Klaus Westermann

<Webentwicklung />

Das Skript zur Vorlesung

Stand: Mai 2018

Inhaltsverzeichnis

Informationen zur Veranstaltung 10
Inhalte: Die wichtigsten Auszeichnungssprachen 10
Ablauf: Vorlesung und Praktikum 10
Sprachkenntnisse 10
Mitarbeit und Leistungsnachweis 10
Workload und Prüfungsanmeldung 10
Qualifikationsziele 10
Literatur 11
Print 11
Internetquellen 11
W3C, Empfehlungen und Dokumente 11
W3C Wiki 11
Web-Tools 11
Eine weniger nützliche Seite 11
Sammlung von Websites 11
Ranking international 12
Tests und Vergleiche 12
Daten, Dokumentstruktur und Erscheinungsbild 13
Daten 13
Bezeichner und Werte 13
Datensätze 13
Dokumente 13
Aussehen und Erscheinungsbild 13
Technologien für Daten, Dokumentstruktur, Erscheinungsbild 15
Daten (XML) 15
Dokument (HTML) 16
Aussehen (CSS) 17
Browser-Stylesheet und Benutzer-Stylesheet 17
Weitere Auszeichnungssprachen für Daten, Dokumente, Erscheinungsbild 17
APT 17
BBCode 17
Wikitext, Wiki-Code, Wiki-Syntax 17
JASON 18
LaTeX 18
Auszeichnungssprachen „erfinden“ 18
Warum heißen Auszeichnungssprachen „Auszeichnungssprachen“? 18
„Auszeichnung“ im Print-Medium 18
SGML, „Urmutter“ aller Auszeichnungssprachen 19
Auszeichnungssprachen und Programmiersprachen 20
Datenverarbeitung 20
Datenbeschreibung 20
Die Arbeitsweise des Internets 20
Client und Server 20
Protokolle 20
Serverseitige Sprachen 20
Clientseitige Sprachen 21
Die Arbeitsweise des Browsers 21
HTML-Parser 21
Rendering-Engine 21

Fehlertoleranz von Browsern 21
Validatoren 22
Werkzeuge und Hilfsmittel 22
Kommerzielle Editoren und Freeware 22
Open-Source-Editoren 22
Validatoren 23
Kompatibilität 23
Sammlung von Werkzeugen für Webentwickler 23
Browser und Browser-Add-Ons für die Entwicklungsumgebung 23
Nachschlagen im Internet 23
Die Arbeitsumgebung 24
FTP-Programme 24
Zusammenfassung 24
HTML und die Dokumentstruktur 25
Von den Anfängen bis HTML 5 25
HTML-Zeitleiste 25
Der Dokumenttyp (Doctype) 26
XHTML 26
HTML 5 27
Standardisierungsbestrebungen 27
Proprietäre Lösungen und Zersplitterung 27
Das W3C (World Wide Web Consortium) 27
Grundgerüst eines HTML-Dokuments 28
Minimales Grundgerüst 28
Ein erweitertes Grundgerüst 28
Dokumentenkopf für XML-Parser 28
Professionelles Betrachten von Websites 29
Elemente, Attribute, Events 29
HTML-Elemente 29
Elemente mit Attribut 30
Events und Event-Attribute 31
Element-Arten 31
„Einmalige“ HTML-Elemente 31
Block-Elemente 32
Inline-Elemente 32
Content Modelle 33
Fließender Inhalt (Flow Content) 33
Gliedernder Inhalt (Sectioning Content) 34
Überschriften (Heading Content) 34
Ausdrücke (Phrasing Content) 35
Externer, eingebundener Inhalt (Embedded Content) 35
Interaktiver Inhalt (Interactive Content) 35
Metadaten-Inhalt (Metadata Content) 35
Regeln für das Umschließen von Elementen 36
Regel 1 36
Regel 2 36
Regel 3 36
Regel 4 36
Regel 5 36
Regel 6 36
Semantisches HTML 37

HTML und sein Erscheinungsbild ... 37
Browser-Stylesheet ... 38
Benutzer-Stylesheets ... 38
Zusammenfassung ... 39
Modelle der Visualisierung ... 40
Das DOM und das Zusammenspiel von HTML und CSS ... 40
Knoten, Kinder, Eltern und Geschwister ... 40
Vererbung und Nicht-Vererbung ... 41
Vererbbar als Default: Color-Eigenschaft (property) ... 41
Nicht-vererbbar als Default: Border-Eigenschaft (property) ... 41
Vererbung im Dokumentenbaum ... 41
Selektoren ... 42
Das style-Attribut ... 42
Stildateien, Stylesheets ... 42
Aufbau einer Stilregel ... 42
Arten von Selektoren ... 42
Element-Selektoren ... 42
Klassen-Selektoren ... 43
ID-Selektoren ... 43
Die Berechnung der Spezifität ... 43
Universal-Selektor ... 43
Kombinatoren ... 43
Kombinator für Kind-Elemente ... 44
Listen-Selektoren ... 44
Der !important-Zusatz ... 44
Kaskade ... 45
Die Trennung von Form und Inhalt ... 45
Visueller Aufbau eines HTML-Dokuments ... 45
Das Box-Modell ... 47
Breite und Höhe eines Block-Elements ... 47
Rahmen, die "Randlinie" ... 47
Innenabstand, "Polster" ... 48
Außenabstand ... 48
Weitere Eigenschaften des HTML-Elements ... 48
Illustration des Box-Modells ... 48
Der tatsächliche Platzbedarf ... 49
Das alternative Boxmodell ... 50
Maßangaben und Maßeinheiten ... 50
Längenmaße ... 50
Relative Schriftgrößen ... 50
Farbangaben ... 51
Beispiele für Farbcodes ... 51
Das Visual Formatting Modell ... 52
Floaten ... 52
Positionierung "static" ... 52
Positionierung "relative" ... 52
Positionierung "absolute" ... 52
Positionierung "fixed" ... 53
Übersichtstabelle Positionierung und Floaten ... 53
Verschachteln von HTML-Elementen ... 53
Zusammenfassung ... 53

Embedded Content und die Arbeit mit Daten ... 54
Vorgehen beim Aufbau eines HTML-Dokuments ... 54
Scribble ... 54
Schritt 1: Das HTML-Grundgerüst erstellen oder kopieren ... 54
Schritt 2: Der semantische Seitenaufbau ... 55
Schritt 3: Der visuelle Seitenaufbau ... 55
Schritt 4: Validieren ... 55
Die Einbettung von Nicht-Text-Content ... 55
Bilder ... 55
Vordergrundbilder – das img-Element (HTML) ... 55
Bildformate für Vordergrund- und Hintergrundbilder ... 56
Hintergrundbilder – die background-Eigenschaft (CSS) ... 56
Sprite-Grafik ... 57
Weitere Elemente für Embedded Content ... 57
Audio-Einbindung ... 58
Video-Einbindung ... 58
Codecs und MIME-Types ... 58
YouTube-Einbindung ... 59
Video API ... 59
JavaScript und die HTML5 Video API ... 59
Media-Events ... 59
Das Canvas-Element und seine API ... 60
Drag & Drop API ... 60
Geolocation API ... 60
Formulare und die Verarbeitung von Benutzereingaben ... 60
Aufbau eines Formulars ... 61
Visuelle Gestaltung des Formulars ... 62
Formular-Elemente ... 63
Das *form*-Element ... 63
Das *label*-Element ... 63
Versenden eines Formulars als E-Mail ... 63
Eingabefelder ... 63
Auswahlfelder ... 63
Weitere Formular-Elemente und -Attribute ... 64
Das Registrierungsformular ... 65
Das Formular als Sicherheitsrisiko ... 65
Speicherung von Daten auf dem Client ... 65
Cookie ... 65
Webstorage ... 66
Cookies/Webstorage im Vergleich ... 66
Sicherheit und Risiken ... 66
Webstorage Objekte ... 66
Zusammenfassung ... 67
Funktionale Elemente, Selektoren, Event-Attribute ... 68
Tabellarische Ausgabe von Daten ... 68
Aufbau einer Tabelle – die Tabellen-Elemente ... 68
Ausgewählte CSS-Eigenschaften für Tabellen ... 69
Listen ... 70
Nummerierte Liste, geordnete Liste (ordered list, ol) ... 70
Unnummerierte Liste, ungeordnete Liste (unordered list, ul) ... 70
Listen schachteln ... 70

Definitionsliste ... 71
Listen als Grundlage der Navigationsstruktur ... 71
Das Anchor-Element (<a>) ... 71
Zielangaben bei Links ... 72
Buttons ... 72
Menüs auf der Basis von Listen ... 72
Aktive Schaltflächen ... 73
Systematisierung der Selektoren ... 74
Ausgewählte Beispiele für Selektoren in CSS Level 3 ... 74
Pseudo-Klassen und Pseudo-Elemente ... 75
Pseudo-Klassen ... 75
Pseudo-Elemente? ... 76
Anwendungsbeispiele ... 76
Anführungszeichen, Zähler, Tooltips ... 77
Für CSS Level 4 geplante Selektoren ... 78
Interaktivität mit JavaScript ... 78
Events, Event-Attribute, Event-Handler ... 78
Weitere Events ... 79
Arbeiten mit jQuery ... 79
Zusammenfassung ... 79
Gestalterische Aspekte und Usability ... 80
Schriften ... 80
Merkmale von Schriften ... 80
Auflösung ... 81
Schriftgrößen ... 81
Systemfonts und Webfonts ... 81
Schriftarten vergleichen ... 82
Textblöcke ... 82
Absätze ... 82
Einrückungen ... 82
Textblöcke: Listen (HTML) ... 83
Typografische Regeln ... 83
Farben ... 83
Farbräume und Farbmodelle ... 83
Farbschemata ... 85
Spalten ... 87
Echte und unechte Spalten ... 87
Unechte Spalten mit float ... 87
Echte Spalten mit CSS 3 ... 88
Elemente des Kopfbereichs ... 88
Tags und Meta-Tags ... 88
Aufbau eines Meta-Tags ... 89
Von W3C empfohlene Meta-Tags (Draft) ... 89
HTML-Forwarding ... 89
Robots ... 89
Gestaltgesetze ... 89
Usability ... 91
Was ist Usability? ... 91
EN ISO 9241 - Interaktion zwischen Mensch und Computer ... 92
Die sieben Grundsätze der Dialoggestaltung ... 92
Irreführende Gestaltung ... 93

Barrierefreiheit ... 93
Was heißt Barrierefreiheit? ... 93
Web Accessibility Initiative ... 93
Barrierefreies Webdesign und multimediale Inhalte ... 94
Das Role-Attribut ... 94
Zusammenfassung ... 95
Responsive Webdesign ... 96
Printdesign und Webdesign: Einige Unterschiede ... 96
Papierformat – Bildschirmformat ... 96
Seitenumfang ... 96
Interaktion ... 96
Usability ... 96
Suchmaschinen ... 97
Herausforderungen des Webdesigns ... 97
Webdesign-Philosophien ... 97
Web Style Guides ... 98
Web Style Guide, 3rd Edition (Yale) ... 98
Media-Attribute ... 99
Notierung gerätespezifischer Media-Attribute ... 99
Liste gerätespezifischer Media-Attribute ... 100
Der Viewport ... 100
Media Querys – Medienmerkmale abfragen ... 100
Anwendungen ... 101
„Hamburger-Menü“ ... 101
Nützliche Verfahren für das Responsive Webdesign ... 102
Bilder skalieren ... 103
Menüs anpassen ... 103
Schatten für Container und Schrift ... 104
Zusammenfassung ... 104
Einführung in XML ... 105
Was ist XML? ... 105
Spezifische und generische Codierung ... 105
Spezifische Codierung ... 105
Abgrenzung von XML zu XHTML und CSS ... 105
Anwendungsfelder für XML ... 106
Ein XML-Dokument ... 106
XML-basierte Auszeichnungssprachen: ... 106
XML-Software ... 106
Aufbau eines XML-Dokuments ... 107
Dateiformat ... 107
Prolog ... 107
XML-Daten, Baumstruktur, Wohlgeformtheit ... 107
Elementnamen ... 108
Wohlgeformtheit ... 108
Die Dokumenttyp-Definition ... 108
Validierbare XML-Dokumente ... 109
Mit DOCTYPE valides XML erzeugen ... 109
Mit XML-Schema-Definition valides XML erzeugen ... 110
Namensräume ... 111
„Problematische“ Element-Definitionen ... 111
Namespaces und Qualified Names ... 111

Gültigkeitsbereich der Namespace-Deklaration ... 112
Das Schema-Element und seine Referenzierung ... 112
XSD: Das XML-Schema-Element ... 112
XML: Referenzierung der Schema-Definition ... 113
Elementtypen einer DTD ... 113
Bestandteile der Dokumentstruktur ... 113
Elementtyp: EMPTY ... 113
Elementtyp: ANY ... 114
Elementtyp: (#PCDATA) ... 114
Kommentare ... 114
XML-Verknüpfungszeichen ... 114
Verknüpfungszeichen "," (Komma) ... 114
Verknüpfungszeichen "|" ... 114
Verknüpfungszeichen "*" (Multiplikationszeichen) ... 115
Verknüpfungszeichen "?" ... 115
Verknüpfungszeichen "+" ... 115
Zusammenfassungszeichen "( )" (Klammer) ... 116
Das visuelle Erscheinungsbild von XML-Elementen ... 116
CSS und XSL ... 116
CSS und XML ... 116
Eine XML-Bücher-Tabelle erstellen ... 117
Aufgabenstellung ... 117
Schritt 1 ... 117
Schritt 2 ... 117
Schritt 3 ... 117
Schritt 4 ... 118
Schritt 5 ... 118
Schritt 6 ... 118
Zusammenfassung ... 118
Erzeugen valider XML-Dokumente ... 120
DTD: Verfeinerte Dokumenttyp-Definitionen ... 120
Element-Attribute definieren ... 120
Attribut-Typen ... 120
Attribute in der Übersicht ... 121
Modifikatoren ... 121
Regeln für Attribute ... 122
Beispiel für "gutes" und "weniger gutes" XML ... 122
DTD: Entitys ... 123
Entity-Beispiele ... 124
Modulare DTD mit Hilfe von Parameter-Entitys ... 124
XSD: Erstellen und anwenden von Schema-Definitionen ... 125
Vergleich XSD und DTD ... 125
XSD: einfache Elemente ... 126
XSD: komplexe Elemente ... 126
XSD: Datentypen ... 127
Übersicht: Merkmale der XML-Syntax ... 127
XML-Syntax ... 127
Wohlgeformtheit ... 127
Gültigkeit ... 127
Bestandteile eines XML-Dokuments ... 127
CDATA ... 128

#PCDATA ... 128
Zusammenfassung ... 128
XSL-Transformationen ... 129
XSL (Extensible Stylesheet Language) ... 129
XSL-Merkmale ... 129
XSL-Transformation von XML nach HTML ... 129
XSL-Transformation von XML nach XML ... 130
Das XSL-Dokument ... 130
XML-Schema und XSLT ... 132
XML-basierte Sprachen ... 132
XML-Organisationen ... 132
DocBook ... 133
ODF ... 133
RELAX NG ... 134
Epub-Format ... 134
RSS Feed ... 134
XHTML ... 134
SVG ... 134
MathML ... 135
Beispiel MathML und SVG in HTML ... 135
Die XSL-Familie ... 135
Drei Arten, XML-Dokumente anzuzeigen ... 136
Zusammenfassung ... 136
Anhang ... 136
Abbildungsverzeichnis ... 136
Tabellenverzeichnis ... 137
Verzeichnis der Links ... 137

Informationen zur Veranstaltung

Das Modul „Webentwicklung“ ist für Studierende der Angewandten Informatik eine Pflichtveranstaltung. Es werden 6 ECTS-Punkte erreicht.

Inhalte: Die wichtigsten Auszeichnungssprachen

- XML (Daten)
- HTML (Dokumentstruktur)
- CSS (Aussehen)
- Exkurs Javascript (Interaktion)

Mit Hilfe von Auszeichnungssprachen werden aus Daten gestaltete Dokumente.

Ablauf: Vorlesung und Praktikum

- Die Vorlesung führt durch die maßgeblichen Inhalte
- Das Praktikum ermöglicht es, die Inhalte anzuwenden
- Auf Wunsch findet eine Zwischenklausur statt

In der Hochschule für angewandte Wissenschaften kommt es vor allem auf die Anwendungsorientierung an – also auf die Praxis.

Sprachkenntnisse

- Lehr- und Prüfungssprache: deutsch
- Gute Englischkenntnisse sind zum Studium von Originaldokumenten erwünscht.

Mitarbeit und Leistungsnachweis

Mitarbeit

- Keine Anwesenheitspflicht (außer bei der Klausur)
- Spezielle Fragen beantworte ich ausschließlich während des Praktikums
- E-Mail-Anfragen nur öffentlich im Moodle-Diskussionsforum

Benotung

- Dreistündige, praktische Abschlussklausur
- Die Abschlussklausur setzt sich aus den ca. 100 Praktikumsaufgaben zusammen

Ort: PC-Pool-Geräte mit dem Editor notepad++
Hilfsmittel: Vorlesungsskripte, Internet-Dokumente

Workload und Prüfungsanmeldung

Berechnung der ECTS-Punkte

- Vorlesung (2 SWS) = 30 Stunden Anwesenheit
- Praktische Übungen (2 SWS) = 30 Stunden Anwesenheit
- Vor- und Nachbereitung

Qualifikationsziele

- Aufbau von XML-Dokumenten
- Umsetzung auch umfangreicherer Webseiten in HTML
- Gestaltung mittels CSS
- Kenntnis grundlegender Usability-Prinzipien

Literatur

Print

- *Brian P. Hogan (deutsch von Stefan Fröhlich)*, HTML5 & CSS3, Webentwicklung mit den Standards von morgen, ISBN 978-3-89721-316-6, O'Reilly, 2011
- *Kai Günster,* Schrödinger lernt HTML5, CSS3 und JavaScript, Das etwas andere Fachbuch, ISBN 978-3-8362-2020-0, Galileopress, 2013
- *Helmut Vonhoegen*, Einstieg in XML, Grundlagen, Praxis, Referenz, ISBN 978-3-8362-2620-2, Galileopress, 2013

Internetquellen

- *Hrsg. W3C*, All Standards und Drafts, http://www.w3.org/TR/ (zuletzt aufgerufen am 8.9.2014)
- *Hrsg. W3C*, HTML5, A vocabulary and associated APIs for HTML and XHTML, W3C Recommendation 28 October 2014, (zuletzt aufgerufen am 7.1.2015)
- *Patrick J. Lynch, Sarah Horton*, Web Style Guide, 3rd Edition, http://webstyleguide.com/ (zuletzt aufgerufen am 8.9.2014)

W3C, Empfehlungen und Dokumente

- Styles | CSS versus XSL
- Zum Boxmodell (CSS3)
- Zur Seite Selektoren (CSS3)
- Zur Seite HTML5.1 (Nightly)
- Zur XML-Empfehlung
- Zur XML-Schema-Empfehlung
- Zur XSLT-Empfehlung

W3C Wiki

- Hauptseite
- Beispiel: floaten
- Beispiel: positionieren

Web-Tools

- Color Scheme Designer (Farbgestaltung, Farbharmonien)
- Blindtext-Generator (Platzhalter für Text)
- Free Formatter (Code Formatierer, Generatoren, Validatoren)
- http://www.utilities-online.info/ (XML-Tools, Validatoren)
- Schriftarten-Tester (Vergleich unterschiedlicher Textformate)
- Ultimate CSS Gradient Generator (Hintergrundverläufe erstellen)
- Schattengenerator (Schatten erstellen)
- CSS3 Generator (CSS3-Effekte erstellen)
- Online-Menumaker (CSS Drop-Down Menu Framework, GNU- und MIT-License)

Eine weniger nützliche Seite

- OMF (mit Sound)

Sammlung von Websites

- Beispiele für responsive Webdesign
- Awwward (Kommerzielle Preisverleihung)
- Best Web Gallery (Als besonders gut empfunden Websites)

- Best Designs (Ebenfalls als gut empfundene Websites, geordnet nach Kategorien)

Ranking international

- Top Sites (Alexa)
- Top Sites (Thema, Suchbegriff)

Tests und Vergleiche

- http://www.quirksmode.org (Browser-Kompatibilitätsinformationen)
- http://www.browserscope.org/ (Browser-Standard-Tests)
- http://www.browserscope.org/alltests (Browser-Standard-Tests im Autorun-Mode)
- http://haz.io/ (Testet den aufrufenden Browser auf CSS- und HTML-Fähigkeit)
- http://caniuse.com/ (Vergleicht ausgewählte Features unter verschiedenen Browsern)
- http://mobilehtml5.org/ (HTML- und CSS-Fähigkeiten der Browser auf mobilen Endgeräten)
- http://www.css3.info/modules/ (Status der W3C Recommendations)
- http://tools.css3.info/selectors-test/test.html (Selektoren-Test)

Alle Links wurden zuletzt am 31. März 2018 geprüft.

Daten, Dokumentstruktur und Erscheinungsbild

Aus Daten werden präsentable Dokumente

Informatiker und Informatikerinnen, die Web-Anwendungen programmieren, stehen stets vor dem Problem,

- **Daten**
- als **Dokumente** auszugeben
- und ihnen das gewünschte **Aussehen**

zu verleihen.

Daten

Daten bestehen immer aus einem *Bezeichner* und einem *Wert*. Zusammengehörende Bezeichner und Werte heißen *Datensatz*.

Bezeichner und Werte

```
Bezeichner: Wert /* Definition */
Gegenstand: Strandkorb
Tagespreis: 6 Euro
Standort: Strandabschnitt 14
Beschreibung: Bei dem Strandkorb handelt es sich um ein wetterfestes Modell mit
    ausziehbarer Schublade.
Bild: Strandkorb.png
```

Datensätze

CSV-Datei mit Feldbezeichner in der ersten Zeile, Semikolon als Werte-Trennzeichen, insgesamt drei Datensätze

```
Gegenstand; Tagespreis; Standort; Beschreibung; Bild
Strandkorb; 6 Euro; Strandabschnitt 14; Bei dem Strandkorb handelt es sich um ein
    wetterfestes Modell mit ausziehbarer Schublade. ; Strandkorb.png
Sonnenschirm; 3 Euro; Standhalle; Nur für Sonne, nicht für Regen geeignet;
    Schirm.jpg
Ruderboot; 24 Euro; Hafenmole; Ruderboot für maximal vier Personen; Ruderboot.jpg
```

Die Daten sind in der **Datenbank** des Strandkorbverleihers gespeichert. Sie sind unsichtbare Bit-Werte. Um sie sichtbar zu machen, wird ein Dokument erzeugt, dem dann ein bestimmtes Erscheinungsbild (Aussehen) verliehen wird.

Dokumente

Dokumente enthalten die Teilmenge der Daten, die zur Darstellung ausgewählt werden. Sie bestehen aus den **Gliederungselementen** der Seite. Die Gliederungselemente werden mit den Bezeichnern und Werten aus den Datensätzen gefüllt. Auch das Dokument ist zunächst unsichtbar. Erst wenn die Gliederungselemente ihr Erscheinungsbild erhalten, wird das Dokument sichtbar.

Aussehen und Erscheinungsbild

Durch das Erscheinungsbild erhalten die Gliederungselemente ihr Aussehen. Ein mögliches Erscheinungsbild könnte das Bild linksbündig zu zeigen und die Titelzeile „Strandkorb“ ins Zentrum des Interesses zu rücken.

Dokument, Variante 1

```
Überschrift: Gegenstand
Abschnitt-1: Bild
Abschnitt-2: Tagespreis, Standort, Beschreibung
```

Abbildung 1: Das Dokument erhält sein Erscheinungsbild (Dokument (1))

Dokument, Variante 2

Im zweiten Beispiel sind den Gliederungselementen des Dokuments andere Bezeichner und Werte zugeordnet. Hier steht der Preis – vielleicht ein Sensationspreis? – im Vordergrund.

```
Überschrift: Tagespreis, Standort
Abschnitt-1: Bild, Gegenstand, Beschreibung
```

Abbildung 2: Ein alternatives Erscheinungsbild (Dokument (2))

Dokument, Variante 3

Die Werte der Datenbank werden im dritten Beispiel verwendet, um eine Produktliste auszugeben. Der Liste liegt eine völlig andere Dokumentstruktur zugrunde. Die Daten sind jedoch dieselben.

```
Überschrift: Produktliste
Tabelle: Bild, Gegenstand, Tagespreis, Standort, Beschreibung
```

Produktliste

Bild	Gegenstand	Tagespreis	Lage	Beschreibung
	Strandkorb	6 Euro	Strandabschnitt 14	*Bei dem Strandkorb handelt es sich um ein wetterfestes Modell mit ausziehbarer Schublade*
	Sonnenschirm	3 Euro	Standhalle	Nur für Sonne, nicht für Regen geeignet
	Ruderboot	24 Euro	Hafenmole	Ruderboot für maximal vier Personen

Abbildung 3: Ein drittes Erscheinungsbild (Dokument (3))

Aber auch wenn kein besonderes Erscheinungsbild mit dem Dokument verbunden ist, wird das Dokument dennoch im Browser angezeigt. Dies kommt daher, dass jeder Browser über eingebaute Regeln verfügt, wie Dokumentstrukturen darzustellen sind.

Strandkorb

Tagespreis: 6 Euro

Lage: Strandabschnitt 14

Bei dem Strandkorb handelt es sich um ein wetterfestes Modell mit ausziehbarer Schublade.

Abbildung 4: Anzeige des Dokuments mit Hilfe der eingebauten Darstellungsregeln

Technologien für Daten, Dokumentstruktur, Erscheinungsbild

Daten (XML)

Die Auszeichnungssprache XML (*Extensible Markup Language*) beschreibt und strukturiert Daten und ist gleichzeitig eine Meta-Sprache, mit der sich andere Auszeichnungssprachen deklarieren lassen. Das folgende Listing zeigt die Datenstruktur der Produktliste in der Sprache XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Autor: Klaus Westermann -->
<Produktliste>
  <Datensatz>
```

```
    <Gegenstand>Strandkorb</Gegenstand>
    <Tagespreis>6 Euro</Tagespreis>
    <Standort>Strandabschnitt 14</Standort>
    <Beschreibung>Bei dem Strandkorb handelt es sich um ein wetterfestes Modell mit
ausziehbarer Schublade.</Beschreibung>
    <Bild>Strandkorb.png</Bild>
  </Datensatz>
  <Datensatz>
    <Gegenstand>Sonnenschirm</Gegenstand>
    <Tagespreis>3 Euro</Tagespreis>
    <Standort>Standhalle</Standort>
    <Beschreibung>Nur für Sonne, nicht für Regen geeignet.</Beschreibung>
    <Bild>Schirm.jpg</Bild>
  </Datensatz>
  <Datensatz>
    <Gegenstand>Ruderboot</Gegenstand>
    <Tagespreis>24 Euro</Tagespreis>
    <Standort>Hafenmole</Standort>
    <Beschreibung>Ruderboot für maximal vier Personen.</Beschreibung>
    <Bild>Ruderboot.jpg</Bild>
  </Datensatz>
</Produktliste>
```

Dokument (HTML)

Die Auszeichnungssprache HTML/XHTML (*Hypertext Markup Language/Extensible Hypertext Markup Language*) beschreibt die Struktur eines Dokuments. Weiterhin ist es möglich, über Element-Attribute das visuelle Erscheinungsbild des Inhalts festzulegen. Das sollte jedoch vermieden werden. Seitenstruktur und visuelles Erscheinungsbild sollten voneinander getrennt sein. Das visuelle Erscheinungsbild ist die Aufgabe der *Cascading Style Sheets* (CSS).
Das hier gezeigte Listing strukturiert einen Datensatz als HTML-Dokument.

```
<!DOCTYPE html>
<html lang="de">
<head>
<meta charset="utf-8" />
<title>Strandkorb-1</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link href="_reset.css" rel="stylesheet" type="text/css">
<link href="strandkorb.css" rel="stylesheet" type="text/css">
</head>
<body>
<section>
<h1><img src="strandkorb1.jpg" width="400" height="400"
alt="Strandkorb">Strandkorb</h1>
<p>Tagespreis: 6 Euro</p>
<p>Lage: Strandabschnitt 14</p>
<p class="info"><em>Bei dem Strandkorb handelt  es sich um ein wetterfestes Modell
mit ausziehbarer Schublade</em><em>.</em></p>
<p class="clearer"> </p>
</section>
</body>
</html>
```

Aussehen (CSS)

CSS (Cascading Style Sheets) ist eine Sprache, die Stil-Eigenschaften von HTML-Elemente definiert. Damit ist es möglich, das visuelle Erscheinungsbild eines Dokuments pixelgenau festzulegen. Erst die im Stylesheet definierten Regeln machen das Dokument sichtbar. Es ist sogar möglich, die Sichtbarkeit je nach Ausgabemedium unterschiedlich zu regeln (Siehe *Media-Attribute* und *responsive Webdesign*).
Die hier notierten Stilregeln beschreiben das Aussehen des *img-Elements* und des *section-Elements*.

```
img {
	float: left;
	margin-right: 10px;
}
section {
	border: 3px solid #516683;
	padding: 10px;
	height: 400px;
	width: 765px;
	margin-right: auto;
	margin-left: auto;
	margin-top: 10px;
}
```

Browser-Stylesheet und Benutzer-Stylesheet

Alle Browser verfügen über eingebaute Stylesheets (*Browser-Stylesheet*), die das Erscheinungsbild der HTML-Elemente festlegen, sofern Web-Designer keine weiteren Regeln für das Erscheinungsbild definieren. Je individueller das Erscheinungsbild, desto ausgefeilter die Stil-Regeln, die im Web-Design erstellt werden (*Benutzer-Stylesheet*).

Weitere Auszeichnungssprachen für Daten, Dokumente, Erscheinungsbild

APT

APT (*Almost Plain Text*) ist eine vereinfachte Auszeichnungssprache für technische Dokumentationen.

```
Section title
* Sub-section title
** Sub-sub-section title
*** Sub-sub-sub-section title
```

BBCode

BBCode ist ein HTML-ähnlicher Auszeichnungscode mit geringem Sprachumfang.

```
[b]fett[/b]
[i]kursiv[/i]
[u]unterstrichen[/u]
[s]durchgestrichen[/s]
```

Wikitext, Wiki-Code, Wiki-Syntax

Wikitext ist eine nicht-standardisierte, vereinfachte, schnell zu lernende Auszeichnungssprache für Wiki-Texte.

```
1st level heading: = ... =
```

```
2nd level heading: == ... ==
3rd level heading: === ... ===
4th level heading: ==== ... ====
5th level heading: ===== ... =====
```

JASON

Jason ist eine auf *Javascript* basierende Auszeichnungssprache für Daten und Datenstrukturen.

```
{
"firstName": "John",
"lastName": "Smith",
"age": 25
}
```

LaTeX

LaTeX dient dem Erzeugen von Dokumentationen.

```
%% Auszug
\maketitle
\tableofcontents
\section{Einleitung}
Hier steht der Einleitungstest. Die Überschrift kommt automatisch ins
Inhaltsverzeichnis.
\subsection{Gliederungsebene 2}
```

Auszeichnungssprachen „erfinden“

Auszeichnungssprachen können in nahezu beliebiger Weise erfunden werden. Kennen Sie schon „Gallien“, die Sprache für Speisenkarten?

```
!!! Sprache: Gallien Version 0.01 alpha%
!majestix! Frühstück %majestix
!miraculix! Mittagessen %miraculix
!obelix! Abendessen %obelix
!römer! Nachtisch %römer
```

In dem nicht ganz ernst gemeinten Beispiel geht es um ein wichtiges Frühstück, ein wundersames Mittagessen, ein üppiges Abendessen und einen sportlichen Nachtisch… Die Auszeichnungssprache „Gallien“ ist hiermit erfunden.

Warum heißen Auszeichnungssprachen „Auszeichnungssprachen“?

„Auszeichnung“ im Print-Medium

Unter „Auszeichnung“ verstehen Schriftsetzer und Layouter die Kennzeichnung von Text, damit sie den Text sinngemäß richtig „setzen“ können. Autoren und Korrektoren verfügen normalerweise nicht über die Hilfsmittel, um ein Text ansprechend und vor allem über die Gesamtheit der Seiten hinweg mit einem konsistentem Erscheinungsbild zu versehen. Nach dem Setzen gaben die Drucker die erste „Fahne“ an den Autor zur Durchsicht zurück, damit er neben den Korrekturen auch die gewünschten Auszeichnungen vornimmt.

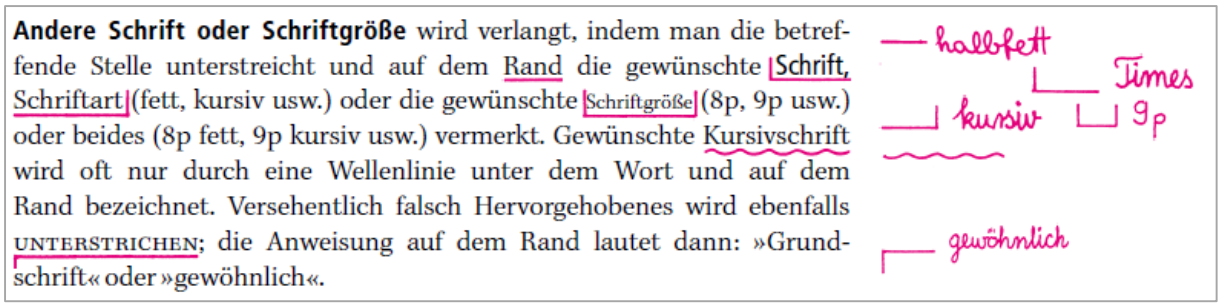


Abbildung 5: Korrekturzeichen nach DIN 16511

Die Abbildung zeigt die Korrekturzeichen nach DIN 16511. Die Auszeichnung – das *Markup* – ist zwar nicht maschinenlesbar, zeigt aber das Prinzip einer Auszeichnungssprache. Nichts anderes machen XML und HTML. Sie markieren Text und weisen den markierten Abschnitten eine Rolle in der Daten- und Dokumentstruktur zu.

```
<Buchtitel>Vom Winde verweht</Buchtitel>
```

Die XML-Notierung weist „Vom Winde verweht“ die Bedeutung eines Buchtitels zu.

```
<p>Scarlett <strong>liebt</strong> Rhett Butler.</p>
```

Die HTML-Notierung weist den Browser an, „Scarlett liebt Rhett Butler“ als eigenständigen Absatz (Paragraph <p>) zu behandeln und das Wort „liebt“ mit der Bedeutung „wichtig“ zu versehen.

SGML, „Urmutter“ aller Auszeichnungssprachen

In den Fünfzigerjahren des 20. Jahrhunderts, als die Computertechnik den Kinderschuhen entwuchs, erkannten Forscher und Techniker die Notwendigkeit von maschinenlesbaren Auszeichnungssprachen, die von allen Computersystemen in gleicher Weise interpretiert werden.
In der Folge schuf *Charles Goldfarb* für *IBM* die erste Generalized Markup Language, kurz *GML*. Es ging dabei nicht um das Aussehen von Dokumenten, sondern um die Kennzeichnung ihrer logischen Struktur wie zum Beispiel die Gliederung eines Textes in Überschriften, Unterüberschriften oder Abschnitte. Die Weiterentwicklung von GML wurde 1986 als *SGML* (*Standard Generalized Markup Language*) zum *ISO-Standard* (ISO 8879:1986). SGML wurde somit zur „Urmutter“ aller standardisierten Auszeichnungssprachen.

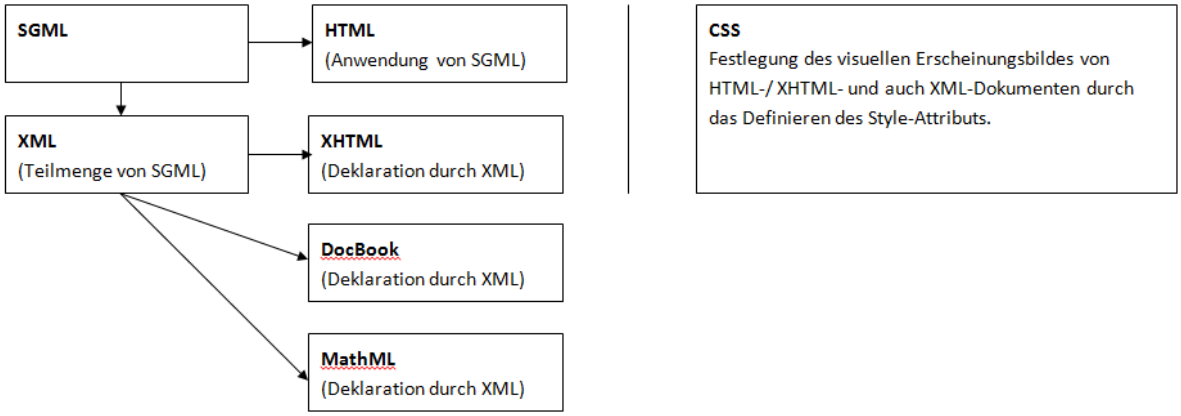


Abbildung 6: Auswahl einiger aus SGML abgeleiteter Auszeichnungssprachen

SGML ist als gesamte Spezifikation technisch nicht implementierbar. Jedoch sind weitere standardisierte Auszeichnungssprachen aus ihr hervorgegangen. So ist *HTML* eine Anwendung von SGML, *XML* ist eine Teilmenge von SGML ("Best of...").

Auszeichnungssprachen und Programmiersprachen

Programmiersprachen verarbeiten Daten, Auszeichnungssprachen beschreiben Daten.

Datenverarbeitung

Ein Rechenprogramm macht aus den beiden Eingabedaten „1“ und „1“ und dem Verarbeitungszeichen „+“ das Ausgabedatum „2“ oder „11“, je nachdem ob das „+“-Zeichen als String-Verkettung oder als Rechenzeichen interpretiert wird.

```
1 + 1 = 2 (Addition)
oder
1 + 1 = 11 (String-Verkettung)
```

Datenbeschreibung

Eine Auszeichnungssprache tut dagegen – grob gesagt – „nichts“. Sie ist im Vorfeld der Berechnung nötig, um die Bedeutung der Eingabedaten zu beschreiben, damit die Programmiersprache weiß, wie sie verfahren soll. Im folgenden Beispiel beschreibt sie die Daten (<Erste_Zahl>, <Zweite_Zahl>), den Operator und das Verkettungsverfahren (<Verfahren>), das auf die Daten angewendet werden soll.

