergrundbild wird so oft wiederholt, wie es ganz in den verfügbaren Platz passt, der Rest bleibt leer) und `round` (Hintergrundbild wird so oft wiederholt, wie es ganz in den verfügbaren Platz passt, dabei wird die Grösse so angepasst, dass es aufgeht). Weitere Eigenschaften sind `background-clip`, `background-origin` und `background-size`. Interessierte können die Details in der Spezifikation nachlesen: *CSS Backgrounds and Borders Module Level 3*.<sup>73</sup>

## CSS-Sprites

Wenn in einem HTML-Dokument viele Bilder geladen werden müssen, zum Beispiel viele kleine Icons, kann dies die Ladezeit des Dokuments ziemlich vergrössern. Für jedes einzelne dieser Bilder muss ein HTTP-Request an den Server geschickt werden. Auch bei Navigationen mit Roll-Over-Images – also Bilder, die je nach Zustand (normal, mouseover, active) ausgetauscht werden – fallen mehrere Bilddateien an. Das Laden zahlreicher

---

<sup>73</sup><http://www.w3.org/TR/css3-background/>

Einzelbilder kann man vermeiden, indem man diese Bilder neben- oder untereinander in einem grösseren Bild platziert. Für eine Navigationsleiste könnte dies so aussehen:

iPhone	iPad	iTunes	Support	
iPhone	iPad	iTunes	Support	
iPhone	iPad	iTunes	Support	
iPhone	iPad	iTunes	Support	

Abbildung 89: CSS-Sprites für die Navigation (Ausschnitt), Quelle: apple.com

Nun muss man nur nach dafür sorgen, dass an den gewünschten Stellen der passende Bildausschnitt, also etwa ein Navigationselement im gewünschten Zustand, angezeigt wird. Dazu wird an allen Stellen, wo ein Teil des Bildes sichtbar werden soll, das gesamte Bild als Hintergrundbild eingefügt. Um den richtigen Ausschnitt zu zeigen, wird es zusätzlich mit `background-position` geeignet positioniert.

Ein einfaches Beispiel soll dies verdeutlichen: Es wird ein Bild mit drei Navigationsbuttons verwendet: *Home*, *Previous*, *Next* (Abbildung 90).



Abbildung 90: Beispiel für CSS-Sprites, Quelle: w3schools.com

Im CSS-Ausschnitt für einen der Navigationsbuttons sehen Sie, dass die Breite des Elements und die Position des Hintergrundbilds so angepasst werden, dass genau der Pfeil nach rechts angezeigt wird. Beim Überfahren mit dem Mauszeiger (`:hover`) wird das Hintergrundbild so verschoben, dass die dunklere Variante angezeigt wird:

```

1 #next {
2     left: 129px;
3     width: 43px;
4     background: url('img_navsprites_hover.gif') -91px 0;
5 }
6 #next a:hover {
7     background: url('img_navsprites_hover.gif') -91px -45px;
8 }
```

Das Beispiel kann auf der Website W3Schools genauer untersucht und ausprobiert werden.<sup>74</sup> CSS-Sprites können die Ladezeiten deutlich reduzieren, wenn viele kleine Bilder benötigt werden. Daher wird diese Technik auf vielen Websites eingesetzt. Vor allem mit CSS eingefügte Hintergrundbilder werden mit Sprites zusammengefasst. Theoretisch könnte die Technik auf für Vordergrundbilder eingesetzt werden, indem der anzuzeigende Bereich mit `overflow: hidden` begrenzt und das Bild entsprechend positioniert wird.

<sup>74</sup>[https://www.w3schools.com/css/tryit.asp?filename=trycss\\_sprites\\_hover\\_nav](https://www.w3schools.com/css/tryit.asp?filename=trycss_sprites_hover_nav)

Dies widerspricht aber der Vorstellung, dass das Bild Teil des Inhalts sein soll und auch mit ausgeschaltetem CSS der Seiteninhalt noch sinnvoll genutzt werden kann.

## Mehr zu CSS

Wie bereits ausgeführt wurde CSS nach CSS2 in vielerlei Hinsicht erweitert. Diese Erweiterungen – häufig unter der Bezeichnung CSS3 zusammengefasst – sind auf der Seite *CSS current work & how to participate* des W3C zusammengestellt.<sup>75</sup> Einigen dieser neuen Features sind wir bereits begegnet:

- Es gibt zahlreiche neue Selektoren, um gezielt einzelne Elemente aus einer HTML-Datei auswählen zu können.
- Neu können auch Font-Dateien geladen werden, um auch Schriftfamilien einzusetzen zu können, für die man nicht unbedingt annehmen kann, dass sie die Website-Besucher auf ihren Systemen installiert haben.
- Flexbox und CSS Grid erlauben es, komplexe Seitenlayouts zu erstellen.
- Elemente können mehrere Hintergrundbilder haben. Außerdem gibt es weitere Möglichkeiten, die Darstellung von Hintergrundbildern zu beeinflussen.

Die neuen CSS-Möglichkeiten sind auf eine Sammlung von Modulen aufgeteilt, welche sich in sehr unterschiedlichem Zustand befinden. Einzelne Module sind bereits fertig spezifiziert und befinden sich im Status einer *Recommendation* (REC), andere sind immerhin schon *Candidate Recommendations* (CR). Die meisten Module sind aber noch in einer (teilweise frühen) Konzeptphase (*Working Draft*, WD). Abbildung 91 zeigt einen kleinen Ausschnitt der aktuellen Arbeiten an CSS-Modulen.

## Boxen und Text verschönern

### Abgerundete Ecken

Ein grosses Defizit von CSS2 war nach Ansicht vieler Webdesigner, dass man Boxen nicht auf einfache Weise mit *abgerundeten Ecken* versehen konnte. Das lässt sich schon daraus schliessen, dass mit allen möglichen Tricks versucht wurde, solche abgerundeten Ecken zu erzeugen. Zum Beispiel mit Hintergrundbildern, die auf mühsame Weise in den Ecken platziert wurden – man hatte ja auch nicht die Möglichkeit, mehrere Hintergrundbilder anzugeben. Das war relativ viel Aufwand für ein in der Regel recht bescheidenes Ergebnis. Mit CSS3 geht es nun elegant mit der **border-radius**-Eigenschaft:

```
1 section.eins {  
2     border-radius: 25px;  
3 }  
4 section.zwei {  
5     border-radius: 50%;  
6     padding: 35px;  
7 }
```

Positionierung, Grösse, Farben, Rahmeneigenschaften sind bereits an anderer Stelle definiert worden. Abbildung 92 zeigt, wie die beiden Boxen mit diesen Regeln aussehen.

---

<sup>75</sup> <https://www.w3.org/Style/CSS/current-work.en.html>

## TABLE OF SPECIFICATIONS

Ordered from most to least stable:

Completed	Current	Upcoming	Notes	
CSS Snapshot 2020	NOTE		Latest stable CSS	(i)
CSS Color Level 3	REC	REC		(i)
CSS Namespaces	REC	REC		(i)
Selectors Level 3	REC	REC		(i)
CSS Level 2 Revision 1	REC	REC	See Errata	(i)
Media Queries	REC	REC		(i)
CSS Style Attributes	REC	REC		(i)
CSS Cascading and Inheritance Level 3	REC	REC		(i)
CSS Fonts Level 3	REC	REC		(i)
CSS Writing Modes Level 3	REC	REC		(i)
CSS Basic User Interface Level 3	REC	REC		(i)
CSS Containment Level 1	REC	REC		(i)
Stable	Current	Upcoming	Notes	
CSS Backgrounds and Borders Level 3	CR	PR		(i)
CSS Conditional Rules Level 3	CR	CR		(i)
CSS Multi-column Layout Level 1	WD	CR		(i)
CSS Values and Units Level 3	CR	PR		(i)
CSS Flexible Box Layout Level 1	CR	PR		(i)
CSS Counter Styles Level 3	CR	PR		(i)

Abbildung 91: CSS3 Module (Auszug)

