Webengineering

Eigenes Skript

Studiengang Angewandte Informatik

Duale Hochschule Baden-Württemberg Karlsruhe

von

Andre Meyering

Kurs:	TINF16B2
Dozent:	Prof. Dr. Jürgen Röthig
Semester:	1. und 2. Semester (09.01.2017 - 27.02.2017, 14.05.2017 - 20.06.2017)
letzte Änderung:	*2. Dezember 2017*

Dies ist das eigene Skript für „Webengineering“ bei Herrn Prof. Dr. Jürgen Röthig für das 1. und 2. Semester im Jahr 2017. Es enthält fast alles, was im Unterricht an die Tafel geschrieben oder besprochen wurde. Die LaTeX-Dateien sollten sich im gleichen Share befinden, in dem du diese PDF-Datei gefunden hast.

Bei Fragen, Fehlern oder Ergänzungen – oder sollten die LaTeX-Dateien fehlen – wende dich bitte an dhbw@andremeyering.de. Ich hoffe, diese PDF hilft dir beim Lernen.

Inhaltsverzeichnis

1 Vorwort

Herr Röthig schreibt alles, was für seine Klausuren von Bedeutung ist, an die Tafel. Es ist daher nur zu empfehlen, alles mitzuschreiben, da er kein Skript besitzt und auch keinen Foliensatz. Der Unterricht im Vergleich zu anderen Dozenten unterscheidet sich darin, dass während der Klausur keine Hilfsmittel verwendet werden dürfen. Dafür besteht die Klausur zu 90% nur aus Abfrageaufgaben.

Dieses Skript enthält *alles*, was Herr Röthig 2017 an Wissen voraussetzt. Auf den letzten Seiten dieses Skripts findet sich zusätzlich noch eine Übungsklausur. Die Klausuren unterscheiden sich jedes Jahr nur um einige wenige Aufgaben. Ist man zwei, drei Übungsklausuren durchgegangen, so ist die Klausur einfach zu bestehen.

Zusammen mit meinem Kurs TINF16B2 haben wir dieses Skript ausgedruckt und korrigiert. Inhaltliche Fehler sollten daher (fast) keine mehr enthalten sein.

Das aktuellste Skript findest du unter:
`https://gitea.ameyering.de/DHBW_AI_16/Webengineering_Roethig`

Ich wünsche dir viel Erfolg bei Herrn Röthig in der Vorlesung Webengineering. Solltest du dieses Skript erweitern wollen, so kannst du dich an dhbw@andremeyering.de wenden.

2 Einstieg

Dozent: Prof. Dr. Jürgen Röthig
Raum: C560
Tel.: 0721 9735-883
E-Mail: roethig@dhbw-karlsruhe.de

Umfang: 48h (1. und 2. Semester).
Prüfungsleistung: Klausur (90min; ohne Hilfsmittel); 80% Klausur | 20% Projekt

2.1 Inhaltsübersicht der Vorlesungen

1. Motivation/Historie
2. Anwendungsprotokoll HTTP
3. Dokumentenbeschreibungssprache (HTML)
4. CSS für „Designvorgaben“
5. XML als „abstrakte Datenbeschreibungssprache“
6. XSLT als Transformationssprache für XML-Dateien
7. SVG als Grafikformat

3 Motivation/Historie

Das Wort „Engineering“ zeigt den Bezug auf praktische Anwendbarkeit.

3.1 Historie des Internets

- 1965/67/69: Arpanet-Projekt-Initierung/-Idee/-Prototyp
- Motivator/erste Anwendung: entfernte Bedienung/Benutzung von Rechnerresourcen, `telnet`, `ssh`

3.2 Das Web / WWW

Mit „Web“ ist das World Wide Web (WWW) gemeint.

3.2.1 Was ist das WWW?

- Eine Anwendung, welche das Internet als Kommunikationsplattform nutzt
- Eine Anwendung, welche Informationen „verteilt“ auf verschiedenen Rechnern zur Verfügung stellt; diese Informationen sind untereinander „verknüpfbar“ (über spezielle Verweise) Beispiel für Verweise (aus dem täglichen Leben):
  - Seitenverweise im Text („siehe Seite 102, Zeile 23“, „siehe oben“)
  - Verweise auf Abbildungen, Tabellen
  - Hinweis-Straßenschilder („Frankfurt 130km ↑“)
  - Weiterleiten/Verbinden zum geeigneten Ansprechpartner
  - Fahrplan
  - Literaturquellen/-verweise, weitere Informationsquellen
  - Schulverweis („auf andere Schule!“)
  - Adressbücher/Telefonbücher
- Spezialität der Verweise im „WWW“: automatische Verweisauflösung durch Standardaktion (z. B. der Mausklick).

Der Hyperlink

Der „Hyperlink“ ist ein Verweis mit automatischer Verweisauflösung. Er besteht im „WWW“ aus zwei Teilen: sichtbare Beschreibung und (meist) unsichtbares Verweisziel.

- „Erfinder“ des Hyperlinks: Vannevar Bush, 1945
- siehe „As we may think“, Atlantic Monthly, 1945
- siehe Maschine „memex“ (Memory extension) ⇒ Information auf Mikrofilm mit Verweismöglichkeit inkl. automatischer Verweisauflösung

3.2.2 Zwei Eigenschaften des WWW („Erfolgsfaktoren“)

1. **Lokationsunabhängigkeit**

   Unabhängig von der Lokation der Information und des Suchenden, kann die Information immer „auf die gleiche Weise“ beschafft werden.

   Beim Design des WWW von Tim Berners-Lee geplant.

2. **Medienunabhängigkeit**
   Die Information kann grundsätzlich in verschiedenen Medienformen (Text, Bild, Video, Audio, ...) vorliegen.

   Ursprünglich nicht geplant und nicht im Web enthalten. Belege:

   - HTML: Hyper **Text** Markup Language
   - HTTP: Hyper **Text** Transfer Protocol

   ⇒ in ersten Versionen **nur** Unterstützung von Text vorhanden! (**Klausurrelevant!**)

3.3 ISO-OSI-7-Schichten-Architektur

Die 7 Schichten des ISO-OSI-7-Schichten-Architektur werden in Tabelle 3.1 dargestellt.

Hinweis: Schicht 2 kann unterteilt werden.

- Ab der Transportschicht werden nur noch Ende zu Ende Verbindungen betrachtet.
- *siehe* Wikipedia (`https://de.wikipedia.org/wiki/OSI-Modell`)

3.4 Internet-4-Schichten-Modell

Schicht 5-7 ⇒ Application Layer
Schicht 4 ⇒ Transport Layer
Schicht 3 ⇒ Internet Layer
Schicht 1-2 ⇒ Network Access-Layer

HTTP als Anwendungsprotokoll des 4-Schichten-Internet-Modells benötigt als unterliegendes Protokoll der Transportschicht TCP (UDP geht nicht!).

Nr.	Deutsch	Englisch	Beschreibung
7	Anwendungsschicht	Application Layer	Realisierung des Kommunikationsmodells für die jeweilige Anwendung (Client/Server; Peer-to-Peer)
6	Darstellungsschicht	Presentation Layer	Zeichen-/Anwendungskodierung
5	Kommunikations-steuerungsschicht (besser nicht: Sitzungsschicht)	Session Layer	Sychronisierung der Anwendung; Transaktionssteuerung; (z. B. RPC Protokoll)
4	Transportschicht	Transport Layer	Verbindungen; Vollständigkeit/Fehlerfreiheit/Reihenfolge; Port-/Anwendungsadressierung; (TCP/UDP)
3	Vermittlungsschicht	Network Layer	Wegewahl/Routing; Adressierung (mit IP-Adresse)
2	Sicherungsschicht	Data Link Layer	LLC: Logical Link Control (Integritätsüberprüfung)
			MAC: Medium Access Control (Adressierung mit MAC-Adresse); Regelung des Zugangs auf das „shared-medium“
1	Bitübertragungs-schicht	Physical Layer	Signalkodierung; Stecker/Kabelform

Tabelle 3.1: OSI-7-Schichten-Modell

4 Hyper Text Transfer Protocol (HTTP)

Was ist ein Protokoll?
Ein Protokoll ist verbindliche Vereinbarung (Regeln), wie zwei (oder mehr) Teilnehmer miteinander kommunizieren.

4.1 HTTP 0.9 (1991) - Erste Version

Request: `GET /pdf/zur/ressource`
Response: `<Inhalt der Ressource>`

Problem: Nur Plain-Text und HTML (erkennbar am Anfang `<html>`) sind erlaubt und möglich.
Problem: Fehler sind nicht sicher erkennbar.

4.2 HTTP 1.0 (1992)

Request: `GET /pdf/zur/ressource HTTP/1.0`
Response: beginnt mit einer Statuszeile: `HTTP/1.0 <statuscode> <textuelle Beschreibung des statuscodes; Reason String>`, danach Response-Header.

4.2.1 Status Codes

Hinweis zur Klausur

Die Codes müssen nicht für die Klausur auswendig gerlernt werden. Es reichen die Gruppen.

Für alle Codes siehe auch: `https://de.wikipedia.org/wiki/HTTP-Statuscode`.

- **`1xx`**: Informational Codes
  ⇒ Zur Anzeige, dass weitere Daten folgen (Verhinderung von Timeouts).
- **`2xx`**: Successful Client Requests
  - `200` ⇒ Ok
  - `204` ⇒ Ressource leer
  - `206` ⇒ Partial Content
- **`3xx`**: Client Requests Redirected
  - `301` ⇒ Moved Permanently
    ⇒ z. B. bei Domainnamensänderung, Neuordnung der Website
  - `302` ⇒ Moved Temporarily
    ⇒ z. B. bei Überlastung der Website, Wartungsmodus, temporärer Ausfall
- **`4xx`**: Client Request Errors
  - `400` ⇒ Bad Request ⇒ Der Request war nicht korrekt.
  - `401` ⇒ Authorization Required ⇒ Eine Autorisierung ist erforderlich.

  - `403` ⇒ Forbidden ⇒ Der Webserver hat z. B. keinen Zugriff (auf ein Verzeichnis). Oder man greift auf ein Verzeichnis zu, ohne eine Datei anzugeben (und es gibt keine index.html und das Verzeichnislisting ist deaktiviert).
    „Wenn ich eine solche Meldung erhalte, erhält sie jeder!“
  - `404` ⇒ Not Found: Der Pfad wurde nicht gefunden
- **`5xx`**: Server Error
  - `500` ⇒ Internal Server Error
  - `505` ⇒ HTTP Version Not Supported ⇒ Der Server wird mit einer Version angesprochen, die er nicht unterstützt. Heute eher selten.

4.2.2 HTTP Header

Die Statuszeile wird gefolgt von einer oder mehreren HTTP-Header-Zeilen im Format: `<HTTP-Headername>: <Wert>`, z. B.

`Content-Type: text/html` Der Wert ist ein Multimedia Internet Mail Extension Type (MIME-Type)

- besteht immer aus zwei Teilen: tlmt/slmt

  Top-Level-Media-Type (tlmt) ⇒ grundsätzliche Medienform (text, image, video, audio, application, [multipart])

  Sub-Level-Media-Type (slmt) ⇒ Angabe der konkreten Codierung (Anwendungscodierung), z. B. `text/html`, `text/plain`, `image/jpeg`, `video/mp4`, ...
- **Achtung**: Alle MIME-Types sind standardisierte Werte! Diese werden standardisiert von der „IANA“ (Teil der „IETF“)
  Ausnahme: slmt beginnt mit „`x-`“, z. B. `application/x-meinespezielleirgendwas`
- Wird immer mitgegeben, außer man hat *keinen* Content (z. B. bei `204` [siehe S. 6])!

`Content-Length: <Größe in Bytes>` Größe der Antwort in Bytes

- Wichtig für Anzeige von Download-Fortschrittsbalken
- Wichtig für Caches in Browsern und Proxies

`Last-Modified: <US Datum>` z. B. `Jan 23rd, 2017 ...EST`
`Expires: <US Datum>` ebenfalls wichtig für Caches und Proxies
`Content-Language:<Sprache des Inhalts>` Menschliche Sprache des Inhalts. Kann auch bei Bildern angegeben werden, wenn dort z. B. Text drauf steht.

- 2-Letter-ISO-Language-Code ggf. mit Sprachvariante, z. B. `de/DE-AT`, `fr`

Eine Leerzeile beendet die HTTP-Header-Zeilen, danach folgt das Dokument. Die Leerzeile enthält **keine** Zeichen, auch keine Leerzeichen!

4.2.3 Weitere Request-Typen

Mit HTTP 1.0 wurden weitere Request Typen eingeführt.

HEAD-Request

`HEAD /pfad/zur/ressource HTTP/1.0` ⇒ liefert nur die Statuszeile und HTTP-Headerzeilen.

POST-Request

`POST /pfad/zur/ressource HTTP/1.0`
⇒ gefolgt von Parametern (signalisiert wie bei einer Response über HTTP-Headerzeilen).

4.2.4 Was funktioniert nicht?

HTTP 1.0 funktioniert gut, aber was funktioniert nicht bzw. was geht besser?

Ist es möglich, mehrere FQDN auf eine IP-Adrese zu leiten?
⇒ FQDN1 und FQDN2 zeigen beide auf `1.2.3.4`; per DNS möglich und erlaubt!

`GET /pfad/zur/ressource HTTP/1.0` ⇒ Problem: Der FQDN steht nicht im Request bei HTTP 1.0 und wird nicht einmal für den TCP-Verbindungsaufbau verwendet
⇒ *Der Webserver weiß nicht, welche Präsenz angefragt ist!*

Achtung

HTTP 1.0 kann nicht erkennen, von welchem FQDN der Request kommt! Siehe Abschnitt 4.5 für Möglichkeiten, dennoch verschiedene Webpräsenzen zu unterscheiden.

4.3 HTTP 1.1 (seit 1997)

Pflicht-Header im Request:

`Host:` $\underbrace{\texttt{www.dhbw-karlsruhe.de}}_{\text{angefragter FQDN}}$

Jetzt gibt es persistente Verbindungen, d. h. eine TCP-Verbindung wird nacheinander für mehrere Request/Response-Paare zwischen Client und Server genutzt.
Es muss somit nicht jeweils eine neue Verbindung aufgebaut werden. Bereits vorher bei HTTP 1.0 gab es ähnliches, nämlich den „Connection: Keepalive“-Header. Dieser ist jedoch nicht standardkonform und nur teilweise implementiert!

Dieses Abarbeiten von Requests und Responses benötigt Zeit und verzögert das Laden eines Dokuments. Mit HTTP2 gibt es andere Möglichkeiten.

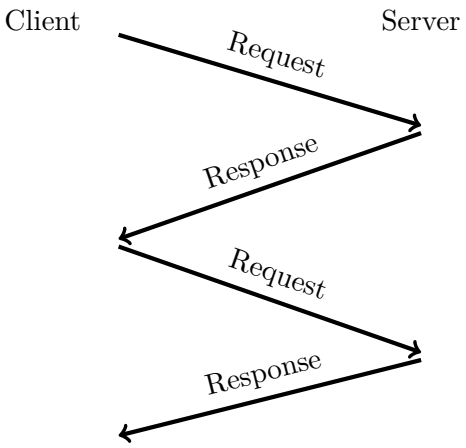


Abbildung 4.1: Sequentielle Abarbeitung von Requests und Responses

4.4 HTTP 2 (seit ca. 2015)

Siehe Wikipedia: `https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol#HTTP.2F2`

- Multiplexen mehrerer HTTP-Requests über eine TCP-Verbindung per „Stream-ID“ ist möglich
- Komprimierung (auch) der HTTP-Header möglich.
- „Server-Push“[1], d. h. ein Webserver kann nach dem Request auf eine Ressource weitere damit in Zusammenhang stehende Ressourcen mitschicken, ohne dass diese explizit vom Client angefordert wurden.

[1] `https://en.wikipedia.org/wiki/HTTP/2_Server_Push`

4.5 Bereitstellung/Unterscheidung von mehreren Web-Präsenzen auf einem (großen) Webserver

1. über den Pfad/das Verzeichnis
   - `http://www.uni.de/~<nachname>/` ⇒ Mit Tilde: Unterverzeichnis im Home-Verzeichnis. Nicht für professionelle Web-Angebote geeignet.
2. über verschiedene Portnummern ⇒ auch nicht professionell
   - `http://www.uni.de:4711/`
   - `http://www.uni.de:0815/`
3. per Fully Qualified Domain Name (FQDN) „Rechner“ oder „Domain“-Name, welche per DNS auf eine IP-Adresse umgesetzt werden (A-Record oder Alias-Eintrag),
   z. B. `http://dhbw-karlsruhe.de`, `http://www.uni-karlsruhe.de` ⇒ Dies gilt als professionell
   a) mehrere IP-Adressen (je eine eigene IP-Adresse pro FQDN)
      i. mehrere Netzwerkkarten
         - teuer (Hardware)
         - Anzahl Steckplätze für Netzwerkkarten
      ii. mehrere IP Adressen für eine Netzwerkkarte (⇒ bei allen halbwegs modernen Betriebssystemen möglich, bei unixoiden Betriebssystemen „schon immer“, bei Windows seit WindowsNT [nicht bei Windows 3.1])
         - Reichen die IP-Adressen aus?
         - Wie viele IP-Adressen gibt es?
         - Siehe Abschnitt 5.1 auf Seite 11
   b) mehere FQDN für eine IP-Adresse:
      Unterscheidung durch `Host`-Header im Request (ab HTTP Version 1.1).

5 IP Adresse

5.1 IPv4

Bei einer IPv4 `a.b.c.d` mit $a,b,c,d \in 0,...,255$ (oder anders gesagt: IPv4 Adresse = 4-Byte-Wert bzw. 32 Bit) gibt es theoretisch $2^{32} = 2^2 \cdot 2^{30} = 2^2 \cdot (2^{10})^3 \approx 2^2 \cdot 1000^3 \approx 4 Mrd.$ verschiedene IPv4-Adressen.

- Tatsächlich gibt es jedoch weniger als 4 Mrd. IP-Adressen, welche für Webserver potentiell nutzbar wären. Nicht benutzt werden können:
  - `127.0.0.0/8` ⇒ Loopback (der eigene Rechner)
  - `192.168.0.0/16` ⇒ Private Netzwerkbereiche
  - `172.16.0.0/12` ⇒ Private Netzwerkbereiche
  - `10.0.0.0/8` ⇒ Private Netzwerkbereiche
  - `224.0.0.0/4` ⇒ Multicast
  - `240.0.0.0/4` ⇒ Future/Experimental Use
- Außerdem sind in jedem Teilnetz die erste und die letzte Adresse als Netzwerkadresse bzw. Broadcast-Adresse reserviert.
  ⇒ insgesamt stehen deshalb weniger als 4 Mrd. Adressen zur Verfügung.
- Außerdem: IP-Adressen sind äußerst ungerecht auf der Welt verteilt:
  - Nordamerika: sehr viele
  - Europa, Australien: ordentliche Menge
  - Rest (Afrika, Asien, Südamerika): sehr wenig
- Außerdem: immer mehr Geräte der Haushalts- und Unterhaltungselektronik sind „internetfähig“ (haben einen Webserver).