```
<Rechenaufgaben>
	<Datensatz>
		<Erste_Zahl>1</Erste_Zahl>
		<Zweite_Zahl>1</Zweite_Zahl>
		< Operator>+</ Operator>
		<Verfahren>Rechnen</Verfahren>
	</Datensatz>
	<Datensatz>
		<Erste_Zahl>4</Erste_Zahl>
		<Zweite_Zahl>3</Zweite_Zahl>
		< Operator>*</ Operator>
		<Verfahren>Verketten</Verfahren>
	</Datensatz>
</Rechenaufgaben>
```

Das Beispiel soll als Anhaltspunkt dienen. In der Praxis ist die Abgrenzung zwischen Programmier- und Auszeichnungssprachen jedoch nicht immer einfach.

Die Arbeitsweise des Internets

Client und Server

Das WWW basiert auf dem *Client-Server-Prinzip*. Der *Server* (Bediener, Daten-“Diener”) ist ein Softwareprogramm, das auf die *Requests* (Anfragen) eines *Clients* (Kunde) reagiert und die gewünschten *Responses* (Antworten) bereitstellt. Die Server-Software, zum Beispiel *Apache*, wird auf einer Computer-Hardware installiert, die ebenfalls als Server bezeichnet wird. Millionen von Servern halten weltweit Daten bereit.

Protokolle

Um auf unterschiedlichen Rechnerarten und Betriebssystemen den Datenaustausch zu ermöglichen, sind einheitliche Protokolle für die Datenübertragung notwendig, hier die *TCP/IP* - Protokollfamilie (*Transmission Control Protocol* auf dem *Internet Protocol*).

Serverseitige Sprachen

Die Responses des Servers werden von Sprachen abgearbeitet, die für den Client unsichtbar sind.

Beispiele sind *PHP*, *Perl*, *Phyton* und viele andere. Sie formulieren die Responses als HTML-Dokumente und senden diese an den Client zurück. Der Client sieht nur das ausgelieferte HTML (und CSS, JavaScript), nicht aber die sprachlichen Anweisungen, mit denen diese Dokumente erstellt wurden.

Clientseitige Sprachen

Auf der Seite der Clients, also auf den lokalen Computern, ist es die Sache des Web-Browsers, die ankommende Information darzustellen. Der Browser setzt den HTML- und CSS-Code in sichtbare Information um. Häufig ist im Datenstrom auch JavaScript-Code enthalten, der ebenfalls im Client ausgeführt wird. Je nach den Fähigkeiten des Browsers kommt es zu unterschiedlichen Ergebnissen. Webseiten müssen deshalb vor der Freigabe unbedingt unter verschiedenen Bedingungen mit verschiedenen Browsern getestet werden.

Die Arbeitsweise des Browsers

Der HTML-Code wird im Browser von Maschinencode in eine für Menschen lesbare Form gebracht. Dies geschieht in zwei Schritten.

- Die Analyse und Bewertung des HTML-Codes leistet der *Parser*.
- Für die visuelle Darstellung sorgt die *Rendering-Engine*.
- Das *Javascript-Interpreter* führt Javascript-Programmieranweisungen aus.

HTML-Parser

HTML-Code ist für Software zunächst einmal nichts Anderes als reine Textinformation, eine Aneinanderreihung von Zeichen. Der in einem Webbrowser enthaltene HTML-Parser analysiert die ankommenden Zeichen und erkennt anhand der Syntaxregeln - zum Beispiel der spitzen Klammern - die Strukturelemente. Das Ergebnis des Parsens ist das *DOM*, das *Document Object Model*. Ein Parser-Modul, das in beliebiger Software verwendet werden kann, um einen Datenstrom hinsichtlich seiner HTML-Elemente zu untersuchen, ist der *HTML-Parser*.

➡ ***HTML-Parser*** **(http://htmlparser.sourceforge.net/).**

Rendering-Engine

HTML schreibt nicht vor, wie ein Dokument angezeigt werden soll, HTML gibt nur die semantische Struktur vor. Die Rendering-Engine ordnet der semantischen Struktur das Aussehen zu. Je nach Ausgabemedium (Monitor, Beamer, TV-Gerät, Drucker, Braille-Leser…) kann das visuelle Erscheinungsbild den spezifischen Geräteanforderung entsprechend *gerendert* werden. Die Rendering-Engine wird von den notierten CSS-Eigenschaften gespeist. Fehlen diese - ist also kein Benutzer-CSS vorhanden - wird das *Browser-CSS* verwendet, das in jeden Browser als Default eingebaut ist. Je nach Hersteller existieren unterschiedliche Engines:

- *Gecko* (u.a. Mozilla Firefox, SeaMonkey, Thunderbird), entstanden aus Netscape-Navigator, siehe auch https://wiki.mozilla.org/Gecko:Home_Page
- *KHTML* (u.a. Konqueror), vom KDE-Projekt entwickelt
- *WebKit* (u.a. Safari, Google Chrome), Abspaltung und Weiterentwicklung von KHTML, siehe auch http://www.webkit.org
- *Presto* (Opera), in neueren Versionen ebenfalls *Webkit*
- *Trident* (Internet Explorer für Windows), erstmals 1997 vorgestellt, mit IE 8 komplett überarbeitet und endlich standardkonform
- *Tasman* (u.a. Internet Explorer für Mac), seit 2000 Rendering-Engine im IE für Mac.

Fehlertoleranz von Browsern

HTML-Parser werden auf den meisten Web-Seiten mit zum Teil schwerwiegenden *Syntaxfehlern*

konfrontiert, sie müssten die Anzeige solcher Web-Seiten eigentlich abbrechen. Unter Marktgesichtspunkten wäre das jedoch fatal. Deshalb sind HTML-Parser mehr oder weniger tolerant gegenüber fehlerhaftem Code. Standardkonformes HTML ist heute jedoch von großer Bedeutung. Nur so sind die Forderungen nach raffinierter Gestaltung und nach Auffindbarkeit durch Suchmaschinen zu vereinbaren.

Validatoren

Validatoren prüfen den Code auf Konformität zu seinen Regeln und Standards. Bekannt ist der Validierungsservice des W3C. Darüber hinaus existieren zahlreiche weitere Möglichkeiten der Validierung von HTML/XHTML-Code.

Werkzeuge und Hilfsmittel

Zum Editieren von W3C-konformen Auszeichnungssprachen ist keine aufwändige Software nötig. Es genügt ein einfacher Text- bzw. Zeilen-Editor, der die *UTF-8-Codierung* unterstützt. Mehr Spaß macht allerdings die Arbeit mit Editoren, die das *Syntax-Highlighting* beherrschen, die also Schlüsselwörter, Variablen, Werte und Eigenschaften farbig hervorheben.
Textverarbeitungsprogramme wie Word oder Open Office sollten keinesfalls benutzt werden, um Auszeichnungssprachen zu editieren. Die genannten Programme enthalten selbst bereits Auszeichnungscode.

Kommerzielle Editoren und Freeware

Im professionellen Arbeitsumfeld von Webdesign-Agenturen sind häufig Produkte der Firma *Adobe* im Einsatz wie die *Creative Suite (CS)* und die *Creative Cloud (CC).* CS und CC sind die Sammlung einer Vielzahl von Programmen zur Erstellung von Print- und Onlinemedien. Häufig benutzte Programme im Webdesign sind *Dreamweaver* (Dokumentenauszeichnung) und *Photoshop* (Grafik). Die Konkurrenzprodukte der Firma *Microsoft* hießen lange Zeit *Frontpage* und *Expression Web 4. Heute ist es Microsoft Visual Studio für das Web.*

- **Adobe Dreamweaver** (*Adobe*), kostenpflichtige Software für Windows und Mac
- **Brackets** (Adobe), Freeware für Windows, Mac, Linux
- **Expression Web 4** (*Microsoft*), Freeware seit 2012, Windows
- **Microsoft Webmatrix** (*Microsoft*), Freeware mit Tools für den Bau von Websites
- **Phase 5** (*Ulli Mehbohm*), Freeware für Privatpersonen und allgemeinbildende Schulen, Windows, letzte Version 5.6.2 von 2008
- **Visual Studio Express für das Web** (*Microsoft*), Freeware für Websites
- **Webocton-Scriptly** (*Benedikt Loepp*), zurzeit Freeware in der Version 0.8x von 2010, Windows, kein UTF-8

und andere...

Open-Source-Editoren

Leistungsfähige Editoren gibt es auch mit Open Source Lizenzen. Bei der Auswahl von Open-Source-Programmen bitte auf die Größe der Entwicklergemeinde und die letzte Aktualisierung achten.
Notepad++ ist der einzige, in den Klausuren zugelassene Editor für HTML- und CSS-Dokumente. Für XML-Dokumente kann auch der XML Copy Editor benutzt werden.

- **Bluefish**, Mac, Windows, Linux
- **Notepad++**, Windows.
- **Smultron**, Mac
- **XML Copy Editor** (mit eingebautem Validator), Windows

Validatoren

Validatoren prüfen den Code auf Konformität zu den Syntax-Regeln und Standards. Bekannt ist der Validierungsservice des W3C. Darüber hinaus existieren weitere Möglichkeiten der Validierung von HTML/XHTML-Code.

- WWW-Consortium, HTML-Validator, HTML-Validator
- WWW-Consortium, CSS Validator, CSS-Validator
- WWW-Consortium, Link-Checker, Link-Checker

Kompatibilität

- Darstellung auf alten Internet-Explorern simulieren, http://netrenderer.com
- Site-Performance, Dr. Watson, http://watson.addy.com/
- Screenshots auf anderen Browsern (Browsershots), http://browsershots.org/
- A-Prompt (http://wob11.de/apromptkomplett.html) Validierung und Prüfung hinsichtlich Barrierefreiheit

Sammlung von Werkzeugen für Webentwickler

- UI-Test

Browser und Browser-Add-Ons für die Entwicklungsumgebung

Zurzeit sollten Webseiten noch auf verschiedenen Browsern getestet werden. Die HTML- und CSS-Standards sind unterschiedlich implementiert. HTML-Dokumente müssen auch auf dem Internet-Explorer ab Version 9 im gleichen Erscheinungsbild angezeigt werden. Der Internet-Explorer ist wegen seines großen Marktanteils in Großunternehmen und im Business-Umfeld von besonderer Bedeutung.
Viele Anzeigeunterschiede und -Abweichungen beruhen jedoch auf fehlerhaftem HTML-Code und unterschiedlicher Fehlertoleranz der Browser.

Mindestausstattung für Browser

- IE
- Firefox
- Chrome
- Edge

Add Ons

- *Web-Developper Toolbar* (Firefox und Chrome)
- *Color-Picker*

Bei den verwendeten Browsern für die Webseitenerstellung sollte man sich unbedingt der Browser des Zielpublikums bedienen. Für Medienkonsumenten und den Massenmarkt sind die Ergebnisse von Linux-Browsern irrelevant.

Nachschlagen im Internet

Gerade am Anfang fällt es häufig schwer, sich die verschiedenen HTML-Elemente und CSS-Eigenschaften zu merken. Folgende und andere Ressourcen seien empfohlen:

- HTML 5: W3C-Dokument | W3C-Schools html-Referenz
- HTML 5.1: W3C-Dokument
- HTML 4 und XHTML: Selfhtml
- CSS 2.1: Selfhtml | CSS4YOU
- CSS 3: W3C-Dokumente | W3C-Schools CSS Referenz

Die Arbeitsumgebung

Bevor Sie mit einem Webprojekt beginnen, richten Sie auf Ihrem Computer die Arbeitsumgebung ein, die Ihre Webseiten und Grafiken in derselben Anordnung enthält, wie der Webspace im Internet. Alle Dateien eines Webprojekts müssen innerhalb, bzw. unterhalb eines einzigen Ordners liegen. Der Ordner, ist also eine lokale Kopie ihrer Website. Legen Sie einen Ordner an, in dem Sie alle Webprojekte (zum Beispiel: *Webprojekte)* sammeln möchten. Achten Sie auf die Groß- und Kleinschreibung. Das kann online von Bedeutung sein. Erstellen Sie für *jedes Projekt einen eigenen Unterordner*, damit sich die Verweise zu den Daten verschiedener Kunden nicht überkreuzen oder auf Ordner verweisen, der später nicht in den Webspace geladen werden.
Prinzipiell ist es möglich, Umlaute zu verwenden, doch in der Praxis führen sie immer wieder zu Überraschungen und unnötigem Arbeitsaufwand.

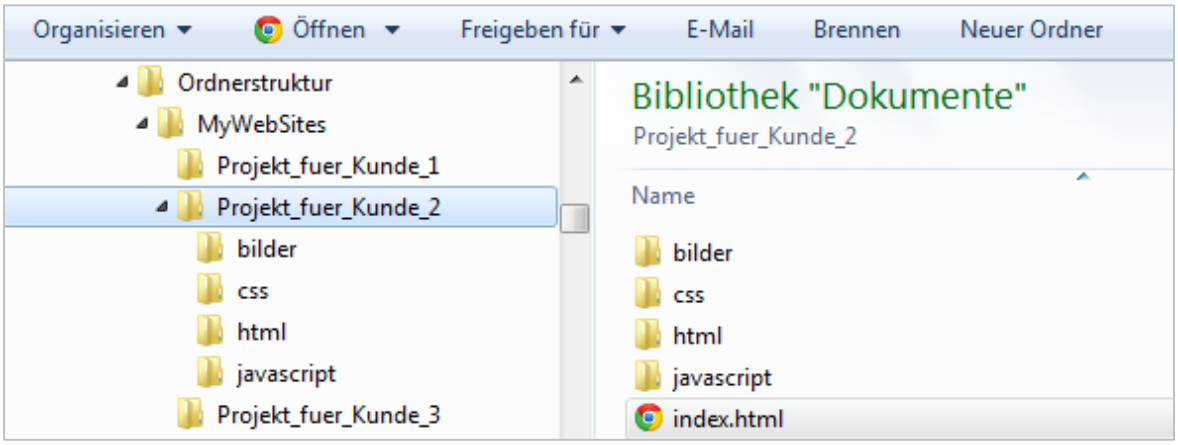


Abbildung 7: Ordnerstruktur für mehrere Webprojekte

Die Ordner können Sie mit einem beliebigen Tool anlegen. Achten Sie auch hier auf die Groß- und Kleinschreibung, vermeiden Sie Umlaute! Verbinden Sie dann den Projektordner mit dem Ordner in Ihrem Webspace, auf den der Domainname zeigt.
Achtung: Speichern Sie in diesen Ordnern nur die Dateien, die Sie später auch veröffentlichen möchten, nicht also zum Beispiel die umfangreichen Ebenen-Bilder aus Ihrem Photoshop.

FTP-Programme

Mit FTP-Programmen laden Sie die Kopie der lokalen Website in den Webspace. Einige Editoren haben bereits eingebaute FTP-Programme, mehr Möglichkeiten bieten jedoch gesonderte FTP-Programme.

- *WINSCP*, Windows
- *FileZilla*, Windows, Mac
- *Cyberduck*, Mac

Zusammenfassung

Auszeichnungssprachen markieren Text und verleihen ihm dadurch die gewünschte formale Bedeutung. XML strukturiert Daten, während HTML Dokumente strukturiert und die Strukturelemente mit Daten füllt. CSS – eigentlich keine Auszeichnungssprache – macht die Information sichtbar. Der Parser des Browsers „fischt“ Auszeichnungs-Elemente aus dem Datenstrom und errichtet die Dokumentstruktur. Die Rendering-Engine macht die Daten innerhalb der Dokumentstruktur sichtbar. Der Javascript-Interpreter führt im Browser Programmbefehle aus. Zum Editieren von Auszeichnungssprachen dienen einfache Bearbeitungswerkzeuge.

HTML und die Dokumentstruktur

Von den Anfängen bis HTML 5

Das „World Wide Web“ ist nicht das Internet, es ist neben E-Mail der vielleicht bekannteste Dienst, der mit Hilfe des Internets existiert. *Tim Berners-Lee* und *Robert Cailliau* erkannten Ende der Achtzigerjahre des zwanzigsten Jahrhunderts den Bedarf, wissenschaftliche Artikel und Forschungsergebnisse über verschiedene Computerwelten hinweg auszutauschen. Beide Wissenschaftler arbeiteten damals am CERN in Genf, der europäischen Organisation für Kernforschung. Auf der Basis bereits existierender, ähnlicher Konzepte entwickelte Berners-Lee das *HTTP-Kommunikationsprotokoll* und die Auszeichnungssprache HTML. Die Entscheidung, auf Patentierung oder Lizenzzahlungen zu verzichten, führte zu der explosionsartig schnellen Verbreitung.

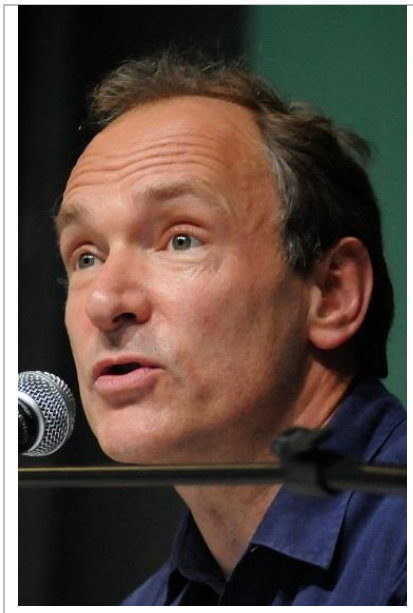


Abbildung 8: Tim Berners-Lee

Bild: Silvio Tanaka (originally posted to Flickr as Tim Berners-Lee) [CC BY 2.0 (http://creativecommons.org/licenses/by/2.0)], via Wikimedia Commons

Abbildung 9: Marc Andreessen

Bild: Brian Solis - IMG_8642. Lizenziert unter CC BY 2.0 über Wikimedia Commons - http://commons.wikimedia.org/wiki/File: Marc_Andreessen.jpg

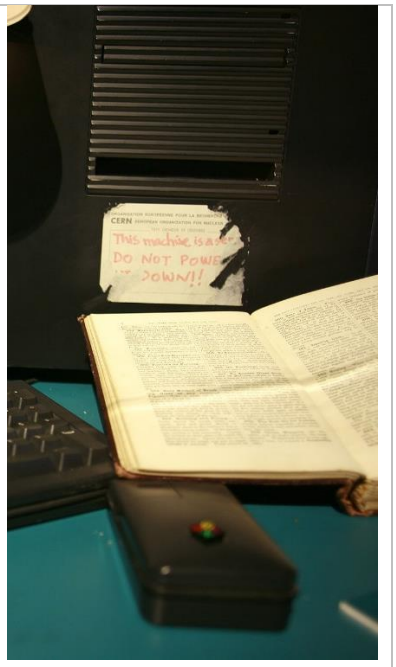


Abbildung 10: Tim Berners-Lee's PC at CERN: “Bitte nicht ausschalten. Das ist ein Server!“

Bild: Robert Scoble from Half Moon Bay, USA - Birthplace of the Web (the computer that Tim Berners-Lee used to invent the World Wide Web). Lizenziert unter CC BY 2.0 über Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Tim_Berners-Lee%27s_computer_at_CERN.jpg

HTML-Zeitleiste

Im Jahr 1989 beschreibt *Tim Berners-Lee* erste Funktionalitäten von HTML. Am 3. November 1992 findet die Präsentation der HTML-Urversion statt. Die Urversion trägt noch keine Versionsnummer, sie kann nur Texte anzeigen.

Marc Andreessen befindet sich im Jahr 1992 zur richtigen Zeit am richtigen Ort, als er von HTML erfährt. Als Student der *Universität von Illinois* ist er am *National Center for Supercomputing Applications* an der Entwicklung des ersten Browsers beteiligt, der aufgrund seiner Usability populär werden kann (*Mosaic-Browser*). Frühere Web-Browser-Applikationen sind für Nicht-Programmierer viel zu kompliziert.

Andreessen verlässt die Hochschule und gründet zusammen mit Kommilitonen die Firma *Netscape*. 1994 erscheint der Netscape Navigator 1.0. Der Browser, programmiert von einem professionellen Team für das Windows-Betriebssystem, kostet 35 Dollar. Ein knappes Jahr später bietet Microsoft den *Internet Explorer* als Zusatzprodukt für *Windows 95* zum Preis von 100 DM an. Noch kann der Internet Explorer keiner Bilder darstellen.
Zwischen 1992 und 1995 entstehen mehrere nicht-nummerierte HTML-Weiterentwicklungen, die Grafiken anzeigen und Schrift fett oder kursiv darstellen. Zwischen 1995 und 1999 kommt es zu einem Wettlauf zwischen HTML-Standard und proprietären HTML-Interpretationen von Netscape und Microsoft. HTML 2.0 ist bei Erscheinen veraltet, ebenso HTML 3.0.
Mit der HTML 3.2-Version vom Januar 1997 erscheint zum ersten Mal eine brauchbare Arbeitsgrundlage für die Standardisierung. Die Syntax von HTML 3.2 ist in einer *DTD* (*Document Type Definition*) beschrieben. Weiterhin ist beschrieben, wie Formulare, Tabellen, Textfluss um Bilder und die Einbindung von Applets realisiert werden müssen.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
```

HTML 4.0 vom 18. Dezember 1997 beschreibt unter anderem das Anlegen und die Einbindung von Stylesheets, und mit der Version 4.01 vom Heiligabend 1999, die zahlreiche kleinere Korrekturen enthält, kommt die Entwicklung von HTML für fast zehn Jahre zum Stillstand. Es gibt mehrere gültige Dokumententyp-Definitionen und somit mehrere "Dialekte" mit unterschiedlichem Sprachumfang:

- HTML 4.01 Strict DTD: Wie HTML 4, jedoch ohne *deprecated* Elemente ("cooles" HTML, weil modern).

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

- HTML 4.01 Transitional DTD Wie HTML 4, jedoch inklusive *deprecated* Elemente.

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

- HTML 4.01 Framset DTD: Wie HTML 4, jedoch inklusive *deprecated* Elemente und Frame-Elemente.

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

Der Dokumenttyp (Doctype)

Der Dokumenttyp, der vor dem Beginn des HTML-Codes notiert wird, verweist auf die gültigen HTML-Elemente und die Regeln, wie diese Elemente innerhalb des Dokuments miteinander in Beziehung stehen. Praktisch wirkt die Angabe des Dokumenttyps wie ein Stichwortgeber, der dem HTML-Parser einen „Tipp“ gibt, nach welchen Regeln der ankommende Datenstrom zu analysieren und auszuwerten ist. Dokumente, die nicht den Regeln entsprechen, sind invalid.

XHTML

Parallel zur Veröffentlichung der HTML 4.01-Recommendation arbeitet das W3C bereits an einer Neuformulierung von HTML. Der während der rasanten Entwicklungsjahre entstandene Wildwuchs von HTML-Elementen und Attributen sowie deren Visualisierungskonzepte in den Browsern soll mit einer vollständigen Neuformulierung auf das Wesentliche zurückgeführt und vereinheitlicht werden. Das Mittel der Wahl, um HTML zu beschreiben ist XML. Das neue HTML heißt demnach *XHTML*. Bereits am 26. Januar 2000 erscheint die Recommendation *XHTML 1.0*.

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

XHTML kann auch von XML-Parser interpretiert werden. Ein Merkmal von XHTML ist der Slash in leeren XHTML-Elementen. Das leere HTML-Element `<br>` (Zeilenumbruch ohne Absatz) wird in XHTML nun als `<br />` notiert. Dasselbe gilt für das `<meta />`-Element und alle anderen Elemente, die keinen Inhalt besitzen, sondern nur ohne oder mit Attributen notiert werden.

```
<img src=“bild-10.jpg“ width=“400“ height=“300“ alt=“Bildbeschreibung“ title=“
Bildbeschreibung“ />
```

Mit XHTML bürgert sich die Kleinschreibung der Elementnamen ein.

HTML 5

XHMTL gilt lange Zeit als zukunftsweisend, jedoch gerät die Weiterentwicklung ins Stocken. Es wird deutlich, dass die Anwender hernach erheblich mehr Hintergrundwissen benötigen würden, um Dokumente zu erstellen und zu editieren. Deshalb kommt es zu Bestrebungen, HTML 4 weiterzuentwickeln. Im Sommer 2009 wird die Arbeit an XHTML 2.0 zugunsten von HTML5 eingestellt. Bereits im Jahr 2010 unterstützen zahlreiche Browser einige Teile des geplanten HTML5-Vokabulars. Vor allem die Browser in mobilen Endgeräten wie Tablets und Smartphones machen sich die neuen und einfachen Möglichkeiten von HTML5 zunutze. Trotzdem dauert es noch bis zum 28. Oktober 2014, bis HTML5 als Recommendation freigegeben ist. Heute ist HTML5 das Mittel der Wahl, um *responsives Webdesign* zu realisieren.

HTML 5 stellt auf der Basis von XHTML 1.0 und HTML 4.01 ein zusätzliches Vokabular zur Verfügung. Mehrere HTML-Elemente wurden abgeschafft. (Sie sind *deprecated*.) Gleichzeitig wurde die DOM-Spezifikation überarbeitet.

Standardisierungsbestrebungen

Proprietäre Lösungen und Zersplitterung

Die Schlacht um Marktanteile zwischen *Netscape* und *Microsoft* führte dazu, dass beide Unternehmen eigene HTML-Elemente entwickelten, die nur von den eigenen Browsern verstanden wurden. Bis 1996 ist der *Netscape-Browser* dem *Microsoft-Browser* technisch überlegen. Mit der Version von Windows 95B integriert Microsoft den Internet Explorer fest ins Betriebssystem und verschafft sich durch diese Machtstrategie einen Wettbewerbsvorteil. Der Marktanteil von Netscape stürzt ab, die Netscape-Browser werden fehlerhaft. 1998 gibt Netscape den Browser-Code als *Open Source* frei und ruft die *Mozilla*-Initiative ins Leben. Es ist der einzige Weg, um ein Konkurrenzprodukt zum Internet Explorer am Leben zu erhalten. In Firefox und anderen Mozilla basierten Browsern lebt Netscape weiter - wenn auch inzwischen nur noch rudimentär.

Das W3C (World Wide Web Consortium)

1994 gründet Tim Berners-Lee das *World Wide Web Consortium* (*W3C*). Sein Ziel ist es, einen einheitlichen Standard durchzusetzen, proprietäre Lösungen einzudämmen und unterschiedliche HTML-Sprachen zu vermeiden.

Im Januar 2015 besteht das W3C aus 403 Mitgliedern. Alle namhaften IT-Unternehmen und rund 30 Universitäten sind vertreten. Trotzdem ist das W3C keine zwischenstaatlich anerkannte Organisation und kann keine offiziellen Standards setzen. Die Arbeitsergebnisse heißen denn auch *Recommendations*, Empfehlungen – sie sind aber quasi Normen.

Neben HTML gibt das W3C auch Empfehlungen für andere Sprachen heraus. Beispiele sind XHTML, XML, CSS, SVG und andere. Es werden nur Lösungen unterstützt, die frei von Patengebühren sind.

Grundgerüst eines HTML-Dokuments

Jedes HTML-Dokument kann auf einem Grundgerüst aufgebaut werden. Am besten ist es, sich ein eigenes, valides Grundgerüst zu erstellen und an einem leicht zugänglichen Speicherort zu lagern, um stets darauf zugreifen zu können.
Jedes gültige HTML-Dokument beginnt und endet mit dem *html*-Element. Der Kopfbereich, alles was zwischen dem öffnenden und schließenden *head*–Element liegt, enthält Angaben über das Dokument: Welcher Titel? Welche Codierung? Und weitere Angaben,
Im Browserfenster werden nur jene Elemente angezeigt, die sich zwischen dem öffnenden und dem schließenden *body*–Element befinden.
Das minimale Grundgerüst sollte zu einem erweiterten Grundgerüst vervollständigt werden, das alle Elemente enthält, die häufig und gerne benutzt werden.

Minimales Grundgerüst

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8" />
    <title>HTML 5 Dokument</title>
 </head>
 <body>
   <h1>Überschrift</h1>
   <p>Fließtext des Dokuments.</p>
 </body>
</html>
```

Ein erweitertes Grundgerüst

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8" />
   <title>HTML 5 Dokument</title>
   <!–Verweis auf externes Stylesheet im gleichrangigen Ordner "css" -->
    <link href="../css/stil.css" rel="stylesheet" type="text/css">
   <!–Verweis auf ein Javascript im gleichrangigen Ordner "js" -->
    <script type="text/javascript" src="../js/skript.js"></script>
 </head>
 <body>
   <h1>Überschrift</h1>
   <p>Fließtext des Dokuments.</p>
 </body>
</html>
```

Dokumentenkopf für XML-Parser

HTML5 kann auch mit einem XML-Parser gelesen werden. Dazu wird im Dokumentenkopf der Namensraum bekannt gegeben. Das HTML-Parser ignoriert die Namensraumangabe.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"> <!-- Namensraum -->
 <head>
… weiter wie im HTML-Grundgerüst.
```

Professionelles Betrachten von Websites

Von einer *Website* ist dann die Rede, wenn mehrere HTML-Dokumente desselben Erscheinungsbildes miteinander verwoben sind. Neben HTML-Dokumenten enthält die Website möglicherweise Bilder, Grafiken, Illustrationen, Videos, Animationen, PDF-Dokumente, clientseitig ausführbare Programme (Javascript) und Format-Dateien, die das gleichbleibende Erscheinungsbild gewährleisten. Fünf Fragen helfen dabei, Websites unter professionellen Gesichtspunkten zu betrachten:

- Welche HTML-Elemente sind erkennbar?
  Beispiel: Text, Überschriften, Blöcke, Spalten, Bilder, audiovisuelle Elemente, Titel
- Wie verhält sich die Seite beim Verändern des Browserfensters?
  Beispiel: Seite verkleinern/vergrößern (Strg+-, Strg++), auf- und zuschieben?
- Ist die Seite valide?
  Wenn nein, warum nicht?
- Wie verhält sich die Seite bei verschiedenen Browsern?
- Wie verhält sich die Seite bei verschiedenen mobilen Endgeräten?

Wenden Sie die Fragen auf drei Webseiten-Beispiele Ihrer Wahl an.

Elemente, Attribute, Events

HTML-Elemente

HTML-Elemente strukturieren Text, das heißt: sie erzeugen den Bauplan des Dokumentes, das mit Hilfe von CSS-Stilregeln in der gewünschten Weise sichtbar gemacht wird.

```
<h2>Dieser Text ist eine Überschrift zweiten Grades (heading). </h2>
<p>Dieser Text ist ein Absatz (paragraph). </p>
```

HTML-Elemente sind durch spitze Klammern gekennzeichnet. Schließende Elemente enthalten zusätzlich den Slash.

Notierung der HTML-Elemente

Start-Tags, Auszeichnungen für den Element-Anfang

```
Beispiel: <body>
```

Ende-Tags, Auszeichnung für das Element-Ende

```
Beispiel: </body>
```

Manche Elemente müssen unter bestimmten Bedingungen nicht geschlossen werden (W3C:2013, 8.1.2.4 Optional tags). Dies spart bei großen Dokumenten Traffic.

Elementnamen als Content

Um Elementnamen in HTML zu schreiben, müssen auf Quelltext-Ebene Entitys verwendet werden.
< = :<
> = >

```
Ansicht im Browserfenster: <body><p>Fließtext</p></body>
Ansicht im Quellcode: &lt;body&gt;&lt;p&gt;Fließtext&lt;/p&gt;&lt;/body&gt;
```

Leere Tags

Aus Kompatibilitätsgründen können leere Tags in der XHTML-Schreibweise verwendet werden.

```
Beispiel: <br />, <input />
```

Leere Elemente (Tags) umschließen keinen Text.

Elemente mit Attribut

HTML-Elemente können oder müssen mit Attributen näher beschrieben werden. So ergibt das *img*–Element nur dann einen Sinn, wenn das Attribut die Bildquelle nennt.

```
<img src="bild.jpg">
```

Andere HTML-Elemente funktionieren auch ohne Attribut. "Überschrift" wird auch ohne style-Attribut angezeigt – jedoch nicht in Rot.

```
<h1 style="color:red">Überschrift</h1>
```

Style-Attribute zur Textauszeichnung sollten jedoch im Sinne der Trennung von Format und Inhalt vermieden werden. Besser ist es, das Erscheinungsbild in eine Stilregel zu packen und diese als Attribut „class“ dem Element zuzuweisen. Die Klasse „stil-1“ kann dann darüber hinaus auch anderen Elementen als Attribut dienen.

```
<h1 class="stil-1">Diese Überschrift gehört zur Klasse stil-1, in der eine oder
mehrere Stilregeln definiert sind. </h1>
```

Die Klasse „stil-1“ enthält in diesem Beispiel die Regel, das <h1>–Element in der Schriftart „Verdana“, der Schriftgröße „24px“ und in der Farbe „blau“ darzustellen.

```
.stil-1 {
      font-family: Verdana;
      font-size:24px;
      color:blue;
}
```

Notierung der Attribute

Einfache Werte:

```
<input value=yes>
```

Werte mit Leerzeichen stehen in doppelten oder einfachen Begrenzungszeichen.

```
<img alt="Das Wasser ist blau." >
<img alt='Das Wasser ist blau.' >
```

Falsch ist diese Schachtelung:

```
 <a title="Attributwerte können in "Anführungszeichen" stehen." >
```

Richtig wäre:

```
<a title="Attributwerte können in 'Anführungszeichen' stehen." >
```

Globale Attribute

Globale Attribute können auf jedes HTML-Element angewendet werden.

Attribut	**Funktion**
id	ID-Selektor (Eindeutigkeit)
title	Beschreibung des Elements (erscheint als "Fähnchen")
lang	Sprache des Elements
translate	Legt fest, ob bei einer Änderung der Schreibrichtung das Element verändert wird.
dir	Schreib-/Leserichtung von Text

class	Klassen-Selektor
style	Stil-Attribut
data-*s	Custom Data Attribut (frei definierbares Attribut)

Tabelle 1: Globale Attribute für HTML-Elemente

Events und Event-Attribute

Für viele HTML-Elemente kann festgelegt werden, was bei bestimmten Aktionen geschehen soll, zum Beispiel beim Anklicken mit der Maus. Die Elemente verfügen gewissermaßen über eine eingebaute Aufmerksamkeit hinsichtlich bestimmter Aktionen. Dies ist vor allem für die Programmierung von Interaktionen mit JavaScript von erheblicher Bedeutung.

Beim *Anklicken* des *h1*-Elements mit der Maus wird eine Javascript-Funktion ausgeführt.

```
<h1 onclick="javascript:alert('Hallo Welt!')" >Überschrift </h1>
```

Beim *Laden* des Dokuments wird der Name des Benutzers abgefragt.

```
<body onload="javascript:prompt('Geben Sie bitte Ihren Namen ein', '') " >
```

Element-Arten

„Einmalige“ HTML-Elemente

Genau drei Elemente kommen in einem validen HTML-Dokument nur ein einziges Mal vor.
Das Wurzel-Element *html* umschließt das Dokument, das aus Kopfbereich *head* und Körper *body* besteht. Das Beispiel definiert ein deutschsprachiges HTML-Dokument.

```
<!DOCTYPE html>
<html lang="de">
   <head>
      <meta charset="utf-8" >
     <title>Die Biene</title>
   </head>
   <body>
     <h1>Die Biene</h1>
     <p>Die Biene ist ein Insekt, das Nektar sammelt.</p>
   </body>
</html>
```

Element	Kurzbeschreibung
<html>	Kennzeichnung eines Dokuments als HTML-Dokument
<head>	Kennzeichnung des Kopfbereichs des Dokument, der nicht sichtbaren Metadaten-Inhalt enthält
<body>	Kennzeichnung des Textkörpers, der den „fließenden“ Inhalt enthält

Tabelle 2: Dokument, Dokumentenkopf und Textkörper

➡ **Siehe Länderkürzel.**

Innerhalb des *html* -Elements dürfen alle Arten von HTML-Elementen vorkommen. Das *head*–Element erwartet *Metadaten*-Inhalt, während das *body*–Element auf alle Arten von *Flow-Content* vorbereitet ist.

Block-Elemente

Block-Elemente bilden eigene Blöcke. Sie nehmen in der Breite den maximal zur Verfügung stehenden Raum ein. In der Höhe orientieren sie sich am Inhalt. Block-Elemente können Inline-Elemente umschließen.

Element	**Kurzbeschreibung**
<p>	Element für einen Absatz (Fließtext).
<hr>	Thematischer Break durch eine Linie.
<pre>	Im HTML-Code vorformatierter Text
<blockquote>	Einzüge, Einrückungen
	Nummerierte Liste (nur sinnvoll, wenn es -Elemente umschließt)
	Unnummerierte Liste (nur sinnvoll, wenn es -Elemente umschließt)
	Listenelement für geordnete und ungeordnete Listen
<dl>	Definitionsliste (nur sinnvoll, wenn es <dt>- und <dd>-Elemente umschließt)
<dt>	Eigenschaft der Beschreibungsliste, Definitionsliste ...
<dd>	... gefolgt von einem Wert
<figure>	Erläuterung, Verdeutlichung, zum Beispiel eine Illustration
<figcaption>	Beschreibung der Illustration
	Bedeutungsfreies Struktur-Element

Tabelle 3: Ausgewählte Block-Elemente

Inline-Elemente

Inline-Elemente nehmen nur den Raum ihres Inhalts ein. Sie können innerhalb von Block-Elementen vorkommen, jedoch nicht außerhalb.

Element	**Kurzbeschreibung**
<a>	Link, Verweis
	betont, emphatisch
	stark, wichtig
<small>	kleiner
<s>	veralteter Inhalt
<cite>	Zitieren eines Begriffs, Titels, Person
<q>	Zitat, Quelle
<dfn>	Definition
<abbr>	Abkürzung
<time>	Maschinenlesbarer Zeit-String in einem Zeit/Datumsformat
<code>	Codebeispiel
<var>	Variable, zum Beispiel in mathematischen Dokumenten
<samp>	Beispiel

<kbd>	Erwartete Texteingabe
<sub und sup>	höher/tiefer gestellt
<i>	kursiv
<b>	physisch: fett
<u>	physisch: unterstrichen
<mark>	physisch: Hervorhebung (visuell)
<ruby>, <rt>, <rp>	Asiatische Betonungszeichen
<bdi>	bidirektionales Textformat
<bdo>	Leserichtung in Zusammenhang mit dem Attribut "dir"
	bedeutungsfreies Markierungs-Element, Inline-Pendant zu
 	Zeilenumbruch ohne Absatz
<wbr>	Bedingter Zeilenumbruch

Tabelle 4: Ausgewählte Inline-Elemente

Content Modelle

Neben der Unterscheidung in Block- und Inline-Elemente, die auf HTML4 zurückgeht, ist es in HTML5 sinnvoll, eine differenziertere Unterscheidung nach Funktion, zu treffen (Siehe auch *W3C*, HTML5: 3.2.4 Content models). Jedes HTML-Element erwartet eine bestimmte Art von Inhalt. Elemente, die den gleichen Inhalt oder Teilmengen desselben Inhalts erwarten, können zu Kategorien zusammengefasst werden. Daraus ergibt sich, dass jedes HTML-Element zu keiner, zu einer oder zu mehreren Kategorien gehören kann.

Fließender Inhalt (Flow Content)

Der fließende Inhalt umfasst alle Block- und Inline-Elemente. Damit ist gemeint, dass der Inhalt wie „Flüssigkeit“ fließt und sich den Gegebenheiten des Browserfensters anpasst.

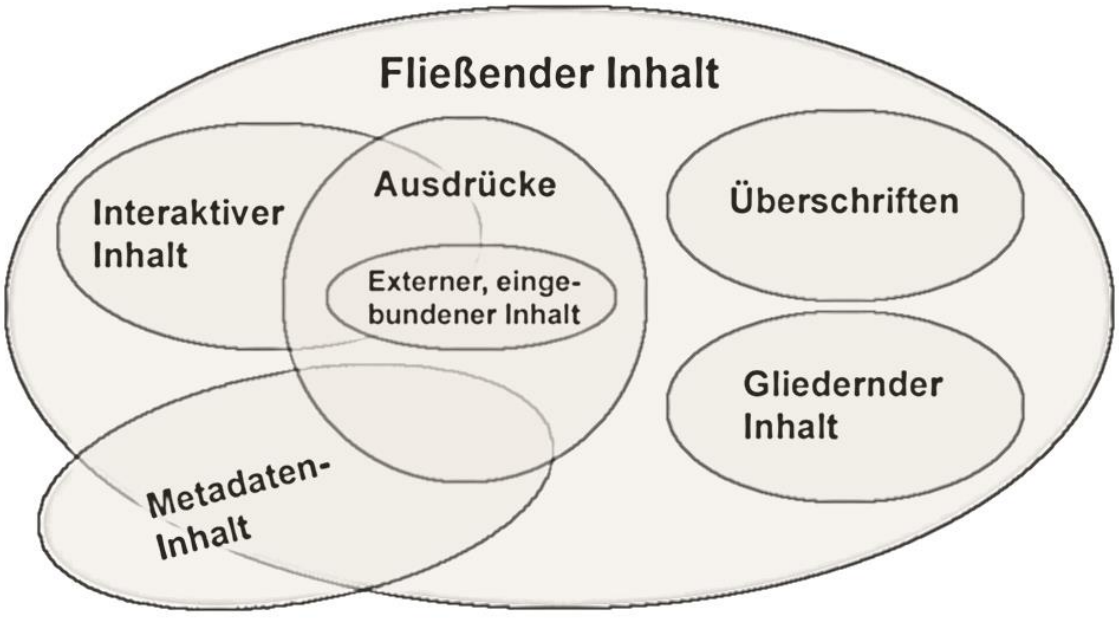


Abbildung 11: Arten von HTML-Inhalt

Die Elementreihenfolge im Browserfenster entspricht der des Quelltextes. Mit Hilfe von Stilregeln kann zwar das Erscheinungsbild im Browserfenster verändert werden, zum Beispiel so, dass im Quelltext später folgende HTML-Elemente im Browserfenster an den Anfang rücken. Der Parser liest jedoch alle Elemente der Reihe nach ihres Vorkommens im Quelltext.
Nahezu alle Elemente, die im Body-Bereich vorkommen, gehören zum fließenden Inhalt.

```
a, abbr, address, area (sofern Abkömmling des map-Elements), article, aside, audio,
b, bdi, bdo, blockquote, br, button, canvas, cite, code, data, datalist, del, dfn,
div, dl, em, embed, fieldset, figure, footer, form, h1, h2, h3, h4, h5, h6, header,
hr, i, iframe, img, input, ins, kbd, keygen, label, main, map, mark, math, meter,
nav, noscript, object, ol, output, p, pre, progress, q, ruby, s, samp, script,
section, select, small, span, strong, sub, sup, svg, table, template, textarea,
time, u, ul, var, video, wbr
```

Es lohnt sich, die sehr allgemeinen Block- und Inline-Elemente anhand ihrer Funktion und dessen, was sie als Inhalt erwarten, weiter zu kategorisieren.

Gliedernder Inhalt (Sectioning Content)

Gliedernder Inhalt beschreibt den Gültigkeitsbereich von HTML-Elementen. Er ist eine Teilmenge des fließenden Inhalts und kann weitere gliedernde Elemente enthalten.
Das HTML-Dokument kann innerhalb des Bodys mehrere Artikel (*article*) enthalten, die in Abschnitte (*section*) gegliedert sind. Den Artikeln und Abschnitten können Zusatzinformationen (*aside*) mitgegeben werden. Jedes HTML-Dokument verfügt über eine oder über mehrere Navigationsleisten (*nav*).

Element	**Kurzbeschreibung**
<article>	Selbständige inhaltliche Einheit, zum Beispiel ein Blog-Post
<section>	Abschnitt innerhalb eines Artikels
<nav>	Navigationsbereich
<aside>	Anmerkungen zum Hauptinhalt, häufig für Werbung verwendet

Tabelle 5: Gliedernder Inhalt

Überschriften (Heading Content)

Überschriften, hier nach Wichtigkeit der Bedeutung geordnet, leiten den Inhalt von Abschnitten ein.

```
h1, h2, h3, h4, h5, h6
```

Die Überschriften können zusammen mit anderen Elementen, zum Beispiel dem Firmenlogo, zum Kopfbereich eines Abschnitts gruppiert werden. Ebenso empfiehlt es sich, einen Fußbereich anzulegen und dort Adressen und Verweise wie Copyright, Privacy oder AGB unterzubringen

Element	**Kurzbeschreibung**
<header>	Kopfbereich eines Abschnitts
<footer>	Fußbereich eines Abschnitts
<address>	Kennzeichnung von Kontakt- und Rechtsinformationen

Tabelle 6: Kopfbereiche und Fußbereiche

Ausdrücke (Phrasing Content)

“Ausdrücke” meint Inhalt, der innerhalb von p-Elementen vorkommen kann, also meist das typische Inline-Element. Ausdrücke können weder Überschriften, noch Gliederungselemente enthalten. “Ausdrücke” sind eine Teilmenge des fließenden Inhalts.

```
a, abbr, area, (sofern Abkömmling des map-Elements), audio, b, bdi, bdo, br,
button, canvas, cite, code, data, datalist, del, dfn, em, embed, i, iframe, img,
input, ins, kbd, keygen, label, map, mark, math, meter, noscript, object, output,
progress, q, ruby, s, samp, script, select, small, span, strong, sub, sup, svg,
template, textarea, time, u, var, video, wbr
```

Externer, eingebundener Inhalt (Embedded Content)

Externer, eingebundener Inhalt importiert Inhalt aus anderen Ressourcen in das HTML-Dokument.

```
audio, canvas, embed, iframe, img, math, object, svg, video
```

Mehr zum Gebrauch von Bildern in HTML-Dokumenten finden Sie unter dem Abschnitt "Embedded Content".

Interaktiver Inhalt (Interactive Content)

Die Elemente repräsentieren Inhalt, der Benutzereingaben und Benutzer-Interkationen erwartet.

```
a, audio (mit controls-Attribut), button, embed, iframe, img, (mit usemap-
Attribut), input (sofern nicht “hidden”) keygen, label, object, (mit usemap-
Attribut), select, textarea, video (mit controls-Attribut)
```

Mehr zum Gebrauch von interaktivem Inhalt finden Sie unter den Abschnitten "Embedded Content" und „Arbeiten mit Daten“.

Metadaten-Inhalt (Metadata Content)

Die Elemente im Kopfbereich, dem *head*–Abschnitt eines HTML-Dokuments werden im Browserfenster nicht angezeigt. Sie enthalten Angaben *über* das Dokument (Meta-Daten), also über die im *body* vorkommenden Elemente.

```
head, meta, base, title, link.
```

Beispiel für Metadata Content im Kopfbereich

```
<head>
   <meta charset="utf-8">
   <base href="http://www.domainname.de/">
   <title>Ein Beispiel</title>
   <link id="standard_css" rel="stylesheet" href="stil-1.css">
   <link rel="stylesheet alternate" href="stil-2.css" title="Ohne Brille">
   <script src="switcher.js" type=” text/javascript”></script>
   <meta name="author" content="Harry Hirsch">
</head>
```

Scripting-Elemente

Das Scripting-Element kann über einen MIME-Type verfügen. Der MIME-Type gibt an, wie mit dem Nicht-HTML-Inhalt des Elements zu verfahren ist. Es gibt MIME-Types, die auf Inhalte verweisen, die zu interpretieren sind (ausgewählte Beispiele).

```
"application/ecmascript"
"application/javascript"
```

```
"text/jscript"
"text/javascript"
```

Die folgende Zeile startet den Javascript-Interpreter des Browsers. Der Interpreter führt die Programmbefehle aus.

```
<script src="switcher.js" type=” text/javascript”></script>
```

Andere MIMETypes verweisen auf Inhalte, die NICHT zu interpretieren, sondern zu parsen sind.

```
"text/plain"
"text/xml"
"text/css"
```

Die folgende Zeile filtert die CSS-Direktiven aus dem Datenstrom und wendet sie auf die HTML-Elemente an.

```
<link rel="stylesheet" href="stil-1.css" type=”text/css”>
```

Regeln für das Umschließen von Elementen

Viele Elemente können nicht beliebig verschachtelt werden. Sie erlauben nur bestimmte andere Elemente innerhalb ihres Geltungsbereichs. Grundsätzlich gilt, dass Block-Elemente zwar Inline-Elemente umschließen können, nicht jedoch umgekehrt. Bei Block-Elementen ist der Sachverhalt schwieriger. Manche Block-Elemente können andere Block-Elemente umschließen, andere jedoch nicht. Die folgenden Faustregeln gelten für den Body-Bereich eines HTML-Dokuments.

Regel 1

Block-Elemente können Inline-Elemente umschließen.

Richtig

```
<p>Dieses Wort ist <strong>wichtig</strong>.</p>
```

Regel 2

Inline-Elemente können niemals Block-Elemente umschließen.

Falsch

```
<strong><p>Dieses Wort ist <strong>wichtig</p>.</strong >
```

Regel 3

Überschriften (Heading Content) können nur Inline-Elemente umschließen.

Richtig

```
<h2>Dieses Wort ist <i>kursiv (italic)</i>.</h2>
```

Regel 4

Gliederungselemente und das `<div>`-Element können andere Block-Elemente und Inline-Elemente umschließen sowie gleiche und andere Gliederungselemente.

Regel 5

Inline-Elemente können Inline-Elemente umschließen.