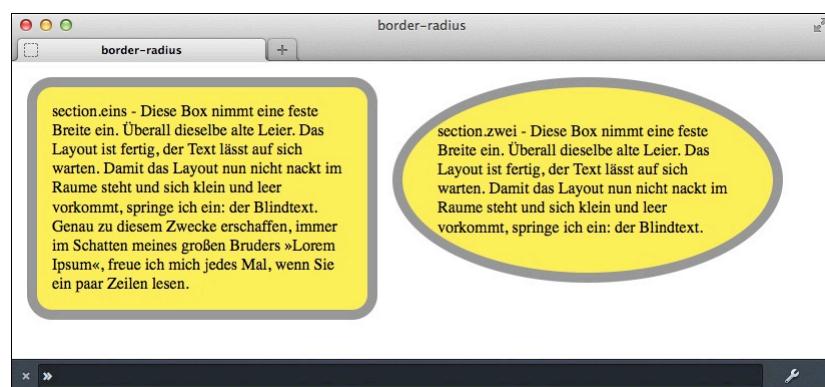


Abbildung 92: CSS3-Eigenschaft *border-radius*

Der Radius kann als Länge oder in Prozent angegeben werden. Die Prozentangabe bezieht sich auf die Höhe und Breite der Box. Es können auch gleich mehrere Werte angegeben werden. Bei vier Werten bezieht sich der erste auf die Ecke oben links, die weiteren auf die im Uhrzeigersinn anschliessenden Ecken. Außerdem können die Ecken auch einzeln spezifiziert werden:

- `border-top-left-radius`
- `border-top-right-radius`
- `border-bottom-right-radius`
- `border-bottom-left-radius`

Jede dieser Eigenschaften kann wieder zwei Werte haben. Der erste ist dann die Abrundung in horizontaler, der zweite in vertikaler Richtung. Weitere Möglichkeiten können Sie bei Bedarf der Spezifikation *CSS Backgrounds and Borders Module Level 3* entnehmen.<sup>76</sup>

## Schatten

Mit der Eigenschaft `box-shadow` lässt sich eine Box mit einem Schatten versehen. Die Spezifikation sieht eine Reihe verschiedener Möglichkeiten vor, wir beschränken uns aber auf ein einfaches Beispiel:

```
1 section {
2     box-shadow: 3px 3px 5px #999;
3 }
```

Der erste Wert ist der horizontale Versatz des Schattens, der zweite der vertikale Versatz und der dritte der Unschärferadius. Als letztes wird die Farbe angegeben. Hier das Ergebnis:

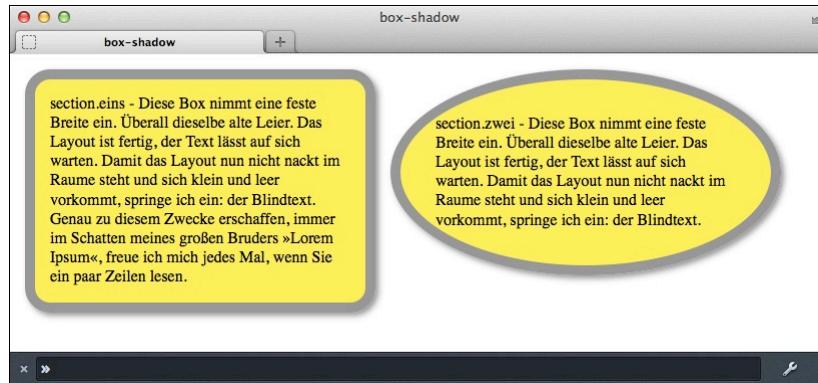


Abbildung 93: CSS3-Eigenschaft `box-shadow`

Auch Text kann mit einem Schatten ausgestattet werden. Dazu wird die Eigenschaft `text-shadow` verwendet. Zur Demonstration wurde das Beispiel etwas angepasst:

---

<sup>76</sup><https://www.w3.org/TR/css-backgrounds-3/>

```

1  section.eins {
2      text-shadow: 0px 1px 0px #fff;
3      background-color: #999;
4  }
5  section.zwei {
6      text-shadow: 2px 2px 4px #999;
7      background-color: #f6e;
8 }

```

Es versteht sich von selbst, dass Textschatten sparsam eingesetzt werden sollten, und ein geeigneter Kompromiss zwischen Lesbarkeit und Gestaltung zu suchen ist. Abbildung 94 zeigt das Ergebnis des Beispiels.

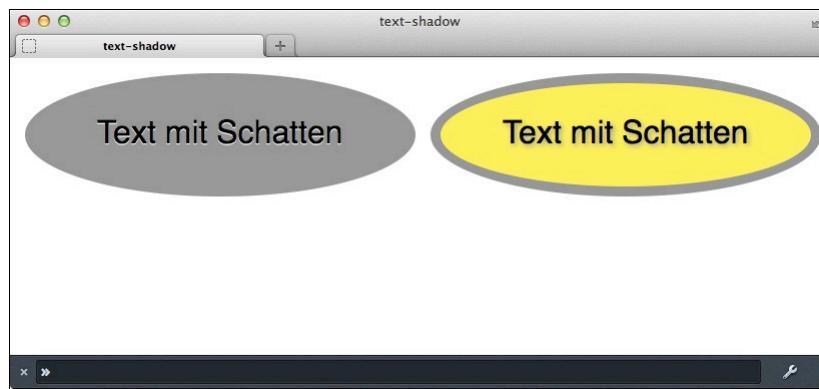


Abbildung 94: CSS3-Eigenschaft *text-shadow*

### Transparenz

Beim Thema Farbangaben wurde bereits darauf hingewiesen, dass CSS3 eine Farbangabe mit Transparenz erlaubt: `rgba(r,g,b,a)` gibt die drei Werte für rot, grün und blau sowie die Deckkraft (Alpha-Kanal, engl.: opacity) an. Die drei Farbwerte können ganze Zahlen zwischen 0 und 255 oder Prozentangaben sein. Der vierte Parameter ist eine Zahl zwischen 0 (durchsichtig) und 1 (undurchsichtig). Damit entspricht die Angabe `rgba(50,60,170,1)` dem Wert `rgb(50,60,170)`.

Ebenfalls neu in CSS3 ist die Eigenschaft `opacity`, mit der einem Element eine gewisse Transparenz gegeben werden kann. Als Wert wird eine Zahl angegeben zwischen 0 (durchsichtig) und 1 (undurchsichtig). Wenn die Eigenschaft nicht spezifiziert wird, gilt der Standardwert 1.

CSS3 stellt neben `rgba()` auch noch die Möglichkeit zur Verfügung, Farben als HSL-Wert zu definieren (hue-saturation-lightness). Ausserdem gibt es viele neue vordefinierte Farbnamen. Alle Details dazu können Sie im *CSS Color Module Level 3* nachlesen, das bereits seit 2011 als W3C Recommendation vorliegt.<sup>77</sup>

---

<sup>77</sup><https://www.w3.org/TR/css-color-3/>

## Farbverläufe

Statt einer Box eine Hintergrundfarbe zu geben, kann man sie auch mit einem Verlauf versehen. Solche Verläufe lassen sich überall hinzufügen, wo im CSS auch ein Bild referenziert werden kann:

- Anstelle von Hintergrundbildern (`background-image`)
- Für Listenpunkte (`list-style-image`)
- Bei Rahmenbildern (`border-image`)
- Bei generiertem Inhalt (`content`)

Der erste Fall wird bei weitem der häufigste sein. Hier ist ein Beispiel:

```
1 .beispiel1 {  
2     background-image: linear-gradient(135deg, yellow, blue);  
3 }
```

Dies ergibt einen linearen Verlauf (entlang einer Geraden), von gelb nach blau, die Richtung ist 135 Grad. Man kann es auch umgekehrt angeben: `linear-gradient(-45deg, blue, yellow)` führt zum gleichen Ergebnis (Abbildung 95).

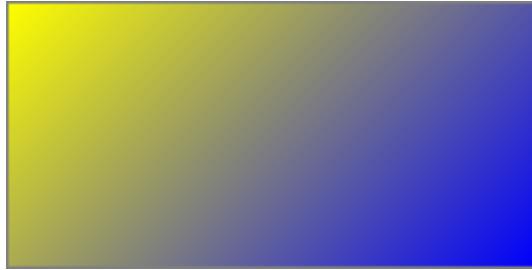


Abbildung 95: Linearer Verlauf (Quelle: W3C)

Neben linearen Gradienten gibt es auch radiale Gradienten (Abbildung 96).

```
1 .beispiel2 {  
2     background-image: radial-gradient(circle, yellow, green);  
3 }
```



Abbildung 96: Kreisförmiger Verlauf (Quelle: W3C)

Verläufe sind in der W3C Candidate Recommendation *CSS Images Module Level 3* beschrieben.<sup>78</sup>

## Transitionen und Transformationen

### Transitionen

Mit Hilfe von Pseudoklassen wie `:hover` kann ein Element seine Darstellung in bestimmten Situationen ändern, zum Beispiel wenn sich der Mauszeiger auf dem Element befindet. Hier ist ein Beispiel:

```
1 a { background: #9c3; }
2 a:hover { background: #690; }
```

Das `a`-Element wird zunächst mit grünem Hintergrund dargestellt. Wenn man mit dem Mauszeiger über das Element fährt, wird der Hintergrund zu einem dunkleren Grün. Die weiteren Eigenschaften des Elements sollen hier nicht interessieren.