Tipp

Grobe Berechnung von 2er Potenzen

Man „splittet“ die Potenz auf, sodass man $(2^{10})^x$ erhält. 2^{10} entspricht ~1000
$2^{32} = 2^2 \cdot 2^{30} = 2^2 \cdot (2^{10})^3 \approx 2^2 \cdot 1000^3 \approx 4 Mrd.$

6 HTML

6.1 Dokumentenbeschreibungssprachen für Textdokumente

```
1            aussehenorientiert
2    ---------------------------------------------------------->
3    |                                              (SGML)
4    |  PostScript                                  LaTeX
5  Z |          WordDOC
6  e |        RTF                                 HTML
7  i |  PDF                                         (XML)
8  t |                   ODT
9    |                   WordDOCX                 Markdown
10   |
```

designorientiert (PDF, ...)	**strukturorientiert** (HTML, ...)
Textpassagen werden mit Aussehen attributiert	Textpassagen mit Bedeutung oder Funktion attributiert
meist binärkodiert	meist textcodiert
⊖ Spezialprogramme notwendig	⊕ mit jedem Texteditor editierbar
⊕ WYSIWYG	⊖ großes Abstraktionsvermögen nötig, um sich das endgültige Dokument vorstellen zu können
⊕ kurze Einarbeitungszeit	⊖ lange Einarbeitungszeit
⊖ Inkonsistenzen bei vergleichbaren Textpassagen möglich	⊕ konsistente Formatierung des gesamten Dokuments
⊖ nur für genau ein Ausgabemedium gedacht	⊕ optional: Anpassung an verschiedene Ausgabemedien möglich

Tabelle 6.1: Dokumentenbeschreibungssprachen

6.2 Hyper Text Markup Language (HTML)

HTML ist eine strukturorientierte Dokumentenbeschreibungssprache

- basiert auf SGML (welches 197X entstanden ist, siehe Abschnitt 6.3)
- textcodiert, basierend auf ASCII (7-bit, 128 Zeichen)
- der meiste Text existiert „nahezu unverändert“ bei der Ausgabe
  Ausnahmen:
  - Entitäten/Entities ⇒ Nicht im ASCII-Zeichensatz enthaltene Zeichen.
    z. B. `&uuml;` ≙ ü („Umlaut“) , `&Ouml;` ≙ Ö, `&szlig;` ≙ ß („Ligatur“ ⇒ Zusammenfassen von mehreren zu einem druckbaren Zeichen)

- spezielle Zeichen:

<	`<`	„less than“
>	`>`	„greater than“
	` `	„non-breaking space“
&	`&`	„Ampersand“
"	`&quot;`	„Quotation“

  ⇒ müssen mit Entitäten umschrieben werden, da sie als einfaches Zeichen eine Sonderfunktion in der Sprache HTML besitzen
  ⇒ bisher alles „named entities“, `&#nnn` ⇒ „numbered entity“ für Zeichennummer `nnn` aus Unicode
- Tags `<tagname [attribut1="Wert1"]>` (nicht angezeigt) `Text` `</tagname>` (nicht angezeigt)
- alle „Metadaten“ im `<head>` (z. B. `<meta>`, `<style>`, `<script>`, `<title>`, . . .)
- Kommentare `<!-- Ich bin ein Kommentar -->`
- Alles andere erscheint als Text in der Ausgabe (im Browser)

6.3 Historie von HTML

~1970 SGML

1992 (Ur-)HTML

- Hyperlinks waren schon möglich (ergibt sich aus HTML: **Hyper** Text. . .)
- Überschriften, Textabschnitte, Listen

1995 HTML 2.0

- Multimediafähigkeit (Bilder [keine Videos, dafür gab es Plugins wie Flash oder Quicktime])
- Formulare (senden geht nun mit `POST`)
- Textgestaltung (ist eigentlich kein Teil der Struktur, sondern Gestaltung!)

1996 HTML 3.2 - Zeit des Browserkrieges (Microsoft vs Mozilla/Netscape)
W3C hat Version 3.0 nicht verabschieden wollen, da die Browserhersteller uneinig waren.

- Tabellen
- (Java-)Applets
- Textfluss um Grafiken (vorher waren Grafiken „inline“ und der Zeilenabstand wurde vergrößert, sodass die Grafik in die Zeile passte. Alternativ konnte man noch Zeilenumbrüche verwenden)

1997 HTML 4.0

- Stylesheets (CSS) können eingebunden werden
- „Unterstrichenes wurde rückgängig gemacht“ (siehe oben)
- Frames
- neue Tags

2000 XHTML 1.0 - keine neuen Inhalte im Gegensatz zu HTML 4.0

- XML-basierte Sprachvariante von HTML 4.0 (strengere Grammatik)

2001 XHTML 1.1

- Modularisierung der Sprache XHTML 1.0 (wichtig für kleine, nicht leistungsfähige Geräte (Handys), welche immer noch standardkonform waren, auch wenn bestimmte Module fehlten)

2014 HTML5

- keine XML-Basierung, aber HTML5 *kann* XML-basiert notiert werden
- Integration von weiteren Medienformen: Audio und Video (nativ, ohne Plugin im Browser)
- Einbindung von SVG und MathML im Quelltext
- Standardisierung des Document Object Model (DOM) und der JavaScript-API zu demselben
- weitere Funktionalitäten als eigenständige Module
  - Canvas (Zeichenfläche, auf die per JavaScript gezeichnet werden kann
  - Geolocation-API (per JavaScript aktuellen Ort erfahren)
  - WebStorage/LocalStorage (Speicherung von Daten im Browser)
  - WebSockets (Kommunikation zwischen Website und Webserver nach Laden der Seite)
- weitere semantische Tags:
  - Navigation (`<nav>`), Menü (`<menu>`), Kopf- und Fußzeilen (`<header>`, `<footer>`), Unterteilung einer „Seite“ in mehrere Abschnitte (`<section>`) oder Artikel (`<article>`)
- viele syntaktische/grammatikalische Vereinfachungen sind nun erlaubt:
  - kein End-Tag (bei leeren Tags, falls der nachfolgende Tag den vorherigen Tag implizit schließt)
  - kein Start-Tag, falls der Tag bedeutet, dass vorher ein anderes Tag geöffnet werden muss (z. B. `<title>`, bevor der `<head>` geöffnet wurde)
  - Anführungszeichen bei Attributwerten können weggelassen werden (sofern keine Trennzeichen enthalten sind, z. B. `<p id=test>`)
  - Attributwerte (bei Attributen mit Wahrheits- oder booleschen Werten) können weggelassen werden (z. B. `<input type="text" disabled["disabled"]>`)
- Multimedia-Tags wie `<video>` und `<audio>`
- HTML5 ist ein „living Standard“
  ⇒ Standard verändert sich ständig, bzw. wird fortlaufend erweitert!

Hinweis zur Klausur

In der Klausur sollten keine Vereinfachungen verwendet werden! Zu hoher Wahrscheinlichkeit wird sowieso XHTML gefordert!

Tipp

Nicht alle neuen HTML5-Module/Tags/Funktionen werden von allen Browsern unterstützt. Was in welchen Browsern verwendet werden kann, kann auf `http://caniuse.com/` nachgelesen werden.

6.4 Der (X)HTML-Doctype

```
1  <?xml version="1.0" [encoding="UTF-8"] ?>
2  <!DOCTYPE html PUBLIC "Public-ID" "System-ID">
3  <!-- Für HTML5 nur:
4    <!DOCTYPE html>   | html: Name des Root-Tags
5  -->
6  <html xmlns="http://www.w3.org/1999/xhtml">
7    <head>
8      <title>Titel des Dokuments</title>
9    </head>
10   <body>
11     <!-- Eigentlicher Inhalt des Dokuments -->
12   </body>
13 </html>
```

Listing 6.1: Beispiel eines XHTML 1.1 Dokuments

`PUBLIC` Es handelt sich um einen öffentlichen Standard.
Alternativ gibt es noch `SYSTEM` oder `PRIVATE` für selbst definierte Grammatiken mit (oder ohne) Angabe der DTD.

„Public-ID“ Textuelle, exakt festgelegte „Beschreibung des Standards“, z. B. `-//W3C//DTD XHTML 1.1//EN` bei XHTML 1.1

„System-ID“ URL (Adresse), welche auf die zugrunde liegende Grammatikdatei (in Form einer DTD) zeigt.
z. B. `http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd` bei XHTML 1.1

xmlns Der XML-Namensraum

Hinweis

Die „System-ID“ muss für die Klausur nicht auswendig gelernt werden!

Für eine Liste verfügbarer DOCTYPEs siehe
⇒ `https://www.w3.org/QA/2002/04/valid-dtd-list.html`

6.5 Wichtige (X)HTML Tags im <body>

<h1> bis <h6> „heading“/Überschriften in verschiedenen Hierarchiestufen, welche aber nicht verpflichtend einzuhalten sind. Dabei ist <h1> die oberste Ebene.

<p> „Paragraph“/Textabsatz

Listen

- „Unordered List“ – Liste mit allgemeinem Aufzählungszeichen, z. B. Spiegelstrich oder Aufzählungspunkt.
- „Ordered List“ – Liste mit Nummerierung als Aufzählungszeichen, z. B. Zahlen, Buchstaben oder römische Zahlen.
- „List Item“ – Aufzählungspunkte innerhalb von <ul>...

⇒ Listen können geschachtelt werden, indem in ein <li> ein weiteres / geschrieben wird.

<dl> „Definition List“ – Definitionsliste

- <dt> „Definition Term“ – Begriff
- <dd> „Definition Description“. Erklärung des Begriffs.

⇒ zwei verschiedene Arten von Listenelementen

⇒ Definitionslisten dürfen ebenfalls geschachtelt werden, allerdings nur im <dd>

⇒ Die paarweise Verwendung von <dt> und <dd> ist üblich, aber nicht gefordert.

Hyperlink

- <a> „anchor“ – Textanker (irreführende Bezeichnung für einen Verweis)
- enthält Beschreibung und technisches Verweisziel.
  - Beschreibung: Inhalt des Tags (<a>Inhalt hier)
  - Verweisziel: Attribute href="..." in Form einer URI (siehe Kapitel 7)
- Beispiel: Homepage der DHBW
- Es können auch relative URLs angegeben werden, die jedoch mithilfe der vollständigen URL des enthaltenden Dokuments ebenfalls zu einer vollständigen (absoluten) URL gemacht werden (siehe Unterabschnitt 7.2.1 auf Seite 30)

Klausur – Hinweis

In der Klausur kann eine Fehlersuche in einem XHTML Dokument vorkommen.

Beispiel:
Es kommt eine unsortierte Liste, worauf ein <ol> kommt ⇒ Ein semantischer, nicht aber grammatikalischer Fehler!

6.6 Bilder <img>

```
<img src="URL" alt="alternative Beschreibung" [width="" height=""] />
```

- Attribut `src` („source“/Quelle) für URL zur Bilddatei.
- Attribut `alt` („alterntive text“) für Alternativtext zur Anzeige, falls das Bild selbst nicht angezeigt wird (technische Gründe und Barrierefreiheit)
- für mehr siehe Abschnitt 6.6 auf Seite 17

`width` und `height` enthalten (meist) die „natürliche Größe“ des Bildes, um dem Browser das entsprechende Platzfreihalten zu ermöglichen, bevor die Bilddatei selbst geladen ist. Eine Skalierung ist möglich, aber:

- die Datenmenge muss dennoch komplett geladen werden
- erhöhter Speicherplatzverbrauch im Web-Browser (Arbeitsspeicher! Es ist nicht der Cache gemeint!)
- die Qualität der Skalierung im Browser ist (meist) schlechter als bei Bildbearbeitungsprogrammen

6.7 Tabellen

Tabellen sind eine zweidimensionale Struktur mit Zeilen und Spalten. Diese müssen für HTML notwendigerweise serialisiert werden, da HTML ein Textformat ist!

Im HTML werden erst die Zeilen und dann die Felder (Spaltenelemente) innerhalb der Zeilen geschrieben.

Für die Tabelle gibt es folgende wichtige HTML 3.2 Tags:

`<tr>` „table row“ – enthält eine Tabellenzeile

`<td>` „table data“ – enthält je ein Tabellenfeld

`<th>` „table heading“

```
1    <table>
2        <tr>
3            <td>a1</td>  <td>a2</td>
4        </tr>
5        <tr>
6            <td>b1</td>  <td>b2</td>  <td>b3</td>
7        </tr>
8    </table>
```

Listing 6.2: HTML 3.2 Tabellen

a1	a2	
b1	b2	b3 |

Tabelle 6.2: Tabelle mit unterschiedlicher Anzahl an Feldern pro Zeile

Die Anzahl der Spalten (und Zeilen) wird nicht vorab definiert. Eine unterschiedliche Anzahl an Feldern in den Zeilen ist außerdem erlaubt! Dies wird in Tabelle 6.2 gezeigt.

Statt `<td>` kann auch überall `<th>` verwendet werden, um eine Art Metadaten (Beschreibung der Tabelleninhalte) anzeigen zu lassen.

Seit HTML 4 kann `<table>` zusätzlich gegliedert werden in:

- `<thead>` Tabellenkopf (optional, max. einmal)
- `<tfoot>` Tabellenfuß (optional, max. einmal)
- `<tbody>` Tabellenkörper (kann mehrfach vorkommen)

Diese Reihenfolge muss eingehalten werden! Diese drei Elemente enthalten dann nur `<tr>`-Tags, in denen wiederum nur `<td>`- und `<th>`-Tags vorkommen dürfen.

Optionale Attribute in `<td>` und `<th>`:

`colspan` für nebeneinanderliegende Felder, welche das `<td>`/`<th>` überdecken soll

`rowspan` für Angabe der übereinanderliegenden Felder, welche das `<td>`/`<th>` überdecken soll

Die überdeckten `<td>`/`<th>` müssen im Quelltext gestrichen/ausgelassen werden. Ein Beispiel für die Ausgabe zeigt Abbildung 6.1 mit dem dazu gehörigen HTML-Quelltext in Listing 6.3 auf Seite 19.

<table>
<tr><th>Geschäftsjahr</th><th>Umsatz</th><th>Gewinn/Verlust</th></tr>
<tr><td>2013</td><td>5,8</td><td rowspan="2">62,8</td></tr>
<tr><td>2014</td><td rowspan="2">60,0</td></tr>
<tr><td>2015</td><td>50,87</td></tr>
<tr><td colspan="3">Zusammengefasste Spalten</td></tr>
<tr><td colspan="2" rowspan="2">Spalten + Zeilen</td><td>44,87</td></tr>
<tr><td>70,78</td></tr>
</table>

Abbildung 6.1: HTML-Tabelle mit zusammengefügten Feldern

```
<table border="1">
    <tr>
        <th>Gesch&auml;ftsjahr</th>
        <th>Umsatz</th>
        <th>Gewinn/Verlust</th>
    </tr>
    <tr>
        <td>2013</td>
        <td>5,8</td>
        <td rowspan="2">62,8</td>
    </tr>
    <tr>
        <td>2014</td>
        <td rowspan="2">60,0</td>
        <!-- weggelassen -->
    </tr>
    <tr>
        <td>2015</td>
        <!-- weggelassen -->
        <td>50,87</td>
    </tr>
    <tr>
        <td colspan="3">Zusammengefasste Spalten</td>
        <!-- weggelassen -->
        <!-- weggelassen -->
    </tr>
    <tr>
        <td colspan="2" rowspan="2">Spalten + Zeilen</td>
        <!-- weggelassen -->
        <td>44,87</td>
    </tr>
    <tr>
        <!-- weggelassen -->
        <!-- weggelassen -->
        <td colspan="3">70,78</td>
    </tr>
</table>
```

Listing 6.3: Tabelle – `colspan` und `rowspan`

6.8 Semantische und strukturelle Tags (meist mit HTML5 eingeführt)

Abschnitte	
`<body>`	gesamter Inhalt
`<section>`	Unterteilung des Dokuments ins (unabhängige) Sektionen
`<nav>`	„navigation“ – Abschnitt für Navigation (Menü, Suchfeld, . . .)
`<article>`	inhaltlich selbstständiger Teil des Dokuments
`<aside>`	Randbemerkung (Seitennotiz), ohne die das restliche Dokument verständlich bleibt
`<h1>` . . . `<h6>`	Überschriften
`<header>`	für typische Kopfzeilen/-bereiche
`<footer>`	für typische Fußzeilen/-bereiche
`<address>`	für Urheberinformationen (eine Art Metainfo, welche explizit angezeigt werden soll ⇒ bereits in Ur-HTML enthalten!)
`<main>`	wichtigster (eigentlicher) Inhalt der Seite (nur ein `<main>` pro Seite erlaubt)
Gruppen	
`<p>`, `<ul>`, . . .	*siehe Abschnitt 6.5 auf Seite 16*
`<figure>`	„für Abbildungen“
`<figcaption>`	Über-/Unterschrift für `<figure>`
`<blockquote>`	Zitat mit eigenem Absatz
`<pre>`	„preformatted“ – für vorformatierten Text (darf keine Tags enthalten, Whitespace bleibt erhalten). Sinnvoll für z. B. Code-Stücke.
`<div>`	„division“ – Gruppe von Text, die ohne spezielle semantische Bedeutung zusammengehalten wird (seit HTML 4 – siehe Abschnitt 6.3)
Fließtext-Elemente	
`<em>`	„emphasize“ – hervorgehobene Bedeutung (seit Ur-HTML)
`<strong>`	wichtiger Text
`<i>`, `<b>`, `<u>`, `<tt>`, . . .	*italic*, **bold**, underline, `teletype`, . . . – für unterschiedliche Arten der Hervorhebung
`<cite>`	für (kurze) Zitate
`<code>`	für Code-Bruchteile
`<span>`	Textteil ohne semantische Bedeutung (seit HTML 4)
	⇒ größtenteils aus früheren HTML-Version revitalisiert

6.9 Audio- und Videoeinbindung

Zur Einbindung von Audio- und Videodateien werden `<audio>` und `<video>` verwendet.

```
1    <video [width="90px" height="160px" controls="controls" ...] >
2        <source src="URL1" type="MIME Type 1" />
3        <source src="URL2" type="MIME Type 2" />
4        <!-- weitere Quellen möglich -->
5        <!-- alternativ anzuzeigender Inhalt (Tags sind möglich) -->
6        Der Browser scheint die Wiedergabe nicht zu unterstützen.
7    </video>
8
9    <audio [controls ...]>
10       <source src="URL1" type="MIME Type 1">
11       <source src="URL2" type="MIME Type 2">
12       Der Browser scheint die Wiedergabe nicht zu unterstützen.
13   </audio>
```

Listing 6.4: Video- und Audioeinbindung

6.10 Aufgaben

Aufgaben von `http://dh.jroethig.de/`

Grundstruktur einer HTML-Datei

Erstellen Sie eine einfache HTML-Datei, welche nur die unbedingt notwendigen Informationen (HTML-Tags) enthält. Rufen Sie diese Datei mit den verschiedenen auf Ihrem System verfügbaren Web-Browsern auf. Was passiert, wenn Sie notwendige Teile der HTML-Struktur weglassen? Warum?