```
<strong>Dieses Wort ist <i>wichtig</i></strong >
```

Regel 6

Bei allen Verschachtelungen ist darauf zu achten, dass sich die Elemente nicht überkreuzen.

Falsch

```
<p>Dieses Wort ist <strong>wichtig.</p></strong>
```

Semantisches HTML

HTML- Elemente sollen gemäß der Bedeutung verwendet werden, mit der sie Text auszeichnen. Sie sind nicht dafür gedacht, willkürlich angewendet zu werden, vielmehr geht es um die ihnen zugrundeliegenden Bedeutungen. So ist eine h1-Überschrift nur dann anzuwenden, wenn sie tatsächlich wichtigen Überschrift-Text markiert. Das Erscheinungsbild ist bei der Entscheidung für ein Element unwichtig. Das Aussehen wird mit Hilfe von CSS festgelegt. Die Semantik ist vor allem deswegen von großer Bedeutung, weil sich die Indexer von Suchmaschinen an den Bedeutungen orientieren, die dem Text durch HTML-Elemente zugewiesen sind. Anhand der Bedeutungen prüfen sie die Plausibilität von HTML-Dokumenten und ordnen sie in die Datenbank ein.

Die folgenden Beispiele sind syntaktisch richtig und valide. Trotzdem handelt es sich um drei negative Beispiele.

```
<h1>Dieser Text ist unwichtig. </h1>
```

H1-Überschriften werden vorrangig behandelt und dienen der richtigen Indexierung. Also mit wichtigem Inhalt füllen.

```
<article>Hier klicken! </article>
```

Die Anweisung ist kein Artikel. Sie ist auch für den Inhalt von geringer Bedeutung. Am besten mit bedeutungslosem Fettdruck auszeichnen.

```
<b>Hier klicken! </b>
```

Der folgende Text erscheint zwar fett, aber für die Suchmaschinen ist der Fettdruck ohne Bedeutung.

```
<b> Im Falles eines Brandes den Feuermelder zu benutzen. </b>
```

Mit dem *strong*-Element erfährt die Suchmaschine dagegen, dass es sich bei dem Satz um etwas Wichtiges handelt.

```
<strong>Im Falles eines Brandes den Feuermelder zu benutzen. </strong>
```

HTML und sein Erscheinungsbild

HTML-Elemente und ihr Inhalt sind zunächst nichts anderes als ein Datenstrom, den der Parser analysiert, um daraus das Modell des Dokuments zu bauen. Ein HTML-Dokument würde nach dem Parsen in folgender oder ähnlicher Weise erscheinen:

```
Die Biene Die Biene ist ein Insekt, das Nektar sammelt. Umgangssprachlich wird der
Begriff Biene meist auf eine einzelne Art, die Westliche Honigbiene (Apis
mellifera) reduziert, die wegen ihrer Bedeutung als staatenbildender
Honigproduzent, aber auch wegen ihrer Wehrhaftigkeit besondere Aufmerksamkeit
erfährt. Dabei handelt es sich bei den Bienen um eine recht große Gruppe mit sehr
unterschiedlichen Arten. Viele davon, vor allem die solitär lebenden, werden unter
dem Begriff Wildbienen zusammengefasst. Aus Wikipedia:
http://de.wikipedia.org/wiki/Bienen und http://de.wikipedia.org/wiki/Honig
```

Das tatsächliche Erscheinungsbild sieht jedoch so aus:

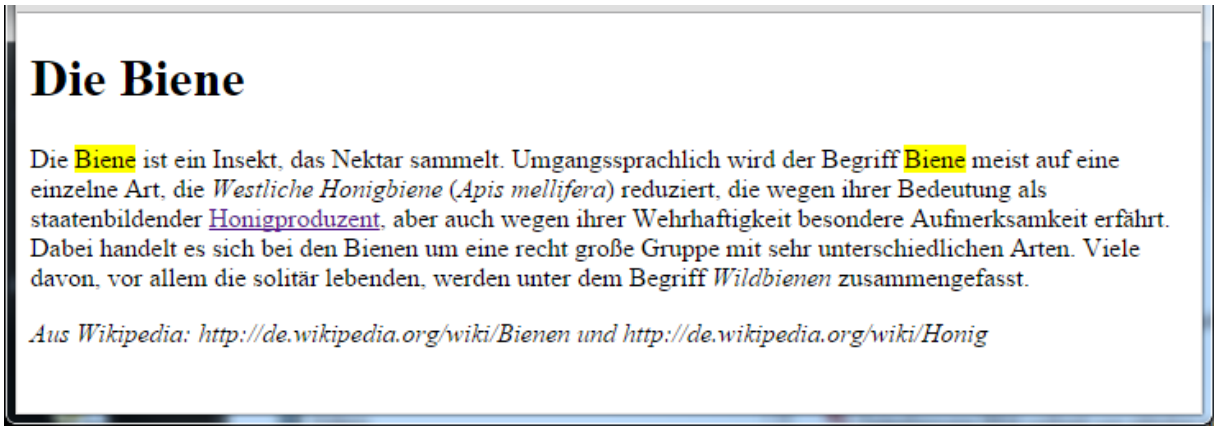


Abbildung 12: Erscheinungsbild der "Biene"-Seite

Browser-Stylesheet

Das Aussehen rührt daher, dass in jedem Browser ein *Stylesheet* fest eingebaut ist. Es legt das Erscheinungsbild der HTML-Elemente fest. Bei Firefox-Browsern finden Sie – je nach Version – das eingebaute Stylesheet, wenn Sie folgende Adresszeile eingeben:

```
resource:///res/html.css (ältere Firefox-Browser)
resource://gre-resources/html.css (neuere Firefox-Browser)
```

In dem rund 750 Zeilen langen Dokument finden Sie beispielsweise die Stilregel, wie *h1*-Überschriften im Browserfenster dargestellt werden (ungefähr bei Zeile 160):

```
h1 {
  display: block;
  font-size: 2em;
  font-weight: bold;
  margin: .67em 0;
}
```

Benutzer-Stylesheets

Die Stilregeln des Browser-Stylesheets werden überschrieben, wenn ein Benutzer-Stylesheet angefügt wird und in diesem andere Werte zu denselben Eigenschaften notiert sind. Spätere Wertzuweisungen überschreiben frühere. So ermöglichen eines oder mehrere Benutzer-Stylesheets, das individuelle Erscheinungsbild eines HTML-Dokuments festzulegen.
Benutzer-Stylesheets werden entweder extern oder im Kopfbereich eines HTML-Dokuments notiert. Die Einbindung erfolgt stets vor eventuell verwendeten Javascript-Dateien.

Einbindung im Kopfbereich

```
<head>
<!–Andere Elemente des Kopfbereichs -->
<style type="text/css">
h1 {
        display: block;
        font-size: 2em;
        font-weight: bold;
        margin: .67em 0;
      }
</style>
<!–Andere Elemente des Kopfbereichs -->
</head>
```

Externe Notierung

```
<head>
<!–Andere Elemente des Kopfbereichs -->
<link href="Dateiname.css" rel="stylesheet" type="text/css">
<!–Andere Elemente des Kopfbereichs -->
</head>
```

Die externe Notierung ist vorzuziehen, da das Stylesheet auf mehrere HTML-Dokumente angewendet werden kann.

Zusammenfassung

Die Aufgabe von HTML ist es, Text-Dokumente zu strukturieren, externe Medien einzubinden und Benutzerinteraktionen zuzulassen. Mit Hilfe von einem oder mehreren Stylesheets erhält das HTML-Dokument sein Aussehen. Die HTML-Elemente werden je nach ihrer Funktion unterschiedlichen Kategorien zugeordnet. Daraus wird ersichtlich, welche Art von Inhalt die Elemente erwarten und wie HTML-Elemente verschachtelt werden können. HTML-Elemente sollen nicht willkürlich zur Textstrukturierung angewendet werden, sondern gemäß ihrer semantischen Bestimmung. Dies ist für die zielgerichtete Auffindbarkeit von Inhalten im weltweiten Netz von erheblicher Bedeutung.

Modelle der Visualisierung

Das DOM und das Zusammenspiel von HTML und CSS

Bereits in vorangegangenen Abschnitten wurde betont, dass der HTML-Parser den Datenstrom im Hinblick auf HTML-Elemente analysiert und daraus die Seitenstruktur erstellt. Beim Zusammenbau der Seitenstruktur orientiert sich der Parser am *Document Object Model (DOM)*. Es beschreibt den logischen Aufbau des HTML-Dokuments, also wie die Elemente und deren Attribute zueinander in Beziehung stehen. Die Darstellung des DOM als Baumstruktur ermöglicht es, die Beziehungen auf einfache Weise zu erkennen.

➡ **Technical Report W3C** http://www.w3.org/DOM/DOMTR

Knoten, Kinder, Eltern und Geschwister

Die Baumstruktur kann man sich aus Knoten aufgebaut vorstellen. Knoten sind jene Stellen, an denen sich die Elemente befinden. Als *Textknoten* werden die Inhalte bezeichnet, zum Beispiel „Mord im Zug“ und „Augenzeugen berichten“.

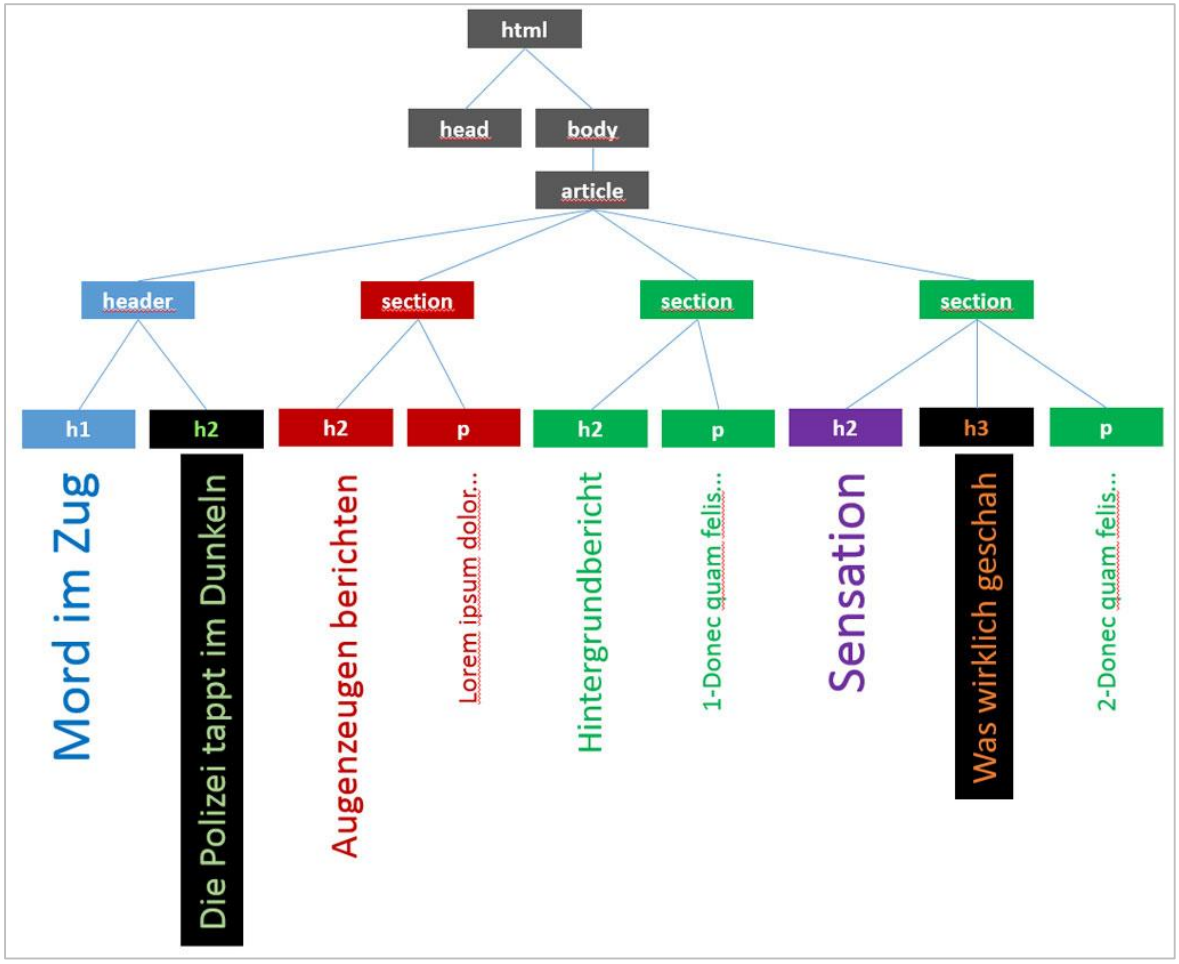


Abbildung 13: Beispiel für eine Baumstruktur, parents, children, siblings

Jedes HTML-Dokument beginnt mit dem *Wurzel-Element*, dem *root-Element*. Das *root-Element* heißt stets „html“, es steht ganz oben am Anfang. Ihm folgen die beiden *Kind-Elemente* (*child-Elemente*) *head* und *body*. Umgekehrt ist „html“ das Eltern-Element (*parent-Element*) von *head* und *body*. Untereinander sind *head* und *body Geschwister-Elemente* (*siblings*).
Das Kind-Element des *body* ist *article*. Dem *article* folgen als *children* die Elemente *header* und die

section-Elemente. *Header* und *section* sind – bezogen auf sich selbst – *siblings*. Auf diese Weise lässt sich das gesamte HTML-Dokument als hierarchische Sammlung von Eltern-, Kind,- und Geschwister-Elementen beschreiben.
Die Hierarchie der Elemente zu verstehen, ist deshalb von Bedeutung, weil die untergeordneten Elemente die Eigenschaften der Eltern-Elemente erben können. Bei manchen Eigenschaften ist die automatische Vererbung der Standard. Dagegen werden andere Eigenschaften nur auf besondere Anforderung vererbt. Welche Eigenschaften automatisch vererbt werden, hängt von der Sinnhaftigkeit der Vererbung ab.

Vererbung und Nicht-Vererbung

Die W3C-Recommendations geben Auskunft, welche Eigenschaften als Default vererbt oder nicht vererbt werden.

Vererbbar als Default: Color-Eigenschaft (property)

Original: Property-		**Deutsch: Eigenschafts-**	
-Name	color	-Name	color
-Value	<color> | inherit	-Wert	Farbwert oder “erben”
Initial	depends on user agent	Anfang-swert	Kontextabhängig
Applies to	all elements	Anwendbar	Alle Elemente
Inherited	yes	Vererbbar	ja

➡ **W3C Color-Modul: http://www.w3.org/TR/css3-color/#foreground**

Nicht-vererbbar als Default: Border-Eigenschaft (property)

Original: Property-		**Deutsch: Eigenschafts-**	
-Name	border	-Name	border
-Value color	<color> | inherit	-Wert Farbe	Farbwert oder “erben”
-Value style	<style> | inherit	-Wert Stil	Stilwert oder “erben”
-Value width	<width> | inherit	-Wert Dicke	Dicke oder “erben”
Initial	individual	Anfang-swert	individuell
Applies to	all elements	Anwendbar	Alle Elemente
Inherited	no	Vererbbar	nein

➡ **W3C-Modul: http://www.w3.org/TR/css3-background/#borders**

Vererbung im Dokumentenbaum

Grundsätzlich werden vererbbare Eigenschaften automatisch vom Eltern-Element auf die Kind-Elemente bis hinunter zu den Enkeln, Ur-Enkeln und Ur-Ur-Enkeln vererbt.

```
body { color: #33F; font-family: Verdana;}
```

Die vererbbaren Eigenschaften wie *color* und *font-family* gelten für das gesamte HTML-Dokument, wenn sie für das body-Element definiert werden. Im Sinne des ökonomischen Codierens empfiehlt es sich deshalb, jene Eigenschaften, die für das gesamte Dokument gelten sollen, für das body-Element zu definieren. Hauptsächlich sind dies die Eigenschaften Schriftfarbe (*color*), Schriftart (*font-family*) und Schriftgröße (*font-size*).

Selektoren

Das style-Attribut

Um die natürliche Vererbung zu unterbrechen – zum Beispiel um Überschriften anders einzufärben als für den *body* definiert – ist es prinzipiell möglich, mit dem *style-Attribut* für jedes HTML-Element eine Stilregel festzulegen. Im folgenden Beispiel wird eine h2-Überschrift rot eingefärbt und die Schriftgröße auf 20px festgelegt.

```
<h2 style=“color:red; font-size: 20px;“>Text der Überschrift</h2>
```

Stildateien, Stylesheets

Mit dem style-Attribut zu arbeiten, ist jedoch nicht elegant. Vor allem wenn in einem HTML-Dokument mehrere h2-Überschriften vorkommen, macht es viel Arbeit, dieselbe Stilregel für jede einzelne Überschrift zu formulieren und bei Bedarf wieder zu ändern. Weniger aufwändig und fehlerträchtig ist es, die Stilregel an den Element-Selektor zu binden. Dies kann dokument-intern im Kopfbereich geschehen, indem dort eine style-Sektion notiert wird.
Das W3C empfiehlt jedoch, die Stilregeln in einer oder mehreren externen Stildateien zu sammeln. Dadurch kann das Erscheinungsbild von HTML-Dokumenten beeinflusst werden, ohne das Dokument selbst zu verändern. Die Arbeitsteilung in Teams fällt leichter. Das media-Attribut erlaubt zudem die Spezifizierung des Ausgabegeräts, wodurch die Gültigkeit der Stilregeln auf spezielle Ausgabegeräte beschränkt wird.

Aufbau einer Stilregel

Stilregeln beginnen stets mit einem *Selektor*. Die Regeln selbst werden in geschweiften Klammern notiert. Sie bestehen aus einem oder mehreren Eigenschafts-/Wertepaaren. Semikolons trennen die Eigenschafts-/Wertepaare.

```
Selektor {
	Eigenschaft: Wert;
	Eigenschaft: Wert;
	Eigenschaft: Wert;
}
```

Beispiel:

```
h2 {
	color: red;
	font-size: 20px;
}
```

Die Regel selektiert alle h2-Elemente des HTML-Dokuments und weist ihnen die notierten Eigenschaften zu.

Arten von Selektoren

Um Stilregeln zu definieren, stehen rund zwanzig Arten von Selektoren zur Verfügung. Mit Selektoren können so Stilregeln gezielt für Elemente im Seitenbaum festgelegt werden. Nachfolgend sind die vier wichtigsten Selektoren beschrieben.

Element-Selektoren

Sind die Selektoren mit den HTML-Elementnamen identisch, spricht man von *Element-Selektoren*. Jedem HTML-Element lassen sich auf diese Weise Stilregeln zuweisen. Die Stilregeln gelten für alle Elemente gleichen Namens, außer sie werden später durch *Klassen*, *Identitäten* und *style*-Attribute überschrieben.

```
Elementname {Regel}
```

```
body {Regel}
article {Regel}
```

Klassen-Selektoren

Die Abbildung auf Seite 40 zeigt, dass die *section*-Elemente unterschiedliche Eigenschaften besitzen. Die *section*-Elemente sind sowohl grün als auch rot. Dies ist nur deshalb möglich, weil sie zwei verschiedenen Klassen angehören. Die Klassen-Regel ist stärker als die Element-Regel.

Der Klassen-Selektor besteht aus dem Klassen-Namen, dem ein Punkt vorangestellt ist.

```
.abschnitt1 {Regel}
.abschnitt2 {Regel}
.rot-gefaerbt {Regel}
```

ID-Selektoren

Das Selektieren von Element-Identitäten geschieht im Style Sheet mit dem sogenannten ID-Selektor. Jede ID darf nur ein einziges Mal in dem Dokument vorkommen. Sinn einer Identität ist es, Verwechslungen und Mehrfachzuweisen auszuschließen. Das h2-Element mit dem Inhalt „Sensation“ (Seite 40) gehört zu einer ID.

Der ID-Selektor besteht aus dem ID-Namen mit vorangestelltem, amerikanischen Nummernzeichen.

```
#wichtig {Regel}
#veraendern {Regel}
#nicht_vergessen {Regel}
```

Die Berechnung der Spezifität

Welche Regel letztlich die Oberhand gewinnt, lässt sich errechnen. Grundlage dieser Tabelle sind die Beispiele von oben.

			Spezifitätswert			
Selektor	**Art**	**Regel**	**1**	**10**	**100**	**1000**
h2	Element	Color: Green	1			
.rot	Klasse	Color: Red		10		
#blau	ID	Color: Blue			100	
	Style-Attribut	Color: Violett				1000

Treffen in einer Regel dieselben Eigenschaften aufeinander, kommt jene mit dem höchsten Spezifitätswert zum Tragen.

Universal-Selektor

Sollen Eigenschaften definiert werden, die für alle HTML-Elemente gelten, muss der Universal-Selektor – der Stern – benutzt werden. Die Regeln beeinflussen direkt jedes HTML-Element. Man sollte sehr bewusst mit dem Universal-Selektor umgehen.

```
* { margin: 0; padding: 0; }
```

Diese Regel bedeutet, dass alle HTML-Elemente keinen Außenabstand und keinen Innenabstand (Polster) aufweisen. Sie wird dazu verwendet, die unterschiedlichen Einstellungen in den Browser-Stylesheets der Hersteller auf denselben Wert zu setzen und ein gleichartiges Erscheinungsbild zu erhalten.

Kombinatoren

Die Eigenschaften von HTML-Elementen lassen sich auch durch den Pfad im Dokumentenbaum definieren. Beispiel ist wieder die Baumstruktur von Seite 40. Die Eigenschaften der h2-Elemente mit den Textknoten „Die Polizei tappt im Dunkeln“ und „Augenzeugen berichten“ werden defi-

niert, indem der Selektor die Reihenfolge der HTML-Element enthält, beginnend bei dem gemeinsamen Ausgangspunkt. Solche zusammengesetzten Selektoren heißen „Kombinatoren“.
Für „Die Polizei tappt im Dunkeln“ gilt der Selektor:

```
article header h2 {
	color: #CF3;
	background-color: #000;
}
```

Für „Augenzeugen berichten“ könnte in gleicher Weise gelten:

```
article section h2 {
	color: Red;
}
```

Die Stilregeln für h2 gelten nur für den im Selektor beschriebenen Weg. Anders ausgedrückt: Nur wenn h2 einem *section*-Element folgt, das wiederum einem *article*-Element folgt ist, nimmt es die rote Farbe an.

Kombinator für Kind-Elemente

Beispiel: Regel auf jene h2-Elemente anwenden, die Kind-Elemente der id="inhalt" sind.

```
#inhalt h2 { color:red; }
```

Beispiel: Regel nur auf jene h2-Elemente anwenden, die Kind-Elemente der id="inhalt" sind und der Klasse class="wichtig" angehören.

```
#inhalt h2.wichtig { color: green; }
```

Listen-Selektoren

Ganz anders zeigt sich das Erscheinungsbild, wenn keine Leerzeichen, sondern Kommas zwischen den Selektoren stehen. Im ersten Beispiel beziehen sich dann die Stilregeln auf jeden durch Komma getrennten Selektor, also sowohl auf den *#inhalt*, als auch auf *h2*.

```
#inhalt, h2 { color:red; }
#inhalt, h2.wichtig { color: green; }
```

Im zweiten Beispiel beziehen sich die Stilregeln sowohl auf den *#inhalt*, als auch auf *h2.wichtig*.

Der !important-Zusatz

Wird in einem Stylesheet dieselbe Stilregel mehrfach definiert (was zum Beispiel vorkommen kann, wenn Benutzer eigene Stilregeln formulieren können) kommt stets die letzte Definition einer Stilregel zur Anwendung.

```
h2 { background-color:red; }
h2 { background-color:green; }
```

Der Hintergrund des h2-Elements erscheint in grüner Farbe, da die letzte Definition alle vorhergehenden Definitionen überschreibt. Das mag trivial erscheinen, in umfangreichen Style Sheets kann das jedoch der Anlass zu einer langen Fehlersuche werden.

Der Zusatz !important verhindert eine nachträgliche Veränderung der Zuweisung.

```
h2 {background-color:red !important; }
h2 {background-color:green; }
```

Die !important-Regel sollte nur in Ausnahmefällen angewendet werden, zum Beispiel dann, wenn es dem Benutzer ermöglicht werden soll, eigene Stilregeln festzulegen, ohne den Grundaufbau eines HTML-Dokuments zu verändern.

Kaskade

Die Tatsache, dass Eigenschaften anhand des Dokumentenbaums gelesen und angewendet werden, heißt Kaskade (Wasserfall). Die Dateien, in denen solche kaskadierenden Regeln gesammelt sind, heißen *Cascading Style Sheets*, kurz *CSS*.

Der Browser, das heißt die Rendering Engine, arbeitet den vom HTML-Parser erzeugten Element-Baum in einer festgelegten Reihenfolge ab:

- Die Rendering Engine des Browsers sucht nach gleichen Selektoren.
- Die Selektoren werden zusammengefasst und nach der letzten, gültigen Definition sortiert.
- Es wird untersucht, ob ein '!important-Zusatz existiert.
- Die Spezifität wird berechnet und die höchste Spezifität wird angewendet.

Die Trennung von Form und Inhalt

Stylesheets – also die Sprache CSS – ermöglichen es, Form und Inhalt in getrennten Dokumenten unterzubringen, im HTML-Dokument und im CSS, dem Stylesheet. Zwar ist es nicht notwendig, Form und Inhalt zu trennen. HTML5 unterstützt auch die direkte Formatierung von HTML-Elementen mit Hilfe von style-Attributen. Diese können jedoch nicht durch externe CSS-Eigenschaften überschrieben werden. Sie behindern den Wechsel von Stilen, wie es zum Beispiel die Barrierefreiheit und die Anpassung an mobile Endgeräte erfordern.

Die Vorteile der Trennung von Form und Inhalt liegen darin:

- Bezug zu Berufen: Informatiker strukturieren, Redakteure schreiben, Grafiker gestalten.
- Änderungen am Erscheinungsbild sind über alle verbundenen HTML-Dokumente möglich.
- Der gleiche Inhalt kann optimiert für verschiedene Anzeigegeräte ausgegeben werden, zum Beispiel für Desktop-PC, Drucker, Smartphone...
- Realisierung von Barrierefreiheit

Visueller Aufbau eines HTML-Dokuments

Jedes HTML-Dokument besteht aus "rechteckigen" Bereichen, die über das Browserfenster verteilt sind. Ihnen liegen zusammenfassende HTML-Elemente zugrunde wie

- article
- section
- div
- header
- aside und andere

Aber auch alle Block-Elemente wie *h1*, *h2*, *p* usw. sind „rechteckig“. Dasselbe gilt für die Inline-Elemente. Die Ecken können zwar durch die Eigenschaft *border-radius* abgerundet werden, trotzdem verhalten sich die Elemente so als wären sie rechteckig.

Abbildung 14: Aufteilung des Browserfensters in rechteckige Bereiche

Die eckigen Bereiche sind manchmal unsichtbar, wie zum Beispiel in der geschwungenen Linie des Fotos bei "White Line/Color Line". In Wirklichkeit liegt dem Bild ein eckiger Bereich zugrunde, der jedoch nicht sichtbar ist.

Abbildung 15: Beispiel für einen scheinbar nicht-eckigen Bereich

Die zusammenfassenden Elemente, auch salopp *Container* genannt, enthalten die weiteren Elemente des HTML-Dokuments:

- Block-Elemente für Überschriften und Fließtext
- Inline-Elemente für Bilder, Hyperlinks, Fett- und Kursivdarstellung
- Listen-Elemente für die Navigationsbereiche
- Tabellen-Elemente für die tabellarische Darstellung von Daten
- Und andere

Damit die HTML-Elemente im gewünschten Aussehen erscheinen, sind im beigefügten CSS-Dokument Stilregeln definiert. Die Modelle für die grundsätzliche Gliederung der Seiten sind

- Das Box-Modell
- Das Visual Formatting Model

Der nächste Abschnitt beschreibt das *Box-Modell*. Die Beschreibung des *Visual Formatting Model* folgt an späterer Stelle.

Das Box-Modell

Das Box-Modell ist grundlegend in der W3C-Empfehlung für *CSS 2.1* beschrieben (Siehe auch http://www.w3.org/TR/CSS2/box.html).
Die Merkmale der Box, des Containers, die den Platzbedarf und die Abmessungen betreffen, sind die folgenden Eigenschaften:

- Breite und Höhe (*width*, *height*)
- Rahmen (*border*)
- Innabstände, Polster (*padding*)
- Außenabstände (*margin*)

Die Eigenschaften gelten für jedes Block-Element. Für Inline-Elemente kommen die Eigenschaften *width*, *height* und *margin* nicht in Frage, da sie über die Zeile (inline) hinausreichen.

Breite und Höhe eines Block-Elements

Mit den Eigenschaften Breite (*width*) und Höhe (*height*) erhält ein HTML-Element seine Abmessungen. Fehlen Angaben zu *width* und *height*, nimmt sich das HTML-Block-Element immer den Platz, den es benötigt. Wer möchte, dass eine Box mit der Textmenge nach unten wächst, darf keinen Wert für die Höhe angeben. Um ein Design zu realisieren, das sich der Menge des Inhalts anpasst, sollte entweder nur die Breite oder nur die Höhe angegeben werden – oder gar keine Abmessung.

```
#box {width: 400px; height: 300px;} /*(Vererbung: nein)*/
```

Dieses HTML-Element ist vierhundert Pixel breit und dreihundert Pixel hoch.

Voreinstellung/Standardwert:

```
#box {width: auto; height: auto;}
```

Die Eigenschaften width und height gelten für Block-Elemente. Inline-Elemente nehmen immer die Abmessungen ihres Inhalts ein.

Rahmen, die "Randlinie"

Ein HTML-Element kann eine Randlinie besitzen, auch Rahmen genannt (*border*). Wenn keine Randlinie vorhanden sein soll, muss der Wert für *border* explizit auf 0 gesetzt werden, um eventuelle Standard-Einstellungen des Browsers (zum Beispiel bei Bildern) zu überschreiben.

```
#box {border: 0;} /*(Vererbung: nein)*/
```

Der Rahmen wird nur dann vollständig angezeigt, wenn auch seine anderen Eigenschaften angeben werden: Die Rahmenart und die Rahmenfarbe. Das folgende HTML-Element besitzt rundherum einen durchgezogenen Rahmen, der 10 Pixel dick ist und blau dargestellt wird.

```
#box {border: 10px solid blue;}
```

Das folgende HTML-Element besitzt rundherum einen gepunkteten Rahmen, der 10 Pixel dick ist und schwarz dargestellt wird.

```
#box {border: 10px dotted black;}
```

Innenabstand, "Polster"

Der Inhalt eines HTML-Element kann an allen Seiten einen Abstand zu seinem Begrenzungsrahmen aufweisen, man spricht vom Innenabstand oder auch vom Polster (*padding*). Wenn kein Innenabstand vorhanden sein soll, muss der Wert für *padding* explizit auf 0 gesetzt werden, um eventuelle Voreinstellungen der Browser-Stylesheets zu überschreiben.

```
#box {padding: 0px; padding-left: 20px; padding-top: 10px;} /*(Vererbung: nein)*/
```

In diesem HTML-Element beträgt links der Abstand zum Text zwanzig Pixel, oben zehn Pixel, rechts und unten jeweils null Pixel.

Außenabstand

Ein HTML-Element hat einen Außenabstand (*margin*) zu den angrenzenden HTML-Element, beziehungsweise zum Browserfenster. Wenn kein Außenabstand vorhanden sein soll, muss der Wert für *margin* explizit auf 0 gesetzt werden, um eventuelle Voreinstellungen des Browsers zu überschreiben. Der Wert für *margin* kann negativ sein, zum Beispiel um Überlappungen zu realisieren.

```
#box {margin: 0px;} /*(Vererbung: nein)*/
```

Das folgende HTML-Element - hier in Kurzschreibweise notiert - hat einen Abstand zum oberen Element von 7px, zum rechten Element von 12px, zum unteren Element von 10px und zum linken Element von 5px.

```
#box {margin: 7px 12px 10px 5px;}
```

Weitere Eigenschaften des HTML-Elements

Es kann die Hintergrundfarbe eines HTML-Elements definiert werden (*background-color*), oder es kann ein Hintergrundbild angegeben werden (*background-image*). Wenn der Inhalt des HTML-Elements mehr Platz benötigt, als ihm die Abmessungen zubilligen, kann die Eigenschaft *overflow* gesetzt werden. Die genannten Eigenschaften zählen jedoch nicht zum Boxmodell.

Illustration des Box-Modells

Die Grafik illustriert das Zusammenwirken von Rahmen, Höhe, Breite, Innenabstand, Außenabstand.

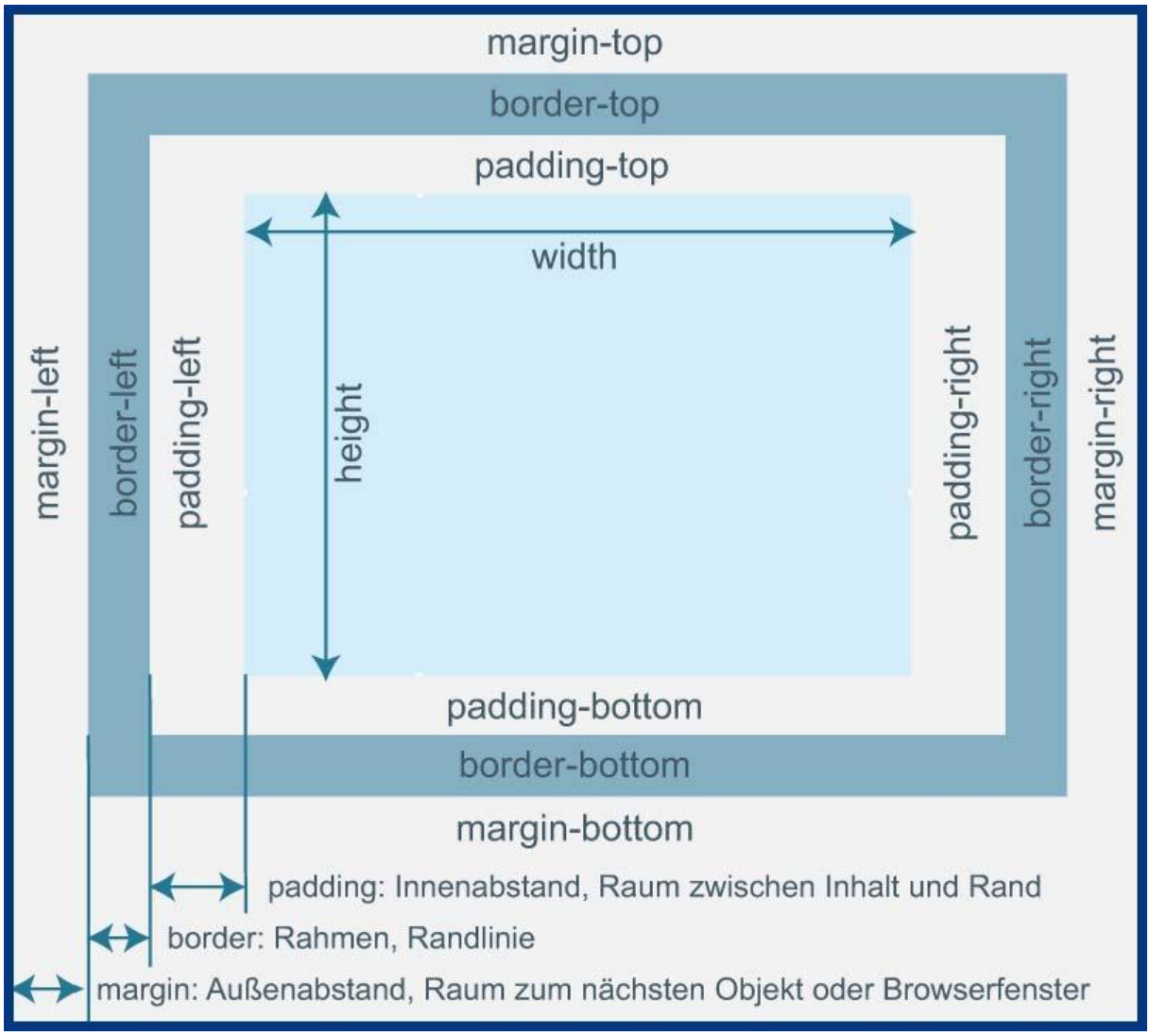


Abbildung 16: Das Box-Modell

Der tatsächliche Platzbedarf

Der tatsächliche Platzbedarf eines HTML-Elements errechnet sich aus den Eigenschaften:

Horizontaler Platzbedarf

Der horizontale Platzbedarf eines Containers errechnet sich aus:
Breite + linker Außenabstand + rechter Außenabstand + linke Rahmenbreite + rechte Rahmenbreite + linker Innenabstand + rechter Innenabstand

Höhe

Der vertikale Platzbedarf eines Containers errechnet sich aus:
Höhe + oberer Außenabstand + unterer Außenabstand + obere Rahmenbreite + untere Rahmenbreite + oberer Innenabstand + unterer Innenabstand.

Beispiel

```
#box { width: 400px;
             height: 300px;
             padding: 10px;
             border: 1px dotted black;
             margin-top: 20px;}
```

Horizontaler Platzbedarf:
400px (width) + 20px (padding, rundherum jeweils 10px) + 2px (border, rundherum jeweils 1px) = 422px

Vertikaler Platzbedarf:
300px (height) + 20px (padding, rundherum jeweils 10px) + 2px (border, rundherum jeweils 1px) + 20px (margin, Außenabstand, nur oben) = 344px

Zusammenfassung
Mit den Eigenschaften *width* und *height* wird die *Nutzbreite* eines HTML-Elements festgelegt. Alle weiteren Angaben werden hinzuaddiert.

Besonderheit
Der Rahmen kann anstelle von *border* auch mit der Eigenschaft *outline* definiert werden. Outline fließt nicht in die Berechnung des Platzbedarfs ein.

Das alternative Boxmodell

Die *CSS 3-Spezifikation* erlaubt ein alternatives Boxmodell mit Hilfe der Eigenschaft *box-sizing*.

```
#box1 {box-sizing: content-box;}
#box2 {box-sizing: border-box;}
```

Box1 verhält sich, wie oben beschrieben. Die Eigenschaften *width* und *height* legen die **Nutzbreite** des HTML-Elements fest. Alle weiteren Angaben werden hinzuaddiert. Box2 verhält sich genau umgekehrt. Die Eigenschaften *width* und *height* legen die *Außenabmessungen* des HTML-Elements fest. Alle weiteren Angaben werden abgezogen.

Maßangaben und Maßeinheiten

Längenmaße

Fürs Webdesign können absolute und relative Längenmaße verwendet werden, um Element-Abmessungen und Schriftgrößen zu definieren.

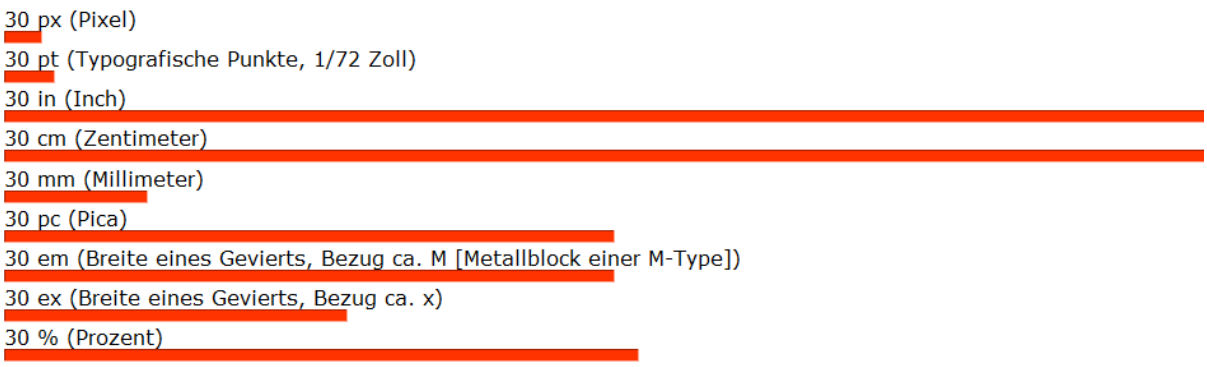


Abbildung 17: Illustration ausgewählter Längenmaße

Hauptsächlich werden verwendet
- px Pixel (feste Längenangabe, Bildschirm)
- pt Punkt (feste Längenangabe, Drucker)
- em (relative Längenangabe bezogen auf die Grundschriftart)
- % (relative Längenangabe bezogen auf das Eltern-Element)

Relative Schriftgrößen

Darüber hinaus gibt es relative Schriftgrößen, die sich jeweils auf die Größe der Grundschriftart beziehen, meist 16 Pixel.

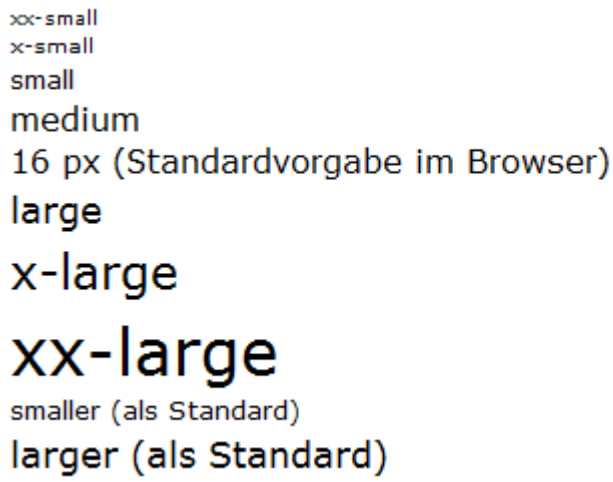


Abbildung 18: Relative Schriftgrößen

Farbangaben

Benannte Farben

Die meisten Browser unterstützen benannte Farben.

Zahlencodes für Farben

Farben werden in der Regel im **Hexadezimal-Format** definiert. Das Format beginnt mit dem amerikanischen Nummernsymbol, dem Raute-Zeichen "#", gefolgt von 6 Ziffern zwischen 0 und F. Immer zwei Stellen stehen für 256 Farbabstufungen. Die ersten beiden für rot, die mittleren beiden für grün und die beiden letzten Stellen stehen für blau (RGB-Schema). Da die Farbwerte bei 0 beginnen, ist der höchste Farbwert einer Farbe 255.

```
color: #AA0000; /* dunkles Rot */
```

Verkürzte Schreibweise

Eine verkürzte Schreibweise mit 3 Ziffern ist möglich, wenn die drei Paare aus jeweils denselben Ziffern bestehen.

```
color: #A00; /* dunkles Rot */
```

Notierung im RGB-Format

Farbwerte können alternativ auch im RGB-Format notiert werden.