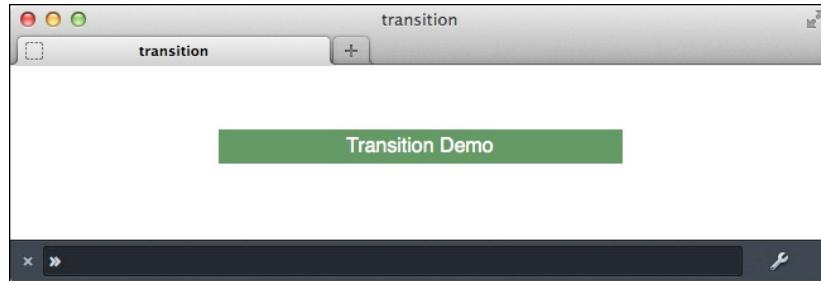


Abbildung 97: CSS3-Eigenschaft *transition*

Der Übergang von einem Zustand zu dem nächsten erfolgt abrupt: Sobald der Mauszeiger über das Element kommt, verändert sich die Hintergrundfarbe. Möchte man einen weicheren Wechsel, kann man CSS-Übergänge (Transitions) verwenden.

```
1 a {
2   background: #696;
3   transition-property: background;
4   transition-duration: 0.5s;
5   transition-timing-function: ease;
6 }
7
8 a:hover {
9   background: #363;
10 }
```

Der Hintergrund ändert sich nun langsam (über eine halbe Sekunde) zu dem dunkleren Grün, wenn der Mauszeiger auf das Element fährt. Beachten Sie, dass die `transition`-Eigenschaften beim Ausgangszustand vermerkt werden und nicht beim `:hover`-Zustand.

---

<sup>78</sup><https://www.w3.org/TR/css-images-3/>

Mit `transition-property` wird angegeben, welche Eigenschaft verändert wird. Angenommen das `a`-Element ändert nicht nur die Hintergrundfarbe sondern auch noch seine Grösse im Zustand `:hover`. Solange `transition-property:background` gesetzt ist, würde trotzdem nur der Hintergrund einen langsamen Übergang zeigen. Mit `transition-property:all` würde sich auch die Grösse langsam verändern.

Die Eigenschaft `transition-duration` gibt die Zeit für den Übergang an.

`transition-timing-function` schliesslich spezifiziert den zeitlichen Ablauf des Übergangs. Mögliche Werte sind unter anderem: `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out`. Man kann mit `cubic-bezier(...)` auch eine eigene Kurve für den zeitlichen Ablauf definieren. Mit `transition-delay` kann man den Übergang verzögern. Diese Eigenschaft wurde im Beispiel nicht verwendet und dürfte insgesamt auch seltener zum Einsatz kommen.

Die einzelnen Eigenschaften zur Definition von Übergängen können auch in *einer* Eigenschaft `transition` zusammengefasst werden:

```
1 a {  
2     background: #696;  
3     transition: background 0.5s ease;  
4 }
```

CSS-Transitions wurden ursprünglich vom Webkit-Team für den Safari entwickelt und sind nun ein W3C Working Draft mit der Bezeichnung *CSS Transitions*.<sup>79</sup>

## Transformationen

Auch die Spezifikation *CSS Transforms* ist aktuell ein W3C Working Draft.<sup>80</sup> CSS-Transformationen erlauben es, die Darstellung eines Elements in der Ebene auf verschiedene Weise zu transformieren. Eine solche Transformation ist das *Skalieren*:

```
1 ul.gallery li a:hover img {  
2     transform: scale(1.5);  
3 }
```

Das Beispiel zeigt, wie Bilder um 50% vergrössert angezeigt werden, wenn der Mauszeiger über ihnen ist. Selbstverständlich kann das in Verbindung mit einer Transition verwendet werden, um ein langsameres Vergrössern und Verkleinern zu erreichen:

```
1 ul.gallery li a img {  
2     float: left;  
3     width: 200px;  
4     transition: transform 0.2s ease-in-out;  
5 }
```

Die Skalierung kann auch auf eine Richtung beschränkt werden: `scaleX`, `scaleY`.

Wenn Sie beim Vergrössern noch einen Schatten hinzufügen (`box-shadow`), sieht es aus, als würde das Bild aus der Ebene herausgehoben. Zusätzlich kann mit `transform-origin` noch der Ursprung der Vergrösserung angegeben werden. `transform-origin: bottom`

---

<sup>79</sup><https://www.w3.org/TR/css-transitions-1/>

<sup>80</sup><https://www.w3.org/TR/css-transforms-1/>

**left** heisst: Die linke untere Ecke ist fest, die Vergrösserung erfolgt nach rechts oben. Eine weitere mögliche Transformation ist das *Rotieren*. Mit `rotate(-10deg)` wird das Element um 10 Grad gegen den Uhrzeigersinn gedreht. Auch eine *Neigung/Verzerrung* ist möglich: Mit `skew()` kann ein Objekt bezüglich der x- und y-Achse geneigt werden:

```
1 div.skew {
2     transform:skew(30deg,10deg);
3 }
```

Der erste Wert gibt an, um wie viel Grad das Objekt um die x-Achse geneigt wird, der zweite Wert bezieht sich auf die y-Achse. Beide können auch einzeln angegeben werden: `skewX`, `skewY`.

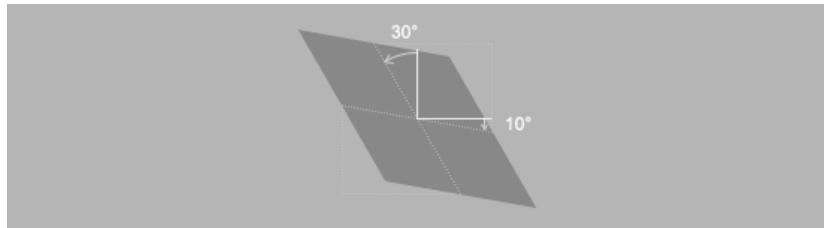


Abbildung 98: CSS transform: skew (Quelle: css-lernen.net)

Mit `translate` kann ein Objekt in x- und y-Richtung verschoben werden. Zum Beispiel verschiebt `transform: translate(-7px, 27px)` das betreffende Objekt um 7px nach links und um 27px nach unten.

Ausser den hier gezeigten gibt es zahlreiche weitere Transformationen, die Sie bei Bedarf in der Spezifikation finden. Mit Transformationen lassen sich interessante Effekte erzielen, wie der Würfel in Abbildung 99, der interaktiv gedreht werden kann.<sup>81</sup> Auf den Seiten steht Text, der auch selektiert und kopiert werden kann.