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Grundstruktur einer HTML-Datei</title>
5   </head>
6   <body></body>
7 </html>
```

Listing 6.5: HTML-Datei – Grundstruktur

- Wenn `<head>` fehlt, gibt es folgende Fehlermeldung: `Element head is missing a required instance of child element title.`

Text

Schreiben Sie nun Text in den Body Ihrer HTML-Datei. Trennen Sie Wörter und Sätze testweise durch unterschiedliche Anzahlen von Leerzeichen und Zeilenumbrüchen. Welchen Einfluss hat dies auf die Darstellung im Web-Browser?

```
<!DOCTYPE html>
<html>
  <head>
    <title>Text</title>
  </head>
  <body>
    <p>
      Hier steht Text
        mit unterschiedlichen     Einsch&uuml;ben
      Test  Text
    </p>
  </body>
</html>
```

Listing 6.6: HTML-Datei – Text

Bei mehreren Leerzeichen, Tabs oder Zeilenumbrüchen wird nur ein Leerzeichen zwischen den getrennten Worten eingefügt! Zeilenumbrüche sorgen *nicht* für Zeilenumbrüche im Browser.

Gliederung

Erweitern Sie Ihre HTML-Datei um mehrere Textabschnitte. Verwenden Sie zur Gliederung des Textes zunächst nur die Tags für Überschriften und Paragraphen. Setzen Sie demgegenüber dann auch die `<span>`- und `<div>`-Tags ein. Wie unterscheidet sich deren Einsatz vom vorangegangenen?

```
<!DOCTYPE html>
<html>
  <head>
    <title>Gliederung</title>
  </head>
  <body>
    <h1>Gliederung</h1>
    <p>Hier steht Text.</p>
    <h2>Unterabschnitt</h2>
    <p>Hier steht noch mehr Text.</p>
    <hr>
    <span>Hier steht Text.</span>
    <div>Noch mehr Text</div>
  </body>
</html>
```

Listing 6.7: HTML-Datei – Gliederung

- Paragraphen und Überschriften erzeugen Abstände (Standard-Design im Browser) ...
- ... `<div>` und `<span>` nicht
- `<span>` erzeugt keine Umbrüche.

Verweise

Bauen Sie sich eine komplette Website auf, indem Sie weitere HTML-Seiten erstellen. Verlinken Sie diese mittels absoluter und relativer URLs. Verändern Sie die Lokation des Dateibaumes (durch Verschieben der Dateien in ein Unterverzeichnis) und prüfen Sie die Funktionsfähigkeit der Links. Bauen Sie in Ihre Seite Verweise ein, mit welchen Sie an bestimmte Stellen im aktuellen Dokument springen können.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Verweise</title>
5    </head>
6    <body>
7      <h1 id="verweise">Verweise</h1>
8      <p>Hier steht Text. Ich verlinke nach <a href="index.html">index.html</a></p>
9
10     <h2 id="unterabschnitt-1">Unterabschnitt</h2>
11     <p>Hier steht noch mehr <span>Text</span>.</p>
12     <div>Noch mehr Text. Verlinkung zur <a ↙
       ↳ href="03_Gliederung.html">Gliederung</a> und zum <a ↙
       ↳ href="#unterabschnitt-2">Unterabschnitt 2</a>.</div>
13
14     <p>[...]</p>
15
16     <h2 id="unterabschnitt-2">Unterabschnitt 2</h2>
17     <p>[...]</p>
18
19   </body>
20 </html>
```

Listing 6.8: HTML-Datei – Verweise

Bilder

Binden Sie nun außerdem graphische Darstellungen ein. Was passiert bei der Darstellung im Web-Browser, wenn die eingebundenen Bilder nicht geladen werden können (z. B. weil sie nicht existieren oder umbenannt wurden) oder wenn Sie das automatische Laden von Bildern abschalten?

```
<!DOCTYPE html>
<html>
  <head>
    <title>Bilder</title>
  </head>
  <body>
    <img src="w3c.png" alt="W3C Logo" width="110" height="61">
  </body>
</html>
```

Listing 6.9: HTML-Datei – Bilder

- Es wird der Alternativ-Text angezeigt
- Eine Box in der Größe von `width` und `height` erscheint

Listen

Erstellen Sie eine ungeordnete sowie eine nummerierte Liste. Bauen Sie nun zusätzlich eine weitere nummerierte Liste in die erste ungeordnete Liste ein. Wie unterscheidet sich diese von der zuerst erstellten nummerierten Liste? Experimentieren Sie mit weiteren vielfach geschachtelten Listen!

Ein weiteres Beispiel für Listen (neben unnummerierten und nummerierten) sind Definitionslisten. Erstellen Sie eine solche! Was passiert, wenn Sie die entsprechenden Tags für die Listenelemente nicht paarweise einsetzen?

```
<!DOCTYPE html>
<html>
  <head>
    <title>Listen</title>
  </head>
  <body>
    <p>Ungeordnete Liste:</p>
    <ul>
      <li>Listeneintrag Nr. 1</li>
      <li>Listeneintrag Nr. 2</li>
    </ul>

    <p>Geordnete Liste:</p>
    <ol>
      <li>Listeneintrag Nr. 1</li>
```

```
16      <li>Listeneintrag Nr. 2</li>
17    </ol>
18
19    <p>Geschachtelte Liste:</p>
20    <ul>
21      <li>Listeneintrag Nr. 1</li>
22      <li>
23        Listeneintrag Nr. 2
24        <ol>
25          <li>Listeneintrag Nr. 1</li>
26          <li>
27            Listeneintrag Nr. 2
28            <ul>
29              <li>Listeneintrag Nr. 1</li>
30            </ul>
31          </li>
32          <li>Listeneintrag Nr. 3</li>
33        </ol>
34      </li>
35      <li>Listeneintrag Nr. 3</li>
36    </ul>
37
38    <dl>
39      <dt>Begriff 1</dt>
40      <dd>Erkl&auml;rung 1</dd>
41      <dd>Erkl&auml;rung 2</dd>
42      <dt>Begriff 3</dt>
43    </dl>
44  </body>
45 </html>
```

Listing 6.10: HTML-Datei – Listen

- Die nummerierte Liste in der unnummerierten Liste wird eingerückt.
- Eine nummerierte Liste in einer nummerierten Liste hat andere Aufzählungszeichen.
- Werden mehrere `<dd>` hinter ein `<dt>` geschrieben, so sind dies alternative Definitionen. Es können mehrere Definitionen für einen Begriff existieren.
- Ein `<dt>` ohne `<dd>` ist ein Begriff ohne Erklärung / ohne Definition

Tabellen

Erstellen Sie eine Tabelle mit vier Zeilen und drei Spalten. Wie wird die Tabelle dargestellt, wenn einzelne Zellen keinen Inhalt besitzen? Fassen Sie zwei einzelne (nacheinander sowohl übereinander als auch nebeneinander liegende) Zellen zusammen. Experimentieren Sie mit weiteren zusammengefassten Tabellenzellen. Was passiert, wenn Sie die gleiche Tabellenzelle sowohl von oben als auch von links mit anderen Tabellenzellen zusammenfassen?

```
1 <!DOCTYPE html>
```

```
2  <html>
3    <head>
4      <title>Tabellen</title>
5    </head>
6    <body>
7      <table border="1">
8        <tr>
9          <td rowspan="2">1</td>
10         <td>2</td>
11         <td>3</td>
12       </tr>
13       <tr>
14         <td colspan="2">2</td>
15       </tr>
16       <tr>
17         <td></td>
18         <td></td>
19         <td>3</td>
20       </tr>
21       <tr>
22         <td>1</td>
23         <td>2</td>
24         <td>3</td>
25       </tr>
26     </table>
27   </body>
28 </html>
```

Listing 6.11: HTML-Datei – Tabellen

- Haben Zellen keinen Inhalt, so werden sie leer dargestellt
- Wird die gleiche Tabellenzelle sowohl von oben als auch von links zusammengefasst, so handelt es sich um undefiniertes Verhalten!

Verschachtelung

Bei Listen haben Sie bereits die Verschachtelung von HTML-Tags verwendet. Probieren Sie dies nun auch bei den anderen bislang verwendeten Tags aus! Wo macht dies Sinn? Wo ist es nutzlos? Und wo ist es (laut Spezifikation) gar verboten?

Semantische Tags

HTML5 wurde vor allem um zahlreiche „semantische Tags“ ergänzt. Bauen Sie diese in Ihren Quelltext ein und schauen Sie sich das Ergebnis im Browser an! Sehen Sie in der Darstellung im Browser, ob und mit welchen semantischen Tags Sie den Quelltext ausgezeichnet haben?

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Semantische Tags</title>
5    </head>
6    <body>
7      <h1>Semantische Tags</h1>
8
9      <nav>
10       <strong>Navigaiton</strong>
11       <ul>
12         <li><a href="./01_Grundstruktur_einer_HTML-Datei.html">Grundstruktur einer ↙
     ↳ HTML Datei</a></li>
13         <li><a href="./02_Text.html">Text</a></li>
14       </ul>
15     </nav>
16
17     <main><p>Hauptinhalt hier</p></main>
18   </body>
19 </html>
```

Listing 6.12: HTML-Datei – semantische Tags

- Nein, man sieht bei den meisten semantischen Tags wie `<main>` oder `<article>` nicht, dass es sich um jene handelt.

Reaktion von Browsern auf Fehler im HTML-Quellcode

Bauen Sie absichtliche Fehler in Ihren HTML-Quellcode ein! Typische Fehler sind beispielsweise fehlende Dokumenttyp-Deklaration, nicht geschlossene Tags, falsch geschachtelte Tags, fehlerhaft aufgebaute Tabellen (sich widersprechende colspan- und rowspan-Attribute) oder falsch geschriebene Tags. Wie reagieren die Ihnen zur Verfügung stehenden Web-Browser auf die Fehler?

- Die meisten (kleinen) Fehler werden ohne Fehlermeldung vom Browser gerendert.
- Fehlende schließende Tags werden geschlossen.

Validation von HTML-Seiten

Zur Validierung der Korrektheit Ihres HTML-Quellcodes dienen HTML-Validatoren. Online im Web gibt es beispielsweise die folgenden:

- W3C Markup Validation Service[1]
- WDG HTML Validator[2]

[1] `http://validator.w3.org/`
[2] `http://www.htmlhelp.com/tools/validator/`

Testen Sie Ihre Seiten (auch und gerade diejenigen mit absichtlichen Fehlern) mit diesen Validatoren auf Korrektheit! Werden alle Fehler (und insbesondere auch nur die wirklichen Fehler) erkannt? In welchen Fällen haben die Validatoren Probleme?

Testen Sie, ob und wie die Validatoren auf Ihre Deklaration der HTML-Version reagieren. Finden Sie beispielsweise Fehler, welche nur bei „XHTML 1.0 strict“ und nicht bei „XHTML 1.0 transitional“ erscheinen! Wie reagieren die Validatoren auf die Deklaration einer „falschen“ (unpassenden) HTML-Version?

Eine Sonderposition nimmt HTMLTidy ein, da es versucht, einige häufig gemachte Fehler automatisch zu beheben. Eine Online-Version gibt es hier:

- W3C Tidy your HTML

Was macht HTMLTidy mit Ihren Seiten, insbesondere denen mit Fehlern? Probieren Sie aus, ob und welche der Fehler behoben werden können!

Untersuchen Sie nun, welche Tags bei HTML5 weggelassen werden können, ohne dass der Validator Fehler meldet! Wie können Sie erkennen, welche Tags der Validator oder der Browser selbst ergänzt?

7 URLs und URIs

Uniform Ressource Locators (URLs) sind eine Teilmenge der Uniform Ressource Identifiers (URIs).

URL Die Adresse einer Ressource (mit allen relevanten Informationen).
Beispiel für URL (und damit auch für URI): http://www.dhbw-karlsruhe.de/

URI Gibt Informationen, wie eine Ressource erreicht werden kann.
Beispiel für URI (aber nicht für eine URL): mailto:xyz@dhbw-karlsruhe.de

7.1 Uniform Ressource Identifier (URI)

Allgemeines Format einer URI: `schema:info`

Beispiele für `schema`:

- `mailto` Info ist die E-Mail Adresse
- `news` Info ist die Newsgroup (und evtl. Artikel-ID)
- `http` Info ist Rest der Adresse

Falls `schema` = Anwendungsprotokoll ⇒ URI ist eine URL.

7.2 Uniform Ressource Locator (URL)

Format einer URL:
`protocol://[uid[:pw]@]host[:port]/[directory/][resource][?parameter][#anchor]`

`protocol` Das Anwendungsprotokoll der Anwendung, z. B. http, https, ftp)

`uid` User-ID, Benutzername (für zugriffsbeschränkte Ressourcen)

`pw` Passwort (in Browsern aus Sicherheitsgründen meist nicht implementiert)

`host` Name (FQDN) oder IP-Adresse des Webservers.

`port` Portnummer (positive ganze Zahl).
Falls nicht angegeben, so ist der Default-Port abhängig vom verwendeten Protokoll (z. B. HTTP auf Port 80, HTTPS auf Port 443)

`/` Der abschließende Schrägstrich (könnte auch als Pfad zur Ressource interpretiert werden).

directory Verzeichnis. Falls nicht angegeben ⇒ Wurzelverzeichnis („root directory“).

resource Name der Ressource im jeweiligen Verzeichnis (meist bestehend aus Name und File-Extension). Falls nichts angegeben wird, so entscheidet der Webserver über die auszuliefernde Ressource (z. B. eine Default-Datei index.html, eine Verzeichnisauflistung oder eine Fehlermeldung).

`parameter` Liste von `GET`-Parametern in der Form `parametername=parameterwert[&paraname2=parawert2[&...]]` (potentiell beliebig viele Parameter, beschränkt durch max. Länge aus URL[1]).

`anchor` Anker: Textstelle, zu welcher der Browser direkt springt.

Achtung

Der Anker (`#xyz`) wird beim Request nicht an den Server mitgeschickt! `anchor` wird lokal im Browser ausgewertet, z. B. durch JavaScript. PHP, welches serverseitig arbeitet, kann dies nicht auswerten.

7.2.1 Relative URLs

Es können auch relative URLs angegeben werden, die jedoch mithilfe der vollständigen URL des enthaltenden Dokuments ebenfalls zu einer vollständigen (absoluten) URL gemacht werden! Es wird zwischen folgenden relativen URLs unterschieden:

Protokoll-relative URL beginnt mit //..., host/... (z. B. HTTPS)

Host-relative (Pfad-absolute) URL: beginnt mit /pfad... (z. B. Verweis auf zentrale Dokumente auf demselben Server wie das Impressum oder die Einstiegsseite)

Pfad-relative URL beginnt mit pfad (ohne Schrägstrich am Anfang; Verweis auf logisch zusammenhängende Ressourcen im selben Verzeichnis oder Unter-/Oberverzeichnis)

Vorteile von relativen URLs:

- (meist) kürzer
- „Umzugsfähigkeit“ der Web-Präsenz...
  - ... auf einen anderen Server (Host-relativ)
  - ... in ein anderes Verzeichnis (Pfad-relativ)

Anwendung von vollständigen (Host-absoluten) URLs

- Verweis auf eine andere Web-Präsenz
- Verweis auf den eigenen Server mit anderem Protokoll (HTTP ⇔ HTTPS ⇔ FTP)

[1] siehe `http://stackoverflow.com/questions/417142/what-is-the-maximum-length-of-a-url-in-different-browsers`

Spezielle Verzeichnisnamen, welche speziell bei relativen URLs wichtig sind:

.. das übergeordnete Verzeichnis (im Falle des Wurzelverzeichnisses wiederum das Wurzelverzeichnis).

. das aktuelle Verzeichnis

Beispiel: URL des Dokuments http://a.b/c/d/e/f.bla

URL in `<a href="...">`	ergibt	
//a.b/z/x	⇒ http://a.b/z/x	
/	⇒ http://a.b/	Einstiegsseite des Servers
./	⇒ http://a.b/c/d/e/	Default-Dokument im aktuellen Verzeichnis
../../../../bla/fasel/./../../da	⇒ http://a.b/bla/da	

7.2.2 Anker

`<a>` für Textanker im Dokument (auf die mit #ankername gesprungen werden kann).

`<a name="ankername">Hierhin kann ich springen</a>` (ursprüngliche Variante) bzw.

`<a id="ankername">Hierhin auch</a>` (neuere/spätere Variante)

Hinweis

`id` ist ein „Universalattribut“ und kann mit (fast) jedem Tag verwendet werden. Der Tag wird dann über #ankername ebenfalls direkt anspringbar!

⇒ Das `<a>`-Tag ist für Verwendung als Textanker heute nicht mehr notwendig! Es reicht die ID z. B. eines `<h1>`-Tags.

8 CSS

Cascading Style Sheet (CSS) ist eine Sprache, welche HTML (und andere Sprachen) mit Aussehen versieht. Es gibt eine Vielzahl an Quellen von CSS-Regeln (Kaskade)

Die Einbindung von CSS in HTML ist möglich durch:

1. Universalattribut `style` für jeden Tag im `<body>` verwendbar
   `<h1 style="color: red;">Rote Überschrift</h1>`
   - ⊖ keine Trennung von Struktur und Aussehen
   - ⊕ gut für Prototyping und für explizit anders aussehende Elemente
2. `<style>`-Tag im `<head>` der HTML-Seite (siehe Listing 8.1 auf Seite 32 als Beispiel)
   - ⊕ Trennung von Aussehen und Struktur klar innerhalb der Datei
   - ⊕ eine Regel hat Auswirkung auf alle selektierten Tags ⇒ einheitliches Aussehen wird ermöglicht
   - ⊖ Design ist separat in jeder Seite vorhanden, bei Designänderungen müssen alle HTML-Seiten angepasst werden
3. `<link>`-Tag im `<head>` der HTML-Seite (siehe Listing 8.2 auf Seite 33 als Beispiel)
   Das Attribut `rel` („relation“) gibt die Beziehung an. Auch möglich: „alternate stylesheet“ für verschiedene auswählbare Aussehensvarianten.
   Das Attribut `media` gibt das Ausgabegerät an. Beispiele:
   - `all` ⇒ für alle Ausgabegeräte
   - `screen` ⇒ für (Computer-)Bildschirme
   - `print` ⇒ für Ausdrucke der Seite
   - `speech` ⇒ für Blinde oder Sehbehinderte. Die Seite wird vorgelesen. Bestimmte Menüpunkte können dafür z. B. ausgeblendet werden.
   - `handheld` ⇒ für sehr kleine Geräte (veraltet)

```
1  <html>
2      <head>
3          <title>Blabla</title>
4          <style type="text/css">
5              h1 { color: red; }
6          </style>
7      </head>
8      <body>
9          <h1>rote Überschrift</h1>
10     </body>
11 </html>
```

Listing 8.1: HTML `<style>`-Tag

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Blabla</title>
5          <link href="URL" type="text/css" rel="stylesheet" title="Name der ↙
   ↳ Auswahlvariante" media="all" />
6      </head>
7      <body>
8          ...
9      </body>
10 </html>
```

Listing 8.2: HTML `<link>`-Tag

8.1 `@import`-Statement

`@import`-Statement zu Beginn eines CSS-Regelblocks (externe CSS-Datei oder innerhalb des Style-Tags). Hiermit wird eine andere CSS-Datei eingebunden.

```
1  @import "URL/zur/datei.css"      [mediatype]; // oder:
2  @import url("URL/zur/datei.css") [mediatype];
```

Listing 8.3: @import-Anweisung in CSS

`mediatype` optional, z. B. `all`, `screen`, `speech`

8.2 CSS Struktur

CSS besteht aus beliebig vielen „Regeln“. Jede Regel besteht aus (mindestens) einer Zeile. Jede Regel besteht aus „Selektor“ und „Deklaration“ im Format:

```
selector { declaration; }
```

Jede Regel kann mehrere Selektoren enthalten, welche durch Kommata voneinander getrennt werden. Jede Regel kann mehrere Deklarationen enthalten, welche in derselben geschweiften Klammer jeweils mit „;“ abgeschlossen werden. Der Selektor wählt die Elemente („Tags“) aus, für welche die Regel gilt. Die Deklaration gibt das „Aussehen“ für die selektierten Elemente vor, z. B. :

```
h1, h2 { color: red; font-size: 16pt; }
```

Ein Selektor ist im einfachsten Fall ein Tagname, z. B. `h1` (oder die Wildcard „`*`“ für alle Tagnamen).