```
color: rgb(170,0,0); /* dunkles Rot */
color: rgb (67%, 0, 0);
```

Auch die Notierung im HSL-Format ist möglich, jedoch unüblich.

Websichere Farben

Als Grafikkarten nur 256 Farben anzeigen konnten, musste auf eine sogenannte **websichere** Farbgestaltung Rücksicht genommen werden. Es gab nur eine begrenzte Anzahl von Farben, die auf PC- und Mac-Systemen in gleicher Weise angezeigt wurden. Moderne Grafikkarten stellen das gesamte elektronisch mögliche Farbspektrum von 16,7 Millionen Farben dar, auf websichere Farben muss keine Rücksicht mehr genommen werden.

Beispiele für Farbcodes

Hexadezimalwert	RGB	Farbe	Kurzschreibweise	Klartextname

#FF0000	Rgb (255,0,0)	Volles Rot	#F00	Red
#00FF00	Rgb (0,255,0)	Volles Grün	#0F0	Green
#000099	Rgb (0,0,153)	Dunkleres Blau	#009	
#FFFFFF	Rgb (255,255,255)	Weiß	#FFF	White
#000000	Rgb (0,0,0)	Schwarz	#000	Black
#C0C0C0	Rgb (192, 192, 192)	Hellgrau	Keine	Silver

Gebräuchlich ist die Schreibweise im Hexadezimalformat.

Das Visual Formatting Modell

Das *Box-Modell* beschrieb, wie sich die Abmessungen der HTML-Elemente auswirken. Das Visual Formatting Modell trifft nun Aussagen, wie sich HTML-Elemente verhalten und wie sie sich im Browserfenster anordnen.

Floaten

Das Floaten ermöglicht es, dass HTML-Elemente von anderen HTML-Elementen umflossen werden können. Die CSS-Eigenschaft *float* besagt, dass das Block-Element vom nachfolgenden Block-Element umflossen werden soll. *Clear* meint, dass das aktuelle HTML-Element das vorhergehende Block-Element nicht mehr umfließen soll. *Float:left* bedeutet, dass der Container *links* steht und umflossen wird. Sinngemäß gilt das gleiche für *float:right*. Das Umfließen gilt so lange, bis es mit *clear:left* bzw. *clear:right* wieder ausgeschaltet wird. Mit *clear:both* können beide Floats gleichzeitig aufgehoben werden. Block-Elemente, die umflossen werden sollen, benötigen eine Breite. Nur Elemente mit den Positionseigenschaften *static* oder *relative* können umflossen werden (siehe unten).

Positionierung "static"

Die Positionseigenschaft *static* ist die Initialeigenschaft eines Block-Elements. Diese Eigenschaft muss nicht extra angegeben werden. Ein static positionierter Container verhält sich keineswegs statisch oder stationär, sondern höchst flexibel. Mit static (statisch) ist gemeint, dass sich der Container zwar seinem Inhalt anpassen kann, sich aber nicht in Bezug auf den normalen Textfluss verschieben lässt. Nicht anwendbar sind deshalb die Eigenschaften *top, right, bottom, left, z-index*. Die automatische Breite lässt den static-Container immer den maximal zur Verfügung stehenden Raum einnehmen. Floaten ist möglich.

Positionierung "relative"

Der *relative*-Container verhält sich zunächst wie ein *static*-Container. Er liegt im normalen Textfluss, nimmt die maximal verfügbare Breite ein und kann floaten. Mit den Eigenschaften *top, right, bottom, left* lässt sich der relative-Container aus dem normalen Textfluss hinausbewegen und relativ zu vorhergehenden Containern verschieben. Auch das Stapeln (*z-index*) ist möglich.

Positionierung "absolute"

Der *absolute*-Container fällt aus dem normalen Textfluss heraus und lässt sich mit den Eigenschaften *top, right, bottom, left* an einer bestimmten Stelle des HTML-Dokuments platzieren. Die Eigenschaften *top, right, bottom, left* beziehen sich auf das HTML-Dokument. Befindet sich ein *absolute*-Container innerhalb eines *relative*-Containers, so orientiert sich der *absolute*-Container nicht am *body*-Element des HTML-Dokuments, sondern an dem ihn umgebenden *relative*-Container. *Absolute*-Container können nicht floaten. Der *z-index* ist anwendbar, der das Überlappen (stapeln) in der z-Achse erlaubt. Die automatische Breite orientiert sich nicht am zur Verfügung stehenden Platz, sondern am Platzbedarf des Inhalts.

Positionierung "fixed"

Der *fixed*-Container orientiert sich am Browserfenster, nicht am HTML-Dokument wie der absolute positionierte Container. Er bleibt an der Stelle stehen, an die er gesetzt wurde, und scrollt nicht mit.

Übersichtstabelle Positionierung und Floaten

Wert/Eigenschaft	**Static**	**Relative**	**Absolute**	**Fixed**
Top	-	Ja	Ja	Ja
Right	-	Ja	Ja	Ja
Bottom	-	Ja	Ja	Ja
Left	-	Ja	Ja	Ja
z-index	-	Ja	Ja	Ja
Width:auto	Orientiert sich am Platz, der zur Verfügung steht	Orientiert sich am Platz, der zur Verfügung steht	Orientiert sich am Platzbedarf des Inhalts	Orientiert sich am Platzbedarf des Inhalts
Float	Ja	Ja	-	-
Clear	Ja	Ja	-	-
Orientierung	am Textfluss	am Vorfahr	am relativen Vorfahr	am Fenster

Tabelle 7: Das Verhalten von positionierten HTML-Elementen

Verschachteln von HTML-Elementen

Reizvoll im Webdesign ist es, HTML-Elemente ineinander zu verschachteln. In der Kombination von verschachtelten, gefloateten und nacheinander angeordneten Containern ergeben sich zahllose Gestaltungsmöglichkeiten.

Zusammenfassung

Der HTML-Parser erzeugt die Dokument-Struktur, die man sich als Baum vorstellen kann. Innerhalb des Baumes werden Eigenschaften automatisch vererbt – sofern sie per Default als vererbbar definiert sind. Anderenfalls können sie auch zwangsweise ererbt werden. Diese Kaskade ist die Grundlage der Trennung von Form und Inhalt, dem wichtigsten Designprinzip von Webseiten. Das Box-Modell beschreibt die Abmessungen und den Platzbedarf von HTML-Elementen, während das Visual Formatting Modell die Möglichkeiten beschreibt, wie HTML-Elemente im Browserfenster positioniert, also angeordnet werden können.

Embedded Content und die Arbeit mit Daten

In diesem Abschnitt soll zunächst planmäßig ein HTML-Dokument mit Text-Inhalt erstellt werden, um danach Nicht-Text-Inhalt – Embedded Content – hinzuzufügen.

Vorgehen beim Aufbau eines HTML-Dokuments

Scribble

Erstellen Sie zunächst ein Scribble oder einen Papierprototyp.

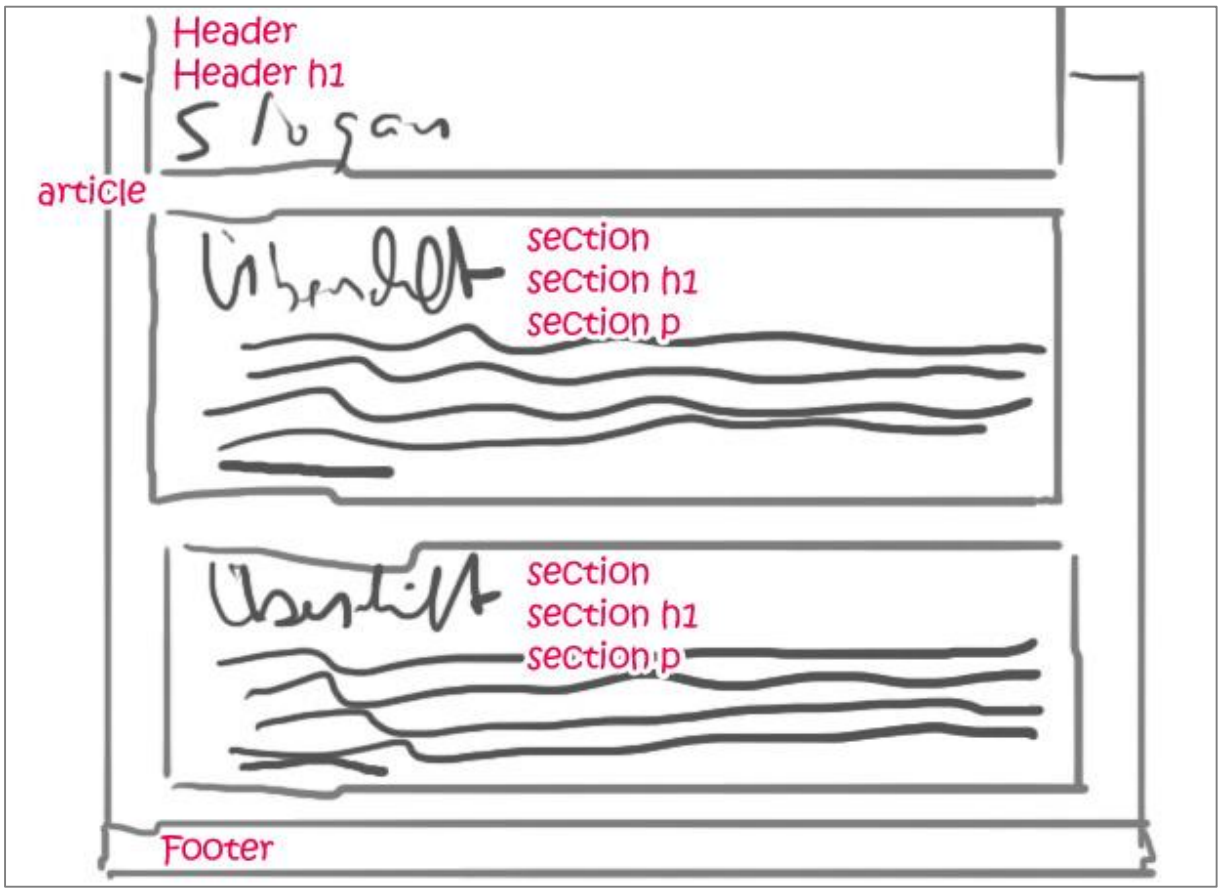


Abbildung 19: Beispiel für ein Scribble

Schritt 1: Das HTML-Grundgerüst erstellen oder kopieren

Erstellen oder kopieren Sie das HTML-Grundgerüst.

```
<!DOCTYPE html>
<html lang="de">
<head>
<meta charset="utf-8">
<title>Mein HTML5 Grundgerüst</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="_reset.css" type="text/css" />
<link rel="stylesheet" href="_styles1.css" type="text/css" />
<link rel="stylesheet" href="_print.css" type="text/css" media="print"/>
</head>
<body>
</body>
</html>
```

Schritt 2: Der semantische Seitenaufbau

Erstellen Sie zunächst den semantischen Seitenaufbau ohne Rücksicht auf das visuelle Erscheinungsbild. Tippen Sie die im Scribble notierten HTML-Elemente der Reihe nach ab. Eventuell möchten Sie bereits Klassen und IDs vergeben. Tun Sie das, aber halten Sie sich noch nicht mit der Ausformulierung der Stilregeln auf. Es bietet sich an, jedem HTML-Element eine Hintergrundfarbe zu geben. Dies erleichtert den gestalterischen Überblick.

Schritt 3: Der visuelle Seitenaufbau

Erst nachdem der semantische Seitenaufbau feststeht, formulieren Sie die Stilregeln für die HTML-Elemente.

Schritt 4: Validieren

Validieren Sie HTML und CSS.
Anmerkung: Die Validierung sollte nicht nur ganz am Ende erfolgen, sondern immer wieder während des gesamten Entstehungsprozesses. Dies erleichtert die Fehlersuche.

Die Einbettung von Nicht-Text-Content

Bilder

Bilder können in HTML-Dokumenten als Vordergrundbilder oder als Hintergrundbilder verwendet werden. Vordergrundbilder werden standardmäßig ausgedruckt, während das Ausdrucken von Hintergrundbildern im Druck-Interface des Browsers explizit eine Bestätigung verlangt.

Vordergrundbilder – das img-Element (HTML)

Das *img*-Element bindet Bilddateien als druckbare Vordergrundbilder ein. Es ist ein leeres Element, benötigt keine schließende Kennzeichnung. Die Attribute des Elements enthalten alle zur Anzeige notwendigen Informationen.

```
<img src="Bilddateiname.jpg" width="400" height="300" alt="Textalternative zum
Bild" longdesc="beschreibung.txt" >
```

Die Attribute des img-Elements

Attribut	Bedeutung
src	Absoluter oder relativer Dateiname des anzuzeigenden Bildes
width	Breite des Bildes, kann durch CSS-Stilregeln überschrieben werden
height	Höhe des Bildes, kann durch CSS-Stilregeln überschrieben werden
alt	Alternativer Text für Suchroboter und Vorlesemaschinen
longdesc	Langbeschreibung in einer externen Datei zur ausführlichen Bildbeschreibung
name	Zugriff auf das Element beim Programmieren (besser das Id-Attribut verwendeen)

Vordergrundbilder können auch in Verweisen verwendet werden.

```
<a href="link.html"><img src="Bilddateiname.jpg" width="400" height="300"
alt="Textalternative zum Bild" /></a>
```

Um semantisch korrektes HTML zu erzeugen, empfiehlt es sich, das gruppierende Element *figure* zu verwenden. Zur Beschreibung des Bildes kann dann das Element *figcaption* dienen.

```
<figure><img src="erdbeere-1.png" width="242" height="300" alt="Erdbeere">
```

```
    <figcaption>Abb. 1: Unsere beste Sorte.</figcaption>
</figure>
```

Um Bilder zu stapeln, also um sie in den Vorder- und Hintergrund zu rücken, wird die Positionierung *absolute* und der *z-index* verwendet.

Bildformate für Vordergrund- und Hintergrundbilder

Ohne Mitwirkung von Plug-Ins oder ActiveX-Komponenten zeigen Browser Bilder an, wenn sie zu einem der folgenden Formate gehören.

Dateiformat	Variante	Merkmale	Animation	Kompression,	Verwendung
*.gif		Pixelgrafik Farben maximal 256, Transparenz	Ja	Verlustfrei (sofern weniger als 256 Farben, sonst Interpolation fehlender Farben), hohe Kompression	Logo und Schmuck-Element mit wenigen Farben, Linien, Animationen
*.jpg		Pixelgrafik Farben maximal 16,7 Mio. keine Transparenz	Nein	Verlustbehaftet Hohe Kompression	Fotos, Illustrationen
*.png	8	Pixelgrafik Farben maximal 256, Transparenz	Nein	Verlustfrei (sofern weniger als 256 Farben, sonst Interpolation fehlender Farben), hohe Kompression	Logo und Schmuck-Element mit wenigen Farben, Linien
*.png	24	Pixelgrafik Farben max. 16,7 Mio. Transparenz, auch im Schatten	Nein	Verlustfrei, hohe Kompression, aber große Dateien aufgrund komplexer Farbalgorithmen	Fotos und Effektbilder, die Transparenz im Schatten erfordern
*.svg		Vektorgrafik, skalierbar	Ja (SMIL), Manipulation des SVG-DOM	Nicht relevant, bzw. nicht vergleichbar	Verlustfreie Skalierbarkeit bei Illustrationen

Tabelle 8: Bildformate

Hintergrundbilder – die background-Eigenschaft (CSS)

Mit Hilfe von CSS-Stilregeln lassen sich HTML-Elementen Hintergrundbilder zuweisen. Hintergrundbilder werden normalerweise beim Ausdrucken nicht berücksichtigt.

```
body { background-image: URL(Bilddateiname1.jpg); }
h1 { background-image: URL(Bilddateiname2.jpg); }
```

Die Standardeigenschaft von Hintergrundbildern ist es, dass das Hintergrundbild sowohl in x-Richtung, als auch in y-Richtung so oft wiederholt wird, bis es die zur Verfügung stehende Fläche vollständig ausfüllt. Der Hintergrund wird „gekachelt“. Ist die Fläche kleiner als das Bild, wird nur der passende Bildausschnitt gezeigt. Weitere Eigenschaften (*background-repeat*) beschränken das Kacheln und positionieren das Hintergrundbild (*background-position*).

```
h1 {background-repeat: no-repeat} /* Hintergrundbild nur einmal zeigen */
h1 {background-repeat: repeat-x} /* Hintergrundbild in x-Richtung wiederholen */
h1 {background-repeat: repeat-y} /* Hintergrundbild in y-Richtung wiederholen */
```

Sprite-Grafik

Unter einer Sprite-Grafik versteht man eine besonders gestaltete Hintergrundgrafik, in der mehrere Design-Grafiken in einer übergreifenden Grafik zusammen untergebracht sind und durch Positionieren an die richtige Stelle geschoben werden. Dies erleichtert die Übersichtlichkeit.
Das Beispiel zeigt eine Sammlung von ca. 30 Facebook-Icons. Der Zugriff auf das Daumen-nach-oben-Symbol geschieht durch das Verschieben der Hintergrundgrafik im HTML-Element. Die Hintergrundgrafik des Elementes mit der ID *#gefaellt_mir* wird 20 Pixel nach rechts und 100 Pixel nach oben verschoben. Das Element #gefaellt_mir muss die gleichen Abmessungen besitzen wie der Daumen-nach-oben-Bildausschnitt.

```
#gefaellt_mir {
	background-image: url(sprite1.jpg);
	background-position: 20px -100px;
	width:16px;
	height:16px;
}
```

Weitere Elemente für Embedded Content

Die Tabelle zeigt ausgewählte weitere Elemente die externe, nicht-HTML-Quellen in einem HTML-Dokument verfügbar machen.
Beispiele sind Medienquellen und andere vollständige Inhaltscontainer. Embedded Content verhält sich meist wie Inline-Elemente.

Element	Kurzbeschreibung
<audio>	Erwartet eine Audiodatei in einem abspielbaren Format, zum Beispiel *mp3*
<canvas>	Erwartet Daten, die auf der Zeichenfläche angezeigt und verändert werden
<embed>	Kennzeichnung von Nicht-HTML-Inhalt aus externen Applikationen
<iframe>	Ermöglicht ein Sichtfenster im HTML-Dokument auf andere HTML-Dokumente
<img>	Erwartet eine Bilddatei im Pixelformat
<math>	Erwartet einen mathematischen Formelsatz
<object>	Kennzeichnet Inhalt, der eines *Plugins* bedarf, häufig in Zusammenhang mit dem <embed>-Element
<svg>	Erwartet Bildinformationen als svg-Vektoren
<video>	Erwartet Bewegtbildinformation

Tabelle 9: Embedded Content

Von den in der Tabelle dargestellten Elementen sollen näher beschrieben werden:

- Audio
- Video
- JavaScript API
- Canvas
- Drag & Drop
- Geolocation

Audio-Einbindung

Das Audio-Element kann auf mehrere Quellen verweisen. Abgespielt wird im Browser die Audio-Datei, die der Browser unterstützt. Eine aufwändige Einbindung mit Hilfe von JavaScript wie in früheren HTML-Versionen ist nicht nötig. Das Attribut „controls“ gibt an, ob eine Steuerleiste zum Abspielen vorhanden sein soll oder nicht.

```
<audio controls="controls">
<source src="song.ogg" type="audio/ogg" />
<source src="song.mp3" type="audio/mpeg" />
Ihr Browser unterstützt nicht das Audio-Element. Your browser does not support the
audio element.
</audio>
```

Video-Einbindung

Auch das Video-Element kann auf mehrere Quellen verweisen. Abgespielt wird im Browser das Video, das der Browser unterstützt. Eine aufwändige Einbindung mit Hilfe von JavaScript ist nicht mehr nötig. Das Attribut „controls“ gibt an, ob eine Steuerleiste zum Abspielen vorhanden sein soll oder nicht.

```
<video width="320" height="240" controls="controls">
<source src="movie.ogg" type="video/ogv" />
<source src="movie.mp4" type="video/mp4" />
Your browser does not support the video element.
</video>
```

Codecs und MIME-Types

Codecs

Die jeweils verwendeten Codecs, die zur Verarbeitung zum webfähigen Video herangezogen wurden, lassen sich zum Beispiel mit dem VLC-Player auslesen (Siehe *Werkzeuge*: Medieninformationen und *Werkzeuge*: Codecs). Die abspielenden Devices müssen die Codecs kennen und deren Informationen verstehen. Das heißt: Nur Videos, deren Codecs im abspielenden System installiert sind, lassen sich auch tatsächlich anzeigen.

Serverseitige MIME-Types

Auch dem Server muss bekannt sein, welche Dateiformate er ausliefert. Beim Apache Webserver sollten für jeden auszuliefernden Dateityp der MIME-Type notiert sein. Sie fügen die folgenden Zeilen in die Apache-Konfigurationsdatei ein, um mp4, ogv und webm -Videos bekanntzugeben:

```
AddType video/mp4 .mp4 .m4v
AddType video/ogg .ogv
AddType video/webm .webm
```

Sie machen diese Änderungen entweder in der Datei *mime.types* oder durch *AddType* in der *httpd.conf*-Datei.

Videos, die mit verschiedenen Codecs erzeugt wurden

```
type="video/ogv" Video: Theora Video | Audio: Opus
type="video/mp4" Video: H.264 MPEG-4 AVC | Audio: MPEG AAC
type="video/webm" Video: Google/On2's VP8 | Audio: Vorbis Audio
type="video/x-ms-wmv" Video: Windows Media Video 8 | Audio: Windows Media Audio 2
type="video/x-flv" Video: Flash Video 1 | Audio: Audio Layer 1/2/3
type="video/x-flv" Video: On2's VP 6.2 | Audio: Audio Layer 1/2/3
```

YouTube-Einbindung

YouTube bietet Einbettungs-Code an, der auf einem *iframe*-basiert. Das iframe-Elemente definiert salopp gesagt eine Art „Loch“ im Browserfenster, durch das ein anderes HTML-Dokument hindurchschauen kann.

```
<iframe width="420" height="315" src="http://www.youtube.com/embed/09PoGVk3eQQ"
frameborder="0" allowfullscreen></iframe>
```

Video API

API heißt *Application Programming Interface* (Programmierschnittstelle). HTML5 APIs erleichtern maßgeblich das Erstellen von Webanwendungen. Viele HTML4 APIs wurden entweder abgeschafft oder verbessert und erweitert. Interessant sind jedoch vor allem die zahlreichen neuen HTML5 APIs.

Beispiele für HTML5 API

- Playback- und Untertitel-API für das *video*- und *audio*-Element
- Drag & Drop-API und das *draggable*-Attribut
- Validierungs-API für Formulareingaben
- und viele andere...

Eine ausführliche Behandlung des API-Themas würde das Thema "Auszeichnungssprachen" sprengen.

JavaScript und die HTML5 Video API

HTML5 verfügt über *Methoden*, *Eigenschaften* und *Event-Handler* (Ereignis-Behandlung), um mit Hilfe von JavaScript die von einer API betroffenen HTML-Elemente zu manipulieren, also zu kontrollieren und zu steuern. Die Basis einer HTML5 API ist das DOM (*Document Object Model*) des HTML-Dokuments.
Die Video-API von HTML5 stellt Methoden, Eigenschaften und Ereignisse (Events) für Playback und Untertitelung zur Verfügung wie

- abspielen
- pausieren
- vergrößern/verkleinern
- Ton an/aus

Das Video enthält in diesem Beispiel keine Steuerleiste (*controls*), um die Funktion des API besser beobachten zu können.

Media-Events

In Aktion sehen Sie alle Methoden, Eigenschaften und Events des *video*-Elements. Bitte beachten Sie dort in der Quelltext-Ansicht das Skript.

➡ **Media-Events des W3C**

Das Canvas-Element und seine API

Das *canvas-Element* (*Leinwand*) stellt eine Bitmap-Zeichenfläche zur Verfügung, die es erlaubt, dass während der Laufzeit mit Hilfe von JavaScript Grafiken erzeugt werden können. Es geht um zweidimensionale (2d) Grafiken wie zum Beispiel Diagramme. Zur API gehören vier Schnittstellen:

- *CanvasRenderingContext2D*: Zum Zeichnen von Linien und Formen,
  siehe auch bei MDN (Mozilla Developper Network)
- *CanvasGradient*: Mit Farbverläufen arbeiten
- *CanvasPattern*: Mit Hintergrundmustern arbeiten
- *ImageData*: Mit Bitmaps arbeiten

Das folgende Beispiel zeichnet ein rotes Rechteck auf eine gelbe Leinwand (*canvas*).

Drag & Drop API

Die *DnD-API* (Formulierung des w3c) stellt einen *Drag & Drop* (ziehen und ablegen) Mechanismus zur Verfügung, der auf der Implementierung des zugrundeliegenden Geräts beruht. Typische Geräte sind Geräte mit Mauszeiger. Auf Geräten ohne Mauszeiger kann *Drag & Drop* auf andere Weise implementiert sein.

```
<p id="ziehen" draggable="true" ondragstart="zieh_mich(event)">ZIEH MICH!</p>
<div id="ablegen" ondrop="lege_ab(event)"
ondragover="erlaube_ablegen(event)">Ablegen</div>
```

Noch ist das Ablegen nicht möglich. Erst einige JavaScript-Zeilen verarbeiten die Mouse-Events.

Geolocation API

Die **Geolocation-API** ist eine Schnittstelle zu den Informationen, die das zugrundeliegende Gerät liefert. Es kann sich um *GPS*-Informationen handeln. Die Informationen können aber auch aus *IP-Adressen*, *WIFI-Daten* und anderen Quellen stammen. Es gibt jedoch keine Garantie dafür, dass die Quelle die Informationen so liefert, dass die Geolocation-API sie sinnvoll nutzen kann. Das Ergebnis sind die geografische Länge (*Longitude, Längengrad*) und Breite (*Latitude, Breitengrad*).

Formulare und die Verarbeitung von Benutzereingaben

Formulare in HTML-Dokumenten erlauben es, dass Benutzer Informationen an den Betreiber der Website senden. Formulare sind die einzige Möglichkeit, um als Webmaster qualifizierte Rückmeldungen zu erhalten.

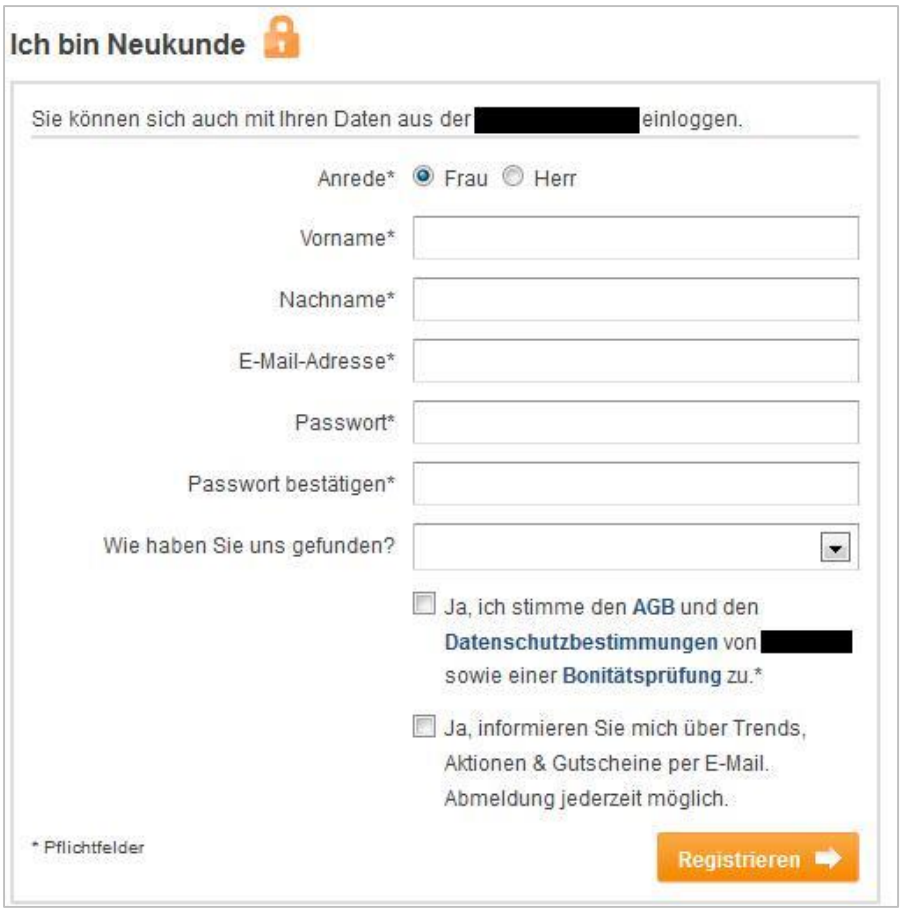


Abbildung 20: Beispiel für ein typisches Formular

Formulare sind mindestens mit einem Eingabefeld und einer Absenden-Schaltfläche ausgestattet. Die Funktionalität der Eingabefelder kann sehr vielfältig sein.
So gibt es Felder für

- Datum
- Dateien
- Einzeiligen und mehrzeiligen Text
- Bilder
- Checkboxen und Radiobuttons

und viele andere mehr. In einem HTML-Dokument können mehrere Formulare vorkommen.

Aufbau eines Formulars

Ein Formular besteht mindestens aus einem Eingabefeld und einer Absenden-Schaltfläche. Hier das Code-Beispiel für ein minimales Formular:

```
<form id="formular1" action="auswerten.php" method="post">
  <label for="textfield">Name</label>
  <input type="text" name="textfield" id="textfield">
  <p><input name="Submit1" type="submit" value="Submit" /></p>
</form>
```

- Das *form*-Element umschließt den Formularbereich.
- Das *id-Attribut* adressiert das Formular. Dies ist zum Beispiel dann von Bedeutung, wenn mehrere Formulare in einem HTML-Dokument enthalten sind.
- Das *action-Attribut* benennt das Serverprogramm, das die Formulareingaben übernimmt

und auswertet.

- Die *input*-Elemente nehmen je nach *type*-Attribut unterschiedliche Benutzeraktionen entgegen. *Type=“text“* erwartet Texteingaben, *type=“submit“* erwartet den Klick zum Absenden oder Bestätigen. Es gibt zahlreiche weitere *type*-Attribute.

Visuelle Gestaltung des Formulars

Ein Formular ist ein Benutzer-Interface, in dem Benutzer aktiv werden. Formulare müssen deshalb übersichtlich gestaltet werden, so dass die Eingaben verständlich und Fehlbedienungen ausgeschlossen sind. Formulare sollten hinsichtlich ihrer Usability unbedingt getestet werden.

Gestaltung mit pre-Element (Vorformatierung)

Das pre-Element stellt das Formular so dar, wie es im Quelltext mit Leerzeichen und Tabulatoren formatiert wird.

```
<form id="formular2" action="auswerten.php" method="post">
<pre><label for="textfield2">Name</label> <input type="text" name="textfield2"
id="textfield2">
</pre>
  <pre>     <input name="Submit1" type="submit" value="Submit" />
</pre>
</form>
```

Gestaltung mit CSS

Hier eine Auswahl von CSS-Stilregeln für die Formulargestaltung:

```
#formular3 {
       border: 1px dotted #666;
       padding: 10px;
       font-family: Verdana, Geneva, sans-serif;
       width: 400px;
       margin-right: auto;
       margin-left: auto;
       text-align: center;
}
#formular3 input {
       width: 300px;
       font-family: Verdana, Geneva, sans-serif;
}
#formular3 label {
       display: block;
       text-align: center;
       margin-bottom: 6px;
}
#formular3 input[type=submit] {
       font-size: 18px;
       background-color: #C00;
       color: #FFF;
       padding: 7px;
       cursor: pointer;
       border: 0px;
}
```

Formular-Elemente

Das *form*-Element

```
<form action="auswertung.xxx" method="post" enctype="text/plain" name="form1"
target="_blank" id="form1">
```

- *form*: Element, das die Formular-Elemente umschließt
- *action*: Attribut, das auf das serverseitige Programm verweist, das zum Auswerten der Formulareingaben dient.
- *method*: Attribut, das die Versende-Methode beschreibt. GET oder POST. GET versendet die Formulardaten in der Adresszeile, POST unsichtbar im Header
- *enctype*: Attribut, das die Codierung der Formularübermittlung beschreibt
- *name*: Attribut, das dem Element einen Namen gibt
- *target*: Attribut, das angibt, in welchem Fenster die Formularauswertung stattfindet
- *id*: Attribut, das eindeutig das Formular benennt

Das *label*-Element

Das label-Element stellt den logischen Bezug der Beschriftung zum Element her. Es zeigt die Zusammengehörigkeit von Beschriftung und Eingabewert an.

```
<label for="textfield3">Reiseziel</label>
<input type="text" name="textfield3" id="textfield3">
```

Das label-Element vereinfacht zudem die visuelle Gestaltung.

Versenden eines Formulars als E-Mail

Das *action*-Attribut des *form*-Elements gibt an, welche Programm-Routine die Formulareingaben beim Betreiber der Website verarbeiten soll. Die Formulardaten lassen sich auch zu einem zentralen E-Mail-Account senden. Dies funktioniert nur, wenn ein E-Mail-Client auf dem Sende-Rechner vorhanden ist.

```
<form action="mailto:schreibtisch@daheim.de" method="post" enctype="text/plain"
name="form1" target="_blank" id="form1">
```

Der Encoding-Type *text/plain* ist hier notwendig.

Eingabefelder

Text-Eingabe, einzeilig

```
<label for="feld1">Feld 1</label>
<input type="text" name="feld1" id="feld1">
```

Text-Eingabe, mehrzeilig

```
<label for="textarea">Bitte schildern Sie den Sachverhalt</label>
<textarea name="textarea" rows="10" id="textarea"></textarea>
```

Unsichtbares Textfeld

```
<input type="hidden" name="hiddenField" id="hiddenField">
```

Auswahlfelder

Listenfeld

Mehrfachauswahl ist wegen des Attributs *multiple* möglich.

```
<label for="feld40">Feld 40</label>
<select name="feld40" size="3" multiple id="feld40">
```

```
<option>keine</option>
<option>Frau</option>
<option>Herr</option>
<option>Mrs</option>
<option>Mr</option>
</select>
```

Dropdownfeld

```
<label for="feld30">Feld 30</label>
<select name="feld30" id="feld30">
<option>keine</option>
<option>Frau</option>
<option>Herr</option>
<option>Mrs</option>
<option>Mr</option>
</select>
```

Entweder-oder-Auswahl mit Radiobutton

```
<p><input type="radio" name="btn1" id="wahl1" value="rot">
<label for="wahl1">Rot</label></p>
<p><input type="radio" name="btn1" id="wahl2" value="gruen">
<label for="wahl2">Grün</label></p>
<p><input type="radio" name="btn1" id="wahl3" value="blau">
<label for="wahl2">Blau</label></p>
```

Sowohl-als-auch-Auswahl mit Check-Boxes

```
<p><input type="checkbox" name="checkbox1" id="checkbox1">
<label for="checkbox">Oliven</label></p>
<p><input type="checkbox" name="checkbox2" id="checkbox2">
<label for="checkbox2">Kapern</label></p>
<p><input type="checkbox" name="checkbox3" id="checkbox3">
<label for="checkbox3">Knoblauch</label></p>
```

Weitere Formular-Elemente und -Attribute

Gruppierung

```
<fieldset>
<legend>Feldgruppe</legend>
<!-- Eingabefelder -->
</fieldset>
```

Die Beispiele bitte auf Chrome/Safari-Browser anschauen!

Das "date"-Attribut

```
<p>Datum: <input type="date" name="datum"></p>
```

Zahlen zum Auswählen

```
<p><label>Anzahl: <input name="Anzahl" type="number" value="1" min="1" max="20"
required></label></p>
```

E-Mail mit Validierung

```
<p><label>E-Mail: <input name="email" type="email" required></label></p>
```

Je nach Browser ist die Anzeige unterschiedlich.

Das Registrierungsformular

Im folgenden Beispiel soll das Registrierungsformular von oben nachgebaut, beschrieben und kommentiert werden.

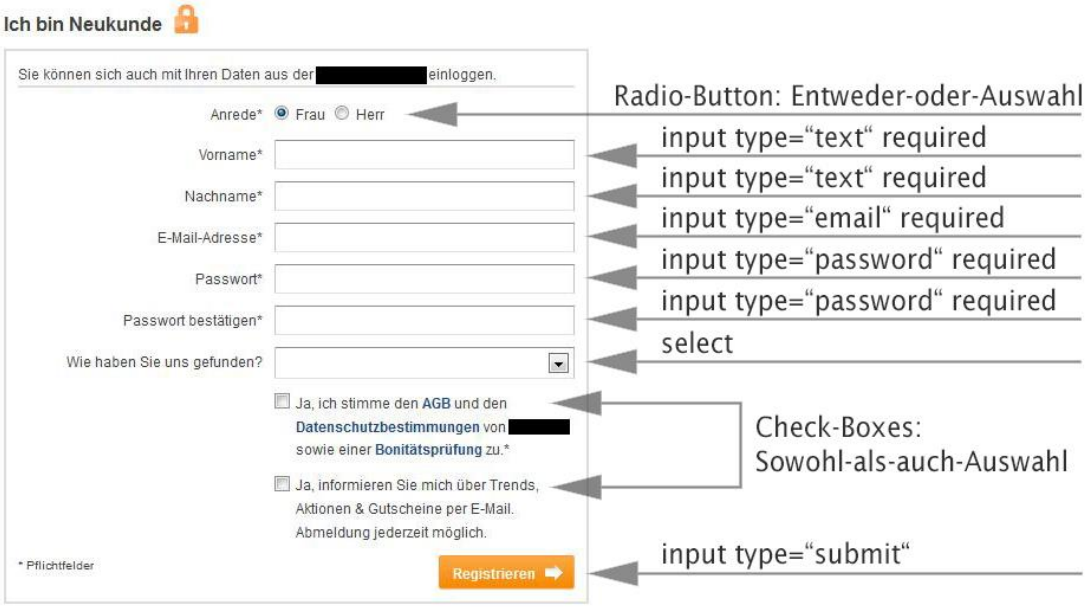


Abbildung 21: Das Registrierungsformular und seine Elemente

Das Formular als Sicherheitsrisiko

Formulare können das Ziel von Angriffen werden, die darauf beruhen, dass anstelle der erwarteten Benutzereingaben *Programm-Code* eingegeben (oder automatisch eingegeben) wird, der dann im Verarbeitungsprogramm unerwünschte Ergebnisse hervorruft.

- Einschleusen von JavaScript-Code (*Cross Site Scripting* (XSS))
- Einschleusen von Mail Header Code (***E**-Mail-Header-Injection*)
- und andere; bitte informieren Sie sich!

Das Auswerten von Formulareingaben ist nicht trivial. Umfangreiche Prüfmaßnahmen sollten durchgeführt werden, bevor die Eingaben tatsächlich ausgewertet, in Datenbanken gespeichert und zurückgegeben werden.

Speicherung von Daten auf dem Client

Die *Cookie*- und *Webstorage*-Technologie erlauben es, Daten auf dem Client-Rechner zu speichern. Solche Daten können Formulardaten sein – zum Beispiel um nicht immer die eigene Postadresse eingeben zu müssen – aber auch Datum und Thema des letzten Zugriffs.

Cookie

Das *Cookie* (Plätzchen) ist eine Textdatei mit zeitlich beschränkter Lebensdauer, die es erlaubt, dass der Betreiber einer Website Informationen auf dem Benutzer-Rechner speichert. Die Nutzer/innen von Webbrowsern können das Speichern von Cookies erlauben oder verbieten.
Häufig werden *Login-Daten* in Cookies gespeichert und beim erneuten Besuch der Website als Anmeldeinformation vorgeschlagen. Ohne Cookies funktionieren viele Websites jedoch nicht oder nur unvollständig, da sie die beim Betreten angelegte *Session-Information* benötigen. Shopsysteme verwenden Cookies in Warenkörben. Cookies können bis zu 4096 Bytes pro Cookie speichern. Jede Domain kann bis zu 20 Cookies ablegen.

Webstorage

Die *Webstorage*-Spezifikation beschreibt, wie der Webbrowser Daten auf dem lokalen Rechner speichern kann. Die Daten können zum Beispiel von Eingabe-Formularen stammen.
Weiterhin stellt die Spezifikation eine API zur Verfügung. JavaScript-Programm-Module greifen auf diese API zu. Die Webstorage-Spezifikation wird das Konzept des *Cookie* ablösen.
Webstorage - auch DOM-Storage genannt - ist eine eigene W3C-Spezifikation, die aus HTML5 ausgegliedert wurde. Je nach Implementierung im Browser können 5 Megabyte pro Domain oder mehr gespeichert werden.

Cookies/Webstorage im Vergleich

	Cookie	**Webstorage**
Lebensdauer	pro Session oder pro definierter Zeitraum	pro Session oder pro definierter Zeitraum
Speicherplatz	pro Cookie 4096 Bytes (4 kb)	pro Datei 5 Megabyte (Firefox), 10 Megabyte (IE)
Anzahl	pro Domain 20 Cookies	1 Webstorage-Objekt pro Domain
Übermittlung	Stets zusammen mit den HTTP-Anfragen (Traffic!)	Browser-kontrolliert auf Anforderung

Tabelle 10: Ausgewählte Merkmale von Cookie und Webstorage

Sicherheit und Risiken

Cookies und Webstorage-Objekte können Benutzer/innen gläsern machen, Stichwort *Privacy* (Datenschutz, Intimsphäre). Wer Cookies zulässt, surft nicht anonym. Das eigene Surfverhalten kann (und wird) kommerziell ausgewertet. Siehe Collusion!

Beispiel

- Sie interessieren sich für ein Hotel bei *HRS* oder *Booking.com*
- Seit dem erscheint auffällig viel Werbung für die begutachteten Hotel

Cookie von German Wings

Name:	s_vi
Content:	[CS]v1|28A864F18501099E-6000011560220011[CE]
Domain:	.germanwings.112.2o7.net
Pfad:	/
Senden für:	Jede Verbindungsart
Für Skript zugänglich:	Ja
Erstellt:	Dienstag, 16. April 2013 20:35:09
Läuft ab:	Donnerstag, 16. April 2015 20:35:09

Tabelle 11: Ein Cookie (entnommen aus Chrome)

Webstorage Objekte

Es gibt zwei Webstorage-Objekte:

- *sessionStorage*: Lebensdauer bis zum Schließen des Fensters
- *localStorage*: Ablage auf dem lokalen Speicher bis zum Löschen
- *Web SQL Database*: (noch sehr unvollständig implementiert)

Das Webstorage-Objekt ist immer aus einem Schlüssel und einem Wert aufgebaut.

```
Schluessel: Wert; Vorname: Otto; Nachname: Kowalsky
```

Webstorage-Objekte setzen und auslesen, Code-Beispiel (HTML)

Im HTML-Formular werden die Daten erfasst und mit Hilfe von JavaScript verarbeitet.

```
<form id="form1" onsubmit="eingaben_speichern()">
       <p><label for="feld1">Vorname</label></p>
       <p><input name="feld1" type="text" id="feld1"
onblur="eingaben_speichern()"></p>
       <p><label for="feld2">Name</label></p>
       <p><input name="feld2" type="text" id="feld2"
onblur="eingaben_speichern()"></p>
       <p><label for="feld2"> </label></p>
       <p><input name="Submit1" type="submit" value="Absenden" id="feld3"></p>
</form>
```

Webstorage-Objekt, Code-Beispiel (JavaScript)

```
function eingaben_speichern() {
   localStorage.name1=document.getElementById("feld1").value;
    localStorage.name2=document.getElementById("feld2").value;
}
function eingaben_laden() {
   vorname = localStorage.name1;
   nachname = localStorage.name2;
   alert(vorname + " " + nachname);
}
```

Anmerkung: In dem Beispiel werden stets Vorname und Nachname beim Speichern vertauscht. Auf den ersten Blick ergibt dies vielleicht keinen Sinn, jedoch wird dadurch auf einfache Weise sichtbar, wie das Speichern und Auslesen funktionieren. Ohne das Vertauschen liefe das Speichern und Auslesen zwar genauso ab, jedoch unsichtbar. Weiterhin sind im Beispiel der CSS-Code und der JavaScript-Code intern notiert.

Zusammenfassung

In diesem Abschnitt ging es um das konkrete Verfahren, wie HTML-Dokumente angelegt, gestaltet und wie externe Daten eingebettet werden.
Beim Erstellen eines HTML-Dokuments entstehen zunächst die semantischen Auszeichnungen des Inhalts. In einem der folgenden Schritte erhalten die Auszeichnungselemente ihr visuelles Erscheinungsbild. Stilregeln werden in CSS-Dateien notiert. Spezielle HTML-Elemente verweisen auf Inhalte, die aus externen Quellen stammen, wie Bilder, Videos und Sounddateien. Für HTML-Elemente, die externen Content einbetten, stellt HTML5 entsprechende APIs zur Verfügung. Mit deren Hilfe lassen sich zur Laufzeit Zeichnungen, Drag-and-Drop-Funktionalität und andere HTML5/Javascript-Anwendungen erstellen. Formulare nehmen Benutzereingaben entgegen und reichen sie an den Betreiber der Website weiter. Spezielle HTML-Elemente, die nur innerhalb des Formular-Elements gültig sind, ermöglichen spezifische Benutzereingaben. Benutzer- und Nutzungsdaten von Websites können auf dem Client-Rechner zwischengespeichert und zu späteren Zeitpunkten wieder ausgelesen werden.

Funktionale Elemente, Selektoren, Event-Attribute

Die folgenden Abschnitte beschreiben weiterführende HTML-Elemente, Selektoren und Event-Attribute, die beim Erstellen von HTML-Dokumenten häufig zum Einsatz kommen. Ein Schwerpunkt liegt auf dem Erzeugen von Buttons und Menüs.

Tabellarische Ausgabe von Daten

Der Ausgabe tabellarischer Daten, zum Beispiel den Ergebnissen von Datenbankabfragen, dienen das *table*-Element und seine Unter-Elemente. In den Anfängen des Internets wurden Tabellen zur Positionierung von Bildern und Texten verwendet. Seit CSS 2.1, das sehr präzise Positionierungen erlaubt, sollen Tabellen nur verwendet werden, wofür sie geschaffen sind: der Ausgabe von Daten in tabellarischer Anordnung. Dies reduziert nicht nur den Code-Overhead, sondern dient auch dem semantisch korrekten Web.

Aufbau einer Tabelle – die Tabellen-Elemente

Eine Tabelle besteht aus *Tabellenzeilen* (Rows, <tr>) und *Tabellenzellen* (Table Data Fields, <td>). Das visuelle Erscheinungsbild wird mit Hilfe von CSS gestaltet.
Die mit * gekennzeichneten Elemente sind für die tabellarische Darstellung von Daten von grundlegender Bedeutung.