## Animationen

Animationen sind im W3C Working Draft *CSS Animations* beschrieben.<sup>82</sup>

Die Möglichkeiten von Animationen sind vielfältig. Hier soll nur ein einfaches Beispiel angegeben werden (Quelle: *CSS3 for Web Designers*). Es soll erreicht werden, dass in einem Formular das Eingabefeld, welches gerade den Fokus hat, mit einem pulsierenden Schatten umgeben ist. Zunächst wird eine Keyframes-Deklaration benötigt, in welcher die einzelnen Zustände festgelegt werden:

```
1 @keyframes pulse {
2     0% { box-shadow: 0 0 12px rgba(51, 204, 255, 0.2); }
3     50% { box-shadow: 0 0 12px rgba(51, 204, 255, 0.9); }
4     100% { box-shadow: 0 0 12px rgba(51, 204, 255, 0.2); }
5 }
```

---

<sup>81</sup> Quelle: <https://paulrhayes.com/experiments/cube-3d/>

<sup>82</sup> <https://www.w3.org/TR/css-animations-1/>

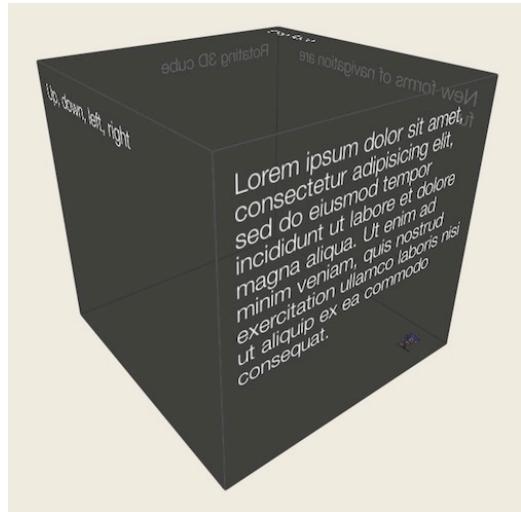


Abbildung 99: CSS transform Demo

Anschliessend wird diese Deklaration zur Beschreibung der Animation verwendet:

```

1 input[type="text"] :focus {
2   animation-name: pulse;
3   animation-duration: 1.5s;
4   animation-iteration-count: infinite;
5   animation-timing-function: ease-in-out;
6 }
```

Wenn das Eingabefeld den Fokus erhält, wird ein pulsierender Schatten um das Feld angezeigt (Abbildung 100).

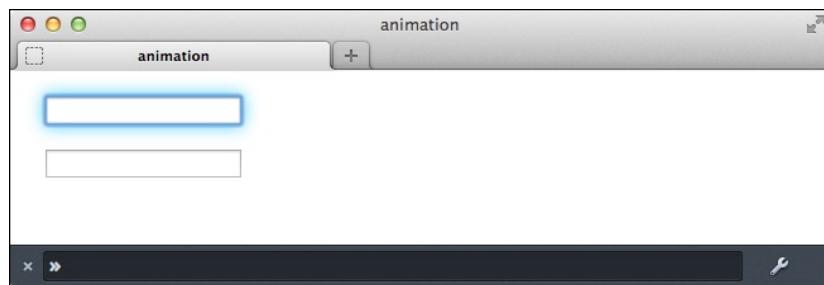


Abbildung 100: CSS3 animation

### Eine Warnung

Übergänge und Animationen können eine Webseite lebhafter machen. Sie können aber auch sehr schnell lästig werden, wenn sie im Übermass oder an unpassender Stelle ange-

wendet werden. Stellen Sie sich also immer, bevor Sie einen Übergang oder eine Animation anwenden wollen, die Frage:

### **Welchen Nutzen hat der Effekt für die Besucher der Website?**

Wenn Sie diese Frage nicht eindeutig beantworten können, sollten Sie auf den Effekt verzichten.

## **CSS-Variablen**

Mit den bisher gezeigten CSS-Möglichkeiten lässt sich das Farbschema einer Webseite nicht an einer Stelle im Stylesheet zentral definieren. Es ist über verschiedene Regeln mehr oder weniger über das ganze Stylesheet verteilt. Zudem kommen die gleichen Farben immer wieder vor. Es wäre gut, man könnte die wichtigsten Farben in ein paar Variablen definieren und dann an verschiedenen Stellen im Stylesheet einsetzen.

Zu diesem Zweck wurden lange Zeit so genannte CSS-Präprozessoren wie LESS oder Sass eingesetzt (mehr zu den beiden Tools weiter unten). Sie erlauben es unter anderem, Variablen zu definieren, diese mit Werten zu belegen und an anderer Stelle im Stylesheet wieder einzusetzen.

Mittlerweile können Variablen auch ohne CSS-Präprozessoren eingesetzt werden. Das ist in der Spezifikation *CSS Custom Properties for Cascading Variables Module Level 1* beschrieben.<sup>83</sup> Hier ist ein Beispiel:

```
1 :root {  
2   --main-fg-color: darkblue;  
3   --main-bg-color: #ffffacd;  
4 }  
5  
6 footer {  
7   color: var(--main-fg-color);  
8   background-color: var(--main-bg-color, white);  
9 }
```

Variablen beginnen mit -- und können mit var() wieder referenziert werden. Die Pseudoklasse :root steht für die Wurzel des HTML-Dokuments, die definierten Variablen sind also im ganzen Dokument nutzbar. Bei Bedarf kann die Gültigkeit aber auch auf eine Teilstruktur eingeschränkt werden. Beim var-Aufruf können auch mehrere Variablen sowie ein Defaultwert angegeben werden für den Fall, dass die davor angegebene Variable nicht definiert ist.

## **Drucken von Dokumenten**

Ausgabemedien, welche auf Einzelseiten basieren, stellen andere Anforderungen an die Gestaltung. Da Webseiten auch ausgedruckt werden, sollte CSS auch dafür Unterstützung bieten. Dazu gehören Dinge wie das Erzeugen von Inhalten wie Fussnoten, internen

---

<sup>83</sup><https://www.w3.org/TR/css-variables/>

Verweisen (s. Abschnitt x auf Seite y), oder Kopfzeilen aus Überschriften. Näheres können Sie in der Spezifikation *CSS Generated Content for Paged Media Module* nachlesen.<sup>84</sup> Wenn Sie diese Möglichkeiten nutzen möchten, ist es wichtig, vorgängig die Browserunterstützung zu klären.

## CSS-Dateien, Werkzeuge

CSS-Dateien können recht schnell unübersichtlich werden. Man sollte sich daher überlegen, wie man Stylesheets am besten organisiert und welche Werkzeuge man für das Bearbeiten von CSS-Dateien und für die Fehlersuche mit Vorteil verwendet.

### CSS-Dateiaufbau

Häufig werden CSS-Regeln wie in diesem Beispiel formatiert:

```
1  /**
2   * Gestaltung der Hauptnavigation
3   */
4
5  ol#hauptnavi {
6      position: absolute;
7      top: 100px; left: -100px;
8  }
9
10 ol#hauptnavi li {
11     width: 220px;
12     height: 4em;
13     position: relative;
14 }
15
16 ol#hauptnavi li a {
17     display: block;
18     position: absolute;
19     bottom: 0; right: 0;
20     padding: 10px 0;
21     font-size: 1.3em;
22 }
23
24 ol#hauptnavi li.impressum {
25     height: 12em;
26 }
```

Also: Selektor auf einer Zeile, die Deklarationen einzeln auf weiteren Zeilen. Das ist offenbar ein guter Kompromiss zwischen Kompaktheit und Lesbarkeit. Kurze, inhaltlich zusammen gehörende Deklarationen wie `top: 100px; left: -100px;` in der ersten Regel kann man auch auf einer Zeile schreiben.

---

<sup>84</sup><https://www.w3.org/TR/css-gcpm-3/>

Bei mehreren zusammen gehörenden Regeln mit nur einer Deklaration kann auch folgende Darstellung sinnvoll sein:

```
1 #subnavi a { background-color: #669; }
2 #subnavi a:hover { background-color: #6a9; }
3 #subnavi a:active { background-color: #6c9; }
```

Auf diese Weise wird weniger Platz verbraucht und die Lesbarkeit bleibt trotzdem erhalten.

CSS-Eigenschaften können mehrere, durch Kommas getrennte Listen von Werten haben. Auch diese sollten gut lesbar geschrieben werden, wobei man am besten freizügig mit Leer- oder Tabulatorzeichen umgeht. Hier ein paar Beispiele:<sup>85</sup>

```
1 .example-1 {
2     background: url(images/example.png) center center no-repeat,
3                 url(images/example-2.png) top left repeat,
4                 url(images/example-3.png) top right no-repeat;
5 }
6
7 .example-2 {
8     transform: scale(.8)
9             skew(20deg, 30deg)
10            translateZ(0);
11 }
```

So viel zur Formatierung der CSS-Regeln. Um den Überblick über ein umfangreiches Stylesheet zu behalten, ist aber auch eine sinnvolle Grobstruktur des Stylesheets erforderlich. Bewährt hat sich eine Aufteilung nach Bereichen auf der Seite:

- Allgemeine Angaben, Schriftarten, Größen, usw.
- Haupt- und Unternavigationen
- Inhaltsbereich
- Seitenleiste

Natürlich kann die Aufteilung auch anders oder weiter untergliedert sein. Das wichtigste Ziel ist, sich in einem Stylesheet schnell zurechtzufinden, auch wenn man es selber geschrieben aber mehrere Monate nicht angesehen hat. Dazu gehören selbstverständlich auch Kommentare über den einzelnen Rubriken und bei Regeln/Selektoren/Deklarationen, die mit Kommentar besser verständlich sind.

## CSS-Werkzeuge

Webdesigner, die viel CSS-Code schreiben, sind mit einem guten Code-Editor in der Regel am produktivsten. Natürlich ist es ein Vorteil, wenn der Editor CSS speziell unterstützt und Hilfsmittel wie Syntax-Coloring, Vervollständigung von Eigenschaften und Werten, oder einen Validator zur Verfügung stellt. Der bereits beim Thema Markup erwähnte

---

<sup>85</sup>Quelle: *Using White Space For Readability In HTML And CSS*,  
<https://www.smashingmagazine.com/2013/02/using-white-space-for-readability-in-html-and-css/>

*Visual Studio Code* kann gut mit CSS umgehen, es gibt aber noch zahlreiche andere für CSS geeignete Editoren.

Für eine Reihe von CSS-Eigenschaften – wie abgerundete Ecken, Schatten, Farbverläufe, Flexbox-Layouts – gibt es Online-Tools, um mit diesen Möglichkeiten zu experimentieren. Abbildung 101 zeigt ein Beispiel zum Anlegen von Farbverläufen.<sup>86</sup> Da immer wieder neue solche Online-Werkzeuge entstehen und diese oft nur für eine bestimmte Zeit existieren, ist es nicht sinnvoll, hier eine Liste solcher Web-Tools anzugeben. Sie sind aber leicht über eine Web-Suche zu finden.

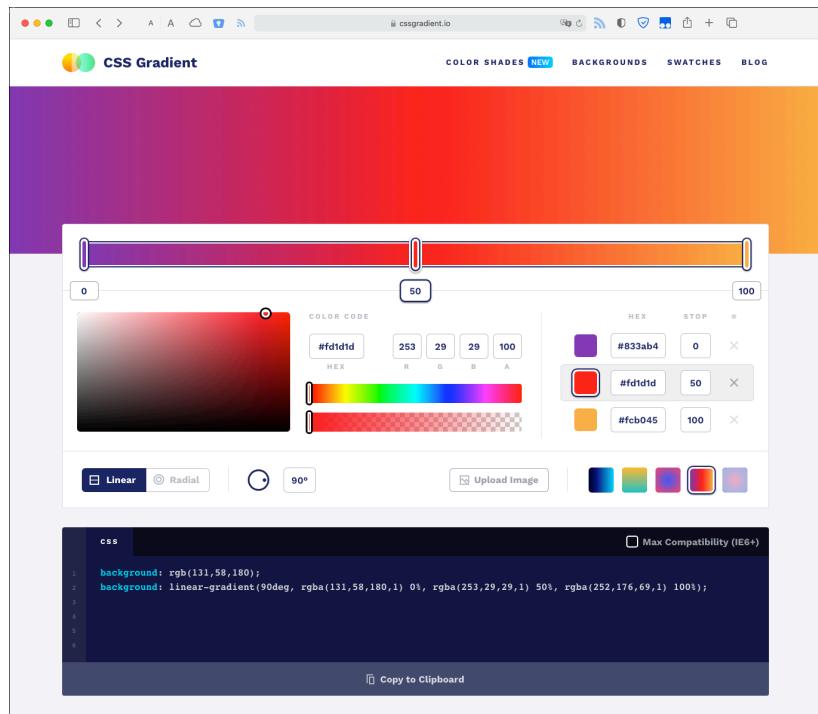


Abbildung 101: CSS Gradient zum Experimentieren

## CSS optimieren

Die Leerzeichen und Zeilenwechsel im CSS-Code sind eigentlich nur nötig, um das Stylesheet besser lesbar zu machen. Für die Verarbeitung im Browser sind sie unnötig. Sie vergrössern nur die Dateien und damit auch die Ladezeiten. Daher gibt es Werkzeuge, um CSS-Code zu optimieren, das heisst unnötige Zeichen und Kommentare zu entfernen oder auch mehrere CSS-Dateien zusammenzufassen (weniger HTTP-Requests zum Server nötig). Manche dieser Werkzeuge arbeiten nicht nur auf syntaktischer Ebene sondern analysieren auch den Inhalt eines Stylesheets. So können sie redundante Deklarationen erkennen und entfernen oder auch mehrere Eigenschaften in den kombinierten Varianten

<sup>86</sup><https://cssgradient.io>

zusammenfassen, also etwa `border-width`, `border-color` und `border-style` in einer Eigenschaft `border`.

Hier ein Ausschnitt eines solchen optimierten Stylesheets:

```
body{font-family:Arial, Verdana, Tahoma, sans-serif;font-size:62.5%;background-color:#eee}#seite{width:920px;margin: 60px auto 20px;position:relative;padding-top:160px;background-image:url(imgs/ logo.png);background-position:700px 0;background-repeat:no-repeat}
```

Das Problem bei der Arbeit mit optimierten CSS-Dateien ist, dass man es dann mit zwei Varianten des Stylesheets zu tun hat: Der Arbeitsversion und der optimierten Version. Man kann wohl aus der optimierten Version wieder eine schön formatierte CSS-Datei erzeugen, aber die sorgfältig erstellten Kommentare sind dann weg. Es braucht also eine Arbeitsweise, in der bei der Entwicklungsarbeit die originalen CSS-Dateien verwendet werden und beim Deployment auf dem Server die optimierten Versionen erstellt werden (am besten automatisch).

## Validieren, Fehlersuche

Beim Entwickeln und Bearbeiten von Stylesheets sollten Sie diese regelmässig mit dem CSS-Validator überprüfen. So erkennen Sie Fehler früh und erhalten auch nützliche Hinweise, wo möglicherweise Probleme auftreten können.

Zur Fehlersuche gut geeignet sind die Entwicklertools in den Browsern, die bereits mehrfach erwähnt wurden.

## LESS und Sass

LESS<sup>87</sup> und Sass<sup>88</sup> wurden im Abschnitt über die CSS-Variablen bereits erwähnt. Neben der Möglichkeit, Variablen zu definieren und zu referenzieren, unterstützen LESS und Sass noch eine Reihe weiterer Funktionen. LESS und Sass werden als Meta-Sprachen zu CSS bezeichnet. Es sind CSS-Erweiterungen, welche zu CSS kompiliert werden, da die Browser LESS und Sass nicht direkt verarbeiten können. Es gibt zwar für beide Sprachen JavaScript-Übersetzer, die eine Übersetzung “on the fly” im Browser ermöglichen. Deren Einsatz ist aber eher während der Entwicklungsphase eines Webangebots sinnvoll.

Sass steht übrigens für *Syntactically Awesome Stylesheets*. Sass unterstützt:

- Variablen
- Mathematische Operationen wie +, -, \*, / oder %
- Funktionen zum Beispiel zum Anpassen von Farbwerten
- Schleifen
- Fallunterscheidungen
- Mixins, eine Art Makros, welche mit Argumenten aufgerufen werden
- Verschachteln von Regeln
- Vererbung

---

<sup>87</sup><https://lesscss.org/>

<sup>88</sup><https://sass-lang.com/>

Von Sass gibt es zwei Syntax-Varianten:

- SCSS (Sassy CSS) verwendet wie LESS die CSS-Syntax, das heisst es ist eine Erweiterung von CSS. Jede gültige CSS-Datei ist somit auch eine gültige SCSS-Datei. Diese Variante schreibt man mit der `.scss`-Erweiterung.
- Die eigentliche Sass-Syntax verwendet eine von CSS abweichende Notation: Statt der geschweiften Klammern werden die Selektoren und Eigenschaften auf eine bestimmte Weise eingerückt. Auch das Semikolon am Ende der Deklarationen entfällt. Diese Variante schreibt man mit der `.sass`-Erweiterung.

Beispiel zu SCSS:

```
1 #main {  
2   color: blue;  
3   font-size: 0.3em;  
4  
5   a {  
6     font: {  
7       weight: bold;  
8       family: serif;  
9     }  
10    &:hover {  
11      background-color: #eee;  
12    }  
13  }  
14 }
```

# Quellen und Verweise

## Material zur Vertiefung

- **CSS Image Sprites** (W3Schools)  
[http://www.w3schools.com/css/css\\_image\\_sprites.asp](http://www.w3schools.com/css/css_image_sprites.asp)
- **Using White Space For Readability In HTML And CSS**  
<http://coding.smashingmagazine.com/2013/02/19/using-white-space-for-readability-in-html-and-css/>

## Standards

- **CSS Specifications** (Überblick)  
<http://www.w3.org/Style/CSS/current-work>

## Quellenangaben

- **CSS3 For Web Designers** (Dan Cederholm, A Book Apart, 2010)  
<http://www.abookapart.com/products/css3-for-web-designers>

Ein Teil der Bilder und Inhalte stammt aus früheren Folien der Fächer WBD (Webdesign), INSY (Informationssysteme) und WEBT (Webtechnologien).