Eine Deklaration besteht aus einer Eigenschaft (property) und einem Wert (value) im Format:

```
property: value;
```

8.2.1 Positionierung: „Box-Model“

In CSS gibt es das sogenannte „Box-Model“. Was es genau bedeutet, wird in Abbildung 8.1[1] gezeigt.

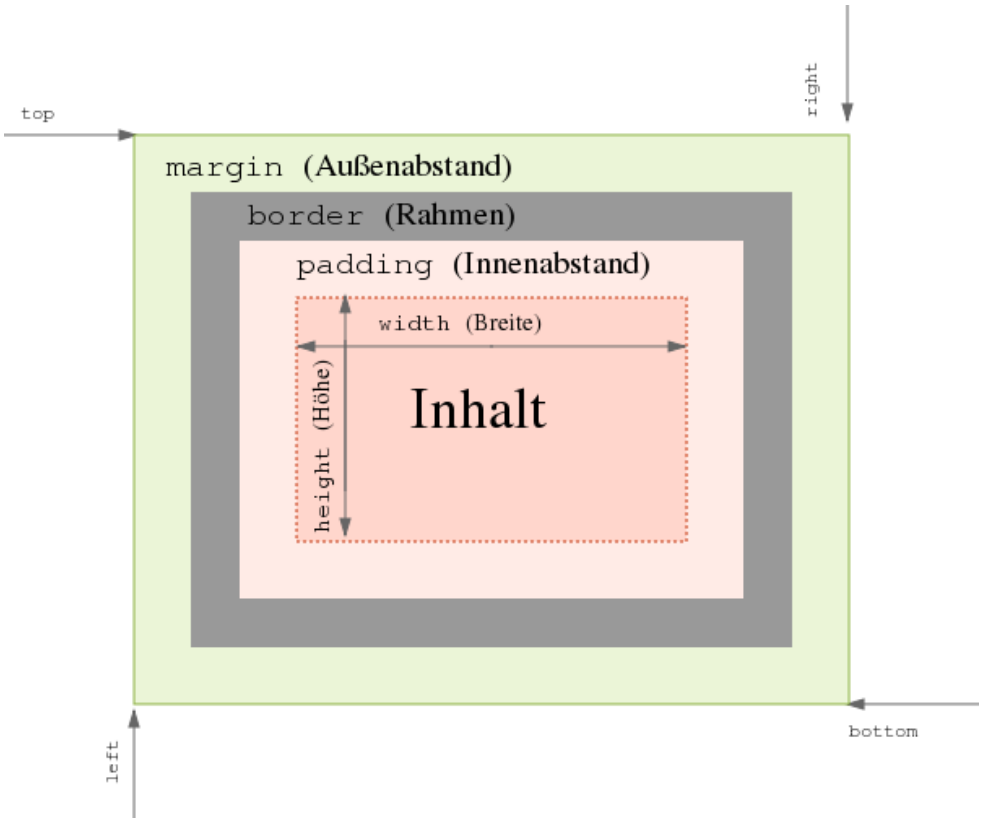


Abbildung 8.1: CSS Box Modell

```
div { width: 40mm; padding: 3mm; border-width: 2mm; margin: 5mm; }
```

⇒ Gesamtbreite: 60mm, da padding, border-width und margin doppelt gezählt werden.

Falls es keine explizite Angabe durch den Autor der Seite gibt, so wird folgendes verwendet:

- Default-Vorgabe (`user-agent`)
- dynamische Anpassung (hier an den Inhalt)

8.2.2 Eigenschaften/„properties“

Eigenschaft „`display`“

Die Eigenschaft `display` gibt an, wie die Elemente im Fluss dargestellt werden:

`display: inline;`	Innerhalb einer Zeile (sofern genug Platz)
`display: block;`	Beginnt und endet auf jeden Fall in einer eigenen Zeile
`display: none;`	Element wird nicht dargestellt und nimmt auch keinen Platz ein

[1] Quelle: `https://wiki.selfhtml.org/wiki/CSS/Box-Modell`

Es gibt noch weitere `values` als Spezialfälle für Tabellen, Listen, ...

Tipp

Eine Alternative zu `display: none;` ist `visibility: hidden;` (anstatt `visibility: visible;`). Damit wird das Element nicht dargestellt, nimmt aber seinen Platz in Anspruch.

Eigenschaft „`position`“

`position: static;`	„normale“ Position, Default-Wert
`position: relative;`	relative Verschiebung zur „normalen“ Position Der Platzbedarf bleibt an der ursprünglichen Position erhalten
`position: absolute;`	Positionierung relativ zum nächsten Vorfahren, welches eine von `static` verschiedene Position aufweist (bzw. `body`)
`position: fixed;`	Positionierung relativ zum Browserfenster („viewport“)

Unterschied `fixed` zu `static/absolute/relative`: scrollt nicht mit bzw. scrollt mit.

Unterschied `absolute/fixed` zu `static/relative`: Element hat keinen Platzbedarf bzw. hat Platzbedarf.

Die tatsächliche „Position“ wird angegeben mittels `top`, `bottom`, `left` und `right`.

`width`, `right`, `margin`, `top`, ... erwarten Längenangaben als Werte. Diese können sein:

- absolute Werte, z. B. $50mm$, $3in$, $2pt$ („Punkt der Schriftsetzer“) ⇒ werden selten verwendet
- relative Werte, z. B.

300px	Pixel, d. h. relativ zur Bildschirmauflösung. Meist ungünstige Wahl für allgemeine Größenangaben, verwendet z. B. für `border` Angaben
50%	Prozentangabe, d. h. relativ zur entsprechenden Bezugsgröße des Elternelements meist oder oft eine gute Wahl (`width` ⇒ `width`, `top` ⇒ `height`)
2em	Breite des Kleinbuchstabens „m“ in der aktuellen Schrift. Ebenfalls eine gute Wahl.
4rem	Breite des Kleinbuchstabens „m“ in der Schrift des `root` Elements (für die Unterstützung siehe `http://caniuse.com/#feat=rem`)
3ex	Höhe des Kleinbuchstabens „x“ in der aktuellen Schrift

Eigenschaften „margin“, „padding“, „border-width“, ...

Diese Eigenschaften sind Sammeleigenschaften für die Dimensionen an allen vier Seiten.

margin-top	padding-top	border-top-width	...
margin-right	padding-right	border-right-width	...
margin-bottom	padding-bottom	border-bottom-width	...
margin-left	padding-left	border-left-width	...

Gesammelt ergibt dies zum Beispiel:

```
margin: 1mm 2mm 3mm 4mm;
```

Die Eigenschaften werden in der Reihenfolge `top`, `right`, `bottom` und `left` mit Leerzeichen getrennt angegeben. Es können aber auch weniger Werte angegeben werden. Fehlende Werte werden ersetzt:

- left = right (3 oder 2 Werte)
- bottom = top (2 Werte)
- left = bottom = right = top (1 Wert)

`border` ist auch eine Sammeleigenschaft für:

1. `border-width` ⇒
2. `border-style` ⇒ jeweils ebenfalls eine Sammeleigenschaft für vier Seiten
3. `border-color` ⇒

`border-style` `solid | dotted | dashed | double | ...| none`

`border-color`

- `red | green | blue|...`
- `#abc1f5` (RGB-Angabe, je zwei Zeichen)
- `#a5f` ≙ `#aa55ff` (RGB-Angabe, je ein Zeichen)
- `rgb(122,231,85)` (RGB-Angabe)
- `rgba(122,231,85,0.5)` (RGB-Angabe + Alpha-Kanal für Deckkraft von 0 bis 1)
- weitere Farbmodelle sind u. a. `hsl(...), hsla(...), ...`
- weiteres Schlüsselwort, manchmal anwendbar (z. B. für Hintergrundfarbe): `transparent` (bei `rgba()` nicht mehr notwendig

Weitere Eigenschaften mit Farbangaben sind u. a. :

- `color` (Schriftfarbe)
- `background-color` (Hintergrundfarbe)

Eigenschaften für den Hintergrund

- `background-image: url("url/zum/bild.jpg");`
- `background-repeat: no-repeat | repeat | repeat-x | repeat-y;`
- `background-position: left|right|center top|bottom|center;`
  oder Längenangabe (mit Positionierung der Bildmitte)
- `background-attachment: fixed | scroll;`

Eigenschaften für Schriften

- `font-style: normal|italic|oblique` schräggestellte Schrift
- `font-variant: normal|small-caps` „Kapitälchen“
- `font-weight: normal|bold|bolder|lighter|X%|Zahl` Schriftdicke: Die meisten Schriften unterstützen nur zwei verschiedene Schriftdicken
- `font-size:` Schriftgröße – Angabe der Längenangabe (vorzugsweise Prozentwerte)
- `font-family: "Name der Schriftfamilie"| serif | sans-serif | cursive | monospace | fantasy`
  Alternativangaben werden durch Kommata getrennt angegeben

Die Sammeleigenschaft dafür ist `font`.

8.3 Ergänzungen – CSS

Positionierung: „dritte“ Dimension: `z-index: <Zahlenwert>;`

⇒ Element mit größeren `z-index` überlagern Elemente mit kleinerem `z-index`

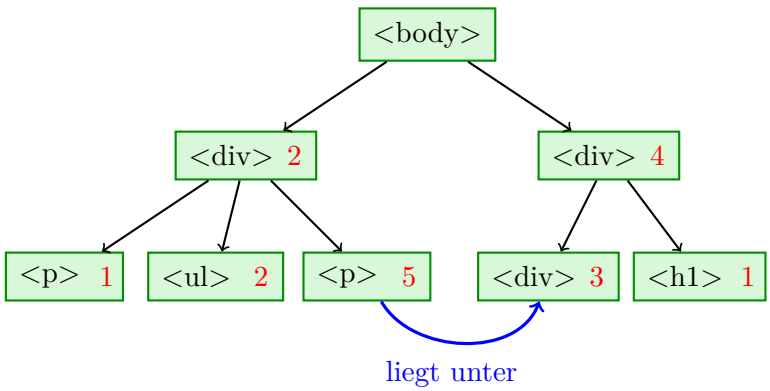


Abbildung 8.2: Beispiel – Z-Index

Wie in Abbildung 8.2 zu sehen ist, bezieht sich der `z-index` auf den aktuellen Kontext und nicht notwendigerweise dokumentenweit.

Umfließen von Inhalten mit `float: none|left|right;` (siehe Abbildung 8.3).

Nachfolgende Inhalte befinden sich bei `left` auf der rechten Seite (das Bild links). Möchte man z. B. die nächste Überschrift nicht neben den `floats` haben, so kann man dies machen mit:

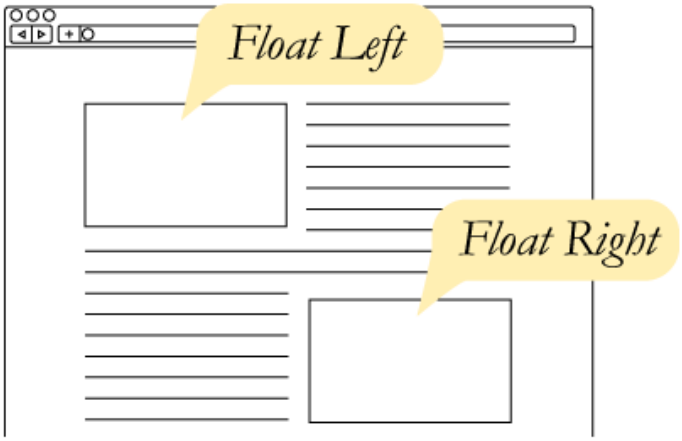


Abbildung 8.3: CSS – `float`

`clear: none|left|right|both;`

Damit werden alle offenen (`left/right`) `floats` vor dem aktuellen Element gesetzt/„geschlossen“.

8.3.1 Selektoren

Regeln werden vererbt (auf Kinder und Kindeskinder).

```
1   body { font-size: 12pt; }
2   h1   { font-size: 18pt; }
```

Listing 8.4: Selektoren

Grundsätzlich gilt die Regel mit dem für das jeweilige Element spezifischsten Selektor. Weniger weit vererbte Selektoren sind spezifischer.

Eigenschaft für Aufzählungszeilen bei Listen:

`li { list-style-type: decimal; }`
⇒ gilt für alle `<li>` (sowohl für `<ul>` als auch für `<ol>`).

Deswegen ist folgendes besser: `ol { list-style-type: decimal; }`

Kontextselektoren

`ol li {...}`	⇒ gilt für *alle* `<li>`-Kinder eines `<ol>`
`ol > li {...}`	⇒ gilt für alle *direkten* `<li>`-Kinder eines `<ol>` (keine Kindeskinder)
`h1 ~ p {...}`	⇒ gilt für `<p>`, wenn es einen vorangegangenen Geschwisterknoten `<h1>` gibt.
`h1 + p { ...}`	⇒ gilt für alle `<p>`, die einen direkten Vorgänger `<h1>` haben.

Die Kombination von Selektoren ist möglich:
`h1 + p + p {...}` ⇒ gilt für das genau 2. `<p>` welches direkt auf `<h1>` folgt.

Kontextselektoren sind spezifischer als Selektoren ohne Kontext!

Attributselektoren

`a[href] {...}`	⇒	gilt für `<a>`-Tags mit `href`-Attribut.
`a[href="/"] {...}`	⇒	gilt für `<a>`-Tags mit Wert von `href`-Attribut „/“ (Verweis auf die Einstiegsseite).
`a[href^="http"] {...}`	⇒	gilt für alle `<a>`-Tags, deren `href`-Attribut mit „http“ beginnt
`a[href^="http://"],` `a[href^="https://"] {...}`	⇒	gilt für alle `<a>`-Tags, die auf externe Seiten verlinken.
`a[href$=".pdf"] {...}`	⇒	gilt für alle `<a>`-Tags, die auf eine PDF verweisen.
`a[href*="google."] {...}`	⇒	gilt für alle `<a>`-Tags, die auf Google verweisen (bzw. die `google.` enthalten).

Klassenselektoren

`div.klassenname {...}` ⇒ gilt für alle `<div>` der Klasse „klassenname“ (`<div class="klassenname"></div>`).

Ein Klassenselektor ist spezifischer als beliebig viele Kontexte eines Selektors.

ID-Selektoren

`div#idwert {...}` ⇒ gilt für einen `<div>` mit ID „idwert“.

Hinweis

`.klassenname` und `#idwert` anstatt `div.klassenname` und `div#idwert` ist auch möglich. Dann gibt es keine Beschränkung auf bestimmte Elementarten!

Ein ID-Selektor ist auf jeden Fall spezifischer als ein Klassenselektor!

Pseudoklassen

Einige Pseudoklassen sind abhängig von „externen Randbedingungen“

`a:link`	⇒	gilt für alle `<a>`-Tags, welche Hyperlinks darstellen (Alternative für `a[href] {...}`).
`a:visited {...}`	⇒	gilt für alle `<a>`-Tags, welche *besuchte* Hyperlinks darstellen).
`a:focus {...}`	⇒	gilt für alle `<a>`-Tags, die den aktuellen Fokus haben (z. B. mit der Tab Taste).
`a:hover {...}`	⇒	gilt für alle `<a>`-Tags, über der sich der Mauszeiger im Moment befindet.
`a:active {...}`	⇒	gilt für alle `<a>`-Tags, die im Moment geklickt werden.

Achtung

Alle Selektoren sind gleich spezifisch, d. h. die Reihenfolge ist wichtig. Die letztnotierte Regel überschreibt die anderen. Im oberen Beispiel darf deswegen `a:link {...}` nicht nach `a:hover {...}` stehen, da dies ansonsten überschrieben werden würde.

Pseudoelemente

```
a[hrefˆ="http://"]:after { content: "Achtung! Externer Link"; }
a[href$=".pdf"]:after { content: url("logo-pdf.png"); }
a[href$=".pdf"]:before { content: url("logo-pdf.png"); }
```

weitere sind u. a.

`:first-letter {...}`	Der erste Buchstabe
`:first-line {...}`	Die erste Zeile
`:first-child {...}`	Das erste Kindelement

Es gibt noch viele weitere Pseudoelemente in aktuellen CSS Versionen, die jedoch nicht für die Vorlesung von Bedeutung sind.

Selektoren mit Pseudoklassen haben grundsätzlich dieselbe Spezifität wie „normalen“ Klassen.

8.3.2 Quellen von CSS-Regeln

Es gibt verschiedene Quellen von CSS-Regeln. Diese sind:

author Vom Webautor

- `<link>`-Tag (am „unspezifischsten“)
- `<style>`-Tag im `<head>` (dazwischen)
- `style`-Attribut (ist am spezifischsten)

(bei gleicher Spezifität der Selektoren)

Grundsätzlich: Eine Regel (bzw. beim `style`-Attribut die Deklaration) überschreibt andere Regeln mit gleicher Spezifität des Selektors, welche sich „weiter weg vom Element“ befinden.

user Betrachter der Website.

user-agent Webbrowser, mit dem der User die Seite anschaut.