```
<table>       *Erzeugt einen Tabellen-Container
<thead>       Erzeugt den Kopfbereich
<tfoot>       Erzeugt den Fußbereich
<tbody>       Erzeugt den Tabellenkörper
<tr>          *Erzeugt eine Tabellenzeile
<td>          *Erzeugt ein Tabellen-Datenfeld
<th>          Erzeugt eine Tabellenüberschrift, Spezialfall eines <td>-Elements
```

HTML-Tabelle, Minimalbeispiel

Das folgende Beispiel eine Tabelle, die aus den mindestens notwendigen Elementen besteht.

```
<table>
    <tr>
      <td>Dufourspitze</td>
      <td>Großglockner</td>
      <td>Zugspitze</td>
   </tr>
    <tr>
      <td>Dom</td>
      <td>Wildspitze</td>
      <td>Schneefernerkopf</td>
   </tr>
   <tr>
      <td>Liskamm</td>
      <td>Weißkugel</td>
      <td>Mittlere Wetterspitze</td>
  </tr>
</table>
```

HTML-Tabelle, Referenzbeispiel

Erheblich komfortabler für Strukturierung und Gestaltung des Erscheinungsbildes ist es, die Tabelle in den Tabellenkopf *thead*, den Tabellenfuß *tfoot* und den Tabellenkörper *tbody* zu gliedern. Selektoren lassen sich so selektiver auf den Inhalt der HTML-Elemente beziehen.

```
<table >
   <thead> <!--Kopfbereich an erster Stelle -->
      <tr>
        <th>Schweiz</th>
        <th>Österreich</th>
        <th>Deutschland</th>
      </tr>
   </thead>
   <tfoot> <!--Fußbereich an zweiter Stelle -->
      <tr>
        <td>Insgesamt 13.706 Meter</td>
        <td>Insgesamt 11.304 Meter</td>
        <td>Insgesamt 8.586 Meter</td>
      </tr>
   </tfoot>
   <tbody> <!-- Tabellenkörper an dritter Stelle -->
      <tr>
        <td>Dufourspitze</td>
        <td>Großglockner</td>
       <td>Zugspitze</td>
      </tr>
       <tr>
        <td>Dom</td>
        <td>Wildspitze</td>
        <td>Schneefernerkopf</td>
     </tr>
      <tr>
        <td>Liskamm</td>
        <td>Weißkugel</td>
        <td>Mittlere Wetterspitze</td>
      </tr>
</tbody>
</table>
```

Das Erscheinungsbild der Tabelle beruht auf CSS-Eigenschaften und deren Werte.

Ausgewählte CSS-Eigenschaften für Tabellen

Abstände und Rahmen von Tabellenzellen

Tabellenzellen werden normalerweise mit einem Abstand dargestellt. Sind zudem Rahmen vorhanden, verdoppeln sie sich dort, wo Zellen zusammenstoßen. Dies sieht nicht schön aus. Die Eigenschaft *border-collapse* beeinflusst die Zelltrennung. Die Eigenschaft gehört zum *table*-Element.

```
border-collapse: collapse; /*Zellen werden ohne Abstände gerendert. */
border-collapse: separate; /* Zellen werden getrennt gerendert */
```

Die *padding*-Eigenschaft der Tabellenzellen beeinflusst den Rundum-Abstand um den Zellinhalt.

```
padding:10px;
```

Tabellenzeilen alternierend einfärben

Alternierende Hintergrundfarben von Tabellenzeilen erhöhen die Übersichtlichkeit.

```
tr:nth-child(2n+1) {background-coor:silver;}
```

Im hier gezeigten Beispiel wird jede ungerade Zeile eingefärbt (2n+1). Wer jede Dritte Zeile verändern möchte, benutzt die Folge (3n).

Listen

HTML bieten Autoren mehrere Mechanismen an, um Information in Form von Listen darzustellen. HTML-Listen können

- Geordnete/nummerierte
- Ungeordnete/unnummerierte
- Zu definierende

Informationen enthalten.
Listen bestehen stets aus einem Element, das Detail-Elemente umschließt, die nur im Kontext des umschließenden Elements sinnvoll sind.

Nummerierte Liste, geordnete Liste (ordered list, ol)

Das *ol*-Listenelement wird für nummerierte Aufzählungen verwendet. Das *start*-Attribut setzt den Startpunkt des Zählers. Das *li*-Element kennzeichnet die Listeneinträge.

```
<ol start=9>
  <li>Schmidt</li>
  <li>Schmider</li>
  <li>Schmalhans</li>
</ol>
```

Per Default beginnt der Zähler bei eins. Anwendungsbeispiele können Ranglisten sein.

Unnummerierte Liste, ungeordnete Liste (unordered list, ul)

Das *ul*-Listenelement wird für Spiegelstrich-Aufzählungen verwendet. Das *li*-Element kennzeichnet die Listeneinträge.

```
<ul>
  <li>Müller</li>
  <li>Meier</li>
  <li>Mahlstein </li>
</ul>
```

Der Wert der CSS-Eigenschaft *list-style-type* bestimmt das Aufzählungszeichen.

```
.quadrat {list-style-type: square;}
.grafik {list-style-image: url(ordner.png);}
```

Der Verweis zu einer Grafik erlaubt beliebige Aufzählungszeichen.

Listen schachteln

Ungeordnete Listen können geschachtelt werden.

```
<ul>
  <li>Datei
    <ul>
      <li>Öffnen</li>
      <li>Speichern</li>
      <li>Speichern unter...</li>
      <li>Schließen</li>
    </ul>
  </li>
```

```
</ul>
```

Die Schachtelung machen sich Webdesigner beim Gestalten von Menüleisten zunutze.

Definitionsliste

Die Definitionsliste (dl) benötigt drei HTML-Elemente. Sie dient zum Auflisten von Eigenschafts-/Wertepaaren oder Begriffs-/Wertepaaren, also zu Definitionen.

```
<dl>
<dt>Zur definierender Term (definition term, dt)</dt> <dd>Seine Definition</dd>
   <dt>Farbe</dt> <dd>rot</dd>
  <dt>Material</dt> <dd>Plastik</dd>
  <dt>Form</dt><dd>Ellipsoid</dd>
</dl>
```

Anmerkung: Die Definitionsliste findet trotz ihrer Möglichkeiten nur selten Anwendung.

Listen als Grundlage der Navigationsstruktur

Listen dienen dazu, um Begriffe, Überschriften und Textblöcke aufzuzählen und anzuordnen. Noch häufiger jedoch bleiben sie unsichtbar – sind aber dennoch überall präsent. Hinter jedem semantisch korrekten Menü, jeder Menüleiste, egal ob senkrecht oder waagrecht angeordnet, verbirgt sich eine *unordered list* (*ul*).

```
<h2>Bekannte Suchmaschinen</h2>
   <ul>
      <li><a href="https://www.google.de" >Google</a></li>
      <li><a href="http://www.bing.com" >Bing</a></li>
      <li><a href="https://de.search.yahoo.com" >Yahoo (=Bing)</a></li>
      <li><a href="http://suche.t-online.de">T-Online (=Google)</a></li>
    </ul>
```

Die Navigation bei Webseiten beruht auf dem Anchor-Element *<a>* und seinem speziell definierten Erscheinungsbild.

Das Anchor-Element (<a>)

Das Anchor-Element war ursprünglich dazu gedacht, um von dem speziell als Anchor ausgezeichneten Wort zu einer anderen Stelle im Dokument oder zu einem anderen Dokument zu springen. Das ist die grundlegende Technologie des Hyperlinks. Die verlinkte Stelle oder Seite kann im selben Browserfenster, in einem neuen Browserfenster oder in einem Fensterrahmen dargestellt werden. Das Anchor-Element funktioniert mit den Attributen *href*, *target*, *title* und den globalen Attributen wie *class*, *id*, *lang*, *style* und anderen.

```
<a href="http://www.gugel.de" >Gugel</a>
```

Das Wort “Google” stellt einen aktiven Link dar. Damit der Link funktioniert, muss das Attribut *href* den Wert einer gültigen URL annehmen.

Weitere Attribute

```
<a href="http://www.gugel.de" target="_blank">Gugel</a>
<a href="http://www.gugel.de" target="_blank" title="zur Wurstfabrik">Gugel</a>
```

Um im selben Dokument zu springen, enthält die URL den Namen einer ID.
Auch eine Grafik kann als Platzhalter für das Link-Wort dienen.

Zielangaben bei Links

Absolute Zielangaben bei Links

Der *absolute Verweis* enthält die vollständige Zieladresse.

```
<a href="http://www.gugel.de" title=" zur Wurstfabrik">Zu Gugel /a>
```

Relative Zielangaben

Der **relative Verweis** enthält die relative Pfadangabe innerhalb des Webseitenordners.

```
<a href="../html/seite01.html" title="Seite 1" target="_blank">Die erste Seite</a>
```

Selektoren für die Darstellung von Links

Das *a*-Element ist zunächst ein Inline-Element, es erscheint stets innerhalb einer Zeile. Wie es letztlich dargestellt, ob innerhalb des Fließtextes oder als eigenständige Zeile, wird mit Hilfe von CSS festgelegt.

Buttons

Die Eigenschaft *list-style-type: none* entfernt das Aufzählungszeichen einer Liste.

```
ul {
	list-style-type: none;
}
ul li {
	margin-top: 2px;
	margin-bottom: 2px;
}
ul li a {
	text-decoration: none;
	background-color: #36C;
	color: #FFF;
	padding-right: 10px;
	padding-left: 10px;
}
```

Die *margin*-Eigenschaften erzeugen im Beispiel vertikale Abstände zwischen den Listen-Elementen. Die text-decoration: none – Eigenschaft entfernt den Unterstrich bei Links. *Padding* erzeugt links und rechts etwas Luft um den Text herum.

Menüs auf der Basis von Listen

Das *ul*-Element für die ungeordnete Liste dient als Container für Menüs. Menüs sind nichts anderes als Links, die mit Hilfe von CSS als Block-Elemente formatiert wurden. In Menüs ist das **a**-Element stets als Block formatiert.

Vertikales Menü (nur die Unterschiede)

```
ul {
	width: 200px;
}
ul li a {
	display: block;
}
```

Die Eigenschaft *display:block* zwingt das Inline-Element, sich wie ein Block-Element zu verhalten. Beim *vertikalen Menü* wird die Breite durch die *width*-Eigenschaft des *ul*-Containers (oder des umgebenden *nav*-Containers) bestimmt.

Horizontal (nur die Unterschiede)

```
ul {
	list-style-type: none;
}
ul li {
	float: left;
	margin-right: 2px;
	margin-left: 2px;
	width: 200px;
}
```

Beim *horizontalen Menü* floatet das *li*-Listenelement. Es ist in seiner Breite definiert, während das *ul*-Element eine automatische Breite einnimmt.

Horizontal, volle Breite (nur Unterschiede)

```
ul {
	list-style-type: none;
	margin:0; padding:0;
}

ul li {
	margin-right: 2px;
	margin-left: 2px;
	width: calc(25% - 8px);
	display: inline-block;
	box-sizing: border-box;
}
```

Anstatt zu floaten, kann das *li*-Listenelement auch als *inline-block* definiert werden. Das Element verhält sich dann einerseits wie ein Block (Breite, Margin), als auch wie ein Inline-Element (Anordnung innerhalb einer Zeile).
Die *box-sizing*-Eigenschaft priorisiert die Abmessungen des HTML-Elements – hier des *li*-Elements. Interessant ist der *calc*-Wert. Dadurch ist es möglich, Prozentangaben und Pixelangaben zu mischen, ohne sich dem Problem der Umrechnung stellen zu müssen.

Kachel-Optik (nur Unterschiede)

```
ul li a {
	padding-top: 200px;
}
```

Die Kachel-Optik entsteht durch das gezielte Einsetzen des *Padding*.

Aktive Schaltflächen

Je nach Maus- und Tastatur-Aktion nehmen Links unterschiedliche Zustände ein, auf die mit Hilfe von sogenannten *Pseudoklassen* zugegriffen werden kann. Auf die Reihenfolge der Notierung kommt es an, da ansonsten Eigenschaften überschrieben werden.

```
ul li a:visited {
	text-decoration: none;
	background-color: #999;
	color: #FFF;
	display: block;
	padding-right: 10px;
	padding-left: 10px;
```

```
        padding-top: 200px;
}
ul li a:hover {
        background-color: #393;
        color: #FFF;
}
ul li a:focus {
        background-color: #CF0;
        color: #000;
}
ul li a:active {
        background-color: #FC3;
        color: #000;
}
```

Notieren Sie zunächst stets die Eigenschaften für das a-Element, dann für die Pseudoklassen in genau dieser Reihenfolge: *visited*, *hover*, *focus*, *active*.

```
a {Regel} (Regel gilt für alle Button-Zustände)
a:link {Regel} (Regel gilt für den Button-Zustand 'als Link')
a:visited {Regel} (Regel gilt für den Button-Zustand 'bereits besuchter Link')
a:hover {Regel} (Regel gilt, wenn der Mauszeiger über dem Link schwebt)
a:active {Regel} (Regel gilt, solange der Link gedrückt wird)
a:focus {Regel} (Regel gilt, wenn der Link mit der Tastatur erreicht wurde)
```

Systematisierung der Selektoren

Der Abschnitt, der sich mit aktiven Schaltflächen befasste, macht es notwendig, die Selektoren näher zu betrachten. Schon die *Pseudoklassen* zeigten, dass neue Notierungen – hier der Doppelpunkt – neue Möglichkeiten bieten. Im weiteren Verlauf dieses Abschnitts sollen noch weitere spezielle Selektoren beschrieben werden, die entweder Neues ermöglichen oder das Schreiben von Stylesheets vereinfachen, in dem sie dessen Komplexität reduzieren.
Neben den *Element-, Klassen-, ID* Selektoren, den daraus erstellbaren *Kombinatoren* und dem *Universalselektor* unterscheidet die W3C-Empfehlung weitere Kombinatoren und zudem *Pseudoklassen* und *Pseudoelemente*.

- Den Nachfolge-Kombinator
- Den Kind-Kombinator
- Zwei Arten von Geschwister-Kombinatoren
- Pseudo-Klassen
- Pseudo-Elemente

➡ **Das zugehörige W3C-Dokument**

Ausgewählte Selektoren werden in den folgenden Abschnitten dargestellt.

Ausgewählte Beispiele für Selektoren in CSS Level 3

Attribut-Selektor
Selektiert Elemente anhand ihrer Attribute (Weitere Beispiele siehe o.g. W3C-Dokument)

```
p[title]{color:red;}
p[title="Ankunft"] {color:red;}
```

Nachfolge-Selektor
Der Nachfolge-Selektor bezieht sich auf alle *p*-Elemente, die in der Baumstruktur dem div-Element an irgendeiner Stelle folgen.

```
div p {color: #03F;}
```

Kind-Selektor
Der Kind-Selektor selektiert das *p*-Element, das in der Baumstruktur dem div-Element direkt folgt.

```
div > p {color: #03F;}
```

Der n-te Kind-Selektor (Strukturelle Pseudoklasse)
Selektiert bestimmte oder mit einer Formel errechnete HTML-Kind-Elemente (Weitere Beispiele siehe o.g. W3C-Dokument).

```
p:nth-child(2) {background-color: #03F}
p:nth-child(2n+1) {background-color: #03F}
```

Geschwister-Selektor (1)
Das Element, das direkt folgt und den gleichen Rang hat

```
h1 + p {font-weight: bold} /* Der erste Absatz nach der Hauptüberschrift */
```

Geschwister-Selektor (2)
Selektiert alle *p*-Elemente, die der Hauptüberschrift folgen

```
h1 ~ p {color:red}
```

Generated Content
Vor oder hinter dem HTML-Element wird Text oder ein Bild eingefügt

```
p:before {content: "-> "} /* Vor jedem Absatz wird ein Pfeil eingefügt */
p:before {url(pfeil.png);} /* Vor jedem Absatz wird die Bilddatei 'pfeil.png'
eingefügt */
```

Pseudo-Klassen und Pseudo-Elemente

Pseudo-Klassen

Klassen beziehen sich auf HTML-Elemente, *Pseudo-Klassen* beziehen sich auf die Zustände, die HTML-Elemente annehmen können. Solche „Zustände“ entstehen beispielsweise beim Überfahren eines HTML-Elements mit der Maus. Der Zustand „Maus schwebt über Element“ (:*hover*) ist keine Klasse wie zum Beispiel *.rot*, die fest mit dem Element verbunden wäre, sondern die nur dann in Erscheinung tritt, wenn der Mauszeiger die entsprechende Position einnimmt.
Pseudo-Klassen beziehen sich also auf Informationen, die nicht von der Dokumentstruktur gegeben sind, sondern die erst durch das Anzeigen des Dokuments sichtbar werden können. Pseudo-Klassen werden mit einem Doppelpunkt notiert.
Beispiele für Pseudoklassen sind :*link*, :*visited*, :*hover*, :*focus*, :*active, :target* und andere.
Die Pseudo-Klasse :hover reagiert beim Überfahren eines HTML-Elements mit der Maus.

```
<#u2:hover {background-color:red; color: white};
```

Anmerkung: Das Hovern sollte auf Links beschränkt bleiben, um keine Irritationen hervorzurufen. Prinzipiell lässt sich auch das *body*-Element hovern, was aber keinen Sinn ergibt.

Die Pseudo-Klasse *:target* erhält dann Bedeutung, wenn sie Ziel-Element eines Link ist. Wird das Element beim Aufruf seiner ID angesteuert - seines Identifikators -, wird die Pseudo-Klasse *:target* aktiv. *Reset* steuert eine beliebige andere ID an, für die keine Pseudo-Klasse *:target* definiert ist.

HTML

```
<a href="#box1">rote Box ändern</a> | <a href="#egal">Reset</a>
```

CSS

```
#box1 {background-color: #F30; height: 100px; width: 100px;}
#box1:target { background-color: #9C0; height: 75px; width: 150px;}
```

Pseudo-Elemente?

HTML-Elemente geben die Dokumentstruktur vor. *Pseudo-Elemente* beziehen sich dagegen auf Teile von HTML-Elementen, es sind gewissermaßen Extrakte. Pseudo-Elemente werden mit zwei Doppelpunkten notiert. Der Doppelpunkt ist noch nicht in allen Browsern implementiert. Häufig werden Pseudo-Klassen und Pseudo-Elemente auf gleiche Weise notiert.

Das *::first-letter*-Beispiel zeigt, wie mit HTML und CSS *Initialen* erzeugt werden können.

```
p::first-letter {Eigenschaft:Wert; Eigenschaft: Wert; usw.}
```

Im *::first-line*-Beispiel wird stets die erste Zeile eines Absatzes in spezieller Weise formatiert.

```
p::first-line {Eigenschaft:Wert; Eigenschaft: Wert; usw.}
```

Anwendungsbeispiele

Menü „Am Meer“, vertikales Menü

Abbildung 22: Webseite "Am Meer"

Menü „Die größten Flughäfen der Welt“, horizontales Menü

Abbildung 23: "Flughäfen", horizontales Menü

Menü „Die größten Flughäfen der Welt“, vertikales Menü

Abbildung 24: "Flughäfen", vertikales Menü

Anführungszeichen, Zähler, Tooltips

Anführungszeichen automatisieren

Das Beispiel zeigt, wie je nach Sprach-Attribut unterschiedliche, landesspezifische Anführungszeichen gesetzt werden, um Zitate zu kennzeichnen. Zunächst werden die Anführungszeichen in der Unicode-Hexadezimal-Schreibweise definiert und zwar hier für die Elemente *q* und *blockquote*, abhängig davon welches Language-Attribut das umgebende *section*-Element besitzt.

```
section[lang="de"] q, section[lang="de"] blockquote {quotes: "\201E" "\201C";}
section[lang="fr"] q, section[lang="fr"] blockquote { quotes: "\00ab" "\00bb";}
```

Die Pseudo-Klassen *:before* und *:after* fügen die Anführungszeichen vor und nach den definierten Elementen ein.

```
q:before, blockquote p:before {content:open-quote;}
q:after, blockquote p:after {content:close-quote;}
```

Einen Zähler mit CSS gestalten
Wem die *ordered list (ol)* zu wenige Möglichkeiten bietet, kann mit Hilfe von CSS einen eigenen Zähler schreiben.

Die Zählereigenschaft im übergeordneten Element setzt den Zähler zurück.

```
body {counter-reset: Ebene1}
```

In jeder zu nummerierenden Überschrift fügt die Pseudo-Klasse *:before* den aktuellen Zählerstand ein und erhöht ihn danach um eins.

```
h2:before {content: counter(Ebene1)") "; counter-increment: Ebene1; }
```

Mit CSS Tooltips gestalten
:Before- und *:After*-Pseudo-Klassen versagen normalerweise beim Auslesen der Attribute des *img*-Elements. Deshalb wurde in diesem Beispiel ein *div*-Element um das *img*-Element gelegt, um dessen Attribute auszulesen.

```
.bild1:hover:after {Eigenschaft:Wert; Eigenschaft: Wert; usw.}
```

Beim Überfahren mit der Maus wird das Attribut des *div*-Elements als Tooltip angezeigt.
Schön wäre es, eine zeitliche Verzögerung anzubringen. Auch das geht mit CSS3.

Für CSS Level 4 geplante Selektoren

Die Arbeit an der Empfehlung *CSS Level 4* sieht vor, das Konzept der Selektoren, Pseudoklassen und Kombinatoren weiter zu verfeinern, vor allem in Hinblick auf die Pseudoklassen.

- Neue Pseudo-Klassen für Links
- Neue Pseudoklassen für die Anzeige von Elementen
- Neue Pseudoklassen speziell für Formulare
- Neue Pseudoklassen für Gitter-Strukturen

➡ **CSS4-W3C-Dokument (Editors Draft)**

Interaktivität mit JavaScript

Events, Event-Attribute, Event-Handler

Alle HTML-Elemente akzeptieren Events (Ereignisse). Damit sie dies auch wirklich tun und auf die Events reagieren, werden Event-Attribute notiert und mit einem Event-Handler, einer JavaScript-Verarbeitungsroutine, verbunden. Events sind beispielsweise:

```
onload (Beim Laden eines Dokuments)
onclick (Beim Klicken auf das Element)
onblur (Beim Verlassen des Elements)
```

Im folgenden Beispiel akzeptiert das *h1*-Element den Maus-Klick und zeigt eine Text-Box an, weil mit dem Event-Attribut onclick der entsprechende JavaScript-Befehl verbunden ist.

```
<h2 onclick="javascript:alert('Hallo Welt!')">Die Überschrift reagiert auf einen
Maus-Klick und liefert als Verarbeitungsergebnis die Text-Box „Hallo Welt!“.</h2>
```

Die CSS-Eigenschaft *cursor:pointer* erzeugt den Hand-Mauszeiger. Er macht darauf aufmerksam, die Zeile anzuklicken, entspricht der Konvention.

```
.klick {cursor: pointer;}
```

Weitere Events

- onchange
- onclick
- ondblclick
- onerror
- onfocus
- onkeydown
- onkeypress
- onkeyup
- onload
- onmousedown
- onmousemove
- onmouseout
- onmouseover
- onmouseup
- onreset
- onselect
- onsubmit
- onunload

Formularbeispiel

Das folgende Beispiel benutzt die Event-Attribute *onmouseover*, *onmouseout* und *onsubmit*. Die JavaScript-Befehle sind in einer externen Datei gesammelt und zu Funktionen zusammengefasst. Die Event-Attribute sind direkt mit diesen Funktionen verknüpft. Die JavaScript-Datei ist im Kopfbereich des HTML-Dokuments eingebunden – nach allen CSS-Dateien. Gelegentlich werden JavaScript-Dateien auch ganz unten, vor dem schließenden *body*-Element notiert.

Arbeiten mit jQuery

Das Motto von jQuery „write less, do more“ bedeutet, dass Web-Entwickler mit weniger Programmieraufwand zu besseren Ergebnissen kommen. Statt für eine clientseitige Web-Anwendung jeden Befehl selbst zu programmieren und in verschiedenen Kontexten zu debuggen, stellt die jQuery-JavaScript-Bibliothek zahlreiche erprobte und ständig gepflegte Programmbausteine zur Verfügung, die kombiniert werden können.

> *„jQuery ist die meistverwendete JavaScript-Bibliothek. Jede zweite Website und drei Viertel der 10.000 meistbesuchten Websites nutzen jQuery (Stand: Juli 2014).“ (http://de.wikipedia.org/wiki/JQuery)*

jQuery-Anwendungen bestehen aus einer Basis-Bibliothek (http://jquery.com/), meist kommen eine zusätzliche spezielle Funktionsbibliothek und selbst erstellter Programm-Code hinzu.

Zusammenfassung

Das Kapitel befasste sich mit ausgewählten HTML-Elementen, die den Seitentext in besonderer Weise strukturiert auszeichnen, Tabellen und Listen. Das Listen-Element dient häufig zum Aufbau von vertikalen und horizontalen Menüleisten. CSS-Eigenschaften, Klassen, Pseudo-Klassen und Pseudo-Elemente helfen dabei, Menüs und andere Text-Elemente so darzustellen, wie es den Sehgewohnheiten bei Menüs und Buttons entspricht. Die Selektoren lassen sich differenzieren und systematisieren. Rund 20 Arten von Selektoren sind in der CSS3-Spezifikation beschrieben. Die Selektoren ermöglichen überraschende Anwendungen und Funktionalitäten. Die Grenze zwischen Auszeichnung und Programmierung verwischt. Clientseitige Webanwendungen lassen sich mit JavaScript realisieren. Die empfohlene Vorgehensweise besteht darin, Funktionsbibliotheken zu verwenden wie zum Beispiel jQuery.

Gestalterische Aspekte und Usability

Das Kapitel befasst sich zunächst mit ausgewählten HTML-Elemente und Stilregeln zur Text- und Textblockgestaltung, um dann auf allgemeine Gestaltungsprinzipien von HTML-Dokumenten und Webseiten einzugehen.

Schriften

Merkmale von Schriften

Die Schriften oder Schriftarten bezeichnen das visuelle Erscheinungsbild von Textinformation. Schriften, die gleichartige Merkmale aufweisen, heißten Schriftfamilien (*font-family*). Die folgenden Buchstaben gehören zur selben Schriftfamilie, sie unterscheiden sich jedoch in ihrem Gewicht (*font-weight*), ihrer Größe (*font-size*), ihrem Stil (*font-style*) und ihrer Darstellung (*font-variant*).

b b b b b b	Unterschiedliche Schriftgröße *font-size*
b **b**	Unterschiedliches Schriftgewicht *font-weight*
b *b*	Unterschiedlicher Schriftstil *font-style*
b B	Unterschiedliche Darstellung *font-variant*
iii	iii Der Buchstabe „i“ zeigt den Platzbedarf derselben Buchstaben. Links: Propotionalschrift, rechts: Schrift mit festem Buchstabenabstand
mmm	mmm
Wetter	Serifenlose Schrift, hier *Century Gothic*
Wetter	Serifenschrift, hier *Century*
Wetter	Schrift mit festem Abstand, hier `Courier`
▶ ✔ ✈ ✖	Symbolschriftart, hier *Webdings*
abc **abc**	Unterschiedlichen Platzbedarf bei gleicher Schriftgröße, hier *Times* und ***Rockwell***.

Tabelle 12: Merkmale von Schriften

Die Schriftfamilien gehören im westlichen Sprachraum vor allem entweder den Gruppen „Serifenschrift“ oder „serifenlose Schrift“ an. Serifen – die angedeuteten Verbindungslinien zwischen den Buchstaben – verbessern vor allem bei Print-Dokumenten mit langen Zeilen und Absätzen den Lesefluss. Auf Monitoren mit geringer Auflösung wie 72 dpi oder 96 dpi erzeugen sie eher ein undeutlich wirkendes Schriftbild. Hier sollte man eher serifenlose Schriften verwenden. Auf

Handy- und Tablet-Displays mit Auflösungen, die an 200 dpi heranreichen, sind Serifenschriften dagegen bereits recht gut lesbar.
Ausgewählte Schrifteigenschaften

```
font-size: 36px; /* Schriftgröße */
font-weight: bold; /* Fett */
font-style: italic; /* Kursiv */
font-variant: small-caps; /* Kapitälchen */
font-family: Verdana; /* Schriftgröße */
color: #C00; /* Schriftfarbe */
background-color: #CF6; /* Schrift-Hintergrundfarbe */
```

Weitere Schriftmerkmale sind

- *Proportionalschriften*: Jeder Buchstabe verwendet die für den angenehmen Lesefluss optimale Breite, geeignet für alle Arten von Texten.
- *Schriften mit fester Zeichenbreite*: Jeder Buchstabe verwendet dieselbe Zeichenbreite; geeignet für Code- und Formelsatz.
- *Symbolschriften*: Anstelle von Buchstaben enthalten die Zeichensätze Formen und Symbole.

Unterschiedliche Schriften haben auch einen unterschiedlichen Platzbedarf sowohl in der Breite als auch in der Zeilenhöhe.

Auflösung

Die Auflösung eines Displays oder einer Grafik bezeichnet die Dichte der Bildpunkte. Sie wird in *Dots per Inch* gemessen (dpi). Je höher die Dichte, desto harmonischer und angenehmer ist der Eindruck.

Schriftgrößen

Die Abstufungen der Schriftgrößen liegen traditionell seit dem 16. Jahrhundert bei: 72, 60, 48, 36, 24, 21, 18, 16, 14, 12, 11, 10, 9, 8, 7, 6. Für mobile Webseiten gilt dies jedoch nur eingeschränkt.

Systemfonts und Webfonts

Ersatzschriftarten

Um System-Schriften darzustellen, müssen die Schriften auf dem Client-Computer installiert sein, der die Website anzeigt. Gängige Schriften, die auf praktischen allen Systemen vorhanden sind, sind *Arial*, *Verdana*, *Times New Roman*, *Helvetica*.
Die *font*-Eigenschaft erlaubt das Notieren von Ersatzschriftarten, die dann automatisch zum Einsatz kommen, wenn die erst genannte Schriftart auf dem Client-Computer nicht vorhanden. Font-Family mit Ersatzschriftarten:

```
body {font-family:Verdana, Arial, sans-serif}
body {font-family: Otto, "Courier New", Courier, monospace;}
```

Als letzte Ersatzschriftart sollte stets *sans-serif*, *serif* oder *monospace* notiert werden, je nachdem ob die gewünschte Schriftart eher einer Serifenschrift, einer serifenlosen Schrift oder einer Schrift mit festem Buchstabenabstand gleicht.

Webfonts

Eine andere Möglichkeit besteht darin, Webfonts zu verwenden. Sie setzen eine Verbindung mit dem Internet voraus, weil Webfonts temporär herunter geladen werden. Eine Offline-Anzeige ist nicht möglich. Inzwischen existiert eine große Auswahl an Webfonts im Internet, auch an kostenlosen Schriftarten. Google liefert gleich den HTML-Code und die CSS-Eigenschaften zur richtigen Benutzung mit. Webfonts werden zunächst importiert, dann definiert. Ausgewählt sind hier die Schriften mit der Stärke 400 (normal) und 700 (fett).

```
@import url(http://fonts.googleapis.com/css?family=Oleo+Script:400,700);
h1, h2 {font-family: 'Oleo Script', cursive;}
```

Alternativ wird ein Verweis im Kopfbereich des HTML-Dokuments notiert.

```
<link href='http://fonts.googleapis.com/css?family=Oleo+Script' rel='stylesheet'
type='text/css'>
h1, h2 {font-family: 'Oleo Script', cursive;}
```

Das *Font Awesome* Projekt (*awesome* dt.: fantastisch, großartig) stellt zahllose Icons und Symbole mit GPL-freundlicher Lizenz zur Verfügung.
Die Awesome-Fonts werden direkt aus dem *CDN* (Content Delivery Network) *MaxCDN* heraus aufzurufen und mit CSS-Klassen verfügbar gemacht.

```
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/font-awesome/4.3.0/css/font-
awesome.min.css">
```

Eine andere Möglichkeit besteht darin, die Schriftschnitte und ein mitgeliefertes CSS-Dokument herunterzuladen und in den HTML-Code einzubinden. Die Awesome-Schriftzeichen sind stets an ein Inline-Element gebunden, hier das *i*-Element.

```
<li><i class="fa-li fa fa-check-square"></i>List icons</li>
```

Schriftarten vergleichen

Ein hübsches Tool, um verschiedene Schriftarten zu vergleichen, gibt es bei *typetester.org*.

➡ **Siehe typetester.org**

Weitere Infos zu Schriften:

➡ **W3C Schrift-Modul, alle Schrifteigenschaften**
➡ **Typografisches Grundwissen (W3C)**

Textblöcke

Absätze

Absätze sind hauptsächlich gekennzeichnet durch ihre Breite (*width*), ihre Ausrichtung (*text-align*) ihre Zeilenhöhe (*line-height*) und den Erstzeileneinzug (*text-indent*). Die Eigenschaften sind im W3C-Textmodul beschrieben.

➡ **W3C Textmodul**

Die Absatzhöhe (*height*) sollte bei Massentexten nicht angegeben werden, da sie stets automatisch mit der Textmenge wachsen sollte. Sind jedoch Textblöcke mit einer genau definierten Höhe erforderlich, ist die Angabe der Absatzhöhe sinnvoll. Die Eigenschaft *text-overflow* legt dann fest, was mit überlappendem oder überlaufendem Text geschehen soll.

Einrückungen

Semantische Einrückungen werden mit HTML realisiert.

```
<blockquote>Eingerückter Text</blockquote>
```

Kommt es nicht darauf an, eine bestimmte Bedeutung mit der Einrückung zu verbinden, werden CSS-Eigenschaften verwendet.

```
p {margin-left: 36px; background-color: #ccc }
p {padding-left: 36px; background-color: #ccc}
```

Textblöcke: Listen (HTML)

Um semantische Listen zu erzeugen, dienen die Elemente *ul* und *ol*. *Ul* liefert ungeordnete, unnummerierte Listen, *ol* geordnete, nummerierte Listen. Listen können auch ausschließlich mit CSS erzeugt werden, sofern sie lediglich der Gestaltung, nicht aber der Bedeutungszuweisung dienen. Hilfreich ist dann die Pseudoklasse *:before* in Verbindung mit der *content*-Eigenschaft.

Semantische Listen, ungeordnet, unnummeriert

```
<ul>
<li>Element 1</li>
<li>Element 2</li>
<li>Element 3</li>
</ul>
```

Semantische Listen, geordnet, nummeriert

```
<ol>
<li>Element 1</li>
<li>Element 2</li>
<li>Element 3</li>
</ol>
```

CSS-Listen, unnummeriert

```
p:before  {content: "\21D2  ";}
```

CSS-Listen, nummeriert

```
body {counter-reset: Liste2;}
p:before {content: counter(Liste2)". "; counter-increment: Liste2; }
```

➡ **Weitere Infos bei W3C, Listen.**

Typografische Regeln

„Typografie ist das Rückgrat guten Webdesigns.“
Und der Usability!

Die Zeilenlänge eines Textes sollte 40 - 80 Zeichen, optimal 65 Zeichen, betragen. Die Faustregel besagt: Schriftgröße x 30.

```
p {font-size: 16px; max-width: 480px;}
```

Der Zeilenabstand sollte mindestens das 1,4-fache der Schriftart-Größe sein. Auf jeden Fall sollte der Zeilenabstand größer als der Wortabstand sein. Weiße Schrift auf dunklem Grund benötigt größeren Zeilenabstand, damit der Eindruck gleichbleibt.

```
body { font-family: Arial; font-size: 16px; line-height: 1.4em;}
```

Die Zeilen sollten sich, auch über mehrere Spalten hinweg, immer an derselben Grundlinie orientieren. Die wird erreicht, indem die Absatzabstände dem Zeilenabstand oder einem Vielfachen davon entsprechen.

```
body {font-family: Arial; font-size: 16px; line-height: 1.4em;}
p {margin-bottom: 1.4em;}
```

Farben

Farbräume und Farbmodelle

Farbräume bezeichnen die Gesamtheit der durch ein Farbmodell darstellbaren Farben.

Das *additive Farbmodell* bezieht sich auf Monitore und elektronische Displays. Ihm liegt der *RGB-Farbraum* zugrunde. Aus den Farben *Rot*, *Grün* und *Blau* entstehen in 16,7 Millionen Abstufungen die wahrnehmbaren Farben sowie *Schwarz* und *Weiß*.

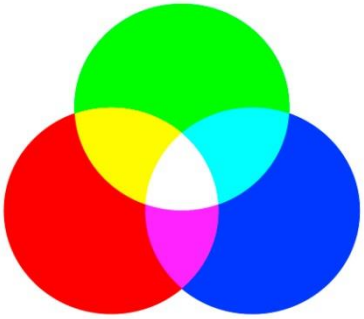
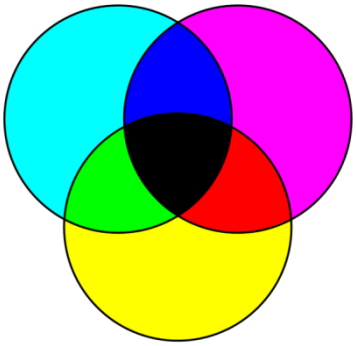
Im Screendesign relevant	Im Printdesign relevant
	
RGB-Farbraum Rot, Grün, Blau	CMYK-Farbraum Cyan, Magenta, Yellow

Tabelle 13: RGB- und CMYK-Farbraum

Das *subtraktive Farbmodell* wird bei der Herstellung von Printerzeugnissen verwendet. Die Grundfarben gehören dem *CMYK-Farbraum* an, *Cyan*, *Magenta*, *Yellow* und einer *Key-Farbe*, die den Dunkelheitsgrad der jeweiligen Farbe definiert. Auch auf diesen Farben entstehen durch subtraktive Farbmischung beliebig viele wahrnehmbare Farben.
Farben, die aus den beiden unterschiedlichen Farbmodellen erzeugt werden, sind nicht hundertprozentig kompatibel, sie sind rechnerisch nicht konvertierbar. Bei der Umrechnung kommt es stets zu geringen Abweichungen, die manchmal stärker, manchmal weniger stark sichtbar sind. Von Werbeagenturen gelieferte Grafiken müssen häufig in den RGB-Farbraum umgerechnet werden, damit sie für die Verwendung im Internet geeignet sind.
Eine mögliche Darstellungsform des RGB-Farbraums ist die Abbildung von *Farbton* (engl.: *Hue*), *Sättigung* (engl.: *Saturation*) und *Helligkeitsstufe* (engl.: *Value*). Der Farbton läuft von 0 Grad - 360 Grad, die Sättigung von 0% - 100%, ebenso die Helligkeitsstufe.

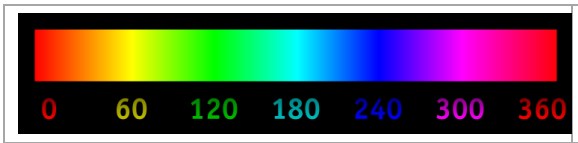
		
Farbton	Sättigung	Helligkeit

Tabelle 14: Farbton, Sättigung, Helligkeit

Abbildung 25: Farb-Rad

Die Farbtöne lassen sich in einem Farb-Rad darstellen. Jeweils gegenüber befinden sich die Komplementärfarben. „Rot“ steht oben, wegen Hue = 0°. Die Farben lösen bei den meisten Menschen dieselben oder ähnliche Empfindungen aus. Die Farben im linken Bereich werden als „kalt“ und die im rechten Bereich als „warm“ empfunden.
Das Farb-Rad hilft beim Erstellen einen Farbschemas. Jedem Screendesign liegt ein Farbschema zugrunde. Farbschemata sorgen für Harmonie und Kontrast. Sie sind entscheidend für die visuelle Identität eines Internet- oder eines Print-Dokuments. Webseiten, die keinem Farbschema folgen, wirken wirr und chaotisch. Es leidet die Übersichtlichkeit. Damit einher geht eine Verschlechterung der Usability.

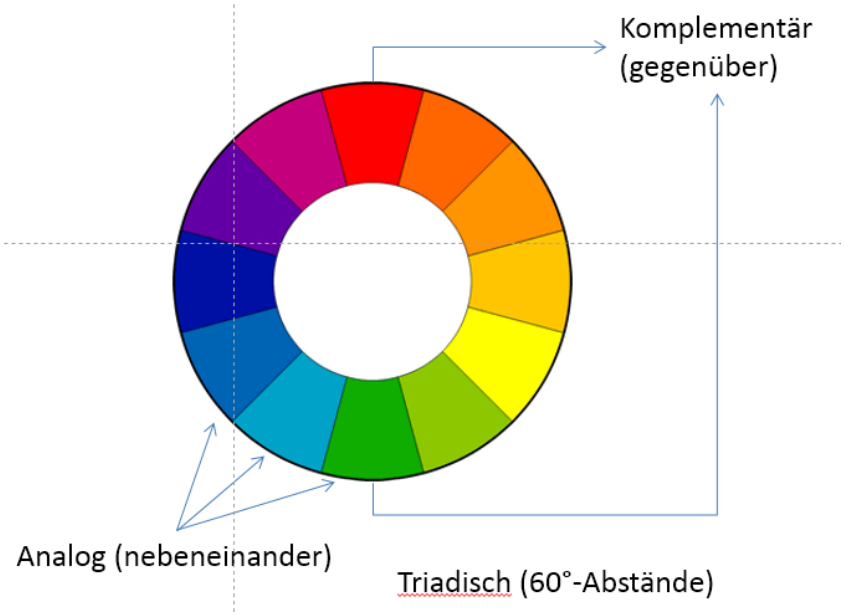


Abbildung 26: Analoge, komplementäre und triadische Farben

Farbschemata

Das Farbschema ist eine Kombination von Farben aus dem Farb-Rad. Diese Farben werden ausschließlich im Screendesign, abgesehen von Fotos, verwendet. Für geübte Designer sind „verrückte“ Schemata machbar, ansonsten sollte man auf gängige Modelle für Schemata zurückgreifen. Beispiele sind:

- Schwarz, Weiß
- Schwarz, Weiß + Schmuckfarbe
- Schwarz, Weiß + Monochromes Schema
- Schwarz, Weiß + Analoges Schema (warm)
- Schwarz, Weiß + Analoges Schema (kalt)
- Schwarz, Weiß + Komplementäres Schema
- Schwarz, Weiß + Natürliches Farbschema
- Schwarz, Weiß + Triadisches Farbschema

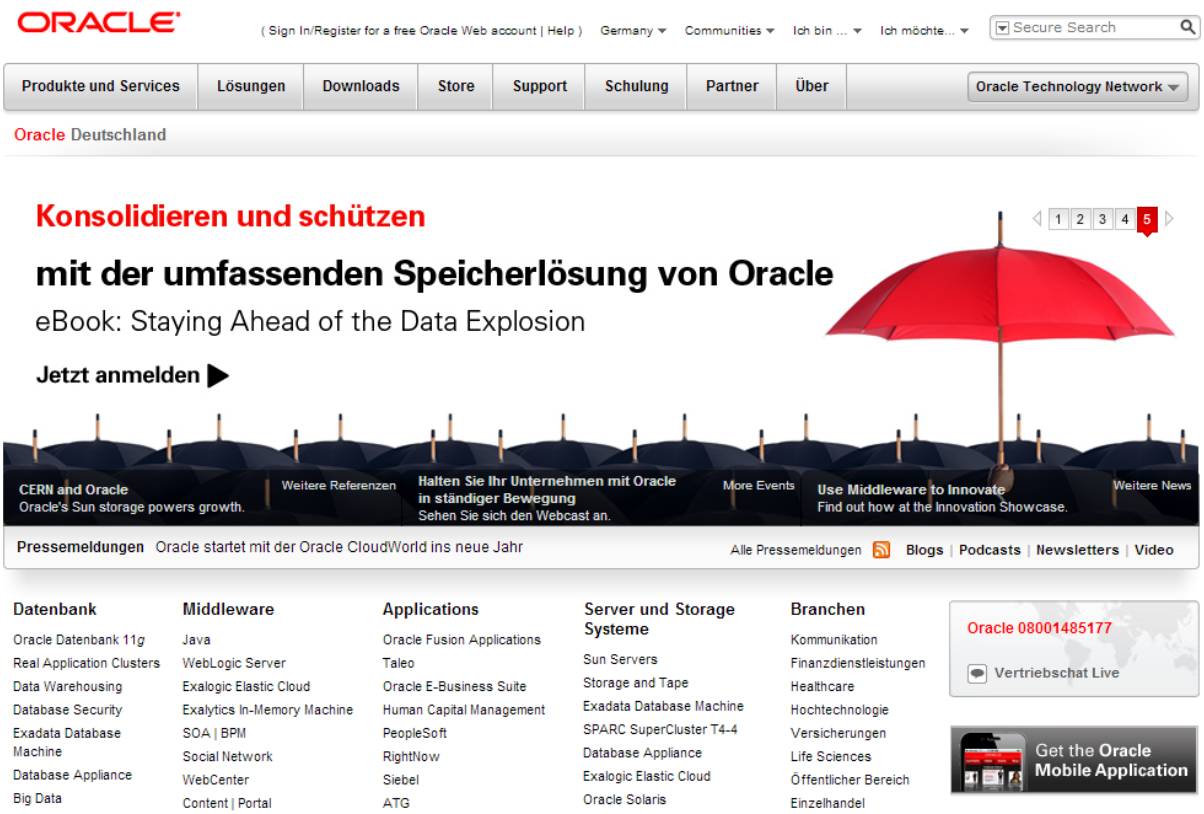


Abbildung 27: Beispiel für ein Farbschema aus Schwarz, Weiß + Schmuckfarbe (Rot)

Im „Oracle“-Beispiel wird ausschließlich die Farbe Rot verwendet und zwar deshalb, weil sie die Farbe des Logos ist, die Hauptfarbe der *Corporate Identity*. Alle anderen Elemente sind schwarz, weiß oder grau dargestellt. Das Orange ist dem international bekannten Icon des RSS-Feeds vorbehalten. Regenschirm, Überschriften, Hervorhebungen und Links tragen allesamt den genauen Farbwert des Logos. Insgesamt ist das Farbschema einfach aber hoch effektiv.

Abbildung 28: Triadisches Farbschema

Die Firma *Bosch* arbeitet mit einem triadischen Farbschema. Auch hier ist das Logo in „Rot“, jedoch tragen alle anderen Schmuck-Elemente den um 60 Grad versetzten Farbwert „Blau“. Texte

und Icons sind in Schwarz-Weiß-Abstufungen ausgeführt. Das Gelb, die dritte um 60 Grad versetzte Farbe wird zunächst nicht verwendet.
Um ein natürlich wirkendes Farbschema zu gestalten, können Fotos farblich zerlegt werden. Palette-Generatoren wie http://www.pictaculous.com/ oder http://www.cssdrive.com/imagepalette/ erzeugen beispielsweise aus jedem Foto Farbkacheln, die den hauptsächlichen Farbanteilen in einem Foto entsprechen. Die generierten Farben dienen zur Gestaltung von Überschriften und Schmuckelementen einer Seite. Das Original-Foto muss nicht weiter verwendet werden.

Abbildung 29: Natürliches Farbschema

Ein beliebtes Tool, um Farbwerte zu ermitteln, ist der *Color Scheme Designer*.

➡ **Zum Color Scheme Designer**

Spalten

Echte und unechte Spalten

Echte Spalten sind dadurch gekennzeichnet, dass bei wachsender Textmenge der Text von einer Spalte in die andere Spalte fließt. Dagegen wachsen bei *unechten Spalten* nur jene Spalten, in der der Text zunimmt.

Unechte Spalten mit float

Unechte Spalten entstehen durch das Floaten von Block-Elementen. Dies funktioniert sehr einfach und zugleich wirkungsvoll. Im Abschnitt „responsive Webdesign“ wird sich dieser Spaltenaufbau ebenso einfach an die Display-Größen anpassen lassen.