# Anhang: Internet-Protokoll-Stack

Wir haben uns bereits mit dem HTTP-Protokoll beschäftigt. Tatsächlich setzt HTTP auf einer Reihe anderer Protokolle auf, um die Übertragung der HTTP-Requests über das Internet zu gewährleisten. Die verschiedenen Protokolle und Schichtenmodelle werden in anderen Kursen des Informatikstudiums noch ausführlich behandelt. Auch als Web-Entwickler wird man regelmäßig mit Fragen dieser Art konfrontiert:

- Warum ist Host xyz nicht erreichbar?
- Welche Ports müssen in der Firewall für Protokoll xyz geöffnet werden?
- Wie lautet die IP-Adresse des Hosts xyz?

Daher sollte man als Web-Entwickler auch die Grundlagen der verwendeten Protokolle kennen. Hier also das Wichtigste in Kürze.

## Ziele

- *Sie kennen die wichtigsten im Internet verwendeten Protokolle (HTTP, FTP, DNS, TCP, UDP, IP), und deren Einordnung im Internet-Protokoll-Stack.*
- *Sie wissen, wie das Domain Name System funktioniert und können DNS-Abfragen auch auf der Kommandozeile durchführen.*

## Relevanz für WBE

Die in diesem Anhang behandelten Themen gehören zwar zum Grundwissen für Web-Entwickler, sind aber nicht zwingend Voraussetzung für WBE.

## Schichtenmodell

HTTP geht davon aus, dass Textdaten korrekt von einem Computer zu einem anderen übertragen werden können. Die Frage ist nun, wie diese Übertragung tatsächlich vorstehen geht.

Im Gegensatz zur *Leitungsvermittlung* (früher in Telefonnetzen verwendet) werden die zu übertragenden Daten im Internet in Pakete aufgeteilt, die getrennt voneinander auf den Weg geschickt werden: *Paketvermittlung*. Am Ziel werden die Pakete dann wieder zur ursprünglichen Nachricht zusammengesetzt. Eine zwischen zwei Endpunkten aufgebaute Verbindung ist damit virtuell, es wird nicht permanent eine Leitung dafür belegt (Abbildung 102).

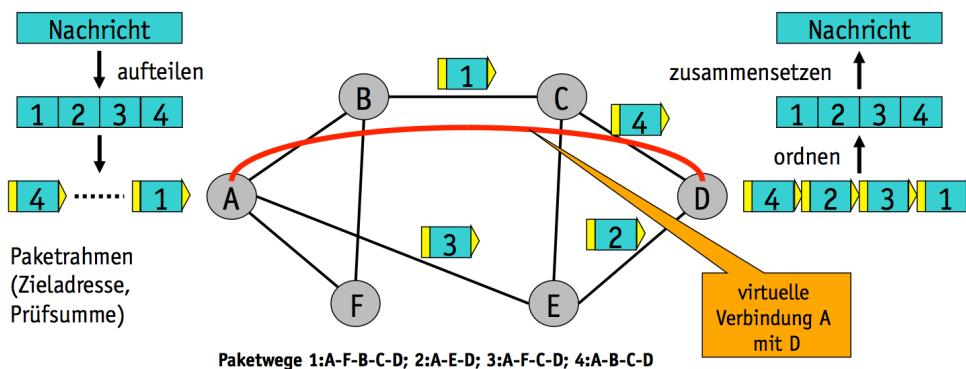


Abbildung 102: Paketvermittlung

Es ist leicht einzusehen, dass es nicht sinnvoll wäre, wenn jedes Programm mit Internet-Kommunikation die Datenübertragung bis ins Detail selbst implementieren würde. Wie in der Software-Entwicklung allgemein üblich wird auch bei der Netzwerkkommunikation die Komplexität reduziert, indem Abstraktionsstufen eingeführt werden. Dies führt zu einem Schichtenmodell. Ein Beispiel soll dies veranschaulichen:

Zwei Philosophen möchten miteinander kommunizieren. Sie sprechen aber erstens nicht dieselbe Sprache und zweitens sind sie nicht in der Lage oder willens, ein Kommunikationsmittel zu bedienen. Die Kommunikation erfolgt also über mehrere Stufen. Die Abbildung 103 zeigt den Ablauf (Quelle: Tanenbaum/Wetherall 2011).

- **Horizontale Kommunikation:** Zwischen den Philosophen besteht eine virtuelle Verbindung, auch wenn sie nicht direkt miteinander kommunizieren. Die tatsächliche Kommunikation erfolgt nur zwischen den Sekretären.
- **Vertikale Kommunikation:** Philosoph spricht mit Dolmetscher, Dolmetscher spricht mit Sekretär, Sekretär „spricht“ mit Faxmaschine.
- **Protokolle der horizontalen Kommunikation:** Sekretäre vereinbaren Medium der Nachrichtenübermittlung, Dolmetscher vereinbaren gemeinsame Sprache, Philosophen vereinbaren gemeinsame Inhalte.

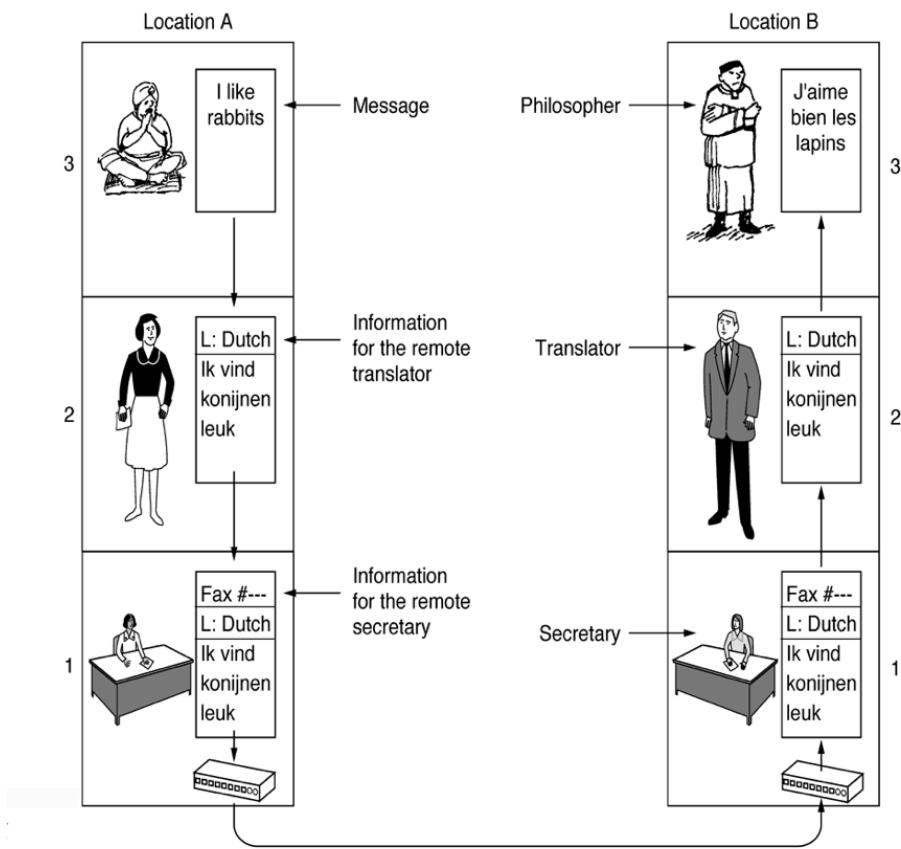


Abbildung 103: Kommunizierende Philosophen

**Hinweis zum Beispiel:** Auf allen Ebenen können die Beteiligten natürlich männlich und weiblich sein; die Bezeichnung *Philosoph* oder *Sekretär* wurden gewählt, um unleserliche Konstrukte wie *PhilosophInnen* oder *Philosophierende* zu vermeiden.

Computernetze funktionieren nach dem gleichen Prinzip:

- **Schichten (Layer):** Die horizontale Kommunikation erfolgt mittels Protokollen, die vertikale Kommunikation über Schnittstellen.
- **Schnittstellen (Interfaces):** Definition der Operationen, welche von der nächst höheren Schicht aufgerufen werden können.
- **Dienste (Services):** Umsetzung der Operationen, welche von der nächst höheren Schicht aufgerufen werden können.
- **Protokoll (Protocol):** Definieren die Regeln, nach denen kommuniziert wird, sowie Form und Art der ausgetauschten Nachrichten.

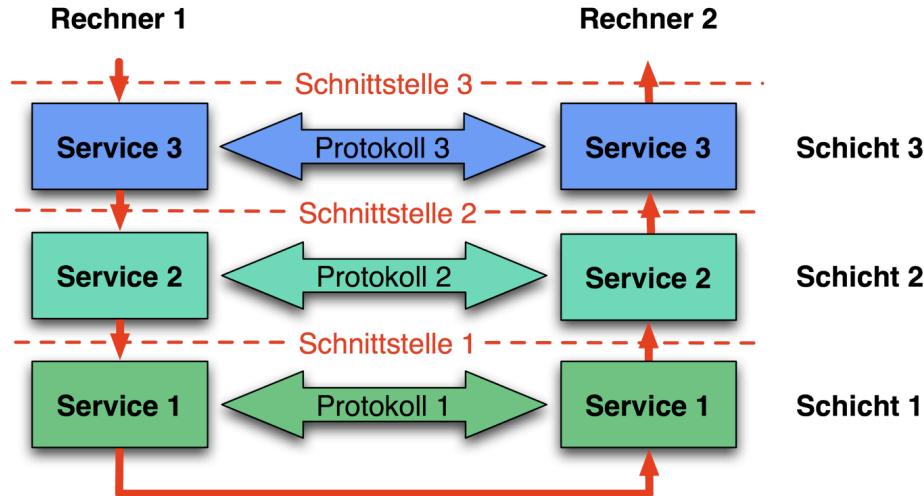


Abbildung 104: Schichtenmodell

Von der Organisation ISO wurde zu diesem Zweck das **ISO/OSI Referenzmodell** für Netzwerkprotokolle standardisiert. Dieses in den 1970er-Jahren entwickelte Modell basiert auf sieben Schichten. Für das Internet wird eine Aufteilung in vier Schichten verwendet, das **TCP/IP-Referenzmodell** oder auch **IETF-Modell** (IETF: Internet Engineering Task Force):

#### Anwendungsschicht (Application Layer):

Protokolle, die mit den Anwendungsprogrammen zusammenarbeiten. Dazu gehört das oben bereits behandelte Protokoll **HTTP**, aber auch **FTP** für die Dateiübertragung oder **SMTP** für die Übertragung von E-Mails.

ISO/OSI	Internet	Protokolle	Typische Angaben
Application (Anwendung)		HTTP FTP SMTP Telnet	URL: <a href="http://www.zhwin.ch">http://www.zhwin.ch</a> Mailadresse: mustepet@zhwin.ch
Presentation (Darstellung)	Application		
Session (Sitzung)		TCP UDP	Portnummer 80 = HTTP 25 = SMTP
Transport	Transport		
Network (Netzwerk)	Internet	IP	IP-Adresse 192.168.0.1
Data Link (Sicherung)		Ethernet Wireless LAN Token Ring	MAC-Adresse 00:0F:7F:23:45:67
Physical (Bitübertragung)	Physical / Access	PPP/(Modem, ISDN, xDSL)	Telefonnummer: 0878/123456

Abbildung 105: Internet-Protokoll-Stack

#### Transportschicht (Transport Layer):

Stellt eine Ende-zu-Ende-Verbindung her. Das wichtigste Protokoll dieser Schicht ist das *Transmission Control Protocol (TCP)*, das Verbindungen zwischen jeweils zwei Netzwerkteilnehmern zum zuverlässigen Versenden von Datenströmen herstellt. Auch das *User Datagram Protocol (UDP)* gehört zu dieser Schicht. Die Portnummer wird verwendet, um bestimmte Anwendungen auf den beteiligten Computern zu adressieren.

#### Internetschicht (Internet Layer):

Diese Schicht ist für die Weitervermittlung von Paketen und die Wegewahl (Routing) zuständig. Kern Verwendet wird das *Internet Protocol (IP)* in der Version 4 oder 6, das einen Paketauslieferungsdienst bereitstellt. Entspricht der Vermittlungsschicht des ISO/OSI-Referenzmodells (Auch Netzwerkschicht genannt).

#### Netzzugangsschicht (Link Layer):

Auch: Physical Layer, Access Layer. Diese Schicht dient als Platzhalter für verschiedene Techniken zur Datenübertragung. Zum Beispiel: Ethernet, FDDI, PPP (Punkt-zu-Punktvorbindung) oder 802.11 (WLAN).

## Ablauf und Routing

Wenn nun ein HTTP-Request vom Browser zu einem Webserver geschickt werden soll, wird dieses der Transportschicht übergeben und mit einem TCP-Header versehen (der unter anderem die Portnummern von Sender und Empfänger enthält). Von dort geht es weiter zur Internet-Schicht, wo die Fragmentierung in Pakete und das Hinzufügen des IP-Headers geschieht. Der IP-Header enthält zu diesem Zweck die Ausgangs- und Ziel-IP-Adressen. Die eigentliche Übertragung erfolgt dann auf dem jeweiligen Medium. Beim Empfänger werden die Pakete in umgekehrter Reihenfolge durch die Schichten übergeben und wieder ausgepackt, so dass nur die verschickten Daten beim Empfänger ankommen (Abbildung 106).

Das Vermitteln der Pakete vom Sender bis zum Empfänger (*Routing*) erfolgt auf der Internet-Schicht. Bei Bedarf können die Pakete dabei auf einem beteiligten Router weiter

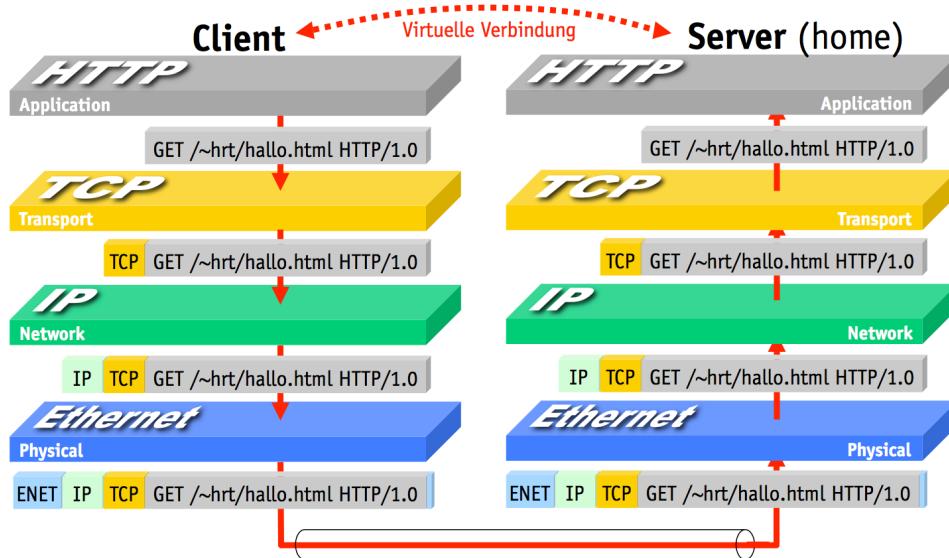


Abbildung 106: HTTP-Request im Schichtenmodell

fragmentiert werden (IPv4). Der IP-Header enthält einen Eintrag TTL (time-to-live), der zunächst auf einen Standardwert gesetzt und dann bei jedem Hop (Schritt von Router zu Router) um eins dekrementiert wird. Wenn TTL den Wert 0 erreicht, wird das Paket als unzustellbar verworfen (Abbildung 107).

Ein **Router** sorgt dafür, dass bei ihm eintreffende Daten zum vorgesehenen Zielnetz weitergeleitet werden. Die Vermittlung geschieht auf der Internet-Ebene (IP-Protokoll). Er hat mindestens eine Schnittstelle und IP-Adresse pro angeschlossenem Netzwerk. Die Entscheidung, wie eintreffende Pakete weitergesendet werden, verwendet der Router Routing-Tabellen. Ist der Weg ins Zielnetz nicht bekannt, wird das Paket an ein Default-Gateway übergeben.

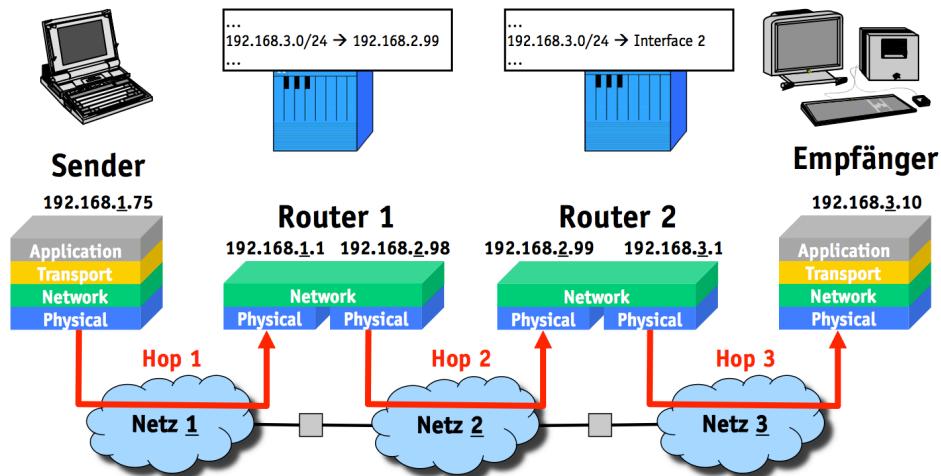


Abbildung 107: Routing

## Unix-Befehle

Mit ping kann festgestellt werden, ob ein bestimmter Host in einem IP-Netzwerk erreichbar ist. Der Befehl traceroute liefert Informationen, über welche IP-Router Datenpakete bis zum abgefragten Ziel vermittelt werden.

```
$ ping 173.194.35.63
PING 173.194.35.63 (173.194.35.63): 56 data bytes
64 bytes from 173.194.35.63: icmp_seq=0 ttl=53 time=20.946 ms
64 bytes from 173.194.35.63: icmp_seq=1 ttl=53 time=19.843 ms
64 bytes from 173.194.35.63: icmp_seq=2 ttl=53 time=21.113 ms
^C
--- 173.194.35.63 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 19.843/20.634/21.113/0.563 ms

$ traceroute 173.194.35.63
traceroute to 173.194.35.63 (173.194.35.63), 64 hops max, 52 byte packets
1  10.0.1.1 (10.0.1.1)  0.445 ms  0.302 ms  0.175 ms
2  * * *
3  217-168-59-34.static.cablecom.ch (217.168.59.45)  11.114 ms  9.353 ms  17.335 ms
4  84.116.211.28 (84.116.211.28)  12.825 ms  11.467 ms  10.464 ms
5  84.116.202.237 (84.116.202.237)  10.774 ms  10.862 ms  17.200 ms
6  ch-zrh01b-ra1-ae-9-0.aorta.net (84.116.134.22)  10.936 ms  12.151 ms  19.866 ms
7  74.125.49.101 (74.125.49.101)  11.653 ms  11.815 ms  11.765 ms
8  72.14.233.44 (72.14.233.44)  24.047 ms  23.333 ms  27.664 ms
9  209.85.241.67 (209.85.241.67)  21.143 ms  21.038 ms  19.950 ms
10 mil01s17-in-f31.1e100.net (173.194.35.63)  19.894 ms  19.923 ms  19.544 ms
```

## DNS

Das Domain Name System (DNS) dient zur Namensauflösung im Internet. Es ist verteilter hierarchischer Verzeichnisdienst, der die Abbildung von Domännamen auf IP-Adressen verwaltet.

Ein Beispiel zum Ablauf: Angenommen der Herr Muster an der ZHAW möchte auf die Seite <http://www.heise.de/newsticker/> zugreifen. Dazu muss zunächst die IP-Adresse von [www.heise.de](http://www.heise.de) herausgefunden werden. Das geschieht im Prinzip so:

- Zuerst wird der DNS-Server der ZHAW kontaktiert, dessen IP-Adresse bekannt ist (zum Beispiel via DHCP beim Anmelden ans Netzwerk).
- Dieser kontaktiert den Root-DNS-Server, um die IP-Adresse für den .de-DNS-Server zu erfragen.
- Der .de-DNS-Server liefert die Adresse des heise.de-DNS-Servers.
- Der heise.de-DNS-Server liefert schliesslich die Adresse für [www.heise.de](http://www.heise.de).

Um nicht jedes Mal die ganze Kette von Abfragen machen zu müssen, werden die Resultate an den verschiedenen Stellen für eine gewisse Zeit zwischengespeichert (*Cache*).

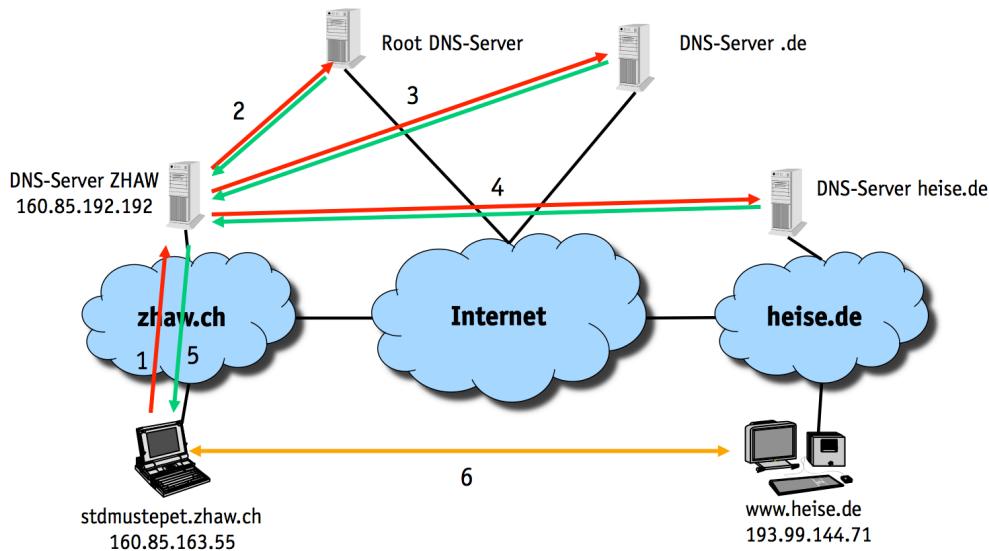


Abbildung 108: DNS-Ablauf

In einem Unix-Terminal können die Kommandos `host` oder `dig` für einen DNS-Lookup verwendet werden:

```
$ host www.zhaw.ch
www.zhaw.ch is an alias for web.zhaw.ch.
web.zhaw.ch has address 160.85.104.111
```

```
$ dig www.zhaw.ch

; <>> DiG 9.8.3-P1 <>> www.zhaw.ch
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49210
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.zhaw.ch.           IN  A

;; ANSWER SECTION:
www.zhaw.ch.      52539   IN  CNAME  web.zhaw.ch.
web.zhaw.ch.      81367   IN  A    160.85.104.111

;; Query time: 3 msec
;; SERVER: 10.0.1.1#53(10.0.1.1)
;; WHEN: Mon Jul 15 18:51:27 2013
;; MSG SIZE  rcvd: 63
```

## Quellen und Verweise

### Material zur Vertiefung

Zum Internet-Protokollstack und den einzelnen Protokollen enthält die Wikipedia ausführliche und gute Artikel. Diese eignen sich zur Vertiefung der behandelten Themen oder zum Beantworten offener Fragen.

- **Wikipedia: Internet protocol suite**  
[http://en.wikipedia.org/wiki/Internet\\_protocol\\_suite](http://en.wikipedia.org/wiki/Internet_protocol_suite)
- **Wikipedia: Internetprotokollfamilie**  
<http://de.wikipedia.org/wiki/Internetprotokollfamilie>
- **Die TCP/IP Protokollfamilie**  
<http://www.linux-praxis.de/linux2/tcpip.html>
- **Movie: Warriors of the net**  
<http://www.warriorsofthe.net/movie.html>
- **DNS: The Good Parts**  
<http://www.petkeen.net/dns-the-good-parts>

### Quellenangaben

- Andrew Tanenbaum and David Wetherall: **Computer Networks**, Fifth Edition, Prentice Hall, 2011.