Prioritätenreihenfolge (bei gleicher Spezifität des Selektors):

```
author > user > user-agent
```

Barrierefreiheit?

Mit `!important` kann die Priorität einer Deklaration erhöht werden:

```
body { font-family: serif !important; font-size:50pt; }
```

Nur die Schriftfamiliendeklaration hat Vorrang vor allen anderen Deklarationen ohne `!important`.

Es gibt eine andere Prioritätsreihenfolge bei `!important`:

```
user > author
```

Der `user-agent` hat keine `!important`-Deklarationen!

Beispiel für eine Klausuraufgabe

Es ist eine HTML- und CSS-Datei vorgegeben. Die HTML-Datei enthält Überschriften unterschiedlicher Art, welche unterschiedlich gestyled wurden (über IDs, Klassen, ...). Auch user- und author-Stylesheets wurden angegeben.

Die Klausurfrage lautet dann: „Welche Farbe wird für welche Überschrift verwendet?“

9 Projekt

Es muss vorher mit Herrn Röthig abgesprochen werden, welches Projekt gewählt wird.

Folgende Projekte stehen zur Verfügung:

1. **Stammbäume und Familientafel** – Datenerfassung, Speicherung und Darstellung

   Aufgabe ist es, Personendaten, wie sie für Stammbäume und Familientafeln verwendet werden, eingeben, speichern und in geeigneter Form darstellen zu können. Zur Eingabe wie zur Ausgabe der Daten ist ein gängiger (moderner) Web-Browser (Firefox, Chrome) vorzusehen, die Datenhaltung erfolgt auf dem Webserver in einer einfachen (strukturierten) Textdatei. Alternativ kann bei Interesse zur Datenspeicherung auch eine Relationale Datenbank (RDB) zum Einsatz kommen.

   Zu verwendende Techniken:

   - Datenhaltungsformat: XML/XML-ähnlich, alternativ RDB
   - Datentransferformat: XML
   - Definition des Datenformats: DTD
   - Wartung per XSLT in das Ausgabeformat
   - Ausgabeformat: XHTML und/oder SVG

   Eingabe der Daten: HTML-Formulare, JavaScript, einfache serverseitige Auswertung und Abspeicherung in beliebiger webserver-fähiger Programmiersprache (PHP, Perl, bash, . . .). Ausgabe der Daten/Transfomation in darstellbare Form: mittels XSLT nach HTML und/oder (alternativ) SVG.

   Die Arbeit ist gruppenweise für geeignete Teilaufgaben zu bearbeiten. Für einzelne Teilaufgaben können auch Alternativlösungen von mehreren Gruppen erarbeitet werden, die dann austauschbar in das Gesamtprojekt integrierbar sein sollten.

   Teilaufgaben wären beispielsweise die Definition der Datenstruktur, die clientseitige Eingabekomponente, die Dateneingabe und -haltung auf dem Server, die Ausgabe im Webbrowser. Letztere sollte idealerweise in verschiedenen Formen und parametrisierbar erfolgen können, beispielsweise die Ahnentafel von X oder alle Nachkommen von Y.

   Es sollte mindestens eine komplette Lösung am Ende als Ergebnis zustande kommen.

2. **Terminverwaltung/Kalender**

   Vergleichbare Bearbeitung, insbesondere unter Nutzung der gleichen Techniken wie für die Aufgabenstellung Stammbaum - XML, XSLT, HTML, CSS, evtl. JavaScript, SVG

3. **Interaktive Kartenanwendung**

   Aufgabe ist, aus als Geokoordinaten vorliegenden Kartendaten (Ländergrenzen, Straßen, Flüsse, Küsten, . . .) eine graphische Darstellung im Web zu erstellen. Die Ausgangsdaten

können aus OpenStreetMap oder anderen frei zugänglichen Datenquellen gewonnen werden, die Zieldarstellung ist in der Sprache SVG zu realisieren. Mindestens eine interaktive Komponente muss enthalten sein, beispielsweise die Angabe von Kenndaten (Name, Lage, Einwohnerzahl, Bedeutung), wenn man auf ein Land klickt.

Einige Teilprobleme:

- Eigene konkrete und genau Zielaufgabenstellung selbst erstellen (!)
- Sichtung und Auswahl der geeigneten Datenquelle(n)
- Reduzierung des Umfangs der Geokoordinaten auf eine im Web (für eine SVG-Datei) praktikable Größe unter Verringerung der Auflösung
- Definition eines geeigneten XML-basierten Datenformats zur Speicherung der relevanten Daten
- Transformation der XML-Ausgangsdaten mit Hilfe von XSLT in eine SVG-Datei
- interaktive Veränderung der SVG-Datei mit JavaScript
- Anforderung nachzuladender Teile vom Server und Einbau in die bestehende SVG-Datei (AJAX)
- serverbasierte dynamische Generierung der nachgeladenen Komponenten (optional)

Achtung

Diese dritte Aufgabenstellung ist relativ unscharf formuliert, was bedeutet, dass das Potential für eigene Kreativität recht gross ist, aber dieses muss auch vorhanden sein und genutzt werden, damit eine „nette“ Anwendung am Ende dabei herauskommt!

4. **Begriffswolken**

Aufgabe ist es, Begriffe und Zusammenhänge zwischen Begriffen mittels XML modellieren zu können. Insbesondere ist eine graphische Darstellung von Begriffswolken am Ende das Ziel. Orientieren kann man sich an Mindmaps und Cognitive Maps sowie ggf. Tag Clouds, wobei keine Beschränkung auf beispielsweise strenge Hierarchien erfolgen soll sowie unbedingt (im Gegensatz zu Tag Clouds) die Beziehungen zwischen Begriffen modelliert und graphisch dargestellt werden müssen.

Zu verwendende Techniken:

- XML (zur Modellierung der Datenstruktur)
- XSLT (zur Transformation der Daten in das Ausgabeformat)
- SVG (als Ausgabeformat)
- weitere Techniken nach Bedarf und Kenntnis

Achtung

Achtung: Auch diese vierte Aufgabenstellung ist relativ unscharf formuliert, was bedeutet, dass das Potential für eigene Kreativität recht gross ist, aber dieses muss auch vorhanden sein und genutzt werden, damit eine „nette“ Anwendung am Ende dabei herauskommt!

Für alle Aufgabenstellungen gilt:

Die Techniken XML (als Datenhaltungsformat oder mindestens als Transferformat im Falle des Einsatzes einer anderen Datenbanktechnik), XSLT (zur Transformation der Daten) sowie SVG und/oder XHTML (als Ausgabeformat) müssen verwendet werden.

Erforderlich ist am Ende ein (einfach installierbares) Archiv (tar, ggf. gzipped, oder zip) mit allen Quellen und ggf. ausführbaren Dateien sowie eine kurze Installationsanleitung, ggf. mit genauer Nennung der Voraussetzungen zur Installation. Außerdem benötige ich für jedes Projekt die Namen der Mitwirkenden bzw. Mitgewirkt Habenden.

Die Bewertung (20% Anteil an der Gesamtnote) umfasst mehrere Kategorien, die sich hauptsächlich aus den drei Bereichen Projektidee, technische Umsetzung und Erfüllung der Anforderungen ergeben.

10 XML – Extensible Markup Language

Extensible Markup Language (XML):

- Vorgaben für die Grammatk einer Sprache
- Tags mit Attributen und Inhalten
- jede XML-Datei muss/sollte wohlgeformt und gültig sein!

  Wohlgeformtheit:

  – Tags immer paarweise, d. h. zu jedem Start-Tag gibt es das passende End-Tag
  – korrekte Schachtelung der Tags, d. h. das zuletzt geöffnete und noch nicht geschlossene Tag muss als erstes geschlossen werden.
  – Es gibt genau einen „Wurzeltag“/„root-Tag“, d. h. genau ein Tag auf oberster Ebene welches das gesamte restliche Dokument enthält.

  ⇒ Kann ohne Kenntnis der konkreten Sprache geprüft werden!

  Gültigkeit:

  – evtl. Name des Wurzelelements
  – Elementnamen
  – Enthaltenseinsmodell für jedes Element (möglicher Inhalt eines Tags)
  – Attributnamen
  – Zugehörigkeit von Attributen zu Tags
  – Attributtyp (mögliche Attributwerte)

  ⇒ Beschreibung der konkreten Grammatik

  Gültigkeit kann anhand einer DTD geprüft werden. Eine Alternative dazu ist XML Scheme Definition (XSD) (deutlich mehr Möglichkeiten für Inhaltsmodell und Typisierung, jedoch viel komplexer).

10.1 DTD

Die Document Type Definition (DTD) ...

... wird referenziert in einer (vollständigen) `DOCTYPE`-Deklaration.
... ist textbasiert, aber nicht XML-basiert
... besteht aus einer Folge von (also beliebig vielen) Deklarationen (siehe Listing 10.1) in der Variante der `DOCTYPE`-Deklaration

```
1  <!DOCTYPE root_element ... mit Param. als Aufzählung der Werte in bestimmter ↙
   ↳ Reihenfolge>
2  <?xml ... mit Parametern als Parname="Parwert" ?>
```

Listing 10.1: DOCTYPE- und XML-Deklaration

Für unsere Zwecke reichen zwei Deklarationen: `<!ELEMENT ...>` und `<!ATTLIST ...>`

Eine DTD wird wie folgend aufgebaut:

10.1.1 ELEMENT

Beschreibt ein Element und seinen Inhalt.

```
1    <!ELEMENT tagname inhaltsmodell>
```

Listing 10.2: DTD - ELEMENT

tagname Name des Tags oder Elements, bestehend aus Buchstaben (Groß- und Kleinschreibung, case-sensitive), Ziffern, manchen Sonderzeichen (z. B. Unterstrich/„_“), beginnend nur mit Buchstabe oder Unterstrich.

Theoretisch kann der `tagname` eine beliebige Länge haben, sollte aber aus praktischen Gründen auf < 256 Zeichen beschränkt werden. Zudem sollten keine Umlaute und sonstige nationale Sonderzeichen verwendet werden.

inhaltsmodell Dies kann sein:

EMPTY Leeres Inhaltsmodell, d. h. der Tag enthält immer genau nichts.
Beispiel: `<!ELEMENT br EMPTY>` für einen Zeilenumbruch in HTML.

ANY Beliebiger Inhalt, d. h. beliebige Mischung aus Text und Tags (welche aber in der DTD deklariert sein müssen). Es gibt kein Beispiel in HTML (und jeder anderen dem Vorleser bekannten Grammatik). ⇒ vor dem Inhaltsmodell „`ANY`“ warnt der Vorleser!

(#PCDATA) „Parsed Character Data“
Zeichenfolgen, welche keine Tag-ähnlichen Strukturen enthalten (z. B. „<“ mit `<` umschreiben).
Beispiel: `<!ELEMENT title (#PCDATA) >`

sequenz `(tagname1, tagname2[, tagname3...])`
Inhalt sind die aufgelisteten Tags (in genau dieser Reihenfolge). Beispiel: `<!ELEMENT html (head, body) >`

auswahl `(tagname1|tagname2[|tagname3...])`
Inhalt ist entweder `tagname1` oder `tagname2` oder ... Es können auch Sequenzen angegeben werden.
Beispiel (aus früherem HTML): `<!ELEMENT html ((head,body)|frameset)`

gemischt `(#PCDATA|tagname1[|tagname2...])`
Inhalt ist entweder Text oder `tagname1` oder `tagname2` ...
Hinweis: Statt `tagname` sind auch weitere Inhaltsmodelle (sequenz, auswahl) möglich!

Häufigkeitsindikatoren

Nachgestelltes Symbol, mit dem ein Inhaltsmodell in der Häufigkeit seines Auftretens beeinflusst werden kann.

- **`*`** Beliebig viele (inkl. keinmal)
- **`+`** Beliebig viele, aber mindestens einmal
- **`?`** Einmal oder keinmal („optional“)

Beispiel:

- `<!ELEMENT p (#PCDATA|em|a|span|...)* >`
- `<!ELEMENT ul li+ >`
- `<!ELEMENT dl (dt|dd)+ >`

10.1.2 `ATTLIST`

Beschreibt die Attribute eines Elements.

```
1   <!ATTLIST tagname attrname attrtyp voreinstellung>
2                  |_____  auch mehrfach  _____|
```

Listing 10.3: DTD – ATTLIST

`tagname` Der Tag, für welche die Attribute deklariert werden

`attrname` Name des Attributs (derselbe Aufbau und dieselben Einschränkungen wie für `tagname`)

`attrtyp`

`CDATA` (Character Data), d. h. beliebige Zeichenfolge (inkl. Tag-ähnlichen Strukturen, welche hier einfache Zeichenfolgen darstellen. Doppelte Hochkommata müssen mit `&quot;` umschrieben werden. *Beispiel*: `<!ATTLIST img alt CDATA #REQUIRED>`

`ID` Dokumentenweit eindeutiger Attributwert, Aufbau/Zusammensetzung wie ein `tagname` (es ist kein rein numerischer Wert möglich). *Beispiel*: `<!ATTLIST a id ID #IMPLIED>`

`IDREF, IDREFS` Ein Attributwert vom Typ ID, potentiell mehrere Attributwerte vom Typ ID (durch Leerzeichen getrennt).

Beispiel aus HTML: Bei Formularen, nicht jedoch bei `<a href="">`

`NMTOKEN, NMTOKENS` „Nametoken“ Aufbau ähnlich wie `tagname`, aber jedes der erlaubten Zeichen kann erstes Zeichen sein! So kann `123abc` kein Tagname sein, da Tags nicht mit einer Zahl anfangen dürfen, ein `NMTOKEN` jedoch schon!
Durch Leerzeichen werden mehrere `NMTOKEN` voneinander getrennt.

Beispiel aus HTML: `<!ATTLIST div class NMTOKENS #IMPLIED>`

Aufzählung `nmtoken1|nmtoken2[|nmtoken3...]`
Aufzählung aller möglichen Werten vom Typ Nametoken

Voreinstellung

`"value"` Ein vorgegebener Standardwert vom selben Typ wie `attrtyp`.

`#IMPLIED` Gibt an, dass das Attribut optional ist.

`#REQUIRED` Gibt an, dass das Attribut Pflicht ist.

`#FIXED "val"` Wenn das Attribut gesetzt wird, darf es nur den Wert `val` annehmen.

Beispiel aus HTML: `<!ATTLIST video autoplay #FIXED "autoplay">`

Hinweis: Manche Attributtypen, wie z. B. ID, können nur die Voreinstellung `#IMPLIED` oder `#REQUIRED` besitzen.

weitere Attributtypen

- `ENTITY`, `ENTITIES`
- `NOTATION`

10.2 Was ist XSLT?

XSL Transformation (XSLT) ist Teil der Extensible Stylesheet Language (XSL)-Spezifikation.

XSLT Eine Sprache zur Umsetzung von XML basierten Dokumenten in andere (meist ebenfalls XML-basierte) Dokumente.

XML Path Language (XPath) Sprache zur Selektion von Knotenmengen) aus einem XML-Dokument

XML-FO Konkrete XML-basierte Sprache zur designgetreuen Ausgabe von Dokumenten.

Wer führt die Transformation durch?

- ein Standalone-Tool:
  - XSL Transformator (Kommandozeilenaufruf: `xslt`)
  - xalan
  - saxon (auch für Version 2, kommerziell)
- Server-Side:
  - Apache-Projekt Cocoon
  - Perl-Modul: AxKit
- Client-Side:
  - gängige WebBrowser (Chrome, Firefox, Edge, Safari, ...)

Die XSLT-Sprache ist XML-basiert . Listing 10.4 zeigt ein Beispiel.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <xsl:stylesheet version="1.0"
3         xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4         xmlns="Namespace der Zielsprache">
5    <xsl:output method="xml|text|html" encoding="KodierungZielformat"
6         doctype-public="Public-Id für DOCTYPE-Dekl."
7         doctype-system="System-ID für DOCTYPE-Dekl." />
8    <!-- Liste von Transformationsvorschriften/Templates -->
9  </xsl:stylesheet>
```

Listing 10.4: XSLT – Aufbau

Editorempfehlung von Herrn Röthig

Editix als FreeEditix. Erhältlich auf: `http://editix.com/`

Wie die Verknüpfung einer XSLT mit einem XML-Dokument aussieht, zeigt Listing 10.5 auf Seite 50. Mit `<?xml-stylesheet href="..." ?>` (Bindestrich, kein Doppelpunkt) wird die XSLT aufgerufen.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <?xml-stylesheet type="text/xsl" href="url/zur/Transformdatei.xsl" ?>
3  <!DOCTYPE bla SYSTEM "http://ref.zur/DTD/fuer/unsere/Gramatik.dtd">
4  <bla>
5  </bla>
```

Listing 10.5: Abstraktes XML-Dokument für die Verknüpfung mit XSLT

10.2.1 Format der XSLT-Templates

```
1  <xsl:template match="bla"> <!-- XPath-Ausdruck -->
2    <!-- Text und Strukturen, welche im Zieldokument den bla-Knoten aus
3         dem Quelldokument darstellen und "ersetzen" sollen, z.B.
4    -->
5    <html><head>...</head><body>...</body></html>
6  </xsl:template>
```

Listing 10.6: XSLT Templates

In Abschnitt 10.4 wird XSLT genauer beschrieben.

10.3 Der XPath-Ausdruck

Der XPath-Ausdruck[1]

- besteht aus einem oder mehreren Lokalisierungssschritte, optional gefolgt von Prädikaten
- mehrere Lokalisierungssschritte werden durch „/“ getrennt
- ein Lokalisierungsschritt besteht aus Achse und Knotentest:
  `axis::node-test[predicate]/...`

[1] siehe auch `https://de.wikipedia.org/wiki/XPath#Achsen`

10.3.1 Achsen

Achsen geben das „Verwandtschaftsverhältnis“ der aufzusammelnden (gesuchten) zum aktuellen Knoten an.

`child`:	direkter Kindsknoten
`parent`:	der Elternknoten
`descendent`:	alle Nachfahrenknoten
`ancestor`:	alle Vorfahrenknoten
`descendent-or-self`:	Vereinigungsmenge von descendent und self
`ancestor-or-self`:	Vereinigungsmenge von ancestor und self
`preceding`:	Vorgängerknoten (ohne ancestor!)
`following`:	Nachfolgeknoten (ohne descendent!)
`preceding-sibling`:	ältere Geschwisterknoten (preceding mit demselben parent)
`following-sibling`:	jüngere Geschwisterknoten (following mit demselben parent)
`attribute`:	am aktuellen Knoten hängender Attributknoten (alle anderen Achsenausdrücke sammeln nur Element- und keine Attributknoten)

Verkürzte Schreibweise

Eine Verkürzte Schreibweise der Lokalisierungsschritte für Achsen[2]

`bla`	≙	`child::bla`
`../bla`	≙	`parent::bla`
`./`	≙	`self`
`@fasel`	≙	`attribute::fasel`

10.3.2 Knotentests

Knotentests schränken die Elementauswahl einer Achse ein:

`tagname`	nur die Knoten mit dem entsprechenden `tagname` (bzw. bei `attribute` der `attrname`)
`*`	Alle Knoten (Wildcard)
`text()`	Alle Textknoten
`comment()`	Für Kommentarknoten
`processing-instruction()`	für Knoten mit „processing instructions“

[2] siehe auch `https://de.wikipedia.org/wiki/XPath#Achsen`

10.3.3 Prädikate

Durch Angabe von Prädikaten kann das Ergebnis weiter eingeschränkt werden. Prädikate werden in eckige Klammern eingeschlossen und können in beliebiger Zahl hintereinander geschrieben werden, wobei die Reihenfolge wesentlich ist. Prädikate können XPath-Ausdrücke enthalten, außerdem kann eine Vielzahl von Funktionen und Operatoren verwendet werden.[3]

```
axis::node-test[predicate][/...]
```

Es handelt sich um eine Bedingung an die Knoten, welche erfüllt („wahr“) sein muss.