```
section {float:left; width:calc(33% - 20px); margin:5px 10px 5px 10px;}
```

Die Eigenschaft *width* in Verbindung mit dem Wert *calc* ermöglicht es, unterschiedliche Maßeinheiten miteinander zu verrechnen.
Wichtig ist, dass in nachfolgenden Abschnitten – zum Beispiel dem Footer – das Floaten durch ein Clear aufgehoben wird. Ansonsten entstünden unerwünschte Umbrüche.

```
footer {clear: both}
```

Ein Anwendungsfall unechter Spalten ist das vertikale Menü. Die linke Spalte erhält eine feste Breite. In ihr wird das Menü wie weiter oben beschrieben aufgebaut.

```
nav {
	float: left;
	width: 200px;
}
```

Bilder in Spalten sollten automatisch skaliert werden, um keine hässlichen Spaltenüberlappungen zu erzeugen.

```
section p img {
	width: 100%;
	height: auto;
}
```

Echte Spalten mit CSS 3

Ab CSS 3 ist es möglich, jedem Block-Element, das Text enthalten kann, die Spalteneigenschaft zuzuweisen. Mehrere weiterführende Eigenschaften bestimmen den Spaltenabstand oder die Art der Linie zwischen den Spalten. Noch muss der Vendor-Präfix vorangestellt werden (Apr 2015).

```
section {
	-webkit-column-count:3;
	-moz-column-count:3;
	column-count:3;
	-webkit-column-rule:solid #CC3300 5px;
	-moz-column-rule:solid #CC3300 5px;
	column-rule:solid #CC3300 5px;
}
```

Die Section-Eigenschaften zeigen einen dreispaltigen Abschnitt. Die Spalten sind durch Linien voneinander getrennt.
Die folgenden beiden Beispiele vereinen unechte und echte Spalten. Im zweiten Beispiel steht das Menü immer an derselben Stelle.

```
nav {width:200px; position:fixed; top:100px;}
```

Die Positionseigenschaft *fixed* „befestigt“ das *nav*-Element auf dem Browserfenster.

Elemente des Kopfbereichs

Tags und Meta-Tags

Für die Gebrauchsfähigkeit eines HTML-Dokuments sind nicht nur die korrekte Struktur und das visuelle Erscheinungsbild von Bedeutung, sondern auch die unsichtbaren Elemente, die im Kopfbereich des HTML-Dokuments notiert sind. Sie enthalten Informationen über (*über = meta*) das Dokument. Somit helfen sie beim Suchen und Finden und bei der Klassifizierung und der Verschlagwortung. Die Daten des Kopfbereichs heißen *Meta-Daten*.
Die einzelnen Meta-Elemente heißen auch *Meta-Tags*.

Das *title*-Element dient **Suchrobots** als Ankerpunkt. Sein Inhalt wird in den Suchergebnissen als erstes angezeigt. Weiterhin erscheint es in der Titelzeile des Browserfensters.

```
<title>Der aussagekräftige Titel der Seite</title>
```

Das *<link>*-Element referenziert ein externes Stylesheet.

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

Das *<script>*-Element verweist auf ein externes JavaScript.

```
<script type="text/javascript" src="script.js"></script>
```

Aufbau eines Meta-Tags

Die meisten Meta-Tags enthalten zwei Attribute
- Das *name*-Attribut (Bezeichnung)
- Das *content*-Attribut (Wert)

```
<meta name="author" content="Otto von Bismarck">
```

Von W3C empfohlene Meta-Tags (Draft)

Das W3C empfielt die folgenden Meta-Tags:

- Name der Anwendung
- Autor des HTML-Dokuments
- Beschreibung des HTML-Dokuments
- Softwareprogramm zur Erzeugung des HTML-Dokuments
- Schlüsselwörter
- Optional: Verlag

```
<meta name="application-name" content="Meta-Tags">
<meta name="author" content="Otto von Bismarck">
<meta name="description" content="Das Dokument enthält die Biografie.">
<meta name="generator" content="Bluefish">
<meta name="keywords" content="Aufbau, Forwarding, Robots">
<meta name="publisher" content="HTML-Verlag GmbH & Co">
```

HTML-Forwarding

Das nächste Beispiel zeigt eine Seite, die nach zwei Sekunden zu einem anderen HTML-Dokument weitergeleitet wird (*forwarding*).

Robots

„Robots“ steuert den Zugriff von Suchmaschinen. Die Angabe wird von den Spider-Programmen der Suchmaschinen wie Google berücksichtigt.

```
<meta name="robots" content="noindex, nofollow">
```

Das Attribut *content="noindex"* bedeutet, dass der Seiteninhalt nicht ausgelesen werden soll. Er bleibt unauffindbar. Das Attribut *content="nofollow"* bedeutet, dass die im Dokument enthaltenen Hyperlinks nicht verfolgt werden dürfen. Auch die Angabe einer Datei robots.txt mit erweiterten Angaben ist möglich (Siehe Internet).

Gestaltgesetze

In den 20er Jahren des 19. Jahrhunderts beschäftigten sich Psychologen ausgiebig mit der menschlichen Wahrnehmung. Sie wollten herausfinden, welche psychologischen Prozesse ablaufen, wenn Sinneseindrücke geordnet und strukturiert werden und wie Material beschaffen sein muss, damit es überhaupt wahrgenommen, geordnet und strukturiert werden kann. Dies war die Geburtsstunde der psychologischen Fachrichtung der Gestaltpsychologie. Die wichtigsten Aussagen bestehen darin, dass das Gehirn beim Erfassen von Information versucht, Strukturen zu bilden, Bild- und Text-Elemente als getrennt oder zusammengehörig zu erkennen und klare, einfache Formen zu finden.

Abbildung 30: Links: Die Kapelle steht als Motiv klar vor dem Hintergrund. Rechts: Keine Figur hebt sich vom Hintergrund ab.

Ein Bild oder ein Screen sollte nicht mehr als fünf bis sieben visuell gruppierte Elemente enthalten. Mehr können Menschen auf Anhieb nicht erfassen. Im folgenden Bild verlieren selbst begabte Menschen jeden Überblick. Sie reagieren mit Ablehnung. Aber erinnert die Gestaltung nicht an viele, unausgegorene Webseiten, wie sie häufig in der Anfangszeit des Internets entstanden?

Abbildung 31: Das Chaosformular

Rund 120 Gestaltgesetze kommen hier zum Tragen. Sie gelten sowohl für realistische Bilder, als auch für Analogie- und logische Bilder. Aus der schier unüberschaubaren Menge seien sieben Gesetze herausgegriffen. Dabei darf es nicht irritieren, wenn sich die Gesetze teilweise überschneiden.

1. **Das Gesetz der Prägnanz, der guten Gestalt oder der Einfachheit**: Menschen interpretieren Formen nach einer möglichst einfachen Struktur. Anstelle eines Elf-Ecks nimmt der Betrachter in *Abbildung 1* eine Kombination aus Dreieck und Viereck wahr – vielleicht aber auch einen Engel?
2. **Das Gesetz der Ähnlichkeit**: Gleiche oder ähnliche Elemente werden als zusammengehörig wahrgenommen. In *Abbildung 2* bilden die weißen Quadrate eine eigene Figur. In *Abbildung 4* besteht die Figur aus vier Reihen Quadrate und aus drei Reihen Kreise.
3. **Gesetz der Nähe**: Elemente mit geringeren Abständen wie in *Abbildung 3* werden als zusammengehörig wahrgenommen.

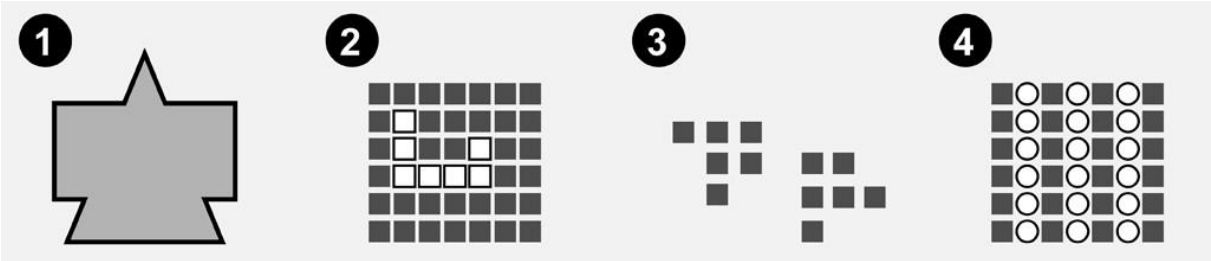


Abbildung 32: Illustration von vier Gestaltgesetzen

Nicht immer sind alle Gestaltgesetze lupenrein anwendbar, häufig überlagern sie sich und müssen in ihrer Gesamtheit und ihrer gegenseitigen Beeinflussung bewertet werden.

Usability

Mit der Gestaltung von Bildschirmoberflächen und Mensch-Maschine-Dialogen befasst sich in grundsätzlicher Weise die internationale „Norm EN ISO 9241 - Interaktion zwischen Mensch und Computer“. Hier geht es um den Gedanken der Usability.

Abbildung 33: Drei gleiche Zeichen für unterschiedliche Bedeutungen

Was ist Usability?

"Die Usability eines Produktes ist das Ausmaß, in dem es von einem bestimmten Benutzer verwendet werden kann, um bestimmte Ziele in einem bestimmten Kontext effektiv, effizient und zufriedenstellend zu erreichen." (DIN EN ISO 9241, 11)

Interaktionseigenschaft, nicht Produkteigenschaft

Beim Usability-Begriff geht es also nicht um eine Produkteigenschaft, sondern um eine Eigenschaft der Interaktion zwischen Mensch und Produkt. Das heißt, dass die Usability eines Produkts der subjektiven Wahrnehmung unterliegt. So ist die Farbe Rot eine objektive Produkteigenschaft, in Bezug auf die Usability kann das Rot subjektiv „Warnung“, „Stopp“, „Halt“ … bedeuten und möglicherweise die Betätigung eines Schaltknopfes verhindern.

Die Usability-Falle

Entwickler laufen immer wieder in die *Usability-Falle*. Sie stecken mit ihrem Kopf so tief in ihrer Entwicklung, dass sie sich nicht vorstellen können, welche Probleme fachfremde Nutzer mit dem Produkt haben können.

Gute Usability

Eine gute Usability ist also dann vorhanden, wenn eine gute Benutzbarkeit für das Zielpublikum des Produkts gegeben ist. Für Webseiten heißt das, dass Benutzer-Interfaces für Kinder anders gestaltet werden müssen als für Greise, um ergonomisch zu sein.

EN ISO 9241 - Interaktion zwischen Mensch und Computer

Die internationale Norm **ISO 9241** beschreibt die ergonomischen Anforderungen für Bürotätigkeiten mit Bildschirmgeräten. Der deutsche Titel lautet **Ergonomie der Mensch-System-Interaktion**. Die Norm besteht aus rund 30 Teilen, die unterschiedliche Aspekte von Hardware und Software beleuchten mit dem Ziel, gesundheitliche Schäden bei der Bildschirmarbeit zu vermeiden und die Gebrauchsfähigkeit zu gewährleisten. In Rechtsfällen ist die Norm Grundlage für Sachgutachten.

Teil-110 Grundsätze der Dialoggestaltung

Für Software- und Webentwicklung besonders interessant ist der Teil 110 der Norm. Unter Dialoggestaltung sind das Agieren und Reagieren von Benutzern auf die Bildschirmdarbietung zu verstehen. Es gibt sieben Grundsätze.

Die sieben Grundsätze der Dialoggestaltung

Die folgenden sieben Grundsätze sind für die Gestaltung und Bewertung eines Dialogs als wichtig erkannt worden.
(Alle Beispiele nach: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten, Teil 10: Grundsätze der Dialoggestaltung, (ISO 9241-10 : 1995)

Grundsatz 1: Aufgabenangemessenheit

„Ein Dialog ist aufgabenangemessen, wenn er den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen.“

Beispiel: Formatierungen wie z.B. Farbe und Informationen wie z.B. Wochentag, Datum usw. werden nur angezeigt, wenn sie die Erledigung der Arbeitsaufgabe erleichtern.

Grundsatz 2: Selbstbeschreibungsfähigkeit

"Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems unmittelbar verständlich ist oder dem Benutzer auf Anfrage erklärt wird."

Beispiel: Das unmittelbare Anzeigen eingegebener Daten und das Anzeigen des Änderungszustands der Daten sind notwendig, dem Benutzer beim Verstehen dessen zu helfen, was in der Anwendung geschieht und was er beeinflussen kann. Können Dialogschritte zurückgenommen werden, zeigt die Anwendung dies an, indem sie eindeutige Informationen darüber gibt, was zurückgenommen werden kann. Kann das Löschen von Daten nicht rückgängig gemacht werden, verlangt das Dialogsystem eine Bestätigung.

Grundsatz 3: Steuerbarkeit

"Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist."

Beispiel: Kein Eingabefeld wird gelöscht, ersetzt oder anderweitig dem Benutzer unzugänglich gemacht, bis der Benutzer die Vollständigkeit der Dateneingabe bestätigt, z.B. durch Drücken der ENTER-Taste.

Grundsatz 4: Erwartungskonformität

"Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht, z.B. seinen Kenntnissen aus dem Arbeitsgebiet, seiner Ausbildung und seiner Erfahrung sowie den allgemein anerkannten Konventionen."

Beispiel: Zustandsmeldungen des Dialogsystems werden stets an derselben Stelle ausgegeben. Der Dialog wird stets durch das Drücken derselben Taste beendet.

Grundsatz 5: Fehlertoleranz

"Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.“

Beispiel: Das Dialogsystem zeigt eine Fehlermeldung an, die Informationen über das Auftreten des Fehlers, die Art des Fehlers und mögliche Methoden der Korrektur in dem Maße enthält, in dem das Dialogsystem diese Informationen geben kann.

Grundsatz 6: Individualisierbarkeit
"Ein Dialog ist individualisierbar, wenn das Dialogsystem Anpassungen an die Erfordernisse der Arbeitsaufgabe sowie an die individuellen Fähigkeiten und Vorlieben des Benutzers zulässt.“
Beispiel: Für sehbehinderte Benutzer stehen größere Schriftzeichen zur Verfügung, die Maus kann an die Benutzung mit der linken oder der rechten Hand angepasst werden.

Grundsatz 7: Lernförderlichkeit
"Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen des Dialogsystems unterstützt und anleitet."
Beispiel: „Learning-by-doing“ wird dadurch unterstützt, dass der Benutzer ermutigt wird: zu experimentieren, in unterschiedlichen Situationen Beispiele durchzuspielen, „Was wäre wenn?“-Alternativen anzuwenden (z.B. Fehlerkorrektur zuzulassen, ohne dass die Gefahr besteht, potentiell katastrophale Ergebnisse herbeizuführen).

Irreführende Gestaltung

Obwohl das Internet noch relativ jung ist, haben sich bereits Konventionen und Verlässlichkeiten herausgebildet, gegen die Webdesigner nur in begründeten Ausnahmenfällen verstoßen sollten. Hier einige Beispiele:

Barrierefreiheit

Was heißt Barrierefreiheit?

Barrierefreiheit heißt, behinderten oder gesundheitlich eingeschränkten Menschen die Teilhabe an Informationsangeboten zu ermöglichen. Die rund 7,5 Millionen (Stat. Bundesamt 2013) zeitweise (z.B. nach Unfällen) oder dauerhaft Schwerbehinderten in Deutschland verlangen ein Recht auf Teilhabe. Benutzer haben möglicherweise gesundheitliche Einschränkungen/Behinderungen, Verständnisprobleme bei Computer-Slang, mangelnde technische Ausstattung, langsame Internet-Verbindung. Sie sind möglicherweise in einer Situation, in der ihre Augen, Ohren oder Hände beschäftigt oder behindert sind (Fahrt zur Arbeit, laute Umgebung o. Ä.).
Die Aufgabe für Web-Entwickler ist es, dafür zu sorgen, dass Webseiten unter den verschiedensten Bedingungen zugänglich bleiben. Inhalte müssen leicht navigierbar sein, es bedarf verständlicher Mechanismen zur Navigation zwischen und innerhalb der Seiten.

Web Accessibility Initiative

Innerhalb des W3C befasst sich die *Web Accessibility Initiative* (*WAI*) mit der Zugänglichkeit des Internet. Die *Web Content Accessibility Guidelines* (*WCAG*) liegen in der Version 2.0 vor, Stand Februar 2015. Die technischen Empfehlungen beschreiben verschiedene Vorgehensweisen, um Webseiten für weitesten Teile der Menschheit nutzbar zu machen.

➡ **Working Group Note, Techniques for WCAG 2.0**

Die technischen Vorschläge richten sich an Webdesigner und Programmierer. Unter anderem geht es um folgende Themen:

- HTML and XHTML Techniques
- CSS Techniques
- Client-side Scripting Techniques
- Server-side Scripting Techniques
- Flash Techniques

- Silverlight Techniques
- PDF Techniques

Ausführliche Code-Beispiele runden die Hinweise ab.

Barrierefreies Webdesign und multimediale Inhalte

Die meisten Gestaltungsregeln für barrierefreies Webdesign gehören inzwischen zum guten Stil eines jeden Webdesign. Beispielsweise ist die Regel, dass jede Grafik mit Hilfe des alt-Attributs über ein Textäquivalent zu verfügen habe, auch eine der Grundregeln der Suchmaschinenoptimierung. Nur so können Bildinhalte auch Vorlese-Maschinen bekannt gegeben werden. Blinde Menschen werden dadurch in die Lage versetzt, sich Bildbeschreibungen anzuhören, anhand derer sie sich das Bild im Kontext des Inhalts vorstellen können. Das Textäquivalent dient gleichzeitig dazu, die Indexer der Suchmaschinenanbieter auf den richtigen Weg zu führen und zur gewünschten Kategorisierung der Bildinhalte beizutragen.
Einer der ehernen Grundsätze barrierefreier Seitengestaltung ist heute jedoch kaum noch aufrecht zu halten. So verlangten die *Web Content Accessibility Guidelines (WCAG) 1.0* aus dem Jahr 1999, dass Webdesign ohne JavaScript funktionieren müsse. Seit 2008 arbeitet die *WAI* parallel an *ARIA (**A**ccessible **R**ich **I**nternet **A**pplications)*, einer Spezifikation, die seit 2014 zur *W3C-Recommendation* geworden ist. Eine der Säulen der *WAI-ARIA* ist das *role-Attribut*, das in allen W3C-Auszeichnungssprachen eingesetzt werden kann.

➡ **W3C-Working-Draft *WAI-ARIA***

Das Role-Attribut

Das *Role-Attribut* erweitert die HTML-Elemente, in dem es ihnen dezidierte Rollen zuweist. Mit dem *Role-Attribut* ist es möglich, die durch die HTML-Semantik festgelegte Rolle eines Elements zu überschreiben. Im folgenden Beispiel wird die h1-Rolle als Bildplatzhalter umdefiniert.

```
<h1 role=“figure“><img href=“bild-001.jpg“></h1>
```

Es ist zwar nicht Sinn der Sache und entgegen den WAI-ARIA-Regeln, die durch die HTML-Semantik bestimmte Rolle eines HTML-Elements zu ändern, prinzipiell aber verfügt das Role-Attribut die Macht, den HTML-Elementen eine neue, andersartige Bedeutung zuzuweisen.
Beim Role-Attribut geht es vielmehr darum

- ganzen Teilen von Dokumenten
- und nicht-semantischen Elementen

eine technisch interpretierbare Rolle zu geben, die entsprechend angezeigt oder auch umgangen werden kann.

➡ **Zur W3C-Spezifikation des Role-Attributs**
➡ **Kategorien von Rollen (nach W3C)**

Das Grundgerüst eines HTML-Dokuments, das den Kriterien der Barrierefreiheit genügt, könnte zum Beispiel wie folgt aussehen:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Dokument mit Role-Attribut</title>
</head>
<body>
```

```
<header role="banner">
<img src="p1020461.jpg" width="900" height="300" alt="Die Theme in London mit
London Eye und Big Ben">
<h1>Dokument mit Role-Attribut</h1>
</header>
<article role="main">
<div role="navigation">
<a href="#">Stadt</a> | <a href="#">Land</a> | <a href="#">Fluss</a> | <a
href="#">Berge</a> | <a href="#">Täler</a></div>
<figure>
<img src="p1020459.jpg" width="600" height="350" alt="Omnibusverkehr in London">
</figure>
</article>
</body>
</html>
```

Zusammenfassung

Beim Aufbau von Webseiten sollte auf die strikte Trennung von Form und Inhalt geachtet werden. Alle Regeln zum Erscheinungsbild gehören in eine externe CSS-Datei oder in einen style-Abschnitt im Kopfbereich. Die Gestaltung von Text, Textblöcken, Farben und Spalten orientiert sich im Normalfall an wenigen, allgemein gültigen Regeln und Empfehlungen. Komplizierte Designs sollten Grafik-Designern überlassen werden. Theoretisch fundiert sind der visuelle Aufbau von Webseiten durch die Gestaltgesetze aus der Wahrnehmungspsychologie und den Empfehlungen zur Usabilty. Die Web Accessibility Initiative des W3C konzentriert die Usability einer Website auf den Aspekt der Barrierefreiheit. Damit ist gemeint, dass alle Menschen, egal ob gesundheitlich eingeschränkt oder nicht, Zugang zum Internet und seinen Angeboten erhalten.

Responsive Webdesign

Printdesign und Webdesign: Einige Unterschiede

Wer Drucksachen erstellt, macht sich zunächst Gedanken über das Papierformat, den Satzspiegel, die Typografie, über Farbharmonien, die Positionierung von Bildern. Die Vorgehensweise beim Erstellen von Webseiten ist ähnlich. Jedoch sind einige wichtige Unterschiede zu beachten.

Papierformat – Bildschirmformat

Abbildung 34: Printdesign und Webdesign

Eine auf **DIN A4** gedruckte Seite behält für immer ihre Größe von **genau 210 mm x 297 mm**. Im Webdesign ist es dagegen nicht vorhersehbar, wie die erstellte Website betrachtet wird. Monitore haben unterschiedliche Größen, unterschiedliche Farbeinstellungen. Internetnutzer verkleinern ihre Browserfenster, um mehrere Fenster parallel anzuschauen.
Moderne Browser ermöglichen es, mit den Tastenkombinationen *Strg++*, bzw. der S*trg+-*, die Seitendarstellung zu skalieren. Darauf muss das Webdesign vorbereitet sein.
Das Anzeigeformat ist beim Webdesign eine variable Größe. Die Seite darf nicht „kaputt“ gehen, wenn sich die Anzeigebedingungen ändern. Auf Netbooks muss eine Seite genauso nutzbar sein wie auf einem 27-Zoll-Monitor oder einem Smartphone.

Seitenumfang

Flyer und Plakat, die Drucksachen, die vielleicht die meiste individuelle Gestaltungsfreiheit lassen, haben selten ein Pendant im Webdesign. Es gibt sie fast nicht, die einseitige Website. Websites bestehen aus vielen Seiten, sind Sammlungen von Webdokumenten. Webdesign ist eher mit dem Design von Broschüren, Zeitungen oder Katalogen vergleichbar. Das Erscheinungsbild muss über viele Seiten hinweg wiedererkennbar sein. Wer eine weiter hinten liegende Seite aufruft, soll auf den ersten Blick sehen, dass auch diese Seite zur Site gehört.

Interaktion

Webseiten können mit Skripten verknüpft werden, die auf Eingaben der Benutzer/innen reagieren. Die Möglichkeiten, interaktive Webseiten zu erstellen, sind nahezu unbegrenzt. Naturgemäß machen sich Printdesigner/innen über derartige Herausforderungen keine Gedanken.

Usability

Printprodukte werden umgeblättert, man orientiert sich am Inhaltsverzeichnis und am Schlagwortregister, manchmal führen farbliche Hervorhebungen durch die Seiten. Die Technik des Hyperlinks stellt erhöhte Anforderungen an die Vorkehrungen, die getroffen werden müssen, damit sich die Benutzer auf der Website zurechtfinden. Die unsystematische Platzierung von Verweisen

(Links) erschwert die Orientierung. Auf vielen Websites finden sich Benutzer nicht zurecht, sie „verlaufen“ sich. Die Benutzbarkeit der Website ist eingeschränkt. Bei Webprojekten ist es deshalb unumgänglich, sich Gedanken über die Benutzerführung zu machen, über Leitsysteme, die es Menschen mit und ohne Handicap (Barriere-Freiheit) gleichermaßen ermöglichen, sich auf einer Website zurechtzufinden.

Suchmaschinen

Gutes Webdesign unterstützt Suchmaschinen, beziehungsweise deren Roboter, bei der Klassifizierung und Verschlagwortung von Webseiten. Ein Webdesign, das auf den Standards des W3-Konsortiums beruht, ist im Vorteil gegenüber rein intuitiv erstellten Webseiten. Das *Wichtigste an einer Website* ist es, von den Suchmaschinen gefunden und im Kontext der gewünschten Schlüsselworte und Suchbegriffe angezeigt zu werden.

Herausforderungen des Webdesigns

Die Herausforderung beim Webdesign liegt darin, Texte, Bilder und Medienobjekte auf dem Display des jeweiligen Endgeräts ergonomisch zu platzieren. Das Ganze soll

- „gut aussehen“
- und unter den *verschiedensten Anzeigebedingungen* in wiedererkennbarer Weise angezeigt werden.

Das Design einer Site ist perfekt, wenn es auch Menschen einbezieht, die in gesundheitlicher Hinsicht das eine oder andere *Handicap* aufweisen. Damit tut man auch dem *Ranking* der Site einen Gefallen, denn ein *barrierefreies Webdesign* entspricht der Logik und der Funktionalität der Roboter und Spider, die eine Site zur Auffindbarkeit im Netz indexieren.

Webdesign-Philosophien

Als Webdesigner/in besteht die Möglichkeit, auf unterschiedliche Weise auf verschiedene Bildschirmabmessungen und Größen von Browserfenstern zu reagieren.

- Statisches Webdesign
- „Flüssiges“ Webdesign
- „Elastisches“ Webdesign
- Adaptives Webdesign
- Responsives Webdesign

Statisches Webdesign

In Analogie zu Druckerzeugnissen, die eine flexible Größe nicht kennen, erscheint die Anordnung „layouttreu“ auf unterschiedlichen Bildschirmen. Das absolute Webdesign hat auf den ersten Blick einige einleuchtende Vorteile, wie zum Beispiel das immer gleiche Seitenformat. Wer jedoch keine Kontrolle über die Inhaltsmenge einer Seite hat – und das ist bei Datenbank basierten Anwendungen der Fall – muss ständig nacharbeiten und neu positionieren. Das statische Webdesign wird auch Smartphone-Anwendungen nicht gerecht.

Elastisches Webdesign

Viele Webseiten benutzen sowohl Elemente des absoluten, als auch des flüssigen Designs und kombinieren diese zu größtmöglicher Funktionalität. Häufig stehen die Haupt-Inhalte zentral in der Mitte. Sie bleiben auch dann in der Mitte, wenn das Browserfester auf- und zugeschoben wird.

Flüssiges Webdesign

Je nach Größe des Browserfensters – schieben Sie es einmal auf und zu! – verändern sich die Umbrüche der Textzeilen auf der Seite. Die Inhalte passen sich also „flüssig“ der Fenstergröße an. Das flüssige Design ist die Grundform des Webdesigns.

Adaptives Webdesign

Beim adaptiven Webdesign werden meist drei bis vier Displayformate angenommen. Für diese Formate werden in Stufen verschiedene Layout erstellt. Dabei helfen *Media-Attribute* und *Media Querys.* In der Praxis werden die Begriffe *adaptiv* und *responsiv* meist synonym verwendet.

Responsives Webdesign

Responsives Webdesign: Responsive Webdesign ist ein Ansatz zur Webseiten-Gestaltung mit dem Ziel, dass sich Inhalt und Gestaltung einer Website möglichst nahtlos an das Ausgabegerät anpassen, nicht umgekehrt. Dabei helfen *Media-Attribute* und *Media Querys*. Responsives Webdesign beruht auf dem „flüssigen“ Webdesign.

Dieser Abschnitt wird sich im Wesentlichen mit dem adaptiven und dem responsiven Webdesign befassen. Wie nun tatsächlich die Webdesign-Philosophie eines Unternehmens aussieht, ist in sogenannten Style Guides festgehalten.

Web Style Guides

Style-Guides enthalten die Regeln für das *Corporate Design* eines Unternehmens.
Beispiele:

- **BBC GEL, British Broadcasting Corporation**
- **Uni Kassel**
- **Uni Wuppertal**

Der Style Guide für den Web-Auftritt enthält Aussagen über die verwendeten Farben, Schriften über Spaltenmaße und vieles andere mehr. Auch CSS-Fragmente zum raschen Einbinden in eigene HTML-Dokumente sind oft zu finden. Insgesamt erleichtert ein Style Guide den Entwurf und die Realisation von Webseiten.

Ein Style Guide besonderer Art ist der Web Style Guide, 3rd Edition (Yale).

- **Yale-Web Style Guide**

Web Style Guide, 3rd Edition (Yale)

Patrick J. Lynch (*1953) und *Sarah Horton* brachten im Jahr 2002 eine Publikation heraus, die sie denen Herz legten, die „dauerhaften Inhalt“ im Internet zu publizieren gedachten.

„Dauerhafter Inhalt unterliegt keinen Trends; gutes Design hat Bestand, während Trends bald unseriös wirken. Erfolg im Webdesign geht über Technologie und Mode hinaus“ (Lynch, Horton, 2002).

Lynch ist Künstler, Designer, Schriftsteller und Fotograf. An der renommierten *Yale University* leitet er spezielle Technologie-Projekte. Für seine – meist medizinischen – Illustrationen erhielt er zahllose Anerkennungen und Preise.

Horton ist Schriftstellerin, Juristin und Spezialistin für Usability und benutzerzentriertes Design. Am *Dartmouth College*, das ebenso wie Yale in der „Efeu-Liga“ der amerikanischen Universitäten spielt, leitete sie die Abteilung für Webstrategie und Design.

Die Kurzbiografien der Autoren zeigen, dass beide keineswegs der Informatikszene entstammen, sondern dass sie das Internet aus der Sicht der Benutzer und der medialen Kommunikation betrachten.

Ihre gemeinsame Publikation nannten sie „Web Style Guide“. Das Buch war viele Jahre *der* Bestseller der *Yale University Press*. Das Besondere an diesem Werk ist es, dass Lynch und Horten auf alle maßgebenden Erfolgsfaktoren eines Webprojektes Bezug nehmen, ohne sich in Details zu verlieren. Für Lynch und Horten steht der gesamte Entstehungsprozess im Vordergrund. Das bürgt für die Qualität, die sich an der Dauerhaftigkeit des Inhalts bemisst – und das macht den Web Style Guide auch noch heute aktuell.

Neben Aussagen zu Produktionsprozess, Informationsarchitektur, Design, Grafik und Multimedia

enthält der Web Style Guide auch einen Abschnitt über die Usability. Sie ist im Sinne von Lynch und Horton als universell zu verstehen. Eine gute Benutzbarkeit heißt „*Moving beyond the typical user*. Anders gesagt: Nicht für den angenommen Durchschnittsuser produzieren, sondern möglichst viele Usergruppen einbeziehen.
Im Abschnitt über die *Site Structure* (5) zitieren Lynch und Horton den Apple-Gründer *Steve Jobs* (1955-2011), der sinngemäß sagte: „Design zeigt, wie etwas funktioniert.“
All diese Aussagen bilden die theoretische und praktisch geforderte Grundlage für ein Webdesign, das sich den zahllosen Varianten der Präsentation anpasst. Beispiele sind:

- Fernsehgeräte
- Drucker
- Smartphones
- Screenreader
- Tablets
- Großbildleinwände

Alte und neue Computer
…und vieles andere mehr.

Media-Attribute

Schon früh befasste sich das W3C damit, die Stilregeln von Stylesheets auf bestimmte Geräteklassen zu beziehen und nur dort gültig zu sein. In einem ersten Schritt wurden Media-Attribute für Stilregeln definiert.

Notierung gerätespezifischer Media-Attribute

Die beiden folgenden Media-Attribute beziehen sich einmal auf die Druckausgabe *(@media print),* zum anderen auf die Darstellung auf dem Bildschirm *(@media screen)*. Zum Drucken wird die Schriftart *Times* verwendet, die Bildschirmdarstellung benutzt die Schriftart *Arial*. Hier sind die Media-Attribute direkt im Stylesheet notiert. Typisch sind die verschachtelten, geschweiften Klammern.

```
@media print {
body {
	font-family: Times, serif;
}
}
@media screen {
body {
	font-family: Arial, sans-serif;
}
}
```

Die Media-Attribute können auch bei der Verlinkung der CSS-Dateien im Kopfbereich angegeben werden. Sie sind dann Attribute des *link*-Elements.

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

Die Eigenschaften und Stilregeln für die Geräte stehen in den verknüpften Stildateien „drucker.css“ und „screen.css“.

Liste gerätespezifischer Media-Attribute

media-Attributwert	Bedeutung
all	Alle Ausgabemedien, Standardwert
braille	Geräte für Braille-Schrift zum Ertasten
print	Papier-Drucker/pdf-Drucker
screen	Farbige Computerbildschirme
speech	Wird den Medientyp „aural“ ersetzen

Tabelle 15: Gerätespezifische Media-Attribute

Ein Problem besteht darin, dass sich Smartphones und Tablets nicht als *handheld*-Geräte melden, wie einst geplant, sondern als normale Computer. Mit Media-Querys und dem „viewport“-Attribut ist es möglich, Stilregeln präziser auf das präsentierende Anzeigegerät zuzuschneiden.

Der Viewport

Mit den hochauflösenden Smartphone- und Tablet-Displays tritt ein Anzeige-Problem auf, das auf der unterschiedlichen Pixeldichte der Displays beruht. Ein HTML-Element, das über eine bestimmte Breite verfügt, erscheint auf Displays mit hoher Pixeldichte verkleinert, während Displays mit sehr geringer Pixeldichte das Element vergrößern.
Das Meta-Tag-Attribut *name="viewport"* und sein Wert weisen die Seite an, wie sie das Display nutzen soll. Sie sorgen für eine korrekte Skalierung der Website beim ersten Aufruf. Das Viewport-Meta-Tag-Attribut ist noch kein W3C Standard, ist aber allgemein gebräuchlich (iOS | Android) und liegt zurzeit (März 2015) als Entwurf vor (*CSS Device Adaptation Module Level 1*).

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Der Wert *width=device-width* bedeutet, dass die Seite die Breite des Bildschirms in geräteunabhängigen Pixeln nutzt. Dadurch kann sie Inhalte neu anordnen und sich an verschiedene Displaygrößen anpassen. Sie reagiert *responsiv*, egal ob es sich um das Display eines Smartphones oder das eines Desktopcomputers handelt. Der Wert *initial-scale=1* weist den Browser an, eine 1:1-Beziehung zwischen CSS-Pixeln und geräteunabhängigen Pixeln herzustellen und auch dann zu gewährleisten, wenn sich die Orientierung des Geräts von Hochformat auf Querformat ändert.

Media Querys – Medienmerkmale abfragen

In Verbindung mit dem Viewport-Meta-Tag-Attribut empfiehlt es sich, sogenannte *Media Querys* zu nutzen, um das HTML-Dokument, bzw. die Website, auf verschiedene Display-Größen und deren Orientierung vorzubereiten.

Die Notierung von Media Querys

Media Querys können entweder im HTML-Dokument oder im CSS-Dokument notiert werden.

```
<link href="phone.css" rel="stylesheet" type="text/css" media="screen and (max-
width: 480px)">
```

Die Media Query *screen and (max-width:480px)* besagt, dass das Stylesheet *phone.css* nur dann wirksam ist, wenn die maximale Bildschirmbreite weniger als 480 Pixel beträgt.
Die andere Möglichkeit besteht darin, die Media Query direkt im Stylesheet zu notieren.

```
@media screen and (max-width:480px) {
	Selektor {
		Eigenschaft: Wert;
		Eigenschaft: Wert;
		Usw…
	}
}
```

Ausgewählte Media Querys

Das Stylesheet lässt bei Browserfenstern, die maximal 800 Pixel breit sind, den Seitenhintergrund blau erscheinen. Außerdem wird hinter die h1-Überschrrift der Inhalt „800px“ eingefügt.

```
@media only screen and (max-width:800px) {
body {
	background-color: #06C;
	color:white;
}
h1:after {
	content: ": 800px";
}
}
```

Im folgenden Beispiel gilt die Hintergrundfarbe nur, wenn das Browserfenster zwischen 480 und 800 Pixel breit ist.

```
@media only screen and (min-width:480px) and (max-width:800px) {
body {
	background-color: #06C;
	color:white;
}
}
```

Auch das Reagieren auf die Orientierung des Displays, die der Lagesensor meldet, ist mit Media-Querys möglich. Im folgenden Beispiel erscheint der Hintergrund im Querformat weiß, im Hochformat blau.
Weitere Media Querys und ihre Werte finden Sie unter anderem bei W3C Schools.

Anwendungen

„Hamburger-Menü“

Wenn sich bei schmalen Displays das Menü in einen Stapel verwandelt, spricht man häufig von einem Hamburger-Menü. Dies ausschließlich mit CSS befriedigend zu lösen, ist leider nicht möglich. Es ist ratsam, die Javascript-Bibliothek *jQuery* zu benutzen. Das folgende Beispiel zeigt das Zusammenspiel von HTML, CSS und Javascript, wenn es um das Erstellen eine rudimentären Hamburger-Menüs geht.