Bedingung

XPath-Ausdruck	nicht-leere Knotenmenge ergibt „wahr“
Zahl	(natürliche Zahl) ergibt den einen Knoten mit der entsprechenden Nummer aus der Knotenmenge beginnend mit 1.
„Vergleich“	zweier XPath-Ausdrücke auf Gleichheit (=), kleiner (`<`), größer (`>`), kleinergleich (`<=`), größergleich (`>=`)
Verknüpfen	mit logischen Operatoren `and`, `or`, `not`.
Zahlenoperationen	+, -, *

Funktionen

`number(Knotenmenge)`	⇒	nummerischer Wert des „Werts“ einer Knotenmenge
`count(Knotenmenge)`	⇒	Anzahl Knoten in der Knotenmenge
`substring(...)`	⇒	Teilzeichenkette
...		

10.4 XSLT - Aufbau

```
1  <xsl:template match="XPath-Ausdruck">
2    <!-- Inhalt, welcher vom Template ausgegeben wird -->
3  </xsl:template>
```

Listing 10.7: XSL Template

Mögliche Inhalte:

- Tags der Zielsprache
  `<html><head><title>...</title></head><body>...</body></html>`
- Text „bla fasel blubber“. Whitespace wird auf ein Trennzeichen („space“) reduziert.
- Text `<xsl:text> leer zeichen</xsl:text>` ⇒ Whitespace bleiben erhalten!
- „Werte“ aus dem Quelldokument: `<xsl:value-of select="Pfadausdruck" />`

[3] Quelle: `https://de.wikipedia.org/wiki/XPath`

- „Wert“ einer Knotenmenge
  - Konkatenation der Werte aller Knoten in der Knotenmenge
- „Wert“ eines Knotens
  - bei Textknoten: der Text
  - bei Attributknoten: der Text des Wertes des Attributknotens
  - bei Elementknoten: rekursive Ermittlung über alle Kindknoten, welche Element- oder Textknoten (*nicht* Attributknoten) sind (Tiefensuche, keine Breitensuche entsprechend der Notation im XML-Dokument).
    `<a>bla fasel blubba</a>` ⇒ Ein Textknoten an `<a>`
    `<a>bla<i>fasel</i>blubba</a>` ⇒ zwei Textknoten an `<a>`

Attribute

`<p style="color:red;">Dieser Text ist rot</p>` ⇒ fester Attributwert

`<xsl:attribute name="Attributname">` ⇒ erzeugt einen Attributknoten am soeben neu geöffneten (noch nicht geschlossenen und noch nicht mit Texten oder Elementknoten als Kinder versehen) Elementknoten.

Wert des Attributs als XPath-Ausdruck:
`<xsl:attribute name="style" select="pfad/im/Quelldokument"/>`

Wert des Attributs als „errechneter“ Wert im Quelltext:
`<xsl:attribute name="style">color:<xsl:value-of select="..." ↙`
`↳ /></xsl:attribute>`

10.4.1 Aufruf der Templates

- muss explizit erfolgen
- das `match`-Attribut sorgt *nicht* für den Aufruf/die Ausführung des Transformators
- der XSLT ruft ein Template für den Wurzeltag auf doch
- mittels `<xsl:apply-templates ...>` können Templates rekursiv aufgerufen werden
  - `<xsl:apply-templates />` Template Aufruf für alle Kindelemente
  - `<xsl:apply-templates select="Pfadausdruck />` Template-Aufruf für alle Knoten der adressierten Knotenmenge (nicht nur Kinder- und Elementknoten)
- „schnelle“ Alternative zur Rekursion: iterativer Durchgang durch eine Knotenmenge

```
1   <xsl:for-each select="Pfadausdruck">
2     <!-- Ausgabe für jeden Knoten der Knotenmenge -->
3   </xsl:for-each>
```

Listing 10.8: XSLT – for-each

- Sortierung der Knoten bei `<xsl:apply-templates/>` und `<xsl:for-each ...>` vor Durchgang durch die Knotenmenge:
  `<xsl:sort select="XPath-Ausdruck als Sortierkriterium" />`
  als erste Kinder des `<xsl:apply-templates ...>` bzw. `<xsl:for-each ..>`
  ⇒ mehrere Sortierkriterien sind möglich (durch mehrere `<xsl:sort>` nacheinander)
- weitere Möglichkeiten zu Templates:

```
1   <xsl:template match="..." mode="Bezeichner">
2     <!-- dieses Template wir nur bei Aufruf mit gleichem mode-Attributwert ↙
    ↳ ausgeführt -->
3   </xsl:template>
```

Listing 10.9: XSLT – Mode

  Aufruf per: `<xsl:apply-templates select="..." mode="Bezeichner" />`

- Falls kein passendes selbstgeschriebenes Template existiert, existiert ein Default-Template (*ohne* mode-Attribut), welches alle Textknoten und für Elementknoten Templates rekursiv aufruft.

Zweite Art von Templates: „named templates“/Templates mit Namen

```
1   <xsl:template name="Bezeichner">
2     <!-- Ausgabe -->
3     <!-- Pfadausdrücke werden relativ zum aktuellen Knoten, welcher sich beim
4          Template-Aufruf nicht verändert hat, berechnet
5     -->
6   </xsl:template>
```

Listing 10.10: XSLT – Benannte Templates

Aufruf per: `<xsl:call-template name="Bezeichner" />`

10.4.2 Parameter und Variablen

Parameter können im Template definiert werden:
`<xsl:param name="Bezeichner">Default-Wert</xsl:param>`

Aufruf mittels: `<xsl:with-param name="Bezeichner">Wert</xsl:with-param>` innerhalb von `<xsl:apply-templates>` oder `<xsl:call-template>`.
Nutzung per `$Bezeichner` innerhalb von XPath-Ausdrücken.

„Variablen“ können in einem Block (und auch direkt innerhalb von `<xsl:stylesheet>` als „globale Variable“) definiert werden mittels:
`<xsl:variable name="Bezeichner">Wert</xsl:variable>` und Verwendung mittels `$Bezeichner`. *Aber*: Der Wert einer „Variablen“ ist nicht veränderbar, sondern fest. Es handelt sich also eher um Konstanten.

`<xsl:param>` kann auch direkt im `<xsl:stylesheet>` genutzt werden, um Parameter beim Aufruf des Stylesheets zu übergeben!

10.4.3 Bedingtes Ausführen

```
1   <xsl:if test="Bedingung">
2       <!-- Ausgabe/Ausführung, falls Bedingung erfüllt ist -->
3   <xsl:if>
4
5   <xsl:choose>
6       <xsl:when test="Bedingug 1">...</xsl:when>
7       <xsl:when test="Bedingug 2">...</xsl:when>
8       <xsl:otherwise>
9           <!--Default-Zweig, falls keine Bedingung erfüllt wurde -->
10      </xsl:otherwise>
11  </xsl:choose>
```

Listing 10.11: XSLT – Bedingtes Ausführen

Bei der XSLT If-Abfrage gibt es kein `else`!

11 Web-Engineering Übungsaufgaben

Achtung

Wenn du die XML-Datei lokal öffnest, kann es sein, dass die XSLT nicht geladen wird. Viele moderne Browser (z. B. Chrome und Edge) verbieten das Laden. Umgehen kann man dies, indem man einen lokalen Server aufsetzt und die Dateien über `localhost` aufruft oder einen anderen Browser (z. B. Firefox, Internet Explorer) installiert und nutzt.

Tipp: Nutze folgende Befehle, um mit Python[a] einen lokalen Server im aktuellen Ordner über ein Terminal zu starten (der Ordner ist dann erreichbar unter `http://localhost:8000`)

- Python 3: `python -m http.server -cgi 8000`
- Python 2: `python -m SimpleHTTPServer 8000`

[a] siehe `https://www.python.org/`

Hinweis

Es wurden nicht alle Aufgaben vollständig bearbeitet! Die Aufgaben, welche jedoch bearbeitet wurden, sollten soweit korrekt sein.

11.1 XML, DTD und Validierung

Konzipieren Sie eine XML-Struktur, mit welcher Sie einen einfachen CD-Katalog abbilden können. Die Datei soll für jede erfasste CD einen Eintrag erhalten, welcher mehrere Felder enthält. Sehen Sie dafür zunächst (mindestens) die folgenden Felder vor:

- Titel des Albums
- Interpret

Erstellen Sie die zur XML-Struktur passende DTD!

Erstellen Sie eine XML-Datei (mit wenigen Einträgen), welche dieser Struktur entspricht. Was passiert, wenn Sie diese Datei in einem XML-fähigen Web-Browser anzeigen?

Validieren Sie Ihre XML-Datei gegen die DTD! Erkennt der Validator, falls die Datei nicht wohlgeformt oder nicht gültig ist, also nicht der DTD entspricht? *Hinweis*: Die meisten Online-Validatoren erfordern, dass Ihre DTD im Internet online zugreifbar ist. Falls Sie keinen Internet-Webspace zur Verfügung haben, wo Sie Ihre DTD ablegen können, können Sie die DTD in Ihre XML-Datei direkt integrieren oder einen Offline-Validator verwenden. Beispielsweise enthält der XML-Editor Free Editix einen solchen.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!DOCTYPE katalog [
3   <!ELEMENT katalog (cd)+>
4   <!ELEMENT cd (titel, interpret)>
```

```
5   <!ELEMENT titel (#PCDATA)>
6   <!ELEMENT interpret (#PCDATA)>
7 ]>
8 <katalog>
9   <cd>
10    <titel>Godiva</titel>
11    <interpret>Heaven Shall Burn</interpret>
12  </cd>
13  <cd>
14    <titel>Ohne Dich</titel>
15    <interpret>Rammstein</interpret>
16  </cd>
17  <cd>
18    <titel>Mutter</titel>
19    <interpret>Rammstein</interpret>
20  </cd>
21  <cd>
22    <titel>Radio/Video</titel>
23    <interpret>System of a Down</interpret>
24  </cd>
25 </katalog>
```

Listing 11.1: XML mit DTD

11.2 Verknüpfung der XML-Datei mit einer XSLT-Datei

Verknüpfen Sie die XML-Datei nun mit einer (vorläufig leeren) XSLT-Datei und lassen Sie die Datei wiederum im Web-Browser darstellen. Was ist der Unterschied zur vorigen Ausgabe?

```
1    XML Parsing Error: no root element found
2        Location: file:///<Dateipfad>/CD_Katalog.xsl
3        Line Number 1, Column 1:
```

Listing 11.2: Ausgabe bei leerer XSLT Datei im Firefox

Da das `root`-Element in der XSLT-Datei fehlt, gibt es einer Fehlermeldung.

11.3 XSLT-Datei Grundgerüst

Füllen Sie die eben erstellte XSLT-Datei mit dem entsprechenden obligatorischen Grundgerüst (XML-Deklaration etc.). Lassen Sie unabhängig von der XML-Eingabedatei eine HTML-Datei (mit Titel und Überschrift) ausgeben.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3    <xsl:template match="/">
4      <html>
5        <head>
6          <title>Mein Katalog</title>
7        </head>
8        <body>
9          <h1>Mein CD Katalog</h1>
10       </body>
11     </html>
12   </xsl:template>
13 </xsl:stylesheet>
```

Listing 11.3: XML Datei mit XSLT Einbindung

11.4 XSLT-Ausgaben und rekursive Transformation

Schreiben Sie eine Transformation, die dafür sorgt, dass alle Inhalte von XML-Tags in der Eingabedatei (unabhängig von deren Struktur) in der Reihenfolge Ihres Auftretens in der Ausgabedatei ausgegeben werden.

11.5 Ausgabe von bestimmten Feldern

Ändern Sie die XSLT so ab, dass nur die jeweiligen Titel der CDs ausgegeben werden.

11.6 Nutzung bestimmter Felder als Attributwerte von HTML-Tags

Sehen Sie am Anfang Ihrer XML-Datei einen Tag „Farbe“ vor und nutzen Sie diesen, um damit die Hintergrundfarbe Ihrer HTML-Datei zu setzen. Was ist der Unterschied zur vorigen (direkten) Ausgabe von Feldern?

11.7 Sortierung

Sortieren Sie die Einträge nun nach dem (nicht angezeigten) Interpreten.

11.8 Transformation abhängig von der XML-Struktur

Erweitern Sie die Ausgabe auf alle vorhandenen Felder. Sorgen Sie dafür, dass jeder CD-Eintrag in einer neuen Zeile beginnt.

11.9 Erstellen einer Tabelle aller Einträge

Ändern Sie die XSLT so ab, dass die CD-Liste in Tabellenform ausgegeben wird. Welches Problem kann auftreten, wenn die CD-Einträge unvollständig sind, wenn also bei einem bestimmten Eintrag nicht alle möglichen Felder (Tags) vorhanden sind?

11.10 Attribute

Erweitern Sie die CD-Datenbank um die jeweilige Musikgattung (Rock, Pop, Jazz, Klassik). Welche Möglichkeiten bieten sich hierfür an? Welche der Möglichkeiten ist sinnvoll, falls der Eintrag abhängig von der jeweiligen Musikgattung unterschiedliche Felder enthalten soll (bei Klassik beispielsweise Komponist, Werk, ... statt Titel, Interpret, ...)?

11.11 Filterung

Geben Sie Listen der CDs in ihrer Sammlung aus. Erstellen Sie zum einen Listen getrennt für die jeweilige Musikgattung. Erstellen Sie außerdem eine komplette Liste Ihrer CDs, bei der die Einträge in der Reihenfolge ihres Auftretens in der XML-Datei aufgeführt sind.

11.12 Rekursive und iterative Transformation

Vergleichen Sie anhand der bisherigen Aufgaben die Verfahrensweise bei iterativer und rekursiver Transformation. In welchen Fällen haben Sie die iterative und in welchen die rekursive Transformation angewandt? Können Sie auch die jeweilige andere Verfahrensweise einsetzen? Realisieren Sie dies, soweit möglich!

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <!DOCTYPE katalog [
3    <!ELEMENT katalog (farbe, (cd)+)>
4    <!ELEMENT cd (titel, interpret)>
5    <!ELEMENT titel (#PCDATA)>
6    <!ELEMENT farbe (#PCDATA)>
7    <!ELEMENT interpret (#PCDATA)>
8  ]>
9  <?xml-stylesheet type="text/xsl" href="CD_Katalog.xsl"?>
10 <katalog>
11   <farbe>#abcdef</farbe>
```

```
12   <cd>
13     <titel>Godiva</titel>
14     <interpret>Heaven Shall Burn</interpret>
15   </cd>
16   <cd>
17     <titel>Ohne Dich</titel>
18     <interpret>Rammstein</interpret>
19   </cd>
20   <cd>
21     <titel>Mutter</titel>
22     <interpret>Rammstein</interpret>
23   </cd>
24   <cd>
25     <titel>Radio/Video</titel>
26     <interpret>System of a Down</interpret>
27   </cd>
28 </katalog>
```

Listing 11.4: XML Datei mit XSLT Einbindung

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3    <xsl:template match="/">
4      <html>
5        <head> <title>Mein Katalog</title> </head>
6        <body>
7          <xsl:attribute name="bgcolor">
8            <xsl:value-of select="/katalog/farbe"/>
9          </xsl:attribute>
10         <h2>CD-Katalog</h2>
11         <table border="1">
12           <tr bgcolor="white">
13             <th>Titel</th> <th>Interpret</th>
14           </tr>
15           <xsl:for-each select="/katalog/cd">
16             <xsl:sort select="interpret"/>
17             <tr>
18               <td><xsl:value-of select="titel"/></td>
19               <td><xsl:value-of select="interpret"/></td>
20             </tr>
21           </xsl:for-each>
22         </table>
23       </body>
24     </html>
25   </xsl:template>
26 </xsl:stylesheet>
```

Listing 11.5: XSLT Beispiel - Iterativer Ansatz

Hinweis

Der rekursive Ansatz fehlt an dieser Stelle!

12 SVG – Scalable Vector Graphics

12.1 Grafikformate (im Web)

In Tabelle 12.1 werden Raster- und Vektorgrafiken miteinander verglichen.

Rastergrafiken (Pixel-, Bitmap-Grafiken)	**Vektorgrafiken**
PNG, JPEG, GIF, (BMP)	SVG
Unterteilung der Darstellungsfläche in (meist gleichgroße, quadratische oder zumindest rechteckige) Teilflächen	Darstellung mithilfe von geometrischen Grundformen (z. B. Kreisen, Linien, . . .)
Zuordnung eines Farb- und/oder Helligkeits- und ggf. Transprarenzwertes zu jeder Teilfläche	Zuordnung von Position, Größe sowie Farb-, Helligkeits-, und Transparentwerten
⊕ gängige Ein- und Ausgabegeräte arbeiten rasterbasiert	⊕ nahezu unendliche Skalierbarkeit
⊕ gut geeignet für fotorealistische Darstellungen	⊕ gut geeignet für schematische, künstlich erzeugte Darstellungen
	⊖ evtl. Zeit und rechenaufwändiges „Rendering“

Tabelle 12.1: Vergleich von Raster- und Vektorgrafiken

12.1.1 RGB-Format (für Rastergrafiken)

RGB hat getrennte Farbwerte für Rot, Grün und Blau, welche jeweils im Wertebereich von 0 bis 255 (8 Bit) liegen. Es bestitzt ein additives Farbmodell, d. h. jeweils Farbe, je nach Helligkeit.

Bei einer gängigen Auflösung von z. B. 20 Megapixel:

- besteht das Bild aus 20 Mio. Teilflächen.
- braucht jedes Bild 60 MByte Speicherplatz!

⇒ die meisten Bildformate komprimieren!

⇒ $2^{24} \approx 16\,Mio.$ Farben

Kompression bei:

GIF Graphics Interchange Format (GIF) (1987-er Jahre) Farbtabelle, d. h. maximal 256 verschiedene Farben aus den insgesamt verfügbaren 16 Mio. sind in einem Bild gleichzeitig verwendbar.