Einbinden von jQuery

Im Kopfbereich immer zuerst das Stylesheet, dann Javascript referenzieren.

```
<style>
	Stilregeln und Media Querys
</style>
<script type="text/javascript" src="jquery-1.9.1.min.js"></script>
<script type="text/javascript" src="jquery.easing.1.3.js"></script>
<script type="text/javascript" src="wd-hamburger.js"></script>
```

Das Beispiel verwendet ein internes Stylesheet. Danach bindet der Kopfbereich die jQuery-Basisbibliothek (hier *jQuery 1.9.1*) ein, weiterhin das Easing-Modul, das für eine visuelle Elastizität sorgt, dann das eigene Script zur Steuerung des Hamburger-Menüs. Die ersten beiden Bibliotheken bleiben unangetastet.

```
$(document).ready(function(){
	$("#hmenu").click(function(){
		$("nav#global").slideToggle("slow");
		return false;
	});
	$(window).resize(function() {
		if (window.innerWidth > 480) {
			$("nav#global" ).show();
		}
		else {
			$("nav#global" ).hide();
		}
	});
});
```

Das Script wird dann aktiv, wenn das Dokument fertig geladen ist (Zeile 1). Wenn auf die id = „hmenu“ geklickt wird, blendet sich die globale Navigationsleiste ein oder aus – je nach aktuellem Zustand. Die *resize*-Funktion ist nötig, um bei den anderen Display-Größen den jeweils gewünschten Zustand herzustellen.
Für das Verhalten des Menüs ist die Media Query im folgenden gekürzten Script verantwortlich.

```
@media screen and (max-width:480px) {
header {
	height: 48px; margin-top:0; position:relative;
}
```

Die header-Eigenschaft *position* wird auf *relative* gesetzt, damit das Menü nicht verdeckt wird.

```
#hmenu {
	width:40px; height:40px;
	top: 4px; right:10px;
	cursor:pointer;
	background-image:url(Hamburger.png);
	background-size:100% 100%;
	position:fixed;
}
nav#global{
	display:none;
	position:relative;
}
}
```

Nützliche Verfahren für das Responsive Webdesign

Die folgenden Abschnitte zeigen einige ausgewählte Standard-Verfahren für das Gestalten von responsiven und adaptiven Webseiten. Zum Prüfen des Ergebnisses eigenen sich – sofern nicht ausreichend Handys zur Verfügung stehen – Simulatoren. So enthält der *Google Chrome Browser* (Rechte Maustaste „Element untersuchen“) ausgezeichnete Simulationen gängiger Devices. Die *Web Developer Toolbar* bei Firefox bietet ebenfalls Simulationstools.

Auch im Internet finden sich zahlreiche Online-Simulatoren. Es lohnt sich, ein wenig zu recherchieren und die unterschiedlichen Qualitäten der Simulatoren kennenzulernen.

Bilder skalieren

Hintergrundbilder wiederholen sich per Default so oft, wie sie in den Hintergrund hineinpassen, sie „kacheln“.

```
body {
	background-image: url(alien-space-734-520.png);
}
```

Zusätzliche Eigenschaften wie *no-repeat*, *repeat-x* und *repeat-y* schränken das Kacheln ein.
Ein echtes Skalieren des Hintergrundbildes ist dadurch zu erreichen, dass das die CSS-Eigenschaft *background-size* verwendet wird. Auch hier gibt es verschiedene Werte, die das Verhalten des Hintergrundbildes beeinflussen.

```
body {
	background-image: url(alien-space-734-520.png);
	background-repeat: no-repeat;
	background-size: 100%;
	background-color: #000;
}
```

Um mögliche Anzeige Fehler zu begrenzen, ist zusätzlich die Hintergrundfarbe definiert, die dem Hauptfarbton des Hintergrundbildes entspricht.
Auch Vordergrundbilder können so definiert werden, dass sie sich an der Größe des Browserfensters orientieren.

```
figure {
	margin-left: 0px;
	width:100%;
}
figure img {
	width: 100%;
	height:100%;
}
```

Das Bild im *figure*-Element passt sich der Größe an.

Menüs anpassen

Beim Verändern des Browserfensters sollten sich ab einer bestimmten Größe die Menüs anders anordnen. Dies hat zwei Gründe:

- Die Größe und damit die Benutzbarkeit der Schaltflächen zu gewährleisten
- Die Information im Fokus zu halten und nicht hinter endlosen Menübäumen zu verstecken

Das erste Beispiel verändert lediglich die Lage der Menüs. Sie bleiben leicht anzutippen. Die Umschaltgröße ist die Bildschirmbreite von 480 Pixel. Ab dieser Größe floatet das Menü. Ist die Bildschirmbreite kleiner, besteht das Menü aus Blöcken. Der eigentliche Inhalt rutscht nach unten.
Im zweiten Beispiel rutscht das Menü nach unten, während der Inhalt weitgehend sichtbar bleibt. Hier wird in der breiten Ansicht das Menü absolute positioniert.
Derselbe Effekt lässt sich auch die *float:left* und *float:right* erzielen wie im nachfolgenden Beispiel.
Das nächste Beispiel zeigt zwei Block-Menüs beim Verkleinern des Browserfensters.

Schatten für Container und Schrift

Für die Lesbarkeit kann es von Vorteil sein, sowohl die Schrift als auch die Text-Boxen mit Schatten zu hinterlegen. Ein heller Schatten von dunkler Schrift, die auf einem dunklen Hintergrund liegt, macht sich immer gut, wie Apple bewiesen hat.

Definition von Text- und Box-Schatten

```
body {
	font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif;
	text-shadow: 2px 2px 3px #F7F7F7;
	background-color:#ddd;
}
```

Definition eines Verlaufs

```
background: -moz-linear-gradient(top, rgba(0,0,0,0) 0%, rgba(0,0,0,0.39) 100%);
background: -webkit-linear-gradient(top, rgba(0,0,0,0) 0%, rgba(0,0,0,0.39) 100%);
background: -o-linear-gradient(top, rgba(0,0,0,0) 0%, rgba(0,0,0,0.39) 100%);
background: -ms-linear-gradient(top, rgba(0,0,0,0) 0%, rgba(0,0,0,0.39) 100%);
background: linear-gradient(to bottom, rgba(0,0,0,0) 0%, rgba(0,0,0,0.39) 100%);
```

Noch sind Vendor-Präfixe nötig, um die Browser verschiedener Hersteller anzusprechen. Eine ausführlichere Darstellung der Präfixe befindet sich im Quellcode des Beispieldokuments.

Zusammenfassung

„Responsive Webdesign“ ist mehr als nur ein paar technische Verfahren. Dahinter steht die Philosophie, HTML-Dokumente so zu gestalten, dass sie sich im Idealfall auf jede nur denkbare Präsentationsbedingung einstellen. Nicht die Benutzer müssen ihre Geräte so einstellen, dass sie Internet-Dokumente lesen können, vielmehr müssen sich die Internet-Dokumente auf die verschiedenen Geräte einlassen. Damit dies möglich ist, wenden Webdesigner Media-Attribute, Media-Querys und das Meta-Viewport-Attribut an. Mehrere Verfahren wie Menüs auf unterschiedliche Displays reagieren, haben sich eingebürgert. Das Verhalten lässt sich mit Simulatoren messen. Bei einfachen Webseiten genügt es schon, das Browserfenster zu vergrößern und zu verkleinern und zu beobachten, ob sich die Website in der gewünschten Weise verhält.

Einführung in XML

Was ist XML?

Spezifische und generische Codierung

Die Idee, den Inhalt elektronischer Dokumente mit „Etiketten“ (Tags) zu kennzeichnen, geht auf *William W. Tunnicliffe* (1922-1996) zurück. Ursprünglich wurden elektronische Manuskripte mit Control Codes und Makros versehen, um die gewünschten Formatierungen zu erhalten. Die Lesbarkeit dieser spezifischen Codierung war maschinen- und anwendungsabhängig; plattformübergreifende Dokumente konnten auf diese Weise kaum realisiert werden.
Im September 1967 äußerte Tunnicliffe auf einer Konferenz die Idee der Auszeichnungssprache. Er schlug vor...

- anstelle spezifischer Codes
- eine aussagekräftige, allgemeingültige Etikettierung

... um elektronische Dokumente unabhängig von der Betriebsumgebung auszuzeichnen. Diese sogenannte *generische Codierung* verbreitete sich allmählich in den späten Sechzigern.

➡ **Nachruf auf William W. Tunnicliffe**

Spezifische Codierung

In Texten eingestreute Control-Codes zeigen nur auf der jeweils spezifischen Maschine das erwünschte Ergebnis.

```
- Epson matrix printers
  CHR(27)+"M
- IBM Proprinter
  CHR(27)+":
- Hewlett Packard LaserJet II
  CHR(27)+"(s16H".
```

Abgrenzung von XML zu XHTML und CSS

1998 gab das W3C Empfehlungen zur XML-Spezifikation heraus. Heute nimmt XML eine überragende Bedeutung ein. XHTML ist ein mit Hilfe von XML definiertes HTML.

XML: Daten

XML wird verwendet:
- Als Metasprache um neue Auszeichnungssprachen für bestimmte Zwecke zu definieren
- Als Beschreibungssprache/Austauschsprache für Datensätze
- Um Daten strukturiert aufzunehmen und weiterzugeben

HTML: Dokumentstruktur

HTML wird verwendet:
- Um Daten strukturiert darzustellen
- Um Dokumente durch semantische Auszeichnung zu strukturieren (Texte, Bilder, Hyperlinks, Medien…)

CSS: visuelles Erscheinungsbild

CSS wird verwendet, um das visuelle Erscheinungsbildes von HTML- und XML-Elementen festzulegen.

Anwendungsfelder für XML

XML heißt *EXtensible Markup Language*. Es ist eine Auszeichnungssprache, keine Programmiersprache, XML speichert und strukturiert Daten. XML ist eine W3C-Empfehlung, enthält jedoch keine vorgefertigten Tags (wie HTML), man muss/darf eigene Tags erstellen. XML ist KEIN Ersatz für HTML. Ein XML-Dokument ist eine in utf-8 codierte Textdatei. Mit XML lassen sich neue Auszeichnungssprachen entwickeln.

Ein XML-Dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset>
	<!-- created with Free Online Sitemap Generator www.xml-sitemaps.com -->
	<url>
		<loc>http://www.abc.de/</loc>
		<lastmod>2011-08-28T12:50:10+00:00</lastmod>
	</url>
	<url>
		<loc>http://www.abc.de/index.html</loc>
		<lastmod>2011-08-28T12:50:10+00:00</lastmod>
	</url>
	<url>
		<loc>http://www.abc.de/html/seite001.html</loc>
		<lastmod>2010-09-26T10:58:21+00:00</lastmod>
	</url>
</urlset>
```

XML-basierte Auszeichnungssprachen:

- DocBook
- *XHTML* (XML-konformes HTML)
- *TEI* (Text Encoding Initiative)
- *SVG* (Vektorgrafiken)
- *Geography Markup Language* (GML)
- *OpenStreetMap* (OSM)
- *MusicXML* (Notendaten, aufgeschriebene Musik)
- *XML Encryption*
- *AutomationML* (Format zur Speicherung von Anlagenplanungsdaten)
- *IODD* (Format zur Beschreibung von Sensoren)
- *SOAP* (Webservices)
- *ONIX* (Verlagswesen)
- *Onyx* (Prüfungs-) fragen)
- u.v.a.m

XML-Software

Software zum Anzeigen, Erzeugen, Verändern, Verarbeiten und Validieren von XML-Dokumenten

XML-Editoren

- Altanova: http://www.altova.com/xml-editor/
- Editix: http://www.editix.com/
- Liquid-XML: http://www.liquid-technologies.com/XML-Editor.aspx
- Stylus-Studio: http://www.stylusstudio.com/
- <oXygen/>: http://www.oxygenxml.com/

- Microsoft XML Notepad 2007: http://www.microsoft.com/download/en/details.aspx?id=7973
- Essential XML-Editor: http://www.philo.de/xmledit/
- XML-Copy-Editor: http://xml-copy-editor.sourceforge.net/

XML-Validatoren (online)

- xmlvalidation: http://www.xmlvalidation.com/
- W3schools: http://www.w3schools.com/xml/xml_validator.asp
- Validome: http://www.validome.org/xml/
- XML-Validator: http://xmlvalidator.new-studio.org/

XML-Validatoren (offline)

- Meist enthalten in kommerziellen XML-Studios.
- Xmllint (Kommandozeilen-Tool, Siehe Google-Code)
- Tk Xmllint (Xmllint mit grafischer Benutzeroberfläche)

Aufbau eines XML-Dokuments

Dateiformat

Ein XML-Dokument besteht aus Textzeichen, es ist nicht binär codiert wie zum Beispiel eine exe-Datei. XML kann mit jedem einfachen Texteditor erzeugt und bearbeitet werden. Der Texteditor muss Unicode-Zeichen unterstützen.

➡ **Siehe unicode.org**

Unicode ist ein internationaler Standard, der für jedes Zeichen aller bekannten Schriftkulturen einen digitalen Code festlegt. Die Verwendung inkompatibler Kodierungen in verschiedenen Ländern soll beseitigt werden. Unicode wird ständig ergänzt.

Prolog

Das XML-Dokument beginnt mit dem Prolog. Als Prolog wird jener Bereich bezeichnet, der vor den XML-Daten notiert wird.

Minimaler Prolog

```
<?xml version="1.0"?>
```

Standard-Prolog

```
<?xml version="1.0" encoding="utf-8" standalone=”yes|no” ?>
```

- *Version="1.0":* Festlegung der XML-Version des nachfolgenden XML-Dokuments.
- *Encoding="utf-8":* Festlegung des Sprachcodierung des XML-Dokuments.
- *Standalone=“yes | no“:* Das XML-Dokument benutzt keine externe *DTD* (Document Type Definition), siehe unten

XML-Daten, Baumstruktur, Wohlgeformtheit

Beispiel für ein XML-Dokument

Jedes XML-Dokument benötigt ein Wurzelelement *(root-Element)*, hier: *< urlset >...</ urlset >.* Ausgehend von dem Root-Element verzweigen die Kind-Elemente, hier *<url>...</url>* und deren Kind-Elemente *<loc>...</loc>* und *<lastmod>...</lastmod>.*

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset>
      <url>
            <loc>http://www.abc.de/</loc>
```

```
		<lastmod>2011-08-28T12:50:10+00:00</lastmod>
	</url>
	<url>
		<loc>http://www.abc.de/index.html</loc>
		<lastmod>2011-08-28T12:50:10+00:00</lastmod>
	</url>
	<url>
		<loc>http://www.abc.de/html/seite001.html</loc>
		<lastmod>2010-09-26T10:58:21+00:00</lastmod>
	</url>
</urlset>
```

Elementnamen

Die Element-Namen sind frei erfunden. Durch die Wortwahl geben sie jedoch einen Hinweis darauf, welche Daten sie beinhalten.
Ein Elementname darf nicht mit einer Zahl beginnen. Alle Buchstaben sind erlaubt und alle Zahlen, jedoch frühestens ab der zweiten Stelle. Minus und Unterstrich sind zulässig sowie höhere Unicode-Zeichen, sofern die Datei direkt als Unicode gespeichert wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<Ölfässer>
	<Öl>Motor</Öl>
	<Öl>Fahrrad</Öl>
	<Öl>Nähmaschine</Öl>
</Ölfässer>
```

Gültige Elementnamen dürfen also auch Umlaute enthalten.

Wohlgeformtheit

Ein XML-Dokument heißt wohlgeformt, wenn gilt:

- Jedes Tag ist geschlossen. *<anfang>*Inhalt***</****anfang>* Ausnahmen sind Tags, die als leer definiert wurden. *<ende />*
- Attributwerte (siehe später) stehen in Anführungszeichen. *<Gruss sprache=“deutsch“>*
- Kein Tag enthält mehr als ein Attribut (siehe später) desselben Namens.
- Kommentare werden stets außerhalb von Tags notiert. Sie beginnen mit *<!--* und enden mit *-->*
- Die spitze Klammer < darf nicht innerhalb von Elementinhalten oder Attributwerten auftreten: *<* stattdessen verwenden.

Die Dokumenttyp-Definition

Beim Auszeichnen von Daten geschehen üblicherweise Fehler:

- Schreib-/Tippfehler in den Elementnamen
- Vergessene spitze Klammern
- Vergessene schließende Tags
- ...

Um Fehlern entgegenzuwirken, werden alle XML-Elemente und ihre Attribute im *Prolog* notiert. Dann kann ein Validator die Gültigkeit der Auszeichnung feststellen. Die Notierung der Elemente heißt *Dokumenttyp-Definition* (*DTD*). Die DTD ergibt den *DOCTYPE* des XML-Dokuments. **Jedes XML-Dokument darf nur aus einer einzigen DTD bestehen!**

- Die DTD beschreibt die Struktur des XML-Dokuments und gewährleistet so den fehlerfreien Datenaustausch.

- Die DTD kann im Prolog notiert oder in eine verknüpfte Datei extern ausgelagert werden.
- Die DTD legt die Strukturmerkmale eines Dokumentes fest und erlaubt die Überprüfung von Dokumenten auf Gültigkeit.
- Die DTD definiert
    - welche Elemente verwendet werden dürfen
    - welche Attribute zu welchem Element erlaubt und/oder erforderlich sind
    - wie Elemente ineinander verschachtelt werden dürfen.

Beispiel für DTD
Die DTD für das Dokument (s.o.) wird folgendermaßen notiert:

```
<!DOCTYPE urlset [
<!ELEMENT urlset (url+)>
<!ELEMENT url (loc, lastmod)>
<!ELEMENT loc (#PCDATA)>
<!ELEMENT lastmod (#PCDATA)>
]>
```

Im Klartext bedeutet die Definition:
Der Dokumententyp für das Root-Element *urlset* besteht aus dem Element *urlset*, das beliebig viele *url*-Elemente enthalten darf. Das Element *url* besteht wiederum aus genau den beiden Elementen *loc* und *lastmod*. Beide dürfen als Inhalt nur geparste Daten enthalten.
Anmerkung: Bei kleinen XML-Dokumenten wird normalerweise wegen des Aufwands kein Dokumententyp definiert.

Validierbare XML-Dokumente

Zurzeit existieren zwei Möglichkeiten, gültige – also validierbare – XML-Dokumente zu erzeugen. Die erste Möglichkeit besteht in der oben gezeigten *Dokumenttyp-Definition (DTD)*. Bei der zweiten Möglichkeit wird eine *XML-Schema-Definition (XSD)* erstellt. Beide Verfahren sind aktuell, wobei das Verfahren der DTD bei Neufassungen oder Neuentwicklungen von XML-basierten Sprachen zunehmend durch das Verfahren der XSD abgelöst wird. Für Sprachen, die in ihrer Konsistenz historisch gewachsen sind (*DocBook 4.xx*, *XHTML*, SVG, *MusicXML* oder *ONIX*), ist die DTD nach wie vor das Maß der Dinge.
Die folgenden beiden Code-Schnipsel zeigen Beispiele für die Validierbarkeit mit Hilfe der DTD und der XSD.

Mit DOCTYPE valides XML erzeugen

Die DTD kann sowohl intern, wie im Beispiel oben, als auch in einer separaten, externen DTD-Datei abgelegt werden.

Die Definition des DOCTYPE

```
<!ELEMENT urlset (url+)>
<!ELEMENT url (loc, lastmod)>
<!ELEMENT loc (#PCDATA)>
<!ELEMENT lastmod (#PCDATA)>
```

Die Einbindung erfolgt im Prolog des XML-Dokuments. Bedeutsam sind dabei die Schlüsselwörter *SYSTEM* und *PUBLIC*.

```
<!DOCTYPE urlset SYSTEM "sitemap-dtd-extern.dtd">
```

Das **Schlüsselwort SYSTEM** steht für eine nicht-öffentliche DTD, deren Speicherort (URL) bekannt ist. Im *urlset*-Beispiel ist es die Definitionsdatei, die alle Element- und Attributs-Definitionen

enthält. Naturgemäß befinden sich solche Definitionsdateien in einem geschützten, nicht-öffentlichen Speicherort. Das **Schlüsselwort PUBLIC** wird verwendet, wenn die DTD eine öffentliche DTD ist, die allgemein zugänglich ist und deren Einzigartigkeit durch einen Identifier (URI) garantiert ist. Dies ist zum Beispiel bei Auszeichnungssprachen der Fall, die mit Hilfe von XML definiert wurden.

```
<!DOCTYPE Katalog PUBLIC "-//OASIS//DTD DocBook V4.1//EN" "docbook.dtd">
```

Mit XML-Schema-Definition valides XML erzeugen

Eine *XML-Schema-Definition* beschreibt die Struktur eines XML-Dokuments und folgt dabei selbst den XML-Regeln. Der Begriff „Schema“ stammt ursprünglich aus dem Bereich der Datenbank-Modellierung. Ein Datenbank-Schema beschreibt die Datenstruktur in (relationalen) Tabellen.

- **XSD** heißt XML- Schema-Definition
- XML-Schema ist eine W3C-Empfehlung (Siehe hier, deutsche Übersetzung)

Das XML Schema:

- Definiert die XML-Elemente und Attribute in einem XML-Dokument
- Definiert Elemente als Kind-Elemente, deren Anzahl und Reihenfolge
- Definiert, ob ein Element leer sein kann
- Definiert Datentypen, Vorgabewerte und Konstanten

Eine der wichtigsten Vorteile von XML-Schema gegenüber einer Dokumentdefinition durch eine DTD ist die Unterscheidung von Datentypen. Dadurch ist es möglich, nicht nur die Datenstruktur, sondern auch die Daten selbst zu validieren.

Die Definition des Schemas (XSD)

```
<?xml version="1.0" encoding="UTF-8"?>
<!–Validiert mit http://www.utilities-online.info/ -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.westermann.de"
           targetNamespace="http://www.westermann.de"
           elementFormDefault="qualified">
  <xs:element name="urlset">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="url" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="url">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="loc" type="xs:string"/>
        <xs:element name="lastmod" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Verknüpfung des XML-Dokuments und seiner XSD

```
<urlset xmlns="http://www.westermann.de"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://www.westermann.de
                            sitemap-xsd.xsd">
… (weitere Elemente)
</urlset>
```

Die Schema-Definition wird an das Root-Element des XML-Dokuments *urlset* gebunden. Anders gesagt: Das XML-Dokument ist eine Instanz (XSI) der in der Schema-Definition (XSD) notierten Elemente und Attribute.
Über die Notation und die Bedeutung des Einbindungs-Codes gibt der nächste Abschnitt Auskunft.

Namensräume

„Problematische“ Element-Definitionen

Das Prinzip von XML beruht darauf, dass nur die Struktur der Sprache definiert ist. Es ist die Sache der Entwickler, die Elemente zu definieren – und zwar so, dass sie dem Anwendungszweck gerecht werden. Dagegen bestehen Sprachen wie HTML, XHTML oder CSS aus einem festen Vokabular, das in den W3C-Recommendations notiert ist.
Die Freiheit der Elementdefinition birgt die Gefahr, dass Elemente nicht eindeutig sind, wenn sie in verschiedenen Kontexten eingesetzt werden. Um zu vermeiden, dass sich dieselben Elementbezeichnungen überschneiden – mit unvorhersehbaren Ergebnissen bei der weiteren Dokumentenverarbeitung – empfiehlt das W3C das Konzept der *Namespaces* (Namensräume) und *Qualified Names* (qualifizierte Names).

Namespaces und Qualified Names

Namespaces ermöglichen es, Elemente in einem bestimmten Kontext (Raum) zu definieren. Die Namen der Elemente werden dazu für einen bestimmten Raum definiert, in dessen Kontext sie Gültigkeit besitzen. Der Name des Raums kann als Präfix vorangestellt werden.

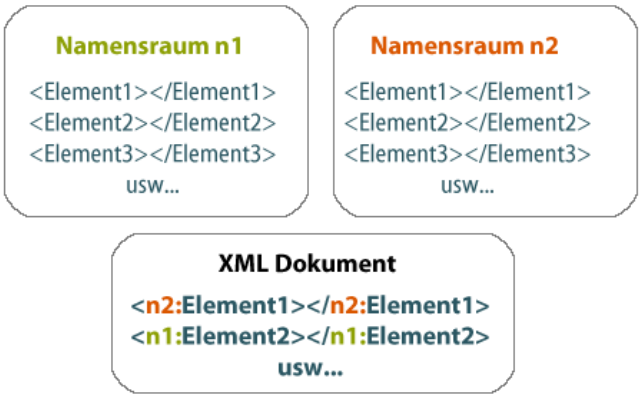


Abbildung 35: Illustration des Namensraum-Konzepts

Damit sich das Problem gleicher Elementnamen nicht auf das Problem gleicher Präfixe verlagert, muss das Namensraumpräfix eindeutig deklariert werden, denn gerade bei trivialen Präfixen wie „n1“ und „n2“ ist es nicht ausgeschlossen, dass sie mehrfach vorkommen, weil verschiedene Entwickler dieselbe Benennung vornahmen.
Um diesem Problem zu begegnen, schlägt das W3C die Verwendung der *QNames* vor. Es sind eindeutige Bezeichner. Die *QNames* (W3C), die qualifizierten Namen, sind der Kern des Namensraum-Konzepts, der *Namespaces*.

Eindeutige Bezeichner, W3C Namensraum-Empfehlung

Prinzipiell kann jeder eindeutige Bezeichner zur Definition eines Namensraums verwendet werden. Jedoch ist es gerade das Problem, eindeutige Bezeichner zu finden. Das W3C empfiehlt dazu folgendes Vorgehen (Aus: *http://www.w3.org/TR/1999/REC-xml-names-1999011*):

- An XML namespace is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names. (…)
- URI references which identify namespaces are considered identical when they are exactly the same character-for-character. (…)

- A namespace is declared using a family of reserved attributes. Such an attribute's name must either be xmlns or have xmlns: as a prefix. (...)

RFC 2396-konforme URI, Beispiel

```
http://www.kw.de/def/kunden
```

http://	www	kw.de	/def/kunden
http-Schema	Internet-Dienst	Domain (registriert)	Pfad
		Eindeutigkeit	Flexibilität

Tabelle 16: RFC 2396-konforme URI

Weiterhin sollen Namespace-Identifikatoren langfristig existent sein.

Die Bindung der Eindeutigkeit an ein Element
Mit Hilfe des reservierten Attributs *xmlns* (*xmlns* heißt: *xml-name-space*) wird der eindeutige Bezeichner an ein XML-Element gebunden.

```
<Präfix:Element xmlns:Präfix="Eindeutige Definition des Präfix">Inhalt</Element>
```

Hinter der URI befindet sich nicht notwendigerweise ein tatsächliches Dokument, vielmehr dient der Bezeichner ausschließlich der weltweiten Eindeutigkeit. Hieraus ergibt sich folgende Notierung für die Eindeutigkeit zweier zunächst identischer Elementnamen:

```
<n1:Id xmlns:Id="http://www.kw.de/def/namensraum1">14<n1:Id>
<n2:Id xmlns:Id="http://www.kw.de/def/namensraum2">001<n2:Id>
```

Die konkrete Wahl des Präfixes ist nicht von Bedeutung, nur die URI ist entscheidend.

Gültigkeitsbereich der Namespace-Deklaration

Die Namensraumdeklaration wird an die **Kind-Elemente** des deklarierten Elements vererbt, nicht jedoch an die Attribute.

```
<n1:Id lieferbar="nein">14</n1:Id>
```

Das Attribut *lieferbar* gehört nicht zum Namensraum. Erst das Präfix bindet es an den Namensraum:

```
<n1:Id n1:lieferbar="nein">14</n1:Id>
```

Das Schema-Element und seine Referenzierung

XSD: Das XML-Schema-Element

Im Folgenden soll des Namensraum-Konzepts anhand des „Sitemap“-Beispiels erläutert werden. Hier noch mal die Links zum Beispiel.
Das <schema>-Element ist das Wurzelelement jeder XML-Schema-Definition.

```
<?xml version="1.0"?>
<xs:schema>
...die weiteren XML-Elemente
</xs:schema>
```

Das Schema-Element enthält meist mehrere Attribute.

```
<xs:schema    xmlns:xs="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.bestimmte-domain.com"
              xmlns="http://www.bestimmte-domain.com"
```

```
        elementFormDefault="qualified">
```

Attribut xmlns:xs=”http://www.w3.org/2001/XMLSchema”
Das Attribut *xmlns:xs* bedeutet, dass alle Elemente und Attribute, die dem Namensraum *http://www.w3.org/2001/XMLSchema* entstammen, mit dem Präfix **xs** gekennzeichnet werden und den dort definierten Regeln folgen.

Attribut targetNamespace="http://www.bestimmte-domain.com"
Das Attribut *targetNamespace* sagt aus, dass alle Element- und Attribut-Definitionen im späteren XML-Dokument dem eigenen Schema entstammen.

Attribut elementFormDefault="qualified"
Mit diesem Attribut wird ausgedrückt, dass alle Elemente in der mit dem Schema instanziierten XML-Dokument auch tatsächlich dem Schema entstammen müssen.

XML: Referenzierung der Schema-Definition

Das XML-Dokument muss erfahren, auf welcher Schema-Definition es beruht. Dies geschieht durch die Referenzierung im Root-Element des XML-Dokuments. Für das Sitemap-Beispiel (s.o.) geschieht die Einbindung, indem auf die beiden Namensräume *xmlns:xs* und *targetNamespace* verwiesen wird.

```
<urlset xmlns="http://www.westermann.de"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.westermann.de
                            sitemap-xsd.xsd">
… (weitere Elemente)
</urlset>
```

Attribut xmlns="http://www.westermann.de"
Das Attribut sagt dem Validator, dass alle Elemente des XML-Dokuments im Namespace von *http://www.westermann.de* deklariert sind.

Attribut xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
Das Attribut besagt, dass im XML-Dokument eine Instanz **xsi** (Xml-Schema-Instanz) des Namensraums *http://www.w3.org/2001/XMLSchema-instance* zur Verfügung steht.

Attribut xsi:schemaLocation
Das Attribut besteht aus zwei Werten, die durch ein Leerzeichen (oder mehrere) getrennt sind. Der erste Wert nennt den eigenen Namensraum *http://www.westermann.de*. Der zweite Wert zeigt auf die physische Definitionsdatei *sitemap-xsd.xsd*.
Anders gesagt: Der Namespace *http://www.westermann.de* soll gegen die Definitionsdatei *sitemap-xsd.xsd* validiert werden.

Elementtypen einer DTD

Bestandteile der Dokumentstruktur

Die gültige Dokumentstruktur entsteht durch Deklarationen. Sie betreffen:
- Elementtypen
- Attributlisten
- Entities
- Notationen

Elementtyp: EMPTY

EMPTY steht für ein leeres Element.

Element-Definition:

```
<!ELEMENT Pflichtfach EMPTY>
```

Notierung im Dokument

```
<Pflichtfach />
```

Elementtyp: ANY

ANY steht für ein Element mit beliebigem Inhalt, auch mit weiteren Elementen als Inhalt.

Element-Definition

```
<!ELEMENT Beschreibung ANY>
```

Notierung im Dokument

```
<Beschreibung>Alles Mögliche</Beschreibung>
```

Elementtyp: (#PCDATA)

(#PCDATA) bedeutet, dass das Element nur *Parsed Character Data* enthalten darf, also zum Beispiel keine Tags.

Element-Definition

```
<!ELEMENT Strasse (#PCDATA) >
```

Notierung im Dokument

```
<Strasse>Bergweg 12</Strasse> | Falsch wäre: <Strasse>Bergweg <Nr>12</Nr></Strasse>
```

Kommentare

Kommentare werden notiert, wie es aus HTML bekannt ist.

Notierung im Dokument

```
<!-- Kommentar -->
```

XML-Verknüpfungszeichen

Verknüpfungszeichen "," (Komma)

Das Komma legt die Elementreihenfolge fest. Elemente müssen in der durch die Kommaseparierung angegebenen Reihenfolge erscheinen.

Element-Definition

```
<!ELEMENT Buch (Autor, Titel, ISBN)>
<!ELEMENT Autor (#PCDATA) >
<!ELEMENT Titel (#PCDATA) >
<!ELEMENT ISBN (#PCDATA) >
```

Notierung im Dokument:

```
<Buch>
    <Autor>Petra Müller</Autor>
    <Titel>Der See ruft</Titel>
    <ISBN>978-3-931871-22</ISBN>
</Buch>
```

Verknüpfungszeichen "|"

Entweder das eine ODER das andere Element kann benutzt werden.

Element-Definition:

```
<!ELEMENT Buch (Autor|Herausgeber, Titel, ISBN) >
```

Es kann entweder der Herausgeber oder der Autor des Buchs angegeben werden

Notierung im Dokument

```
<Buch>
    <Autor>Petra Seemann</Autor>
    <Titel>Der See ruft</Titel>
    <ISBN>978-3-931871-22</ISBN>
</Buch>
<Buch>
    <Herausgeber>Peter Bergmann</Herausgeber>
    <Titel>Der Berg ruft</Titel>
    <ISBN>978-3-931871-0X</ISBN>
</Buch>
```

Verknüpfungszeichen "*" (Multiplikationszeichen)

Das * Element ist optional, kann aber beliebig oft auftreten.

Element-Definition:

```
<!ELEMENT Buch (Autor*, Titel, ISBN)>
```

Es kann sich um ein Buch mit mehreren Autoren oder um eine Autorenlose Kompilation handeln.

Notierung im Dokument

```
<Buch>
    <Autor>Petra Müller</Autor>
     <Autor>Peter Mayer</Autor>
     <Autor>Pater Brown</Autor>
    <Titel>Der Himmel ruft</Titel>
    <ISBN>978-3-931871-22</ISBN>
</Buch>
```

Verknüpfungszeichen "?"

Das ? Element ist optional, es muss nicht oder kann einmal auftreten.

Element-Definition:

```
<!ELEMENT Buch (Autor, Titel, ISBN?)>
```

Die ISBN muss nicht angegeben werden, Autor und Titel müssen innerhalb von <Buch> genau 1x vorkommen.

Notierung im Dokument

```
<Buch>
    <Autor>Petra Müller</Autor>
    <Titel>Der See ruft</Titel>
</Buch>
```

Verknüpfungszeichen "+"

Das + Element muss mindestens 1x vorkommen, kann aber beliebig oft vorkommen.

Element-Definition

```
<!ELEMENT Buch (Autor, Titel, ISBN)+>
```

Notierung im Dokument in x-facher Wiederholung

```
<Buch>
    <Autor>Petra Müller</Autor>
    <Titel>Der See ruft</Titel>
    <ISBN>978-3-931871-22</ISBN>
</Buch>
```

Zusammenfassungszeichen "( )" (Klammer)

Das Zusammenfassungszeichen ( ) lässt das Verknüpfungszeichen (+) für alle Elemente der Klammer wirksam werden.

Element-Definition:

```
<!ELEMENT Buch (Autor, Titel, ISBN)+>
```

Das Element Buch darf beliebig oft vorkommen, muss aber stets aus den in der Klammer enthaltenen Unter-Elementen bestehen. So werden beispielsweise Datensätze definiert.

Das visuelle Erscheinungsbild von XML-Elementen

CSS und XSL

Das visuelle Erscheinungsbild lässt sich auf zwei Weisen festlegen

- **CSS** (wie in HTML)
- **XSL** (spezielle Stylesheet-Sprache für XML)

Auf XSL, bzw. auf die *XSL-Transformation*, gehe ich in einer späteren Veranstaltung ein.

CSS und XML

Der Verweis auf das CSS-Dokument geschieht im Prolog des XML-Dokuments.

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

Beispiel CSS

```
urlset, url, loc, lastmod {
       display:block;
       font-family: Verdana, Arial, sans-serif;
}
urlset {
       margin:1em;
}
url {
       margin-top:1em;
       margin-bottom:1em;
}
loc {
       font-weight:bold;
}
lastmod {
       font-size:0.86em;
       color:darkred;
}
```

Eine XML-Bücher-Tabelle erstellen

Aufgabenstellung

Die Aufgabe besteht darin, die Datensätze einer Bücher-Datenbank mit Hilfe von XML zu formulieren. Die Bücher-Tabelle soll pro Buch die folgenden Felder enthalten:

- Titel
- Autor
- ISBN
- Seiten
- Sprache

In die Tabelle sollen fünf Beispieldatensätze eingegeben und je nach DTD verändert werden, um beim Validieren deren Funktionsfähigkeit zu beweisen.
Es gelten die weiteren Anforderungen:

- Jedes Buch darf nur einmal vorkommen.
- Die XML-Tabelle kann beliebig viele Bücher enthalten.
- Die Datensatzfelder müssen die vorgeschriebene Reihenfolge einhalten
- Titel, Autor und ISBN sind zwingend mit Daten zu füllen
- Für Sammelbände kann anstelle des Autors auch der Herausgeber eingetragen werden.
- Die Angabe der Seitenzahl ist optional.
- Das Buch kann mehrsprachig sein.
- Die XML-Datei soll später formatiert ausgegeben werden.

Schritt 1

- Notieren von Prolog
- Verlinken der CSS-Datei
- Benennung des Root-Elements

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="buch.css" type="text/css"?>
<Literaturverzeichnis>
</Literaturverzeichnis>
```

Schritt 2

- Anlegen einer wohlgeformten XML-Datei
- Eingeben von fünf Beispiel-Datensätzen (hier nur 1 Datensatz)

```
<buch>
  <titel>Grundlagen der Astronomie</titel>
  <autor>Arthur Dent</autor>
  <isbn>978-3-86680-192-9</isbn>
  <seiten>466</seiten>
  <sprache>DE</sprache>
</buch>
```

Schritt 3

- DTD anlegen

```
<!DOCTYPE Literaturverzeichnis [
<!ELEMENT Literaturverzeichnis (buch+)>
<!ELEMENT buch (titel, autor, isbn, seiten, sprache)>
<!ELEMENT titel (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
```

```
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT seiten (#PCDATA)>
<!ELEMENT sprache (#PCDATA)>
]>
```

Schritt 4

- DTD anpassen: Titel, Autor und ISBN sind zwingend
- Anstelle des Autors auch der Herausgeber eingetragen werden

```
<!ELEMENT buch (titel, (autor | herausgeber), isbn, seiten, sprache)>
<!ELEMENT herausgeber (#PCDATA)>
```

Schritt 5

- DTD anpassen: Die Seitenzahl ist optional
- Das Buch kann mehrsprachig sein.

```
<!ELEMENT buch (titel, (autor | herausgeber), isbn, seiten?, sprache*)>
```

Schritt 6

- CSS-Datei erstellen
- Erscheinungsbild prüfen

```
Literaturverzeichnis {
	font-family: Verdana, Arial, sans-serif;
	margin: 16px;
}
buch, titel, autor, herausgeber, isbn, seiten, sprache {
	display:block;
}
buch {
	margin: 1em 0 1em 0;
}
titel {
	font-weight:bold;
}
seiten:after {
	content: " Seiten";
}
sprache:before {
	content: "Sprache: ";
}
```

Wer CSS zusammen mit XML anwendet, sollte beachten, dass XML-Elemente von Hause aus Inline-Elemente sind. Sie werden in einer einzigen Zeile dargestellt. Um XML-Elemente in Absätzen darzustellen, müssen sie als Block-Element definiert werden.

```
buch, titel, autor, herausgeber, isbn, seiten, sprache {
	display:block;
}
```

Zusammenfassung

Die Auszeichnungssprache dient dem Markieren von Daten und damit deren Strukturierung. Es gibt keine vorgefertigten XML-Elemente mit definierten Eigenschaften, vielmehr werden alle benötigten Elemente bedarfsgerecht entwickelt. Für die Logische Ordnung und die Konsistenz der

Daten kann die Dokument-Typ-Definition dienen, kurz DOCTYPE. Man unterscheidet wohlgeformte und valide Daten. Die Darstellung der Daten kann mit Hilfe von CSS erfolgen. Die Struktur der Elemente wird im Wesentlichen durch die Verknüpfungszeichen bestimmt. Das Konzept der Namensräume und der qualifizierten Namen verhindert, dass sich Elemente gleichen Namens zu mehrdeutigen Ergebnissen überlagern.

Erzeugen valider XML-Dokumente

DTD: Verfeinerte Dokumenttyp-Definitionen

Element-Attribute definieren

Attribute (Beifügungen) sind nähere Bestimmungen des Elements. Mit Attributen können XML-Elemente so definiert werden, dass sie zusätzlich zu ihrem Inhalt Informationen enthalten, die diesen näher bestimmen.

Beispiel**:**

```
<Vorname Geschlecht="w" Vorkommen="selten">Patrizia</Vorname>
<Vorname Geschlecht="m" Vorkommen="oft">Helmut</Vorname>
```

Das Element <Vorname> wird durch die Attribute *Geschlecht* und *Vorkommen* näher bestimmt. Die Reihfolge der Notierung von Attributen im XML-Element ist beliebig.

Attribute werden in der DTD mit dem Schlüsselwort *<!ATTLIST* festgelegt.
Beispiel:

```
<!ATTLIST Vorname Geschlecht CDATA #REQUIRED>
<!ATTLIST Vorname Vorkommen CDATA #IMPLIED>
```

Oder:

```
<!ATTLIST Vorname Geschlecht CDATA #REQUIRED
                  Vorkommen CDATA #IMPLIED>
```

Allgemein:

```
<!ATTLIST Elementname Attributname Typ Modifikator>
```

Auch wenn das Element nur ein einziges Attribut enthält, wird es in einer ATTLIST definiert.

Attribut-Typen

```
<!ATTLIST Elementname Attributname Typ Modifikator>
```

Typ: CDATA

Das Attribut kann beliebige Zeichenketten enthalten.

Typ: NMTOKEN

Wie CDATA, jedoch mit den Einschränkungen:

- Keine Leerzeichen im Attributwert
- Attributwert darf weder mit „xml“, noch mit „XML“ anfangen
- Attributwert darf nur mit A-Z, a-z, 0-9 oder _ anfangen

Typ: ID

Das Attribut muss einen Wert enthalten, der im gesamten XML-Dokument einmalig ist.

Typ: Aufzählung

Die Aufzählung wird wie folgt notiert:

```
<!ATTLIST Ampel Farbe (rot| gelb | grün) #REQUIRED>.
```

Der Wert des Attributs kann nur "rot", "gelb" oder "grün" sein - NICHT "blau".

Typ: IDREF
Das Attribut enthält einen Verweis auf eine ID. Es besteht aus einer Zeichenkette, die im XML-

Dokument als ID vorkommt. Mit IDREF und ID können Bezüge zwischen Elementen hergestellt werden.

Anwendung von ID und IDREF

Mit IDREF und ID können Bezüge zwischen Elementen hergestellt werden.
Ein XML-Dokument enthält beispielsweise die Elemente <Datensatz> und dessen Kind-Element <Name>. Man kann <Datensatz> die Attribute *Laufnummer* (Typ ID) und *verwandt mit* (Typ IDREF) zuordnen und dem Element < Beziehung> das Attribut IDREF.

Attribute in der Übersicht

Attribut-Typ	**Bedeutung**
CDATA	Das Attribut kann beliebige Zeichenketten enthalten
NMTOKEN	Wie CDATA, jedoch mit eingeschränkten Zeichen
ID	Das Attribut muss einen Wert enthalten, der im gesamten XML-Dokument einmalig ist.
IDREF	Das Attribut enthält einen Verweis auf eine ID. Es besteht aus einer Zeichenkette, die im XML-Dokument als ID vorkommt.
ENTITY	Der Attributwert muss mit dem Namen einer externen, ungeparsten Entity übereinstimmen.
NOTATION	Siehe unten. Der Wert eines Attributs vom Typ NOTATION, ist der Name einer Notation
Aufzählung	Entweder-Oder-Aufzählung möglicher Werte, zum Beispiel: (rot | gelb | grün)
	Pluralformen
NMTOKENS	Wie NMTOKEN, jedoch darf der Attributwert aus einer durch Leerzeichen getrennten Werteliste bestehen
IDREFS	Das Attribut enthält eine Liste von Referenzen, getrennt durch das Leerzeichen
ENTITIES	Eine Werteliste, deren Werte durch Leerzeichen getrennt sind
NOTATIONS	Eine Liste von Notationen, die durch Leerzeichen getrennt sind

Tabelle 17: XML-Attribute

Modifikatoren

```
<!ATTLIST Elementname Attributname Typ Modifikator>
```

Der *Modifikator* (oder Attributvorgabewert) definiert den Status des Attributs. Er kann drei Werte annehmen.

#IMPLIED

```
<!ATTLIST Name Anrede (Frau|Herr) #IMPLIED>
```

Das Attribut KANN angegeben werden.

#REQUIRED

```
<!ATTLIST Name Anrede (Frau|Herr) #REQUIRED>
```

Das Attribut MUSS angegeben werden.

#FIXED

```
<!ATTLIST Name Anrede CDATA #FIXED "Sehr geehrte Damen und Herren">
```

Das Attribut hat stets denselben einzig erlaubten Wert.

Regeln für Attribute

Ein Attribut darf nicht mehrmals im gleichen Element benutzt werden. Der Wert eines Attributs darf kein < enthalten. Der Wert eines Attributs darf nicht auf ein externes Entity verweisen (siehe unten). Attribute bestehen aus dem Attribut-Namen, dem Attribut-Typ und einem Modifikator.

Attribute versus Elemente

Problem: Die gleichen Informationen lassen sich auf unterschiedliche Weise darstellen.

Beispiel 1, DTD:

```
<!DOCTYPE Kleidung [
<!ELEMENT Kleidung (Kleidungsstück)>
<!ELEMENT Kleidungsstück (#PCDATA)>
<!ATTLIST Kleidungsstück
          Farbe CDATA #REQUIRED>
]>
```

Beispiel 1, Data

```
<Kleidung>
     <Kleidungsstück Farbe="blau">Pullover</Kleidungsstück>
</Kleidung>
```

Oder Beispiel 2, DTD:

```
<!DOCTYPE Kleidung [
<!ELEMENT Kleidung (Kleidungsstück, Farbe)>
<!ELEMENT Kleidungsstück (#PCDATA)>
<!ELEMENT Farbe (#PCDATA)>
]>
```

Beispiel 2, Data

```
<Kleidung>
     <Kleidungsstück>Pullover</Kleidungsstück>
     <Farbe>blau</Farbe>
</Kleidung>
```

Was ist richtig?

Beispiel für "gutes" und "weniger gutes" XML

Die oben genannten Varianten sind beide valide. Es gibt keine festgelegte Regel zur Verwendung von Attributen. Attribute sollen jedoch XML-Elemente näher bestimmen und nicht selbst den Dateninhalt repräsentieren. Letztlich ist es aber jedem selbst überlassen, die passende Form zu finden.

Beispiel "gut" und valide

Die Attribute werden gemäß ihrer Bestimmung eingesetzt und treffen nähere Aussagen zum Element *Buch*.