⇒ Für jede Teilfläche sind nur 1 Byte Information (als Indexwert auf die Farbtabelle) notwendig.

⇒ in dieser Quantisierung auf 256 verschiedene Farbwerte steckt der Verlust bei GIF, *nicht* jedoch in der nachfolgenden Kompression. Das Kompressionsverfahren ist eine modifizierte Lauflängenkodierung (verlustfrei).

Sonderfunktion:

- Animated GIF
- Transparenz (Markieren einer Farbe als „volltransparent“)

12.1.2 JPEG

Joint Photographie Experts Group (JPEG) ist ein weiteres, häufig verwendetes Grafikformat.

Hauptziel von JPEG:

- fotorealistische Darstellungen
- ⇒ viel mehr als 256 Farben
- Farbformat: RGB (ohne Alphakanal)

Komprimierung von JPEG:

- Discrete Cosine Transformation (DCT) als Vorstufe. Es handelt sich dabei um eine Mittelwertbildung über 8×8-Pixelblöcke mit einer Abweichung der $4 \times 4/2 \times 2/1 \times 1$-Pixelblöcke von jeweilig enthaltenden größeren Pixelblock (Abweichungen $\hat{=}$ DCT-Koeffizienten).

  „*Im nächsten Schritt wird das Bild in* 8×8 *große Pixelblöcke aufgetrennt. Jeder dieser Blöcke wird der diskreten Cosinus-Transformation unterzogen. Diese ist mit der Fourier-Transformation verwandt und wandelt die räumliche Information der Helligkeitswerte in eine Frequenz-Darstellung um. Die Umwandlung selbst ist verlustlos bis auf Rundungsfehler.*“ – Quelle: `https://www.tecchannel.de/a/jpeg,401190,3`

- Anwendung der Huffman-Kodierung[1] auf die Koeffizienten

  Prinzip der Huffman-Kodierung/Komprimierung:
  Sortierung der Werte nach ihrer Häufigkeit des Vorkommens, z. B.

1:	häufigster Wert
01:	zweithäufigster Wert
001:	dritthäufigster Wert

  Bei den häufigen Werten wird Speicherplatz gespart, bei seltenen Werten braucht man jedoch mehr Speicherplatz.

  ⇒ Einsparpotenzial nur bei stark unterschiedlich häufig vorkommenden Werten!
  ⇒ verlustfreie Komprimierung

  Verlust durch Ausgleich ähnlicher Koeffizienten auf gleiche Werte (Qualitätsfehler bei JPEG)

[1] siehe auch `https://de.wikipedia.org/wiki/Huffman-Kodierung`

- Nachfolgeformat: JPEG2000
  - Unterstützung von RGB**A**
  - keine allzugroße Verbreitung wegen Lizenzgebühren
  - Unterstützung meist nur von kostenpflichtiger Software (z. B. Photoshop, aber nicht GIMP)

12.1.3 PNG

Portable Network Graphics (PNG) ist ein verlustfreies Grafikformat. Es besitzt folgende Eigenschaften:

- mehrere Farbmodelle
  - indiziert (mit Farbtabelle)
  - RGBA
- immer verlustfrei
- geeignet für jegliche Art der Darstellung (künstliche & photorealistisch)
- Problem bei photorealistischen Darstellungen: manchmal sehr großes Dateivolumen

12.2 SVG

Scalable Vector Graphics (SVG) ist ein Vektorgrafikformat, welches sich auch im Internet durchgesetzt hat. Geschichte:

1998 zwei Ansätze für vektorbasiertes Grafikformat im Web:

1. Microsoft & Macromedia: Vector Markup Language (VML)
2. Adobe, IBM, NS, Sun: Precision Graphics Markup Language (PGML)

10/1998 Anforderungen an vektorbasiertes Grafikformat für das Web

09/2001 SVG 1.0 als W3C-Standard

01/2003 SVG 1.1 mit Fehlerbehebung und Modularisierung

04/2005 SVG 1.2, aber außer „SVG 1.2 Tiny“ zurückgezogen

12/2016 Candidate Recommendation für SVG 2.0

Die Anzeige ist bei allen modernen Browsern „nativ“ möglich. Sie kann auch in eine HTML-Seite eingebettet werden.

`MIME`-Type.	`image/svg+xml`
File Extension:	`svg`, `.svgz` (gzip-komprimiert)
Format/Grundstruktur:	*siehe Listing 12.1*

Tabelle 12.2: Eigenschaften von SVG

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" ↙
   ↳ "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
3  <svg xmlns="http://www.w3.org/2000/svg"
4      version="1.1"
5      width ="<Breite (in Pixeln, Millimetern, etc.)>"
6      height="<Höhe (in Pixeln, Millimetern, etc.)>"
7      viewBox="<x> <y> <Breite> <Höhe>">
8
9      <title>Titel der Grafik</title>
10     <desc>Beschreibung</desc>
11     <defs>
12         <!-- Für weitere Metadaten, z.B. CSS, JS, vordefinierte Objekte, ... -->
13     </defs>
14     <!-- eigentlich anzuzeigender Inhalt bestehend aus grafischen Elementen -->
15
16 </svg>
```

Listing 12.1: Grundstruktur einer SVG Datei

`viewBox` optionales Attribut, um angezeigten Bereich des Koordinatensystems von 0 → Breite und 0 → Höhe auf x → x+Breite und y → y + Höhe zu setzen.

Abbildung 12.1 auf Seite 65 verdeutlicht das Verhalten von `viewBox`[2].

12.2.1 Grundelemente

- Grafische Grundelemente werden jeweils über einen eigenen Tag dargestellt
- (Fast) Alle Parameter werden als Attribute angegeben (außer Textinhalt bei `<text>`)
- (Fast) Alle Tags bei SVG sind leere tags (aber Metadaten `<title>` und `<desc>` können bei jedem Tag als Kindelemente angegeben werden)

Linie `<line x1="0" y1="0" x2="100" y2="100" />`

Kreis `<circle cx="<X-Koord>" cy="<Y-Koord.>" r="<Radius>" />`

Ellipse `<ellipse cx="<x>" cy="<y>" rx="<Radius X>" ry="<Radius Y>" />`

Ellipsen liegen zunächst immer lotrecht im Koordinatensystem.

Rechteck `<rect x="<x>" y="<y>" width="Breite" height="Höhe" />`

Optional gibt es noch `rx` und `ry` um die Ecken abzurunden.

Linienzug/Streckenzug `<polyline points="x1,y1 x2,y2"/>`

Das `points` Attribut enthält Paare von x/y-Koordinaten für die zweidimensionalen Punkte. Getrennt durch Komma und/oder Whitespace, d. h. folgendes wäre syntaktisch korrekt: `<polyline points="5.5,3,8 7.5 1,1,1,11,111,1.1" />`

[2] Quelle: `https://www.sarasoueidan.com/blog/svg-coordinate-systems/`

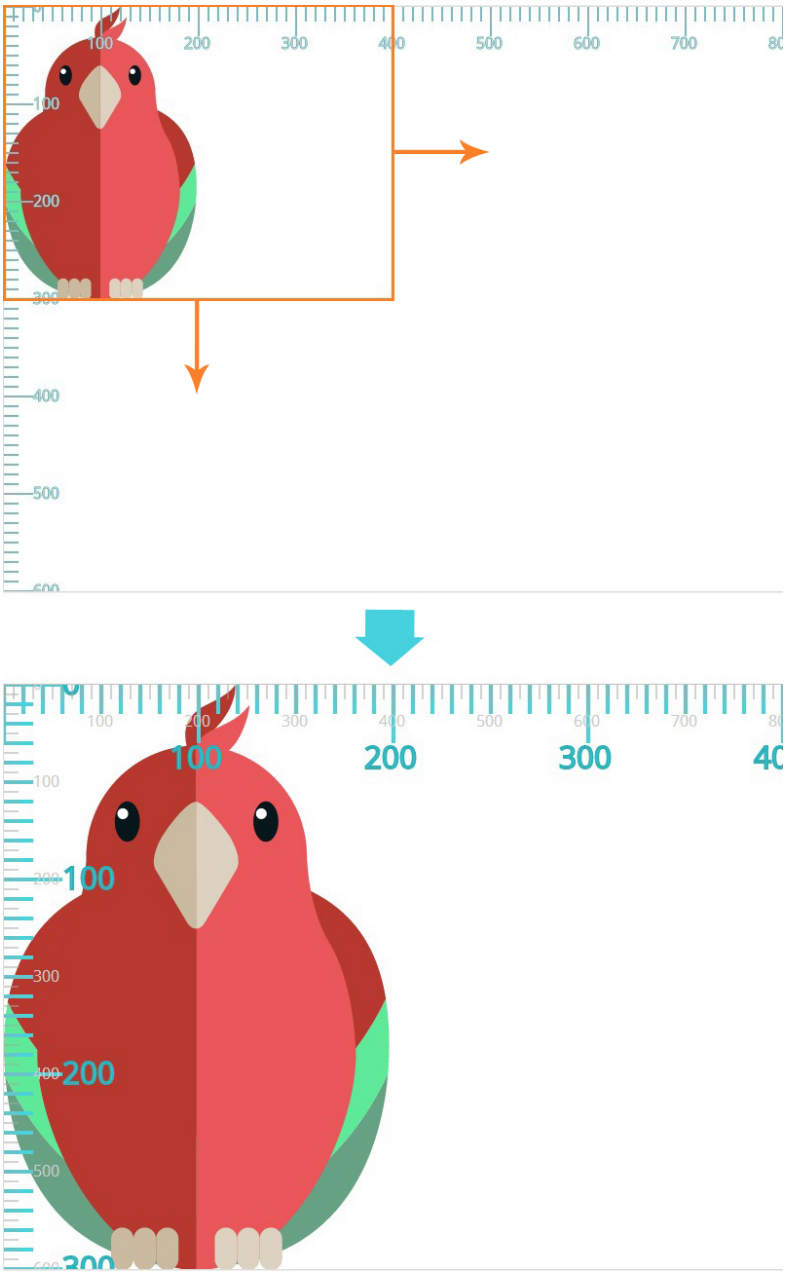


Abbildung 12.1: SVG – `viewBox`

Polygon/geschlossener Streckenzug `<polygon points="x1,y1 x2,y2"/>`

Wie der Linienzug.

quadratische Bézierkurve Drei Punkte beschreiben die Kurve:
Anfangspunkt (AP), Endpunkt (EP) und Stützpunkt (SP)
Siehe Abbildung 12.2 (links) auf Seite 67 und nachfolgend `path Q`.
Unterabschnitt 12.2.3 geht weiter auf Bézierkurven ein.

kubische Bézierkurve Vier Punkte beschreiben die Kurve:
Anfangspunkt (AP), Endpunkt (EP), Stützpunkt 1 (SP1) und Stützpunkt 2 (SP2)
Siehe Abbildung 12.2 (rechts) auf Seite 67 und nachfolgend `path C`

Path „Supertag“ `<path>`: eignet sich für Bézierkurven und (fast) alle anderen Objekte:

Ein Attribut `d` („data“) enthält Zeichenkommandos und Koordinaten.
Kommandos sind einzelne Buchstaben.

- Großbuchstaben: Koordinaten sind absolut
- Kleinbuchstaben: relative Koordinaten zum jeweiligen Bezugspunkt (welcher sich nach der Abarbeitung jedes Kommandos ändert).

`M x y` Move: Bewegung zum Punkt (`x`, `y`), ohne etwas zu zeichnen

`L x y` Line: Zeichne eine Strecke von aktuellen Bezugspunkt zum Punkt (`x`, `y`)

`H y` Horizontal Line: Zeichne eine horizontale Linie zum Punkt (x, $y_{Bezugspunkt}$)

`V y` Vertical Line: Zeichne eine vertikale Linie zum Punkt ($x_{Bezugspunkt}$, y)

`a` arc Kreisbögen (Parameter selbst nachschauen) und Ellipsenbögen

`Q sx sy ex ey` Quadratische Bézierkurve: Bézierkurve vom aktuellem Bezugspunkt über Stützpunkt (`sx`, `sy`) zum Endpunkt (`ex`, `ey`)

`C s1x s1y s2x s2y ex ey` Kubische Bézierkurve: Bézierkurve vom aktuellem Bezugspunkt über die beiden Stützpunkte (`s1x`, `s1y`, `s2x`, `s2y`) zum Endpunkt (`ex`, `ey`)

`T ex ey` ähnlich dem Kommando `Q`, jedoch „smooth“. Als AP wird der vorherige EP gewählt und als erster SP wird der letzte aus der direkt zuvor festgelegten Bézierkurve am AP der neuen Kurve gespiegelt und verwendet.

`S s2x s2y ex ey` ähnlich dem Kommando `C`, jedoch „smooth“. Als AP wird der vorherige EP gewählt und als erster SP wird der letzte aus der direkt zuvor festgelegten Bézierkurve am AP der neuen Kurve gespiegelt und verwendet.

`Z` Schließt den Pfad zum letzten Kontrollpunkt, der durch `M` gesetzt wurde (es gibt keinen Unterschied zwischen Groß- und Kleinschreibung).

Text `<text x="x" y="y">zu schreibender Text</text>`

Stellt eine Textzeile mit linkem oberen Eckpunkt (`x`, `y`) dar.

Gruppierung `<g><!-- andere Tags welche gruppiert werden sollen --></g>`

Sinnvoll für Universalattribute

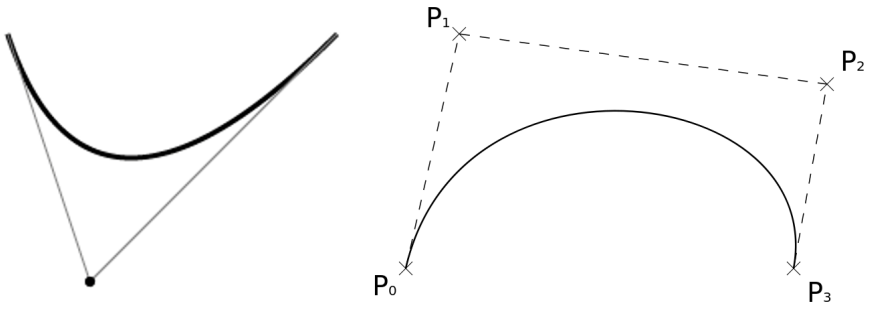


Abbildung 12.2: Quadratische und kubische Bézierkurve

12.2.2 Universalattribute

style für CSS-Angaben. Es gibt u.a. folgende CSS-Properties:

stroke	Strichfarbe
fill	Füllfarbe
opacity	Deckungsgrad (zwischen 0 und 1)
stroke-opacity	Deckungsgrad für Strich
fill-opacity	Deckungsgrad für Füllfläche

Statt über CSS-Properties gibt es auch die Möglichkeit, Darstellungsattribute direkt anzugeben, z. B.