```
<Buch Genre="Fantasy" lieferbar="ja">
   <Titel>Die Schlacht am Toaster</Titel>
   <Autor>Heiner Bauer</Autor>
   <Preis>12,80</Preis>
</Buch>
```

Beispiel "weniger gut" und valide
Die Attribute repräsentieren den gesamten Inhalt.

```
<Buch Genre="Fantasy" Titel="Die Schlacht am Toaster" Autor="Heiner Bauer"
Preis="12,80">lieferbar</Buch>
```

DTD: Entitys

Allgemeine Entities
Entitäten (*Entities*) repräsentieren Informationen. Sie enthalten komplexe Information in prägnanter Kurzform.

```
<!ENTITY text1 "Zu Risiken und Nebenwirkungen fragen Sie Ihren Arzt oder
Apotheker.">
<!ENTITY text2 "Nicht apothekenpflichtig.">
```

Sowohl der Element-Inhalt *(<Bemerkung>&text1;</Bemerkung>*) als auch das Element-Attribut *(<Produkt Rezept="&text2;">Hustenbonbons</Produkt>*) kann auf diese Verkürzungen verweisen. Der Verweis beginnt mit dem vorangestellten *&*-Zeichen und endet mit dem nachgestellten *;*-Zeichen, siehe unten. Die Entity spart Speicherplatz. Erst beim Parsen wird die Verbindung zwischen Entity in XML-Element hergestellt.

```
<Datensatz>
    <Produkt>Nasentropfen</Produkt>
    <Bemerkung>&text1;</Bemerkung>
</Datensatz>
<Datensatz>
    <Produkt Rezept="&text2;">Hustenbonbons</Produkt>
    <Bemerkung>wohlschmeckend</Bemerkung>
</Datensatz>
```

Vordefinierte Entities
Für die Syntax-Zeichen (aber auch für andere Zeichen) liegen vordefinierte Zeichenfolgen vor. Durch sie wird es möglich, spitze Klammern und Anführungszeichen innerhalb des Inhalts von XML-Elementen zu benutzen.

```
&gt; größer-als (>)
&lt; kleiner-als (<)
&amp; kaufmännisches-und (&)
&quot; doppeltes Anführungszeichen (")
&apos; einfaches Anführungszeichen oder Apostroph (')
```

Notierung im XML-Dokument:

```
<wert>a > b</wert> <!–falsch -->
<wert>a &gt; b </wert> <!–richtig -->
```

Um sicherzugehen, dass der Browser die vordefinierten wirklich erkennt, können diese Zeichen auch als Entities zusätzlich aus nummerischen Zeichenreferenzen definiert werden.

```
<!ENTITY apos "&#39;">
<!ENTITY quot "&#34;">
```

Parameter-Entity
Eine **Parameter Entity** unterscheidet sich zunächst von der **allgemeinen Entity** durch ihre Notierung, siehe **"% "-Zeichen** plus Leerzeichen.

Entity-Beispiele

Allgemeine Entity
Hier noch mal das Beispiel einer allgemeinen Entity. *Betriebssystem1* repräsentiert lediglich den Inhalt „Android Version 5.0 (Key Lime Pie)“.

```
<!ENTITY Betriebssystem1 "Android Version 5.0 (Key Lime Pie)">
```

Die **Allgemeine Entity** *&Betriebssystem1;* wird beim Parsen eines XML-Elements wirksam. Umschlossen von XML-Elementen oder von den Anführungszeichen der Attribute verweist sie auf den definierten Inhalt.

Parameter-Entity
Die Parameter-Entity *% Betriebssystem* verweist dagegen auf XML-Elemente, die in einer externen DTD definiert wurden. Sie ergänzt die interne DTD oder eine weitere externe DTD.
Die extern definierten Elemente sind im folgenden Beispiel die Elemente *bsName* und *bsVersion*. Sie stammen aus der DTD „Handy-Betriebssystem.dtd“.

```
<!ELEMENT Betriebssystem (bs+)>
<!ELEMENT bs (bsName, bsVersion)>
<!ELEMENT bsName (#PCDATA)>
<!ELEMENT bsVersion (#PCDATA)>
```

Im XML-Dokument verweist die Parameter-Entity % Betriebssystem auf die DTD.

```
<!ENTITY % Betriebssystem SYSTEM "Handy-Betriebssystem.dtd"> %Betriebssystem;
```

Modulare DTD mit Hilfe von Parameter-Entitys

Bei der Modellierung **komplexer Datenstrukturen** ist es häufig von Vorteil, nicht nur eine einzige, allumfassende DTD zu erstellen, sondern die Definitionen der erforderlichen Datenstruktur auf mehrere DTDs zu verteilen. Dies erhöht die Lese- und die Bearbeitungsfreundlichkeit. Anstatt einer einzigen, allumfassenden DTD hat die modulare DTD mehrere Vorteile:

- Keine Redundanzen
- Man könnte die gleichen DTD in unterschiedlichen Zusammenhängen verwenden
- Erleichterte Pflege der DTD

Kombinieren von DTD in Entities
Die DTD für die Handy-Marke-XML:

```
<!ELEMENT Marke (m+)>
<!ELEMENT m (mHersteller, mModell)>
<!ELEMENT mHersteller (#PCDATA)>
<!ELEMENT mModell (#PCDATA)>
```

Die DTD für das Handy-Betriebssystem-XML:

```
<!ELEMENT Betriebssystem (bs+)>
<!ELEMENT bs (bsName, bsVersion)>
<!ELEMENT bsName (#PCDATA)>
<!ELEMENT bsVersion (#PCDATA)>
```

Die Kombination der DTD-Dateien mit Hilfe von Parameter-Entities:

```
<!DOCTYPE HandyModelle [
```

```
<!ELEMENT HandyModelle (Handy+)>
<!ENTITY % Betriebssystem SYSTEM "Handy-Betriebssystem.dtd"> %Betriebssystem;
<!ENTITY % Marke SYSTEM "Handy-Marke.dtd"> %Marke;
<!ELEMENT Handy (mHersteller, mModell, bsName, bsVersion)>
```

Die Elemente *mHersteller*, *mModell* müssen in der DTD *Handymodelle* nicht neu definiert werden, sie stammen aus der Entity „% Marke“. Das gleiche gilt für *bsName* und *bsVersion*. Sie wurden bereits in der DTD definiert, die mit der Entity „% Betriebssystem“ verbunden ist.
Mit der Kombination von DTD-Dateien mit Hilfe von Parameter- Entities wird das Problem umgangen, dass ein XML-Dokument nur durch eine einzige DTD definiert werden darf.

XSD: Erstellen und anwenden von Schema-Definitionen

Vergleich XSD und DTD

	XML-Schema	**DTD**
Datentypisierung	Ja, alle üblichen Datentypen	Nur rudimentär, wie CDATA und ID
Namen	Globale Namen und lokale Namen	Globale Namen
Mehrere Schemata pro XML-Dokument	Ja, mit XML-Namensräumen	Nein, eine DTD pro XML-Dokument
Dynamische Schemata	Ja, Zuweisung zur Laufzeit möglich	Nein
Validierung	Ja, gegen Schema (Struktur und Daten)	Ja, gegen DTD (Nur Struktur)
… und andere		

Tabelle 18: Gegenüberstellung XML-Schema und DTD

XML-Dokument, definiert mit externer DTD

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE Person SYSTEM "millionaer.dtd">
<Person>
    <Name>Susanne Stegmann</Name>
    <Alter>22</Alter>
    <Millionär>true</Millionär>
</Person>
```

Die DTD

```
<!ELEMENT Person (Name, Alter, Millionär)+>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Alter (#PCDATA)>
<!ELEMENT Millionär (#PCDATA)>
```

XML-Dokument, definiert mit XSD
Die Attribute im Wurzelelement verweisen auf die Namensräume.

```
<?xml version="1.0" encoding="utf-8" ?>
<Person xmlns="http://www.ich.de/def"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.ich.de/def
        millionaer2.xsd">
     <Name>Susanne Stegmann</Name>
     <Alter>22</Alter>
     <Millionär>true</Millionär>
</Person>
```

XSD-Dokument: Die Schema-Definition beschreibt die Beziehung zwischen den XML-Elementen.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified"
           xmlns="http://www.ich.de/def"
           targetNamespace="http://www.ich.de/def">
<xs:element name="Person">
     <xs:complexType>
          <xs:sequence>
                <xs:element name="Name" type="xs:string" />
                <xs:element name="Alter" type="xs:decimal" />
                <xs:element name="Millionär" type="xs:boolean" />
          </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

XSD: einfache Elemente

Sogenannte einfache XML-Elemente (simple Elements) enthalten nur einen Wert, zum Beispiel einen Text, eine Zahl oder ein Datum.

```
<Name>Susanne Stegmann</Name>
 <Alter>22</Alter>
 <Millionär>true</Millionär>
```

Hier die zugehörigen Definitionen im Schema als *simpleTypes*:

```
<xs:element name="Name" type="xs:string" />
<xs:element name="Alter" type="xs:decimal" />
<xs:element name="Millionär" type="xs:boolean" />
```

In der Definition kann ein Default-Wert voreingestellt werden.

```
<xs:element name="Millionär" type="xs:boolean" default=“0“/>
```

Attribute

Attribute werden ebenfalls als *simpleTypes* definiert, hier mit Default-Wert .

```
<xs:attribute name="sprache" type="xs:string" default="DE"/>
```

XSD: komplexe Elemente

Komplexe Elemente, *complexType*, enhalten mehr als nur einen Wert, zum Beispiel andere Elemente und Sequenzen. Im folgenden Beispiel wird das Element „Person“ als *complexType* definiert. Das Element enthält eine Sequenz aus einfachen, allerdings unterschiedlichen Elementen.

```
<xs:element name="Person">
     <xs:complexType mixed="true" >
```

```
          <xs:sequence>
                <xs:element name="Name" type="xs:string" />
                <xs:element name="Alter" type="xs:decimal" />
                <xs:element name="Millionär" type="xs:boolean" />
          </xs:sequence>
    </xs:complexType>
</xs:element>
```

Das Element *<xs:sequence>* ist der *compositor*, er definiert die Reihenfolge der Kind-Elemente.. Die Kind-Elemente enthalten das type-Attribut, das ihnen Datentypen zuweist. Der Wert „zweiundzwanzig“ (als Wort) im Element Alter wird so vom Validator als fehlerhaft erkannt im Gegensatz zum nummerischen Wert „22“. Jedes Daten-Element wird mit *<xs:element>* eröffnet. Über das Attribut wird *name* wird der Bezeichner festgelegt, der Elementname.

XSD: Datentypen

XML-Schema stellt einige *atomare* (einfache, unteilbare) Datentypen bereit. Datentypen sind atomar, wenn sie weder weitere Elemente, noch Attribute enthalten.

Beispiele
- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Eine vollständige Übersicht finden Sie im W3C-Dokument XML Schema Teil 2: Datentypen. Dort finden Sie auch eine Übersichtsgrafik.

Übersicht: Merkmale der XML-Syntax

XML-Syntax

- Alle XML-Elemente beginnen mit einem öffnenden Tag und enden mit einem schließenden Tag (Start- und Ende-Tag).
- Es gibt leere Elemente. Sie vereinigen öffnende und schließende Tags.
- Tags unterscheiden zwischen Groß- und Kleinschreibung (case-sensitive).
- Tags bestehen aus ASCII-Zeichen, sie dürfen Ziffern enthalten, aber nicht mit Ziffern oder Sonderzeichen beginnen. XML-Tags dürfen keine Leerzeichen enthalten.
- Tag-Attribute müssen in Anführungszeichen stehen.
- Es gibt genau ein Wurzelelement.
- XML-Elemente müssen korrekt geschachtelt sein.
- Kommentare dürfen verwendet werden.

Wohlgeformtheit

- Wohlgeformtheit ist dann gegeben, wenn alle Syntax-Regeln eingehalten werden.

Gültigkeit

- XML-Dokumente sind gültig, wenn sie wohlgeformt und gegen eine DTD oder ein Schema-Dokument validierbar sind.

Bestandteile eines XML-Dokuments

- *Prolog* (Version, DTD intern oder extern, Einbindung einer Style-Datei)
- *Datenbereich* (XML-Elemente, die Daten enthalten)

- Jedes XML-Dokument benötigt genau ein *Wurzelelement*
- Die Schema-Definition ist an das Root-Element geknüpft
- XML-Elemente können *Attribute* besitzen
- XML-Dokumente sind hierarchisch strukturiert (*Baumstruktur*), es gibt Eltern-, Kind- und Geschwisterelemente.

CDATA

- Der XML-Parser erfährt, dass die enthaltenen Zeichen nicht als Auszeichnung (Markup), sondern als Character-Data (Text) zu interpretieren sind. CDATA wird beim Definieren von Attributen angewendet.
- Bei der Verwendung von Nicht-Charakter-Data kann das XML-Dokument **nicht geparst** werden.

#PCDATA

- Der XML-Parser erfährt wie bei *CDATA* dass die enthaltenen Zeichen nicht als Auszeichnung (Markup), sondern als Character-Data (Text) zu interpretieren sind. #PCDATA wird beim Definieren von Elementen angewendet.
- Bei der Verwendung von Nicht-Charakter-Data wird das XML-Dokument *dennoch geparst*, es ist aber nicht valide.

Zusammenfassung

Die Auszeichnungssprache XML entfaltet ihr Potenzial durch die Möglichkeit, Attribute und Entitäten zu deklarieren. Attribute verfeinern die Validierung, Entitäten erleichtern die Dateneingabe und erlauben modulare DTDs. Grundsätzlich sollte jedoch die Datenlast auf sinnvoll definierten Elementen liegen, nicht auf den Attributen. Mit der XML-Schema-Definition lassen sich komplexe, mit Datentypen ausgezeichnete XML-Dokumente verfassen. Hier kann nicht nur die Dokumentstruktur, sondern auch die Werte-Inhalt validiert werden.

XSL-Transformationen

Am Beispiel der Sprache *XSL* soll gezeigt werden, wie sich XML-Dokumente umwandeln lassen. Das Kapitel endet mit einem Ausflug nach *XML-Schema*, einer immer häufiger benutzen Sprache, um XML-Dokumente valide zu machen.

XSL (Extensible Stylesheet Language)

XSL ist eine XML-basierte Programmiersprachenfamilie. Der Begriff „Stylesheet“ lässt den Schluss zu, dass es sich bei der Sprache um so etwas Ähnliches wie „CSS“ handelt. Die Begriffsähnlichkeit führt jedoch auf eine falsche Fährte. XSL ist erheblich mächtiger als CSS. Mit XSL lassen sich XML-Dokumente in andere Dokumente umwandeln. Insgesamt ist XSL eine Menge aus Sprachen, die aus drei Untermengen besteht:

- *XSL-T*, Aufgabe sind Transformationen (Umwandlungen)
- *XSL-FO*, Aufgabe ist das Formatieren
- *XPath*, Aufgabe ist es, den Zugriff auf XML-Elemente zu regeln

XSL-FO soll hier nicht besprochen werden, während XPath und XSL-T zusammengehören.

XSL-Merkmale

XSL-Dokumente sind selbst XML-Dokumente, die validiert werden können – und zwar gegen das Schema *xslt*. In vielen Browsern sind Prozessoren integriert, die XSL-Dokumente einlesen und in das gewünschte Ausgabeformat, meist HTML, umwandeln. Die Umwandlungsprozessoren heißen *XSLT-Prozessoren*. Inzwischen liegt die XSL-T-Version 3.0 vor. Hier soll es jedoch um die Version 1.0 gehen. XSL ist eine W3C-Empfehlung.

XSL-Transformation von XML nach HTML

XSL-Dokumente werden im Prolog des XML-Dokuments eingebunden.

```
<?xml-stylesheet type="text/xsl" href="xsl-t-1.xsl"?>
```

Im ersten Beispiel transformiert die XSL-Datei ein einfaches XML-Dokument in HTML.

```
<Meine_Adresse>
    <Name>Gramberg</Name>
    <Vorname>Heinrich</Vorname>
    <Strasse>Uferstraße 12</Strasse>
    <Ort>Hintertupfingen</Ort>
</Meine_Adresse>
```

```
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>XSL-Transformation</title>
</head>
<body><h1>Meine Adresse</h1>
<p>Name: <strong>Gramberg</strong></p>
<p>Vorname: <strong>Heinrich</strong></p>
<p>Straße: <strong>Uferstraße 12</strong></p>
<p>Ort: <strong>Hintertupfingen</strong></p>
</body></html>
```

Öffnen Sie mit dem Firefox-Browser das XML-Dokument, das die Referenz zum Stylesheet enthält, und betrachten Sie das Ergebnis. Lassen Sie sich nicht den Quellcode, sondern den *erzeugten* Code

anzeigen (Siehe *Webdeveloper*-Toolbar).

XSL-Transformation von XML nach XML

Das zweite Beispiel zeigt die Transformation von XML nach XML, jedoch mit veränderter Feldreihenfolge und anderslautenden Elementnamen. Auf diese Weise wäre zum Beispiel die Lokalisierung eines XML-Files realisierbar.

```
<?xml version="1.0"?>
<my_address>
      <city>Hintertupfingen</city>
      <street>Uferstraße 12</street>
      <given_name>Heinrich</given_name>
      <surname>Gramberg</surname>
</my_address>
```

Das XSL-Dokument

Die Transformation beruht auf den folgenden Prinzipien: Das XSL-Dokument enthält das Template für die Ausgabe. Das Template folgt den HTML-Regeln. Das xsl-Element fügt an der gewünschten Stelle den gewünschten Wert ein. Für den Wert wird ein XPATH-Ausdruck verwendet, der den Ast in der Baumstruktur repräsentiert. Das XSL-Element ist ein leeres Element, der Verweis erfolgt über das Attribut *select*.

```
<xsl:value-of select="Meine_Adresse/Vorname"/>
```

Das XSL-Dokument ist ein valides XML-Dokument, das auf einem *XML-Schema* beruht (dazu später). Das Root-Element heißt *<xsl:stylesheet></xsl:stylesheet>*. Der Namensraum *xsl* ist an die URI *http://www.w3.org/1999/XSL/Transform* gebunden.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
....
</xsl:stylesheet>
```

Die Transformation findet im Template *<xsl:template match="/"></xsl:template>* statt. Das Attribut match="/" bedeutet, dass das Template auf das Root-Element des XML-Dokuments angewendet wird. Die Zeichen innerhalb des Template-Bereichs werden ohne Umwandlung an die Ausgabe gesendet. Das xsl-Element *<xsl:value-of select="Meine_Adresse/Name"/>* zeigt auf den Elementnamen des XML-Elements, das an dieser Stelle erscheinen soll. *Meine_Adresse/Name* ist ein *XPath*-Ausdruck, der an den XPath-Ausdruck des *match*-Attributs angehängt wird.

```
<p>Vorname: <strong><xsl:value-of select="Meine_Adresse/Name"/></strong></p>
```

Die Ausgabezeile stellt einen HTML-Absatz *p* dar. Das *strong*-Element formatiert den Inhalt als fett und bedeutend.

```
Vorname: Heinrich
```

Soll der HTML-Transformation ein DOCTYPE mitgegeben werden, ist folgende Textzeile erforderlich.

```
<xsl:text disable-output-escaping='yes'>&lt;!DOCTYPE html></xsl:text>
```

Das XSL-Dokument kann auch ein CSS-File enthalten.Steuerung des Transformationsprozesses, Kontrollanweisungen
Bisher wurde stets nur ein einziger Datensatz transformiert. Soll die Transformation für ein XML-

Dokument mit mehreren Datensätzen stattfinden, muss der XSL-Prozessor wissen, dass er durch alle XML-Datensätze loopen soll. Dies geschieht mit der Kontrollanweisung *xsl:for-each select*.

```
<xsl:for-each select="Adressenliste/Adresse">
      <h1>Adresse</h1>
      <p>Name: <strong><xsl:value-of select="Name"/></strong></p>
      <p>Vorname: <strong><xsl:value-of select="Vorname"/></strong></p>
      <p>Strasse: <strong><xsl:value-of select="Strasse"/></strong></p>
      <p>Ort: <strong><xsl:value-of select="Ort"/></strong></p>
</xsl:for-each>
```

Die folgenden Kontrollanweisungen beschreiben einige ausgewählte Möglichkeiten.

- Element-Knoten auswählen und/oder bearbeiten --> **XPath**
- Durch die Element-Knoten loopen --> **xsl:for-each**
- Die Ergebnisse des Loopens filtern, Vergleichsoperatoren anwenden --> **[Elementwert=Filter]**
- Die Ergebnisse des Loopens sortieren --> **xsl:sort**
- Bedingungen für das Loopen notieren --> **xsl:if**
- Mehrfache Bedingungen für das Loopen noteren --> **xsl:choose**
- HTML-Template anwenden --> **xsl:template** alternativ anstelle des Loopens *for:each* --> **xsl:apply-templates**

Alle folgenden Beispiele basieren auf demselben XML-Dokument und demselben CSS-Dokument.

Loopen

Mit *<xsl:for-each select=" Xpath">…</xsl:for-each>* weisen Sie den XSLT-Prozessor an, einen bestimmen Knoten, bzw. Verzweigung, zu bearbeiten ist. Sie loopen durch alle Kind-Elemente des Knotens. Das *select* -Attribut spezifiziert den Knoten/die Verzweigung durch einen *XPath-Ausdruck.* Ein XPath-Ausdruck funktioniert wie das Navigieren durch die Ordner eines Dateisystems.

Filtern

Auf *xsl:for-each* kann ein Filter angewendet werden. Der Filter-Ausdruck steht in eckigen Klammern:

```
<xsl:for-each select="Titelliste/Track[Interpret='John Lennon']">
```

Auch mehrere Kriterien und Wildcards sind mögliche Filterkriterien. Es lohnt sich, im Internet nach den Begriffen *xsl:for-each select wildcard multiple filter* zu suchen. Als Anhaltspunkt dient die folgende Tabelle der Vergleichsoperatoren.

Vergleichsoperatoren

Zeichen	Bedeutung
=	ist gleich
!=	ungleich
<	kleiner als
>	größer als

Tabelle 19: Vergleichsoperatoren in XSL

Sortieren

Mit *<xsl:sort select="XML-Element"/>* erzwingen Sie innerhalb des Loops eine bestimmte Sortierreihenfolge.

```
<xsl:sort select="Jahr"/>
```

Weitere Attribute erlauben das auf- und absteigende Sortieren sowie das Sortieren nach Zahlen und Strings.

```
<xsl:sort    select="XML-Element"            lang="language-code"
             data-type="text|number|qname"   order="ascending|descending"
             case-order="upper-first|lower-first"/>
```

Bedingung

Mit *<xsl:if Bedingung="Ausdruck">* fügen Sie in den Loop eine Bedingung ein. Es werden alle Titel angezeigt, die neuer sind als 2000.

```
 <xsl:if test="Jahr &gt; 2000">
```

Mehrfache Bedingung

Mit *<xsl:choose>, <xsl:when> und <xsl:otherwise>* ist es möglich, mehrfache Bedingungen bei der Ausgabe eines XML-Dokumentes zu realisieren. Je nach Datierung werden die Begriffe "OLD" oder "NEW" angefügt.

```
<xsl:choose>
      <xsl:when test="Jahr &gt; 2010">
            <td><xsl:value-of select="Jahr"/> NEW</td>
      </xsl:when>
      <xsl:otherwise>
            <td><xsl:value-of select="Jahr"/> OLD</td>
      </xsl:otherwise>
</xsl:choose>
```

Template anwenden

Das *<xsl:apply-template>*- Element kann eine Alternative zum *<xsl:for-each>* -Element sein. Während das <xsl:for-each> -Element durch die Elemente des angegeben Daten-Pfads loopt und in ihrer Reihenfolge abarbeitet, erkennt das <xsl:apply-template>- Element anhand des jeweiligen XML-Tags, welche Ausgaberegel anzuwenden ist.

XML-Schema und XSLT

Die folgenden Beispiele zeigen, wie XML, XSD und XSLT gemeinsam genutzt werden.

XML-basierte Sprachen

XML-Organisationen

W3C (*Word Wide Web Consortium*) ist *die* vielleicht bedeutendste nicht-kommerzielle Organisation, die Empfehlungen (*Recommendations*) für das Internet ausspricht und deren Publikationen das Gewicht von internationalen Normen besitzen.

➡ **Zur Website des W3C**

Nicht nur die Auszeichnungssprachen HTML und XML wurden dort standardisiert, sondern auch zahlreiche andere Sprachen und Verfahren, ohne die das Internet undenkbar wäre. Das W3C ist jedoch nicht die einzige Standardisierungsorganisation.
OASIS (*Advancing Open Standards for the Information Society*) ist eine nicht-kommerzielle Organisation, die sich mit der (Weiter-) Entwicklung von E-Business und Webservice-Standards befasst, von *Application Vulnerability Description Language* (AVDL) bis *XML Localisation Interchange File Format* (XLIFF) (Zum Austausch von Übersetzungsdaten). Bekannt sind unter anderem das *DocBook*-Format und das *Open Document Format* (ODF).

➡ **Die OASIS-Organisation Website**

DocBook

DocBook ist eine auf XML basierende Auszeichnungssprache, die für die Erstellung von technischen Dokumentationen geschaffen wurde. Der Schwerpunkt liegt auf Computerhandbüchern, Hardware und Software. DocBook dient nicht dazu, das Aussehen des Inhalts zu beschreiben, sondern den Inhalt mit Bedeutungen auszuzeichnen.

➡ **Zur Website Docbook.org**

Das folgende Beispiel gibt dem mit *<programlisting>* ausgezeichneten Text die Bedeutung von zitiertem Quellcode.

```
<example>
  <title>HTML mit CSS anzeigen.</title>
  <programlisting>
      ...
  </programlisting>
</example>
```

DocBook weist beispielsweise einem Inhaltsbereich die Bedeutung zu, dass es sich um Quellcode handelt, es ist dann die Aufgabe eines weiteren Tools (z.B. *XSL-FO*), diesem Bereich ein bestimmtes visuelles Erscheinungsbild zu verleihen. DocBook ist also kein Präsentationsformat, sondern ein Tool zur klaren inhaltlichen Strukturierung.
Verschiedene Werkzeuge transformieren DocBook-Dokumente in Präsentationsformate wie *HTML*, *RTF*, *PDF*. Die Umwandlung leisten die Sprache *XSLT* und die XSLT-Prozessoren wie *xslt-proc*, *xalan* oder *saxon*.
DocBook liegt als XML-DTD bis zur Version 4.xx vor, ab Version 5 als Schema RELAX NG (Siehe unten). Weiterhin gibt es ein vereinfachtes Subset *Simplified DocBook* sowie Erweiterungen für *MathML* uns *SVG*.
DocBook ist nicht einfach zu erlernen, es besteht aus über 400 Tags. Mit dem Tool *Asciidoc* wird Wiki-Text in DocBook transferiert.

ODF

ODF (*OASIS Open Document Format for Office Applications*) ein XML- basierter Standard für die Strukturierung von Bürodokumenten wie Texten, Tabellendokumenten, Präsentationen, Zeichnungen, Bildern und Diagrammen. Es wird zum Beispiel in *Open Office* und *Microsoft Word ab 2007* verwendet.
Im Gegensatz zu DocBook, dessen Schwerpunkt auf der professionellen Buchherstellung liegt, wird ODF für weniger anspruchsvolle Bürodokumente verwendet.

Je nach Art der Verwendung existieren unterschiedliche Dateiendungen (kleine Auswahl):

- .odt - OpenDocument-Text
- .ods - OpenDocument-Tabellendokument
- .odg - OpenDocument-Zeichnung
- .ott - OpenDocument-Textvorlage
- .ots - OpenDocument-Tabellenvorlage
- .otg - OpenDocument-Zeichnungsvorlage

Typischerweise besteht ein ODF-Dokument aus einer gepackten Dateisammlung. So lässt sich die Datei „Das ist Text.odt“ (*Open Document Textformat*) entzippen. Man darf sich durch die Dateiendung nicht irritieren lassen, einfach mit *7-zip* oder einem anderen Tool entpacken. Es entsteht eine Ordnerstruktur mit darin enthaltenen Dateien:

```
META-INF
      manifest.xml        // Das Inhaltsverzeichnis
Thumbnails
      thumbnail.png       // Eine Minigrafik der Textseite als Voransicht
content.xml               // Der Text und die Verweise auf Stile und Namensräume
meta.xml                  // Die Metadaten
mimetype                  // application/vnd.oasis.opendocument.text
settings.xml              // Namensraum-Verweise
styles.xml                // Die Stile
```

RELAX NG

In Konkurrenz zu *XML Schema* des *W3C* verabschiedete die OASIS-Organisation *RELAX NG* (*REgular LAnguage Description for XML New Generation*), eine Schemasprache für XML.
Die wesentlichen Eigenschaften von RELAX NG werden vom Entwicklerteam unter anderem als „einfach“, „leicht zu erlernen“ und vereinbar mit anderen XML-Sprachen bezeichnet. Im Gegensatz zu einer DTD, die alle XML-Sprachelemente in der Reihenfolge ihres Vorkommens beschreibt, folgt RELAX NG einem Muster-Konzept (Pattern). Damit eine Regel Anwendung findet, muss ihr Muster irgendwo im XML-Dokument vorkommen.
Weiterhin kann RELAX NG in einer Kurzform deklariert werden. Relax NG Schemas können in XML Schemas und DTDs konvertiert werden. Umgekehrt geht das nicht.

➡ **RELAX NG Homepage**

Epub-Format

Das *epub-Format* ist ein offener Standard für elektronische Bücher, *e-Books*. Die Seiten des e-Books werden als HTML- und CSS-Dokumente abgelegt und über XML-Daten (*container.xml*) verwaltet. Das File *opf* (open packaging format) enthält alle Verweise auf die verwendeten Dokumente, *ncx* strukturiert die Verweise. Alle Files werden gezippt und in *epub* umbenannt. Dasselbe Prinzip wie beim ODF ist hier erkennbar: Pakete machen und zippen. Die Metadaten folgen der Empfehlung von Dublin-Core.

RSS Feed

RSS (*Really Simple Syndication*) dient der einfachen und strukturierten Veröffentlichung von Änderungen auf Websites. RSS benutzt das XML-Format. RSS-Dienste werden meist als *RSS-Channels* angeboten. RSS-Feeds bestehen häufig aus einer Schlagzeile, einem Kurztext und dem Link zur Herkunftsseite.
Anmerkung: Eigentlich keine vollständige Sprache, sondern besonders spezifiziertes XML-Fileformat, Mini-Sprache.

XHTML

HTML war nicht die erste Auszeichnungssprache, ist aber die vermutliche bekannteste geworden. Ihre Systematik beruht auf *SGML*, der seit 1986 standardisierten Metasprache. In den neunziger Jahren des 20.Jahrhunderts wurde HTML auf der Basis von XML neu definiert, jener ebenfalls aus SGML hervorgegangenen Auszeichnungssprache. Damit entstand XHTML.

SVG

„Bilder“ im Internet sind normalerweise *Pixelgrafiken* in den Formaten *jgp*, *png* oder *gif*. Pixelgrafiken sind aus Bildpunkten aufgebaut, es liegt in der Natur der Sache, dass sie nicht skalierbar sind. Vergrößerte, skalierte Pixelgrafiken wirken „verpixelt“, unscharf, verschwommen. Eine scheinbar zusammenhängende Fläche wird in Bildpunkte aufgelöst.
Dagegen sind Grafiken, die mit Hilfe von *Vektoren* (*Vektorgrafiken*) beschrieben sind, unbegrenzt skalierbar, weil die Lage der Grafikelemente je nach Vergrößerungsstufe neu berechnet wird.
SVG ist eine W3C-Empfehlung.

Ein Kreis als SVG, orange mit grünem Rand

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
      baseProfile="full"
     width="300px" height="300px" viewBox="0 0 100 100">
     <circle cx="50" cy="50" r="40.5" fill="#ff9000" stroke="#8ea106" stroke-
width="3px"/>
</svg>
```

MathML

MathML (*Mathematical Markup Language*) ist eine XML-basierte Sprache zur Darstellung mathematischer Formeln und komplexer Ausdrücke. MathML ist eine W3C-Empfehlung.

$$x = \frac{\sqrt{a^2 - 4}}{a - 2}$$

Abbildung 36: Mathematische Ausdrücke mit MathML

Ob ein Browser MathML unterstützt, erfährt man auf der Testseite des W3C.

Beispiel MathML und SVG in HTML

Das folgende Beispiel zeigt ein HTML5-Dokument, das SVG und MathML kombiniert. Die Angabe der Namensräume ist in HTML5 prinzipiell nicht mehr notwendig.

➡ **Zum Namespace MathML**
➡ **Zum Namespace SVG**

Die XSL-Familie

XSL bildet eine Familie: *XSL is a family of recommendations for defining XML document transformation and presentation. It consists of three parts...()*

- *XSL-T* (Transformation)
- Nutzung von *XPath* (Zugriff und Referenzierung), ist eigenständig definiert
- *XSL-FO* (Formatting Objects)

XPATH

XPath ist eine Art Hilfssprache, deren Spezifikation einer Empfehlung des W3 Consortiums (deutsche Übersetzung) folgt. Die Sprache wird dazu verwendet, um auf Elementknoten von XML-Dokumente zuzugreifen und deren Dateninhalte zu manipulieren. Ein weiterer Bestandteil ist die Definition logischer Ausdrücke. Verwendung findet XPath zum Beispiel in XSLT.
Beispiel: Der XPath-Ausdruck *Titelliste/Track* beschreibt den Weg durch den Dokumentenbaum zum Element *Titel*.

XSL-FO

XSL-FO heißt *Extensible Stylesheet Language Formatting Objects*. XSL-FO ist eine Sprache, um XML-Daten für die Ausgabe auf dem Bildschirm, Papier oder anderen Medien zu formatieren, zum Beispiel das Vorlesen auf Sprachsynthesizern.
Im Prinzip hat XSL-FO eine ähnliche Aufgabe wie CSS. Während CSS für die Ausgabe von Bildschirmdokumenten optimiert ist, eignet sich XSL-FO vor allem für die hochwertige Formatierung umfangreicher Druckerzeugnisse. Die Verarbeitungsschritte führen vom XML-Dokument über die XSL-Transformation zur Formatierung.

Unterschied CSS und XSL-FO
Der wesentliche Unterschied ist der, dass XSL-FO als Bezugsgrößen Papierseiten und deren Ränder benutzt, während CSS auf elektronische Displays ausgerichtet ist.

- XML-Dokument --> Transformation zum FO-Baum --> FO-Prozessor (PDF, RTF, Postscript)

CSS oder XSL-FO?
Warum empfiehlt das W3C zwei verschiedene Stil-Sprachen?
Welche sollte ich verwenden?
Folgende „Bauchregel“ gilt:
Benutze CSS wenn du es kannst, und XSL-FO wenn du musst.
Use CSS when you can, use XSL when you must.

➡ ***Bert Bos*, siehe http://www.w3.org/Style/CSS-vs-XSL)**

Drei Arten, XML-Dokumente anzuzeigen

Je nach Aufgabenstellung sind drei Wege gangbar, um XML-Dokumente zu rendern (sichtbar zu machen).

Weg	Ausgangspunkt	Zwischenschritt 1	Zwischenschritt 2	Ergebnis
1	XML-Dokument -->		CSS -->	Präsentation
2	XML-Dokument -->	XSL-Transformation -->	XSL-FO -->	Präsentation
3	XML-Dokument -->	XSL-Transformation -->	HTML+CSS -->	Präsentation

Tabelle 20: XML-Dokumente sichtbar machen

Zusammenfassung

Auf XML basieren auch zahlreiche weitere Auszeichnungssprachen, sowohl komplexe als auch einfache. Die XSL-Familie ermöglicht verschiedene Workflows, um XML-Daten sichtbar zu machen.

Anhang

Abbildungsverzeichnis

Abbildung 1: Das Dokument erhält sein Erscheinungsbild (Dokument (1)) 14
Abbildung 2: Ein alternatives Erscheinungsbild (Dokument (2)) 14
Abbildung 3: Ein drittes Erscheinungsbild (Dokument (3)) 15
Abbildung 4: Anzeige des Dokuments mit Hilfe der eingebauten Darstellungsregeln 15
Abbildung 5: Korrekturzeichen nach DIN 16511 19
Abbildung 6: Auswahl einiger aus SGML abgeleiteter Auszeichnungssprachen 19
Abbildung 7: Ordnerstruktur für mehrere Webprojekte 24
Abbildung 8: Tim Berners-Lee 25
Abbildung 9: Marc Andreessen 25
Abbildung 10: Tim Berners-Lee's PC at CERN: “Bitte nicht ausschalten. Das ist ein Server!“ 25
Abbildung 11: Arten von HTML-Inhalt 33
Abbildung 12: Erscheinungsbild der "Biene"-Seite 38
Abbildung 13: Beispiel für eine Baumstruktur, parents, children, siblings 40
Abbildung 14: Aufteilung des Browserfensters in rechteckige Bereiche 46
Abbildung 15: Beispiel für einen scheinbar nicht-eckigen Bereich 46

Abbildung 16: Das Box-Modell 49
Abbildung 17: Illustration ausgewählter Längenmaße 50
Abbildung 18: Relative Schriftgrößen 51
Abbildung 19: Beispiel für ein Scribble 54
Abbildung 20: Beispiel für ein typisches Formular 61
Abbildung 21: Das Registrierungsformular und seine Elemente 65
Abbildung 22: Webseite "Am Meer" 76
Abbildung 23: "Flughäfen", horizontales Menü 77
Abbildung 24: "Flughäfen", vertikales Menü 77
Abbildung 25: Farb-Rad 84
Abbildung 26: Analoge, komplementäre und triadische Farben 85
Abbildung 27: Beispiel für ein Farbschema aus Schwarz, Weiß + Schmuckfarbe (Rot) 86
Abbildung 28: Triadisches Farbschema 86
Abbildung 29: Natürliches Farbschema 87
Abbildung 30: Links: Die Kapelle steht als Motiv klar vor dem Hintergrund. Rechts: Keine Figur hebt sich vom Hintergrund ab. 90
Abbildung 31: Das Chaosformular 90
Abbildung 32: Illustration von vier Gestaltgesetzen 91
Abbildung 33: Drei gleiche Zeichen für unterschiedliche Bedeutungen 91
Abbildung 34: Printdesign und Webdesign 96
Abbildung 35: Illustration des Namensraum-Konzepts 111
Abbildung 36: Mathematische Ausdrücke mit MathML 135

Tabellenverzeichnis

Tabelle 1: Globale Attribute für HTML-Elemente 31
Tabelle 2: Dokument, Dokumentenkopf und Textkörper 31
Tabelle 3: Ausgewählte Block-Elemente 32
Tabelle 4: Ausgewählte Inline-Elemente 33
Tabelle 5: Gliedernder Inhalt 34
Tabelle 6: Kopfbereiche und Fußbereiche 34
Tabelle 7: Das Verhalten von positionierten HTML-Elementen 53
Tabelle 8: Bildformate 56
Tabelle 9: Embedded Content 58
Tabelle 10: Ausgewählte Merkmale von Cookie und Webstorage 66
Tabelle 11: Ein Cookie (entnommen aus Chrome) 66
Tabelle 12: Merkmale von Schriften 80
Tabelle 13: RGB- und CMYK-Farbraum 84
Tabelle 14: Farbton, Sättigung, Helligkeit 84
Tabelle 15: Gerätespezifische Media-Attribute 100
Tabelle 16: RFC 2396-konforme URI 112
Tabelle 17: XML-Attribute 121
Tabelle 18: Gegenüberstellung XML-Schema und DTD 125
Tabelle 19: Vergleichsoperatoren in XSL 131
Tabelle 20: XML-Dokumente sichtbar machen 136

Verzeichnis der Links

➡ *HTML-Parser* (http://htmlparser.sourceforge.net/). 21
➡ Siehe Länderkürzel. 31
➡ Technical Report W3C http://www.w3.org/DOM/DOMTR 40
➡ W3C Color-Modul: http://www.w3.org/TR/css3-color/#foreground 41

- W3C-Modul: http://www.w3.org/TR/css3-background/#borders 41
- Media-Events des W3C 59
- Das zugehörige W3C-Dokument 74
- CSS4-W3C-Dokument (Editors Draft) 78
- Siehe typetester.org 82
- W3C Schrift-Modul, alle Schrifteigenschaften 82
- Typografisches Grundwissen (W3C) 82
- W3C Textmodul 82
- Weitere Infos bei W3C, Listen. 83
- Zum Color Scheme Designer 87
- Working Group Note, Techniques for WCAG 2.0 93
- W3C-Working-Draft *WAI-ARIA* 94
- Zur W3C-Spezifikation des Role-Attributs 94
- Kategorien von Rollen (nach W3C) 94
- BBC GEL, British Broadcasting Corporation 98
- Uni Kassel 98
- Uni Wuppertal 98
- Yale-Web Style Guide 98
- Nachruf auf William W. Tunnicliffe 105
- Siehe unicode.org 107
- Zur Website des W3C 132
- Die OASIS-Organisation Website 133
- Zur Website Docbook.org 133
- RELAX NG Homepage 134
- Zum Namespace MathML 135
- Zum Namespace SVG 135
- *Bert Bos*, siehe http://www.w3.org/Style/CSS-vs-XSL) 136