```
<line x1="0" x2="5" y1="3" y2="27" stroke="red" />
```

Achtung mit CSS!

CSS-Angaben überschreiben die Werte der Darstellungsattribute!

Hinweis zu SVG 2

Bei SVG 2 werden auch die Geometrie-Attribute `x, y, x1, x2, y1, y2,cx, cy, rx, ry, r,...` zu Darstellungsattributen „promoted“/„befördert“.

Damit sind darauf z. B. auch CSS-Animationen anwendbar.
Jedoch sorgt `* {x:0;}` dafür, dass alle Rechtecke an die X-Koordinate `0` positioniert werden (unabhängig vom Wert des Darstellungsattributs `x` im `<rect/>`.

transform für geometrische Abbildungen

transform="translate(dx dy)" Verschiebung um (`dx`, `dy`) (`dy` kann weggelassen werden, dann wird nur in X-Richtung verschoben)

transform="scale(sx sy)" Skalierung um Faktor (`sx`, `sy`) (wird `sy` weggelassen, so ist `sy = sx`).

Mithilfe von `scale` kann man zudem Elemente spiegeln:

`scale(-1,-1)`	Punktspiegelung
`scale(1,-1)`	Achsenspiegelung x-Achse
`scale(-1,1)`	Achsenspiegelung y-Achse

`transform="rotate(angle cx cy)"` Rotation um Winkel `angle` (Grad, nicht Radianten) um den optionalen Punkt (`cx, cy`)

`transform="skewX(angle)"` Scherung horizontal um `angle` (Grad, nicht Radianten)

`transform="skewY(angle)"` Scherung vertikal um `angle` (Grad, nicht Radianten)

Kombinierte Abbildung: `transform="scale(3 7) translate(50 10)"`
⇒ erst Verschiebung, dann Skalierung!

Matrixtransformation für beliebige affine Abbildungen: `matrix(a b c d e f)`

$$\begin{pmatrix} x_{transf} \\ y_{transf} \\ 1 \end{pmatrix} = \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{orig} \\ y_{orig} \\ 1 \end{pmatrix}$$

Beispiele

`scale(sx sy)` als Matrixtransformation:

$$\begin{pmatrix} x_{orig} \cdot sx \\ y_{orig} \cdot sy \\ 1 \end{pmatrix} = \begin{pmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{orig} \\ y_{orig} \\ 1 \end{pmatrix}$$

`translate(dx dy)` als Matrixtransformation:

$$\begin{pmatrix} x_{orig} + dx \\ y_{orig} + dy \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{orig} \\ y_{orig} \\ 1 \end{pmatrix}$$

12.2.3 Bézierkurve

Konstruktion mithilfe des Algorithmus von de Casteljau[3].

Wie in Abbildung 12.3 gezeigt wird, liegt O auf der Bézierkurve. Die Bézierkurve $A-B-C$ lässt sich als Konkatenation der beiden Bézierkurven $A-P-O$ und $O-R-C$ darstellen. Dadurch ist eine rekursive Anwendung möglich! Dies wird soweit ausgeführt, bis die Auflösung des Ausgabegerätes erreicht ist.

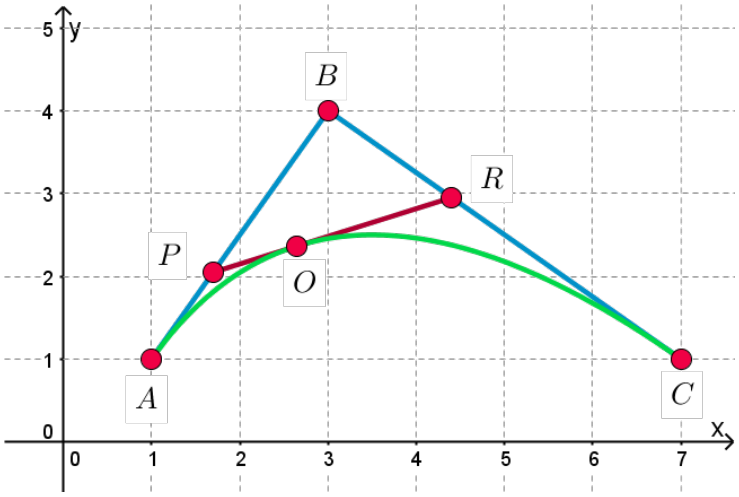


Abbildung 12.3: Erzeugung einer quadratischen Bézierkurve

[3]siehe auch `https://de.wikipedia.org/wiki/De-Casteljau-Algorithmus`

13 Übungsklausur – 2016

13.1 Aufgabe 1 (18 Punkte – 2/6/10)

a) Verweise werden im WWW in einer besonderen Form realisiert. Wie heißt die technische Realisierungsform von Verweisen im WWW und was ist daran anderes als bei den meisten Verweisen in unserem täglichen Leben?

Lösung
Die Realisierungsform von Verweisen heißt „Hyperlink“.
Das besondere ist die automatische Verweisauflösung.

b) Zwei weitere Eigenschaften dieser Verweise (neben der in **a.)** gefragten) machen den Erfolg des WWW aus. Um welche Eigenschaften handelt es sich (mit Erläuterung)? Waren beide Eigenschaften bereits vom Erfinder des Webs, Tim Berners-Lee, geplant? Belegen und begründen Sie ihre Antwort!

Lösung
Es handelt sich um die Lokations- und Medienunabhängigkeit.

Lokationsunabhängigkeit
Unabhängigkeit von der Lokation der Information und der des Suchenden. Dies war von Anfang an geplant, da auch der Tag für Hyperlinks (`<a>`) ab der ersten HTML-Version existiert.

Medienunabhängigkeit
Informationen können in verschiedenen Medienformaten (Bilder, Video, Text, ...) vorliegen. Dies war nicht geplant, da

- das `T` in HTTP für „Text“ steht
- das `T` in der Sprache HTML für „Text“ steht
- die erste Version von `HTTP` keinen `Content-Type`-Header besaß

Vergleiche auch Unterabschnitt 3.2.2 zu den Eigenschaften und Abschnitt 6.3 für die Geschichte von HTML.

c) Relativ und absolut sind zwei Begriffe, welche im Zusammenhang mit Verweisen häufig verwendet werden. Erläutern Sie, was relativ und absolut bei Verweisen bedeutet, in welchen Fällen Verweise relativ und absolut eingesetzt werden und was die jeweilige Form für Vorteile bietet!

Lösung

Absolute Verweise haben ein festes Ziel. Dies hat den Vorteil, dass die angegebene URL immer auf das gleiche Ziel verweist, auch wenn das Dokument verschoben wird. Dies wird zum Beispiel verwendet, wenn man auf die eigene Webseite verlinken möchte, aber unter einem anderen Protokoll (FTP, HTTPS, ...).

Relative Verweise können mithilfe der vollständigen URL des enthaltenden Dokuments zu absoluten Verweisen gemacht werden. Vorteil ist u. a. die kürzere Länge und die „Umzugsfähigkeit“ der Web-Präsenz auf einen anderen Server (host-relativ) oder in ein anderes Verzeichnis (pfad-relativ).

Man unterscheidet zwischen:

- Protokoll-relative URL (falls HTTP/HTTPS automatisch gewählt werden soll)
- Host-relative (Pfad absolute) URL (für zentrale Dokumente, z. B. den CSS-Ordner).
- Pfad-relative URL (für den Verweis auf ein übergeordnetes Verzeichnis)

Siehe auch Unterabschnitt 7.2.1 auf Seite 30.

13.2 Aufgabe 2 (22 Punkte – 10/12)

a) Zur Repräsentation der lieferbaren Produkttypen („Produktkatalog“) eines Schlossherstellers soll eine lineare Darstellung als Text mittels XML Verwendung finden. Entwerfen und beschreiben Sie ein entsprechendes Dateiformat (inklusive einer kurzen XML-Beispieldatei), welches folgende Daten enthält:

- Eindeutige Kennung (numerisch)
- Art des Schlüsselsystems (Einzelschloss, Systemschloss)
- Anzahl möglicher Schlösser
- Sicherheitsklasse (gekennzeichnet mit A für einfache Sicherheit bis E für hohe Sicherheit)
- Verweis auf eine Abbildung

Lösung

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE schloesser SYSTEM "/URL/zur/DTD.dtd">
<schloesser>
    <schloss id="123">
        <system>Einzelschloss</system>
        <anzahl-moeglichkeiten>999</anzahl-moeglichkeiten>
        <sicherheit>A</sicherheit>
        <abbildung href="http://meine.url/grafik.png">
            Abbildung des Schlosses
        </abbildung>
    </schloss>
    <schloss id="124">
        <system>Systemschloss</system>
        <anzahl-moeglichkeiten>9999</anzahl-moeglichkeiten>
        <sicherheit>C</sicherheit>
        <abbildung href="http://meine.url/grafik-2.png">
            Abbildung des Schlosses Nr. 124
        </abbildung>
    </schloss>
</schloesser>
```

Listing 13.1: Schlosshersteller – XML

Hinweis

Bei dieser Aufgabe fehlt die Beschreibung noch. Es muss keine DTD sein, eine schriftliche Beschreibung reicht. Den `DOCTYPE` sollte man jedoch nicht vergessen!

b) Schreiben Sie eine XSLT, welche eine entsprechend Ihrer obigen Grammatik geschriebene Datei in eine von jedem Web-Browser darstellbare HTML-Datei wandelt. Dabei soll die Darstellung als eine Tabelle der obigen Felder erfolgen.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ↙
       ↳ xmlns="http://www.w3.org/1999/xhtml">
3      <xsl:output method="html" encoding="UTF-8" doctype-public="html"/>
4      <xsl:template match="/">
5          <html>
6              <head>
7                  <title>Titel</title>
8              </head>
9              <body>
10                 <table>
11                     <tr>
12                         <th>Kennung</th>
13                         <th>System</th>
14                         <th>Anzahl möglicher Schlösser</th>
15                         <th>Sicherheitsklasse</th>
16                         <th>Verweis (Abbildung)</th>
17                     </tr>
18                     <xsl:for-each select="/schloesser/schloss">
19                         <tr>
20                             <td><xsl:value-of select="@id" /></td>
21                             <td><xsl:value-of select="system"/></td>
22                             <td><xsl:value-of ↙
       ↳ select="anzahl-moeglichkeiten"/></td>
23                             <td><xsl:value-of select="sicherheit"/></td>
24                             <td>
25                                 <a>
26                                     <xsl:attribute name="href">
27                                         <xsl:value-of select="abbildung/@href"/>
28                                     </xsl:attribute>
29                                     <xsl:value-of select="abbildung"/>
30                                 </a>
31                             </td>
32                         </tr>
33                     </xsl:for-each>
34                 </table>
35             </body>
36         </html>
37     </xsl:template>
38 </xsl:stylesheet>
39
```

Listing 13.2: XSLT – Schlosshersteller

13.3 Aufgabe 3 (22 Punkte – 12/10)

a) Gegeben seien die folgenden quadratischen Bezier-Kurven über die folgenden Koordinaten (`x`, `y`) von Anfangspunkt (AP), Stützpunkt (SP) und Endpunkt (EP):

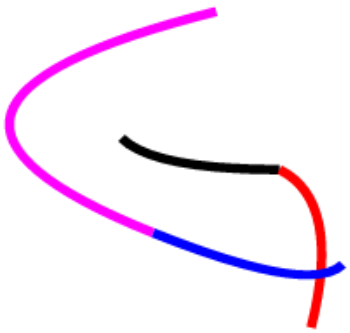
AP	SP	EP
2,4	3,5	7,5
7,5	8,6	8,10
9,8	8,9	3,7
3,7	-7,3	5,0

Aus diesen Kurven wird ein Kurvenzug gebildet. Geben Sie an, an welchen Punkten (AP, SP und EP) der Kurvenzug rund ist und an welchen er „nicht rund“ ist, also Ecken aufweist! Begründen Sie jeweils Ihre Antwort!

Lösung

Der Kurvenzug ist am Punkt:

- `7,5` eckig, da die Steigung der ersten und zweiten Kurve an dem Punkt nicht übereinstimmt (`0` $\Leftrightarrow$ `1`)
- `8,10` und `9,8` nicht stetig, da sich die Koordinaten unterscheiden.
  Somit ist er dort auch nicht rund.
- `3,7` rund, da die Steigung der dritten und vierten Kurve gleich sind ($\frac{2}{5} \Leftrightarrow \frac{4}{10} = \frac{2}{5}$)



Hinweis

Eine alternative Aufgabe könnte sein, dass Punkte vorgegeben sind und man diese erweitern muss, sodass sich eine Rundung ergibt!

b) Schreiben Sie eine vollständige SVG-Datei, welche als Inhalt die in der vorigen Teilaufgabe beschriebenen Kurvenzüge hat!

Lösung

```
<svg width="200" height="150">
    <path stroke-width="3" stroke="black" d="M2,4 Q3,5 7,5 M7,5 Q9,6 8,10 ↙
↳ M9,8 Q8,9 3,7 M3,7 Q-7,3 5,0" fill="transparent"/>
</svg>
```

Listing 13.3: Übungsklausur – SVG

Wenn man die oben stehende Grafik erhalten möchte:

```
<svg width="200" height="150" viewBox="-10 -10 100 150">
  <path stroke-width="3" stroke="black" d="M20,40 Q30,50 70,50" ↙
↳ fill="transparent"/>
   <path stroke-width="3" stroke="red" d="M70,50 Q90,60 80,100" ↙
↳ fill="transparent"/>
   <path stroke-width="3" stroke="blue" d="M90,80 Q80,90 30,70" ↙
↳ fill="transparent"/>
   <path stroke-width="3" stroke="#ff00ff" d="M30,70 Q-70,30 50,0" ↙
↳ fill="transparent"/>
</svg>
```

Listing 13.4: Übungsklausur – Beispiel SVG mit Farbe

Tipp

Da absolute Werte gegeben sind, ist es am sinnvollsten diese auch zu verwenden, d. h. Großbuchstaben für die Bézierkurve. Der erste der drei Werte ist der Anfangspunkt. Dieser wird durch ein „move“ (`M`) erreicht und durch eine quadratische Bézierkurve über den Stützpunkt und Endpunkt mit `Q` fortgesetzt.

13.4 Aufgabe 4 (16 Punkte – 10/6)

a) Die folgende HTML-Seite enthält mehrere syntaktische Fehler und entspricht damit in einigen Punkten nicht dem Standard. Ändern Sie den Code so ab, dass er fehlerfrei und XHTML1.0-konform wird. Geben Sie bei jedem Fehler eine kurze Begründung an!

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns=http://www.w3.org/1999/xhtml xml:lang="en" land="en">
    <kopf>
        <link rel=stylesheet type=text/css href="formate.css">
    </kopf>
    <BODY>
        <ol>
            <li>Dies ist das erste Element einer ungeordneten Liste.</li>
```

```
11                <li>Und hier folgt das zweite Element.
12                <li>Das dritte Element enth&auml:lt eine weitere Liste:
13                    <ul>
14                        <li>bla</li>
15                        <li>fasel</li>
16                        <li>blubber</li>
17                    </ol>
18                    Damit ist dann die innere Liste und auch das dritte ↙
    ↳ Element beendet.
19                </LI>
20            <ul>
21            Das ist fast das Ende.
22            <br>
23            Das ist das Ende.
24        </BODY>
25    </html>
```

Listing 13.5: HTML - Klausuraufgabe

Lösung

Zeile	Fehler und Erläuterung
4	Der Wert des Attributs `xmlns` steht nicht in Anführungszeichen. Attributwerte müssen in XML in doppelten (nicht einfachen!) Anführungszeichen stehen.
4	Das Attribut `land` müsste `lang` heißen.
5 + 7	Der Tag müsste `<head>` sein, nicht `<kopf>`
5	Es fehlt das Pflicht-Tag `<title>Mein Titel</title>`
6	Die Attributwerte von `rel` und `type` müssen in doppelten Anführungszeichen stehen.
6	Beim Tag `<link ...>` fehlt das schließende Tag. Es müsste sein: `<link ... />`
8	`<BODY>` müsste kleingeschrieben sein, da in XHTML Tags kleingeschrieben sein müssen!
9	~~Da es in Zeile 10 heißt, es wäre ein Element einer ungeordneten Liste, müsste der Tag `<ul>` und nicht `<ol>` sein, da „ul“ für „unordered list“ steht.~~ ⇒ semantischer Fehler!
11	Das schließende Tag (`</li>`) fehlt, denn bevor ein neues `<li>` geöffnet werden kann (siehe Zeile 11), muss das vorherige Tag geschlossen werden.
12	Die HTML-Entität muss mit einem Semikolon (`;`) beendet werden, denn mit einem Doppelpunkt würde nicht ein ä dargestellt werden, sondern `&auml:`
17	Das öffnende Tag `<ul>` wird nicht durch ein entsprechendes schließendes Tag `</ul>` geschlossen. Das Tag `</ol>` schließt nicht `<ul>`!
19	Das schließende Tag `</LI>` muss kleingeschrieben werden, da dies vom XHTML-Standard vorgeschrieben ist!
20	Es wird eine ungeordnete Liste geöffnet, bevor die geordnete geschlossen ist. Es müsste sich um ein `</ol>` handeln, bzw. um ein `</ul>`, wenn man den Fehler in Zeile 9 beachtet.
22	Der Zeilenumbruch muss in XHTML geschlossen werden! Es müsste sein: ` `
24	Das schließende Body-Tag muss klein geschrieben werden (`</body>`)!

Hinweis

Gesucht sind syntaktische Fehler! Semantische Fehler wie in Zeile 9 sind **nicht** gesucht!

b) Skizzieren Sie, wie die Darstellung des (korrigierten) HTML-Codes in einem gängigen Webbrowser aussehen könnte.

Lösung

- Dies ist das erste Element einer ungeordneten Liste.
- Und hier folgt das zweite Element.
- Das dritte Element enthält eine weitere Liste:
  - bla
  - fasel
  - blubber

  Damit ist dann die innere Liste und auch das dritte Element beendet.

Das ist fast das Ende.
Das ist das Ende.

13.5 Aufgabe 5 (22 Punkte – 14/8)

Gegeben ist folgendes Stylesheet in der Datei `mystyle.css`:

```
1  body { color:#000000; }
2  em { color:#0000ff; }
3  p em { color:#ffffff; }
4  em.dieseklasse { color:#ff0000; }
5  .dieseklasse { color:#ff00ff; }
```

Listing 13.6: Übungsklausur – CSS

Ferner hat der Benutzer seinen Browser angewiesen, das folgende Stylesheet zu verwenden:

```
1  em { color:#00ff00; }
2  h1 { color:#00ff00 !important; }
```

Listing 13.7: Übungsklausur – CSS – Browser-Stylesheet

a) Geben Sie bei jeder Zeile, welche Text zur Ausgabe enthält, im Quellcode folgender HTML-Datei an, in welcher Farbe der entsprechende Text bei der Darstellung im Browser des Benutzers angezeigt wird! Begründen Sie jeweils kurz ihre Angabe!

Achtung

Der Dozent erwartet, dass keine Hexcodes oder `rgb(r,g,b)`-Farben in dieser Aufgabe angegeben werden. Somit muss man die Hexcodes in Wörtern hinschreiben!

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>CSS-Test</title>     <!-- das wird nicht angezeigt, Dummkopf! -->
5          <link href="mystyle.css" rel="stylesheet" type="text/css" />
6      </head>
7      <body>
8          <h1 style="color: blue">Erster Teil</h1>            <!-- grün -->
9          <p class="dieseklasse" >
10             Das hier sieht wie aus?                         <!-- pink -->
11             <em>Dieser Text hat welche Farbe?</em>          <!-- weiß -->
12             <em class="dieseklasse" > Und dieser hier?</em> <!-- rot  -->
13         </p>
14         <h1 class="dieseklasse" > Zweiter Teil</h1>         <!-- grün -->
15         <ul>
16             <li>Welche Farbe wird hier verwendet?</li>      <!-- schwarz -->
17             <li><em>Und hier?</em></li>                     <!-- blau -->
18         </ul>
19     </body>
20 </html>
```

Listing 13.8: Übungsklausur – HTML mit CSS

Erklärung:

7 Die `!important`-Angabe überschreibt das `style`-Tag!

15 Es wurde keine Farbe angegeben, deshalb wird dies vom Browser festgelegt.

Die folgende Grafik veranschaulicht den HTML Code. Der weiße Text wurde markiert, damit er sichtbar wird.

b) CSS-Regeln und Regelteile können grundsätzlich aus verschiedenen Quellen stammen. Aus welchen? Erläutern Sie die verschiedenen Verwendungsarten!

Lösung

Es gibt:

- `author`: `<link>` und `<style>` im `<head>` und `style` als Attribut. Die Regeln stammen vom Webseiten-Autor.
- `user`: Vom Nutzer festgelegte Regeln, z. B. für sehbehinderte angepasstes Design.
- `user-agent`: Vom Browser festgelegtes Design („default“)

Siehe auch Unterabschnitt 8.3.2 auf Seite 41 für CSS Quellen.

13.6 Weitere mögliche Aufgaben

Die folgenden Aufgaben könnten Klausuraufgaben sein.

1. Nennen und erläutern zwei große Gruppen von Grafikformaten.
2. Zu welcher Gruppe gehört SVG? Nennen sie die wichtigsten Unterschiede zu JPEG.
3. Setzen Sie die gegebene Liste in gültigen HTML-Code um.
4. Neben den zwei Listentypen aus *3*, gibt es einen dritten? Welchen? Wo wird er verwendet? Was ist der Unterschied?
5. Erläutern sie die in den CSS Regeln verwendete Selektoren.

14 Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
AP	Anfangspunkt 65
ASCII	American Standard Code for Information Interchange
CSS	Cascading Style Sheet 32
DCT	Discrete Cosine Transformation 62
DTD	Document Type Definition 45
DNS	Domain Name System
DOM	Document Object Model 14
EP	Endpunkt 65
FQDN	Fully Qualified Domain Name 10
GIF	Graphics Interchange Format 61
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol II
HTTPS	Hyper Text Transfer Protocol Secure
HTML	Hyper Text Markup Language
JPEG	Joint Photographie Experts Group 62
MIME-Type	Multimedia Internet Mail Extension Type 7
PGML	Precision Graphics Markup Language 63
PNG	Portable Network Graphics 63
RDB	Relationale Datenbank 42
RGB	Rot-Grün-Blau
slmt	Sub-Level-Media-Type 7
SVG	Scalable Vector Graphics 63
SGML	Standard Generalized Markup Language
SP	Stützpunkt 65
tlmt	Top-Level-Media-Type 7
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Ressource Identifier 29
URL	Uniform Ressource Locator 29
VML	Vector Markup Language 63
WWW	World Wide Web 3
WYSIWYG	What you see is what you get
XML	Extensible Markup Language 45
XHTML	Extensible HyperText Markup Language
XPath	XML Path Language 49
XSL	Extensible Stylesheet Language 49
XSD	XML Scheme Definition 45
XSLT	XSL Transformation 49
W3C	World Wide Web Consortium

Stichwortverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingsverzeichnis