Version 2024c

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Grundlagen der Web-Programmierung

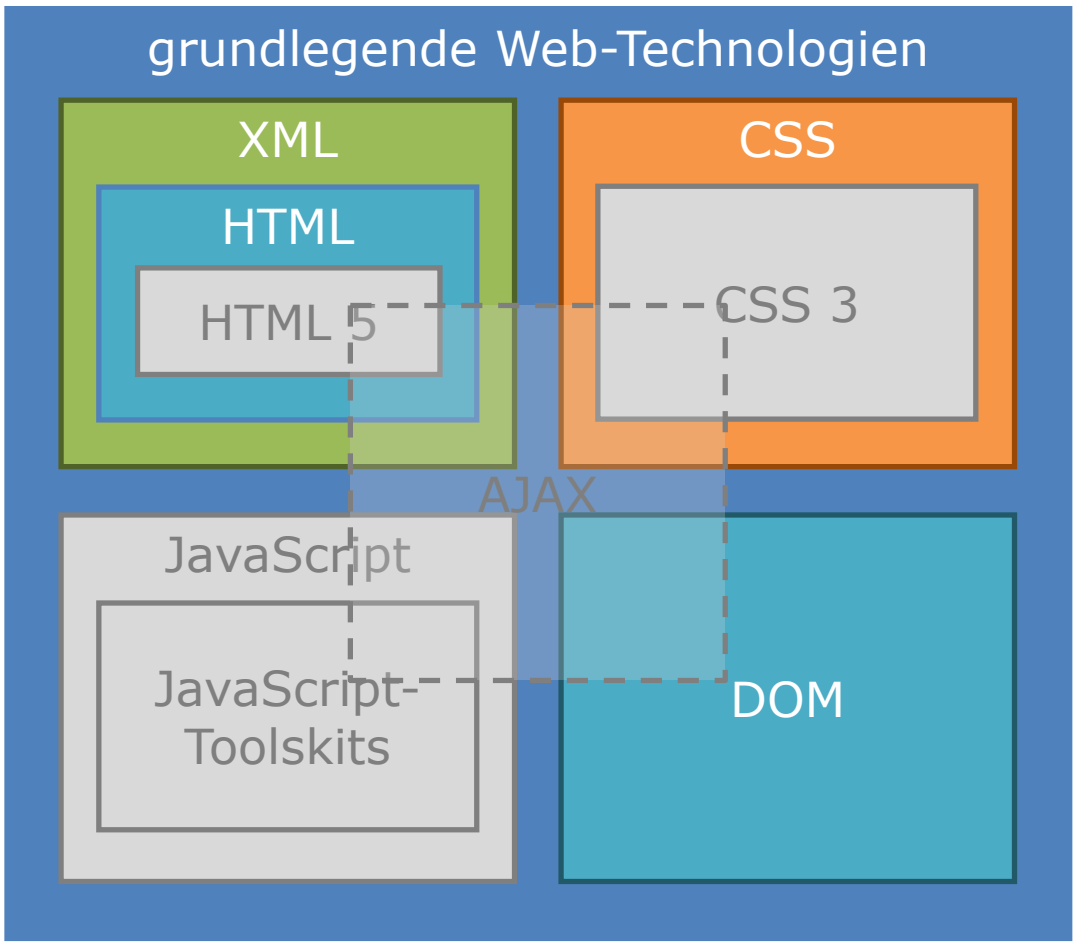
W3C-Standards

Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-sachsen.de

Aufbau der Lehrveranstaltung

Progressive Web Apps

Search Engine Optimization



Inhalte von Web-Seiten

- Bilder
- SVG
- Audio/Video

Proprietäre Standards

World Wide Web Consortium (W3C)

- Gremium zur Standardisierung der Techniken im World Wide Web
  - keine zwischenstaatlich anerkannte Organisation: nicht berechtigt ISO-Normen festzulegen
  - dennoch bilden W3C-Standards oft Basis der ISO-Normen
- organisiert als Mitgliedsorganisation
  (finanziert über Mitglieder; Vorsitz: Tim Berners-Lee)
- Aufgaben
  - Interessengruppen ansprechen
  - Beziehung zur regionalen Politik und Wirtschaft fördern
  - regionale W3C-Mitglieder unterstützen
  - Rückmeldungen geben, über regionale Thematiken
  - regionale Akzeptanz von W3C-Standards fördern
  - Verbreitung von Übersetzungen der W3C-Empfehlungen

Entwicklung von W3C-Standards (1/2)

- Working Draft
  - liegt Öffentlichkeit, W3C-Mitgliedern und allen anderen Interessierten zur Kommentierung vor
  - i.d.R. werden mehrere Arbeitsentwürfe entwickelt (nicht jeder schafft es aber bis zur Empfehlung)
- Last Call Working Draft - letzter geplanter Arbeitsentwurf

Entwicklung von W3C-Standards (2/2)

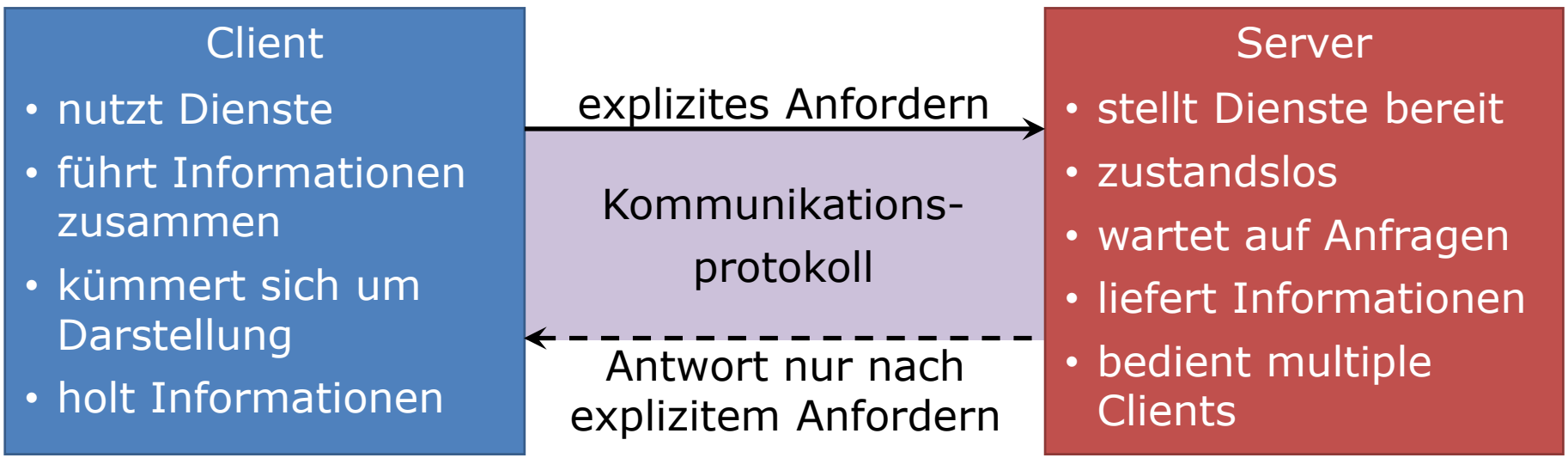
- Candidate Recommendation
  - schließt die eigentliche Arbeit am Dokument größtenteils ab
  - alle technischen Anforderungen sind im Dokument integriert
  - bereits erste Implementationen möglich und gefordert
  - Erfahrungen aus den Implementationen fließen in weitere Dokumente
- Proposed Recommendation
  - durch ersten Implementierungen ergänztes Dokument
  - letzte Stufe im Entwicklungsprozess zur Empfehlung
  - nach Zustimmung der W3C-Mitglieder und des Vorsitzenden folgt Status einer Empfehlung

WICHTIGE* W3C-STANDARDS

* wichtig im Sinne des Moduls 3MI-GLWEB-20

- Hypertext Markup Language (HTML) (seit 1992)
- Cascading Style Sheets (CSS) (1993-1995; W3C seit 1996)
- Portable Network Graphics (PNG) (1994; W3C seit 1996)
- Extensible Markup Language (XML) (seit 1998)
  - XML Schema (XSD)
  - Extensible Stylesheet Language (XSL)
  - XSL Transformation (XSLT)
- Extensible Hypertext Markup Language (XHTML; HTML 4.01) (seit 1999)
- Scalable Vector Graphics (SVG) (seit 2001)

Grundlegendes Kommunikationsparadigma: Client-Server-Kommunikation



In a Nutshell: Extensible Markup Language (XML)

Extensible Markup Language (XML)

- Auszeichnungssprache zur Beschreibung hierarchischer Strukturen
- XML-Dokumente
  - *sollen* aus UTF-8-Zeichen bestehen
  - *können* einer DTD unterliegen
  - sind in Elemente organisiert
  - sind wohlgeformt, wenn alle Regeln eingehalten werden
  - *können* auf eine Grammatik verweisen
  - gültig (valid), wenn wohlgeformt *und* grammatisch korrekt

XML – Elemente

- wichtigste Struktureinheit eines XML-Dokumentes;
  Träger der Information
- Element kann Text und weitere Elemente enthalten
- bildet Knoten des Strukturbaumes eines XML-Dokumentes
- Name ist in DTD-freien XML-Dokumenten frei wählbar, ansonsten
  - muss dieser im DTD deklariert sein und
  - das Element muss sich in einer zugelassenen Position innerhalb des Strukturbaumes gemäß DTD befinden

XML – Wohlgeformtheit

- alle XML-Regeln müssen einhalten werden
- genau ein Wurzelelement
- alle Elemente mit Inhalt besitzen einen Beginn- und einen Endauszeichner
  → Elemente ohne Inhalt können aus nur einem Auszeichner bestehen, der mit /> abschließt
- Beginn- und End-Auszeichner sind paarig, ebenentreu und beachten Groß-/Kleinschreibung
- Elemente dürfen nicht mehrere Attribute mit demselben Namen besitzen
- Attributeigenschaften müssen in Anführungszeichen stehen

XML – Grammatik

- einerseits durch Struktur der DTD definiert
- andererseits in XML Schema (XSD) möglich
  - ähnlich DTD, aber selbst XML-konform und mehr Datentypen unterstützend
  - beschreibt XML-Schema-Instanzen (einzelne Dateien) und Instanzengruppen (Typen von Dateien)
  - Basisdatentypen: xs:string, xs:decimal, xs:integer, xs:float, xs:boolean, xs:date, xs:time
  - Listen und Unions (bestehend aus atomaren Elementen und Listen)

XSD-Beispiel

```
<xs:complexType name="student">
   <xs:sequence>
      <xs:element name="nachname" type="xs:string"/>
      <xs:element name="vorname" type="xs:string"/>
      <xs:element name="geburtstag" type="xs:date"/>
      <xs:element name="semester" type="xs:integer"
                  minInclusive="1" maxInclusive="20"/>
      <xs:element name="studiengang" type="xs:integer"
                  minInclusive="1"/>
      <xs:element name="kommentar" type="xs:string"
                  minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="matrikel" type="xs:integer"/>
</xs:complexType>
```

XML-Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:studenten="file://D:/Verwaltung/studentenschema.xsd">
<studenten>
   <student matrikel="1">
      <nachname>Mustermann</nachname>
      <vorname>Peter</vorname>
      <geburtstag>1996-07-18</geburtstag>
      <semester>2</semester>
      <studiengang>4</studiengang>
   </student>
   <student matrikel="7">
      <nachname>Doe</nachname>
      <vorname>Jane</vorname>
      <geburtstag>1998-02-21</geburtstag>
      <semester>2</semester>
      <studiengang>3</studiengang>
      <kommentar>Austauschstudentin</kommentar>
      <kommentar>keine Mensa-Berechtigung</kommentar>
   </student>
   <!-- u.s.w. -->
</studenten>
```

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Medien und Inhalte mit XML

Medienverarbeitung mit XML

HTML und Javascript sind Beispiele für formale Sprachen, die Medien beschreiben und automatisch verarbeitet (bzw. geprüft) werden können.

→ XML ist eine Metasprache, um solche Sprachen und ihre Interaktion zu formalisieren.

Beispiele:

- HTML beschreibt Text
- SMIL dient zur Synchronisation von Audio, Video mit Text, Bildern
- MusicXML beschreibt Notensatz
- MathML erlaubt den Austausch mathematischer Formeln
- QTI legt Prüfungsfragetechniken fest
- SVG beschreibt Graphiken
- Browser unterstützen Webseiten mit HTML-, SVG- und MathML-Inhalten

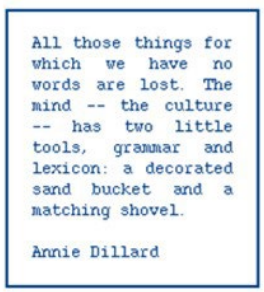
XML und SGML

- XML basiert auf ISO 8879: Standard Generalized Markup Language (SGML) for Information Processing, Text and Office Systems
- XML kann für Langzeitarchivierung geeignet sein, wenn die bedeutungstragenden Elemente der Medien erhalten bleiben (auch, wenn sich Darstellungstechniken verändern)

Wo wird SGML verwendet?

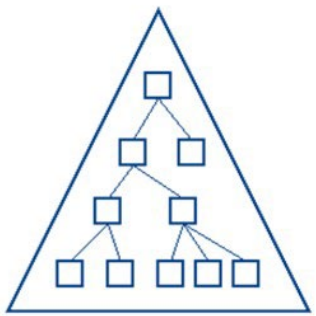
- Euro-Tunnel: 7 GB Daten in SGML
- Airbus: technische Dokumentation von Flugzeugen
- US-DoD verlangt(e) SGML für alle Produktdokumentationen
- Zeitungsverlage verwenden SGML zur Archivierung

Medien in Dokumenten



Inhalt

Semantik



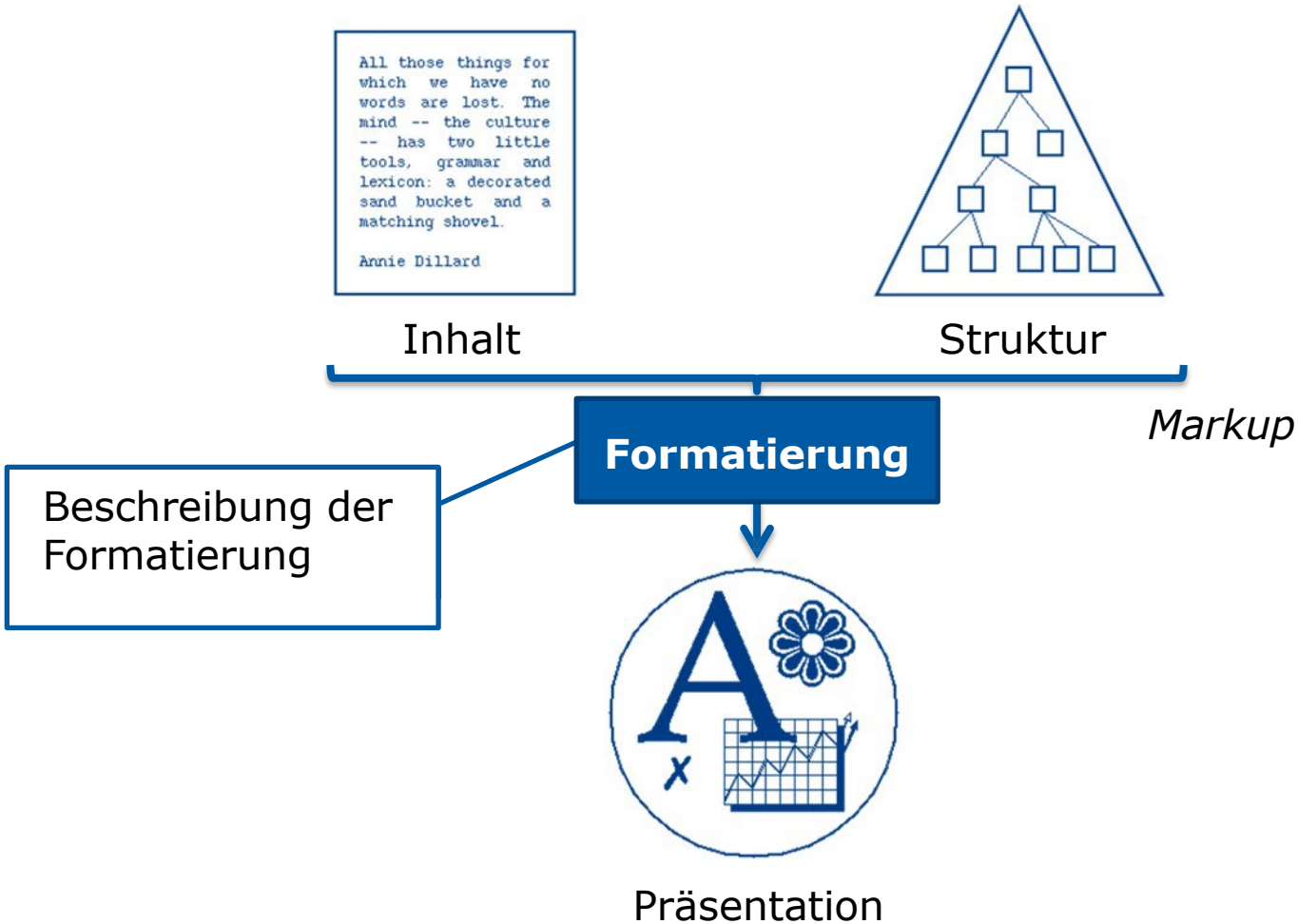
Struktur

Syntax

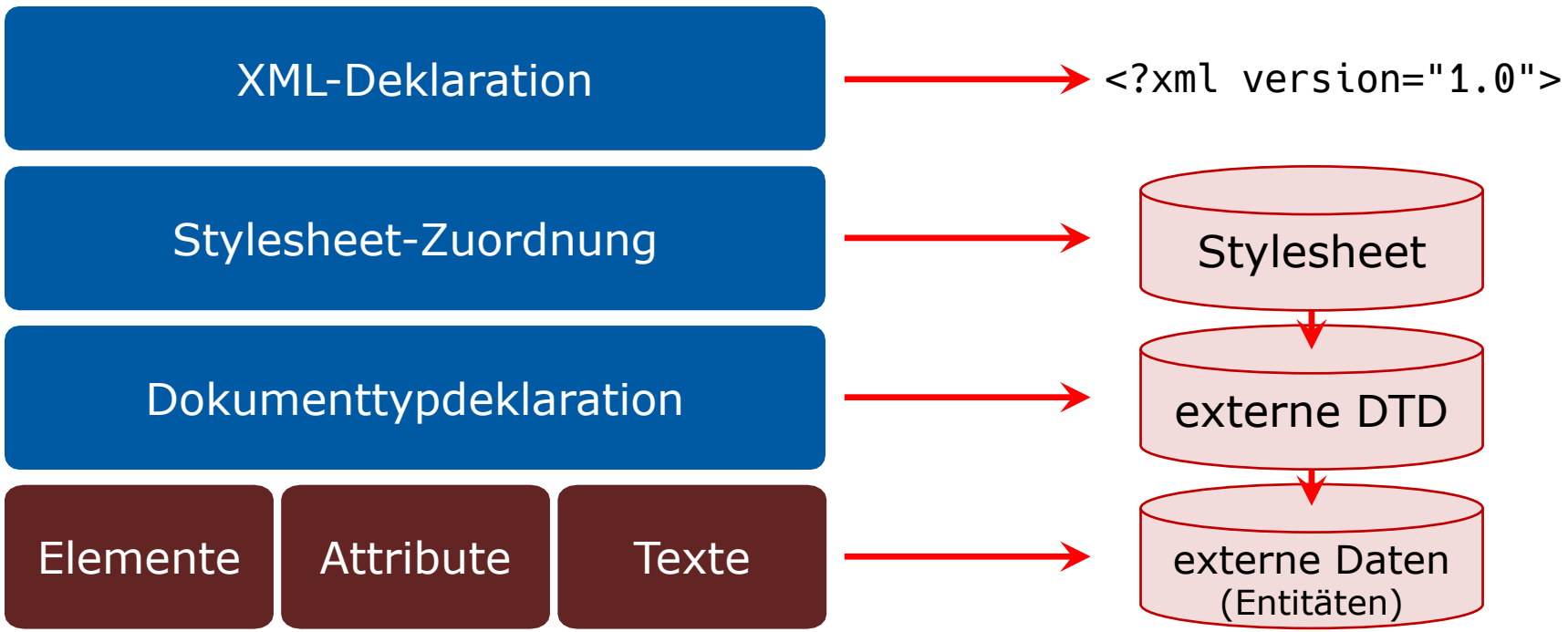
Präsentation

Lexikalische Information

Dokumente mit XML



Vollständiger Aufbau eines XML-Dokuments



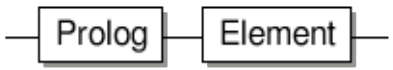
Syntaxdiagramme für wohlgeformtes XML

- XML wurde durch das W3C (www.w3.org/TR/xml) mittels 89 EBNF Regeln beschrieben
- Syntaxdiagramme geben diese Regeln grafisch wieder
  - Es gibt ein (nichtterminales) Startsymbol
  - Symbole in Kreisen sind terminal
  - Symbole in (abgerundeten) Rechtecken sind nichtterminal und beziehen sich auf andere Symbole
  - Ausdruck ist „wohlgeformt“, wenn ein Durchlauf durch die Diagramme möglich ist

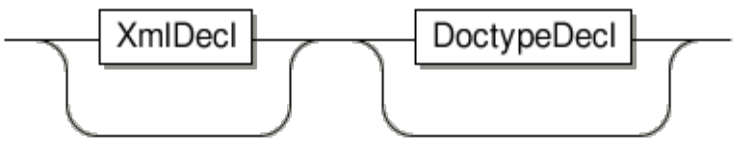
Hinweis: die folgenden Diagramme sind vereinfacht; vollständige Diagramme erzeugt z.B. http://j-algo.binaervarianz.de/

XML-Syntaxdiagramme (1/2)

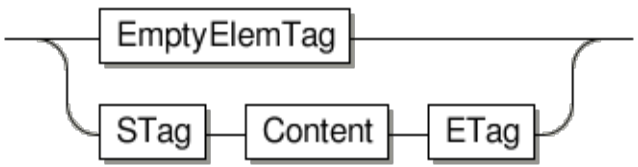
Document



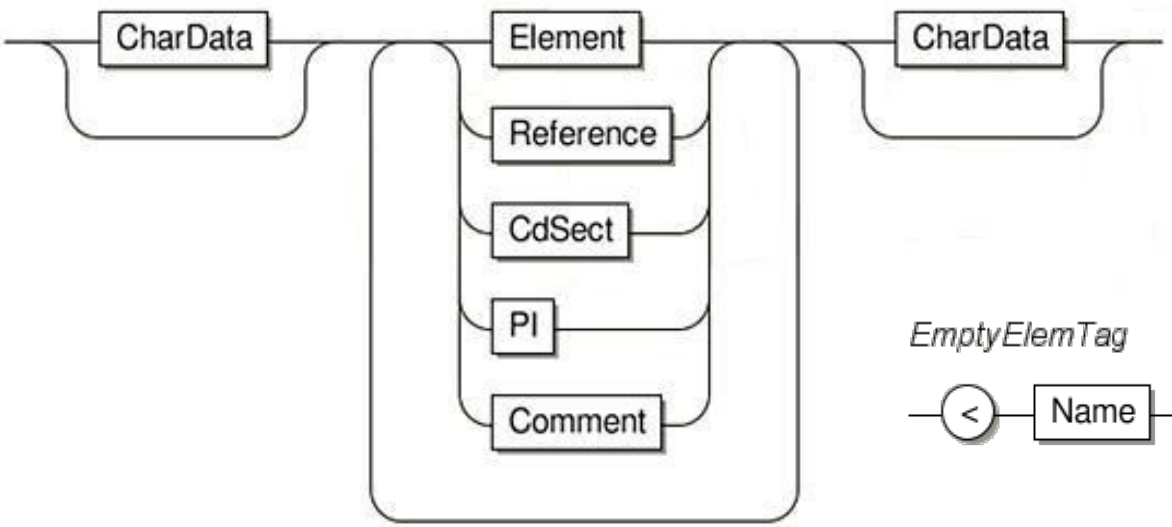
Prolog



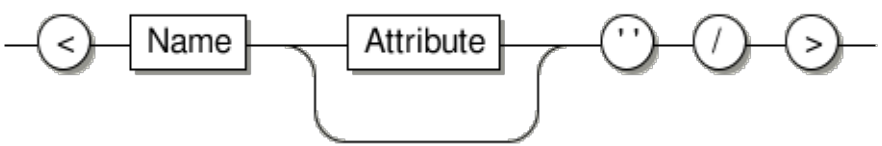
Element



Content

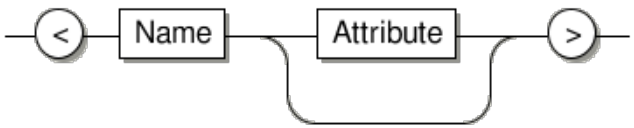


EmptyElemTag

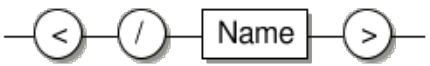


XML-Syntaxdiagramme (2/2)

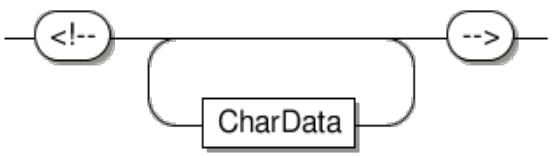
STag



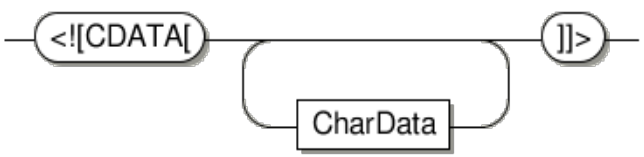
ETag



Comment

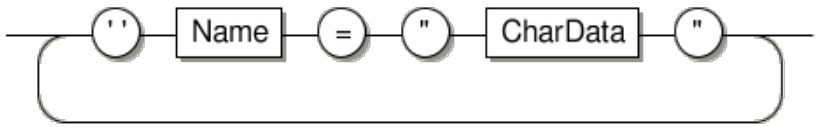


CdSect



PI

Attribute



Namen informell:
beginnen mit Buchstaben oder Unterstrich (_), danach können alphanumerische Zeichen folgen

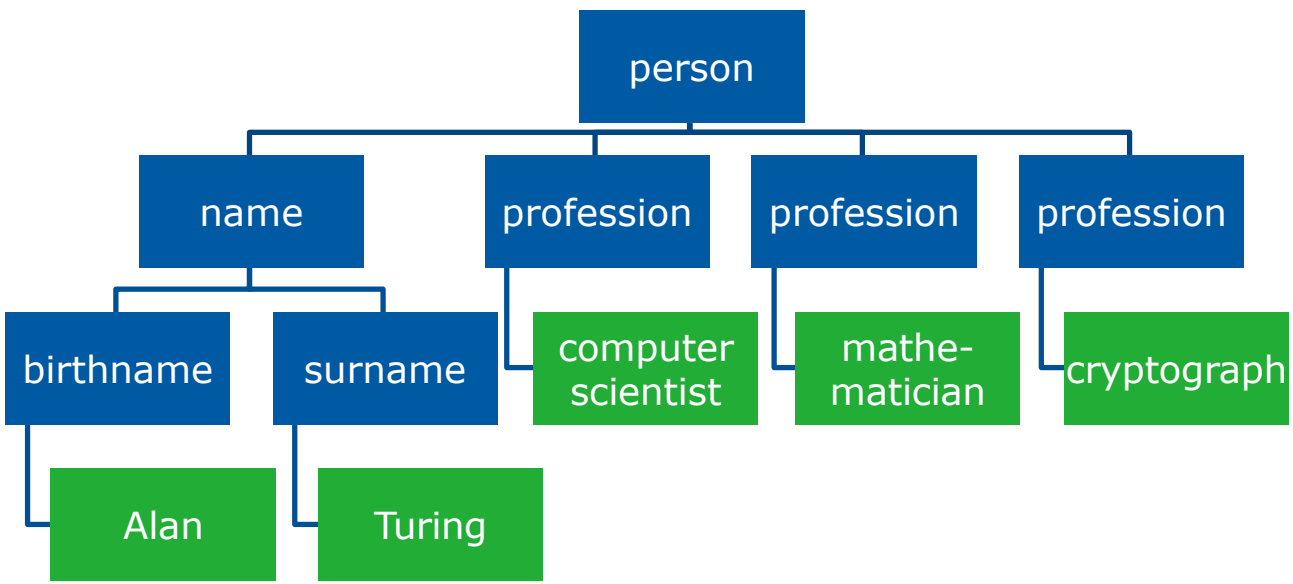
CharData informell:
Zeichen; Metazeichen sind gültig
z.B. ü für 'ü',

(Syntaxdiagramme hier nicht vollständig)

XML-Bäume

```
<person>
   <name>
      <birthname>Alan</birthname>
      <surname>Turing</surame>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptograph</profession>
</person>
```

Syntax:
hierarchischer Aufbau, bestehend aus Elementen und Textzeichen



Begriffe:
- Element
- Knoten (Tag)
- Vorgänger
- Nachfolger
- Nachbar
- Wurzel

Gemischter Inhalt

Textzeichen und Elemente werden als Nachbarn behandelt oder verschachtelt dargestellt.

Beispiel:

```
<biography><name><first_name>Alan</first_name>
<last_name>Turing</last_name></name> war einer der Ersten die
man als <emphasize>Informatiker</emphasize> bezeichnen kann.
W&auml;hrend er eine Vielzahl von Beitr&auml;gen geliefert
hat, ist er vor allem f&uuml;r den <emphasize>Turing
Test</emphasize> und die <emphasize>Turing
Maschine</emphasize> bekannt.</biography>
```

Schnellübung: Zeichnen Sie den XML-Baum!

Elemente tragen Attribute

- ein Attribut ist ein Paar `Attributname="Attributwert"`, das nach dem Anfang eines Elements an den Elementnamen angehängt wird
- Attributwerte werden mit `'` oder `"` begrenzt
- ein Element kann mehrere Attribute haben
- es muss zwischen Element beschreibenden Attributen und Eigenschaften von Elemententitäten unterschieden werden:

```
<person dob="1912/06/23" dod="1954/06/07">
Alan Turing
</person>
```

gegenüber

```
<person>
<dob>1912/06/23</dob>
<dod>1954/06/07</dod>
<name>Alan Turing</name>
</person>
```

Elementnamen sind XML Namen

- erstes Zeichen muss ein Buchstabe sein; nach vereinbarter Kodierung also: a, Ö, Ω, ç, _ (Unterstrich), - (Bindestrich), . (Punkt)
- ab zweitem Zeichen danach auch Ziffern zugelassen
- Leerzeichen, Doppelpunkt, Schrägstrich, … sind reservierte Metazeichen

Beispiele:

- wohlgeformt
  - `<Drivers_License_Number>`
  - `<month-day-year>29/3/2002</month-day-year>`
  - `<_4-lane>I-610</_4-lane>`
- nicht wohlgeformt
  - `<Driver's_licence_number>`
  - `<month/day/year>`
  - `<first name>`
  - `<4-lane>`

Namen für Entitäten

- Entitäten sind Platzhalter für andere Texte bzw. anderen Markup
- Entitäten kodieren Metazeichen:

  < `<`

  & `&`

  > `>`

  " `&quot;`

  ' `&apos;`

→ Vermeiden von Fehlinterpretationen

Beispiel:

```
<image source='oreilly.gif' width='122' height='66'
   alt='erh&auml;tlich bei O&apos;Reilly Books'>
```

CDATA-Abschnitte

- CDATA-Abschnitte werden für beliebige Zeichenketten benutzt
- sie werden durch die Metazeichen !, [, und]gekennzeichnet

- innerhalb eines CDATA Abschnitts sind alle anderen Metazeichen zulässig:

```
<![CDATA[
<svg xmlns="http://www.w3.org/2000/svg"
   width="12cm" height="10cm">
   <elipse rx="110" ry="130" cx="1cm" cy="1cm"/>
   <rect x="4cm" y="1cm" width="3cm" height="6cm"/>
</svg>
]]>
```

Achtung:]] kann nicht innerhalb des CDATA-Abschnitts verwendet werden, diese Zeichen wären sonst mehrdeutig.]] markiert immer das Ende von CDATA.

Weitere Regeln des Mark-up

- Kommentare können überall eingefügt werden (solange die Namen erkennbar bleiben) und sind v.d.F. `<!-- ein Kommentar -->`
- Verarbeitungsanweisungen (processing instructions) weisen auf Verarbeitungsschritte hin

```
<?robots index="yes" follow="no"?>
<?xml-stylesheet href="person.css" type="text/css"?>
```

- XML-Deklaration
  - steht am Anfang einer XML Datei, ist kein Element
  - Kodierung nach UTF-8 ist voreingestellt
  - `standalone="no"` kündigt an, dass die Syntaxbeschreibung (z.B. DTD) beigefügt ist
  - Beispiel:

```
<?xml version="1.0" encoding="US-ASCII"
standalone="yes"?>
```

Validieren von XML Dokumenten

- Wohlgeformtheit ist zur Vermeidung von Fehlinterpretationen notwendig
- Prüfung mittels
  - Websites, z.B.:
    - https://www.xmlvalidation.com
    - www.cogsci.ed.ac.uk/%7Erichard/xml-check.html
  - mittels Programmen wie Xerces: `java sax.SAXCount -v doc.xml`
- nur wohlgeformte XML-Dateien können automatisch verarbeitet werden
- wenn z.B. Browser Ausnahmen zulassen, und die Dokumente darauf bauen, dann werden XML Dokumente unterschiedlich behandelt und dargestellt

Document Type Definition (DTD)

Document Type Definition (DTD)

- Regelsatz zur Definition einer Klasse ähnlicher Dokumente
- besteht aus
  - möglichen Elementtypen
  - erlaubten Attributen von Elementen
  - Entitäten
  - Notationen
- legt fest
  - Reihenfolge
  - mögliche Schachtelung der Elemente
  - Art des Inhalts von Attributen

ELEMENTDEFINITION

- Definition ähnlich EBNF
- Kommentare mittels `<!--` und `-->`
- Inhaltsdefinition

  `#CDATA` – unverarbeitete Zeichenfolgen (Character Data)

  `#PCDATA` – verarbeitbare Zeicheninhalte (Parsed Character Data)

  `EMPTY` – kein Inhalt

  `ANY` – beliebiger Inhalt

- Struktur

  `,` – Reihenfolge (v.L.n.R.)

  `|` und `( )` – wie bei EBNF

- Wiederholungen/Anzahl

  `*` , `+` und `?` – wie bei EBNF

Beispieldefinition: HTML-Elemente

```
<!ELEMENT html (head, body)>

<!ELEMENT hr EMPTY>

<!ELEMENT div (#PCDATA | p | ul | ol | dl | table | pre | hr |
               h1|h2|h3|h4|h5|h6 | blockquote | address |
               fieldset)*>

<!ELEMENT dl (dt|dd)+>

<!-- u.s.w. -->
```

ATTRIBUTDEFINITION

- erlaubte Attributnamen für Elemente
- Typ und Vorgaben der einzelnen Attribute
- Obligation eines Attributs (#REQUIRED oder #IMPLIED)
- Variabilität eines Attributs (#FIXED)
- Standardwert, falls Attribut bei einem Tag nicht angegeben wird
- Beispiel: HTML-IMG-Tag

```
<!ATTLIST img
   id     ID     #IMPLIED
   src    CDATA  #REQUIRED
   alt    CDATA  #REQUIRED
   ismap  IDREF  #IMPLIED
>
```

WEITERE DEFINITIONEN DER DTD

- benannte Entitäten, Abkürzungen (ENTITY)
- Interpretationshinweise für Daten (NOTATION)
- bedingte Abschnitte
  - <![INCLUDE[
    <!ENTITY einbinden "ja">
    ]]>
  - <![IGNORE[
    <!ENTITY einbinden "nein">
    ]]>
  - <!ENTITY % weiche "INCLUDE">
    <![%weiche;[
    <!ENTITY einbinden "ja">
    ]]>
- u.v.a.m.

SYNTAXBESCHREIBUNGEN

- Sprachen wie HTML und SVG basieren auf Syntaxbeschreibungen, um die Art der Elemente und deren Attribute festzulegen
- XML kennt zwei Arten von Syntaxbeschreibungen:
  - Document Type Definition (DTD)
    - definieren die Struktur von XML Dokumenten durch Deklaration von Regeln
    - sind selbst kein XML
    - geben nur die Regeln an, aber nicht die eigentlichen Dokumente
    - geben keine Textzeichen vor
    - beschreiben nicht die Semantik der Elemente bzw. der Daten
  - XML Schema Definition (XSD)

EINFACHES DTD-BEISPIEL

```
<!ELEMENT person (name, profession*)>
<!ELEMENT name (first_name, last_name)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT profession (#PCDATA)>
```

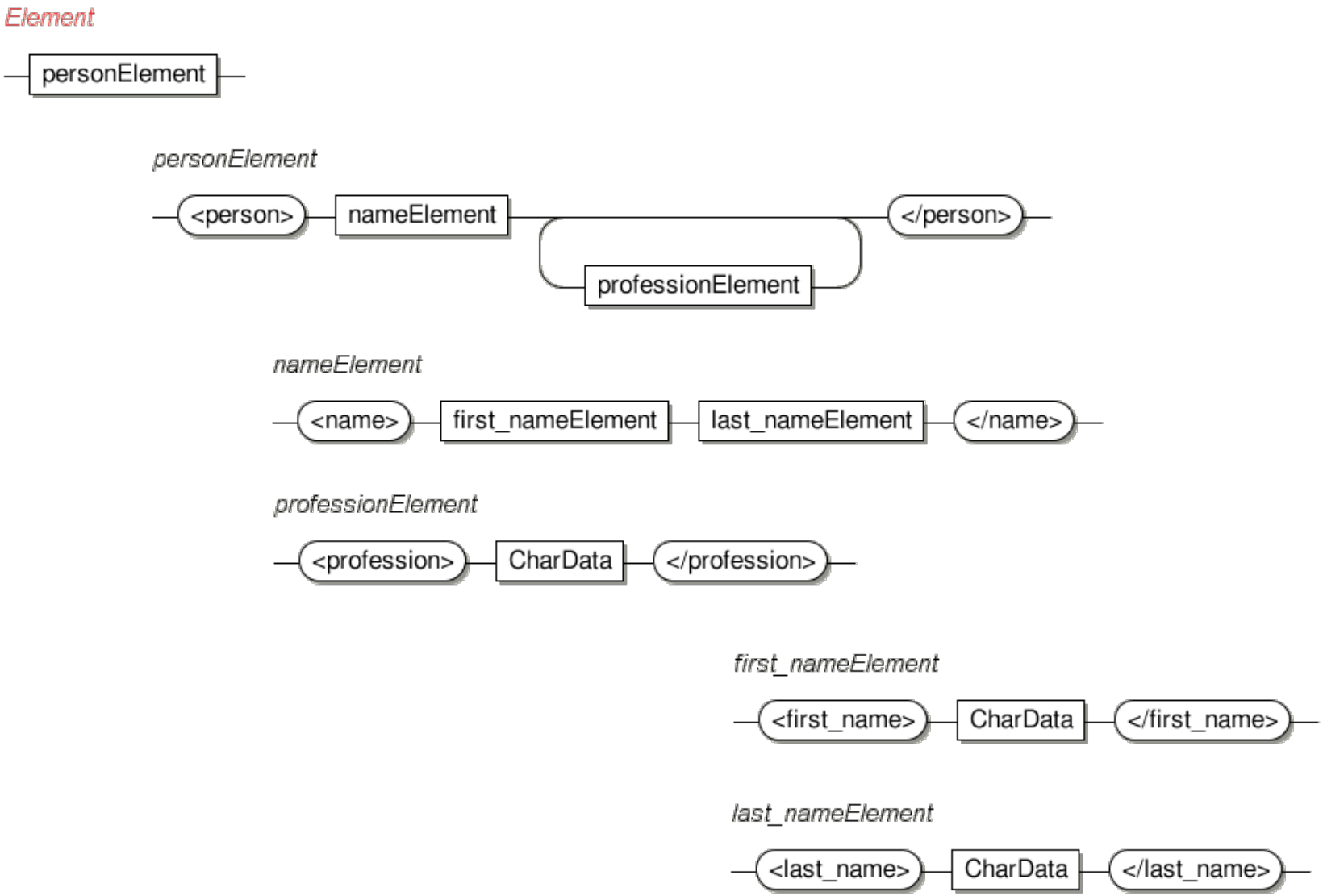
DTD kann vorliegen

- zusammen mit dem XML Dokument in derselben Datei
- an anderer Stelle im Internet
- in einer anderen Datei
- gar nicht (d.h. das verarbeitende Programm trifft implizite Annahme)

Beispiel:

```
<!DOCTYPE person SYSTEM "https://www.ibiblio.org/xml/books/
   biblegold/source/34/person.dtd">
```

DTDs verfeinern Syntaxdiagramme



Interne DTDs

```
<?xml version="1.0"?>
<!DOCTYPE person [
   <!ELEMENT person (name, profession*)>
   <!ELEMENT name (first_name, last_name)>
   <!ELEMENT first_name (#PCDATA)>
   <!ELEMENT last_name (#PCDATA)>
   <!ELEMENT profession (#PCDATA)>
]>
<person>
   <name>
      <first_name>Alan</first_name>
      <last_name>Turing</last_name>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```

Deklaration von Elementen (1/2)

Element besteht

- nur aus Textzeichen einschließlich Entitäten: `#PCDATA`
- einem einzelnen Element: `<!ELEMENT fax (phone_number)>`
- einer Sequenz von Elementen:
  - `<!ELEMENT name (first_name+, last_name)>`
  - in dieser Reihenfolge, ohne Auslassungen, ohne Erweiterung
  - Anzahl der Nachfolger muss durch ein Suffix festgelegt werden
    - `?` kein oder ein Element (d.h. Element ist optional)
    - `*` kein Element oder eine beliebige Anzahl von Elementen
    - `+` mindestens ein Element (oder mehrere)

Deklaration von Elementen (2/2)

- Alternativen fassen Regeln zusammen:

```
<!ELEMENT transport (car | train | plane)>
```

- Klammern fassen Sequenzen zusammen
  - unterstützen Verschachtelung
  - tragen evtl. auch ein Suffix ?, *, +
  - bilden Alternativen: `<!ELEMENT point ( (a,b) | (y,z) )>`

Schnellübungen: Überlegen Sie DTDs für

- Ein Punkt soll entweder durch kartesische Koordinaten (x,y) oder Polarkoordinaten (ρ,ϴ) beschrieben werden
- Ein Polygon besteht aus drei oder mehr Punkten mit kartesischen oder polaren Koordinaten
- Der Name einer Person besteht genau aus einem Vornamen, einer beliebigen Anzahl von weiteren Vornamen, sowie einem Nachnamen

Besonderheiten der Deklaration

- Textzeichen mit Elementen mischen:

```
<!ELEMENT paragraph (#PCDATA | name | profession  |
footnote | emphasize | date)*>
```

- leere Elemente enthalten keine Nachfolger, auch keine Textzeichen

```
<!ELEMENT br EMPTY>
```

  - Vorsicht: muss beendet werden, oderist per Abkürzung zu schreiben, aber es gibt Sonderfälle:
    - `<br/>` → XML
    - `<br />` → XHTML

- any

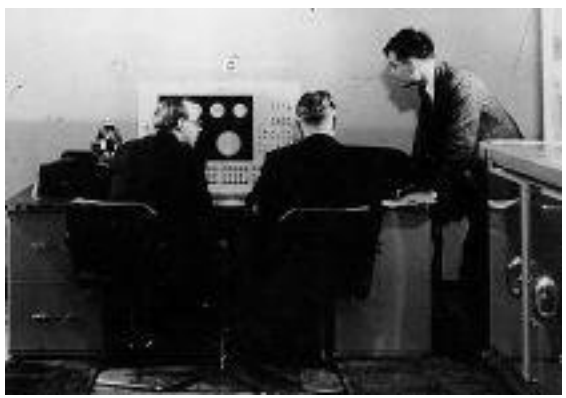
```
<!ELEMENT page ANY>
```

  - ist wenig aussagekräftig, aber kann zu Beginn des Entwurfs von Regeln hilfreich sein
  - würde es erlauben, unbekannte XML Dokumente zu integrieren

Deklaration von Attributen

für jedes Element können Attribute festgelegt werden:

```
<!ATTLIST image source CDATA #REQUIRED
                 width CDATA #REQUIRED
                height CDATA #REQUIRED
                   alt CDATA #REQUIRED
>
```



Beispiel (Instanz):

```
<image source="bus.jpg"
        width="122"
       height="345"
          alt="Alan Turing am Manchester Computer"/>
```

VERSCHIEDENE ATTRIBUTTYPEN

Um den Typ von Attributwerten festzulegen, werden bei der Deklaration folgende Schlüsselwörter verwendet:

- `CDATA` jegliche Zeichenkette, d.h. Text, URL, Preise, E-Mail-Adressen, …
- `NMTOKEN` ein wohlgeformter XML-Name
- `NMTOKENS` ein oder mehrere XML-Namen (durch Leerzeichen getrennt)

  Beispiel zur Aufzählung von Attributwerten:

```
<!ATTLIST date month (Jan | Feb | Mar | Apr | May | Jun |
   Jul | Aug | Sep | Oct | Nov | Dec) #REQUIRED>
```

- `ENTITY` enthält den Namen einer Entität (definiert an anderer DTD-Stelle)
- `ENTITIES` mehrere Entitäten (durch Leerzeichen getrennt)
- `ID` ein eindeutiger XML-Name (nicht an anderer Stelle als ID benutzt)
- `IDREF` reiner XML-Name der sich auf eine ID dieses Dokuments bezieht
- `IDREFS` Sequenz von Ids (durch Leerzeichen getrennt)

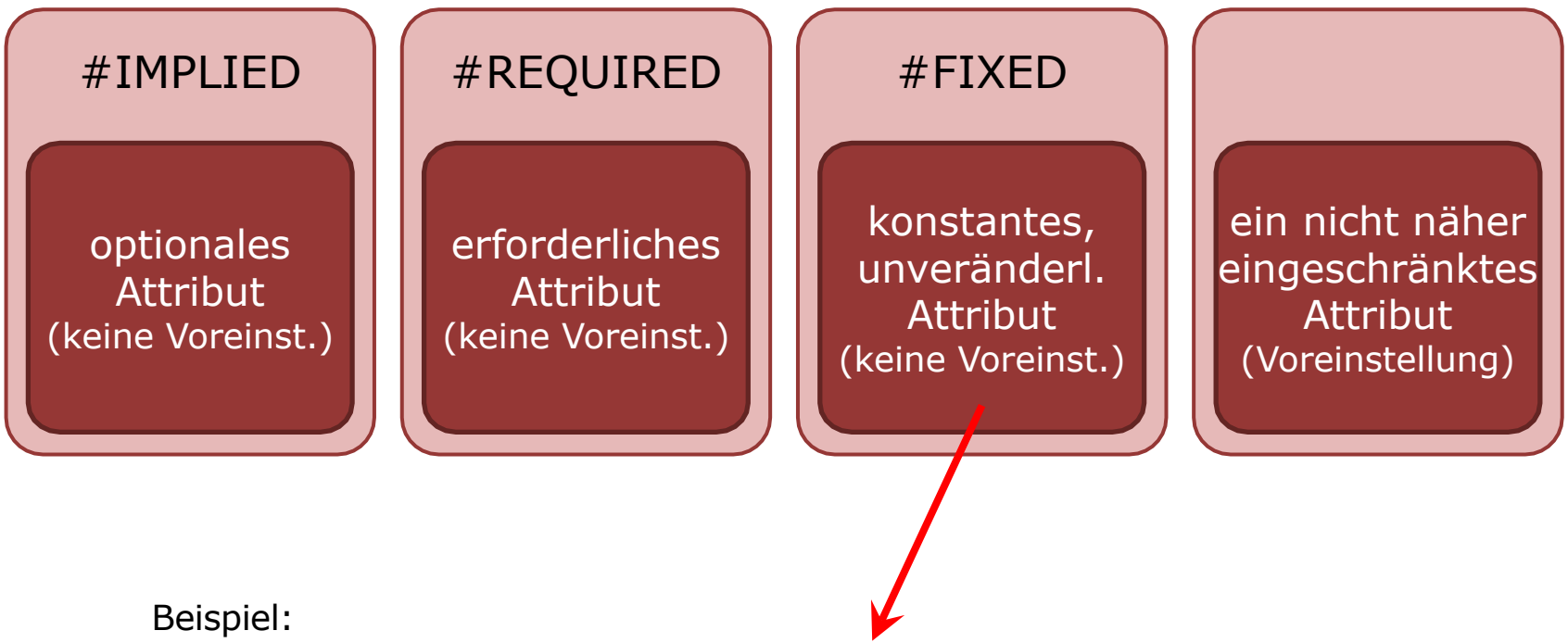
Beispiel für IDREFS Attribut

```
<!ATTLIST project project_id ID #REQUIRED
          team IDREFS #REQUIRED>
<!ATTLIST person ssn ID #REQUIRED
          assigments IDREFS #REQUIRED>

<project project_id="p1" team="ss123-45-6789">
   <goal>Entwicklung eines strategischen Plans</goal>
</project>
<project project_id="p2" team="ss123-45-6789 ss8765-43-210">
   <goal>Einführung von Linux</goal>
</project>

<person ssn="ss123-45-6789" assignments="p1 p2">
   <name>Max Mustermann</name>
</person>
<person ssn="ss8765-43-210" assignments="p2">
   <name>Petra Beispielfrau</name>
</person>
```

Voreinstellungen der Attribute



Beispiel:

```
<!ATTLIST biography xmlns:xlink CDATA #FIXED
"http://www.w3.org/xlink">
```

→ angegebene URL ist immer für das Element biography festgelegt

Beispiel: Personendaten

```
<?xml version="1.0"?>
<!DOCTYPE person [
<!ELEMENT person (name+, profession*)>
<!ELEMENT name EMPTY>
<!ATTLIST name first CDATA #REQUIRED last CDATA #REQUIRED>
<!-- die Attribute first bzw. last sind notwendig, können
jedoch leer bleiben, z.B. <name first="Bobo" last=""/> -->
<!ELEMENT profession EMPTY>
<!ATTLIST profession value CDATA #REQUIRED>
]>
<person>
<name first="Alan" last="Turing"/>
<profession value="computer scientist"/>
<profession value="mathematician"/>
<profession value="cryptograph"/>
</person>
```

Beispiel: Textverarbeitung (DTD-Teil)

```
<!DOCTYPE biography [
   <!ELEMENT biography ( #PCDATA | image | paragraph
      | definition | graph | person | profession | emphasize
      | birthname | surname | footnote | date )* >
   <!ELEMENT footnote (#PCDATA)>
   <!ATTLIST footnote source CDATA #REQUIRED>
   <!ELEMENT person (birthname, surname)>
   <!ATTLIST person dob CDATA #REQUIRED dod CDATA #REQUIRED>
   <!ELEMENT birthname (#PCDATA)>
   <!ELEMENT surname  (#PCDATA)>
   <!ELEMENT image EMPTY>
   <!ATTLIST image source CDATA #REQUIRED alt CDATA #IMPLIED
      width NMTOKEN #REQUIRED height NMTOKEN #REQUIRED>
   <!ELEMENT date (day, month, year)>
   <!ELEMENT day (#PCDATA)>
   <!ELEMENT month (#PCDATA)>
   <!ELEMENT year (#PCDATA)>
   <!ELEMENT profession (#PCDATA)>
   <!ELEMENT emphasize (#PCDATA)>
   <!ATTLIST emphasize xlink:href CDATA #IMPLIED>
   <!ATTLIST emphasize xlink:type CDATA #IMPLIED>
]>
```

Beispiel: Textverarbeitung (XML-Teil)

```
<?xml version="1.0"?>
<biography ><image
source="http://www.turing.org.uk/turing/pi1/bus.jpg" width="152"
weight="345"/><person dob="1912/06/23" dod="1954/06/07">
<birthname>Alan</birthname><surname>Turing</surname></person> war
einer ersten die den Namen <emphasize>Informatiker</emphasize> tragen
durften. Obwohl er unz&auml;hlige Beitr&auml;ge zur Informatik
geleistet hat, ist er am besten f&uuml;r den <emphasize>Turing
Test</emphasize> und die <emphasize>Turing Maschine </emphasize>
bekannt. <surname>Turing</surname> war auch ein ausgewiesener
<profession>Mathematiker</profession> und <profession>Kryptograph
</profession>. Seine Arbeit war ma&szlig;gebend um den Kode der Enigma
Maschine zu entschl&uuml;sseln<footnote source="The Ultra Secret, F.W.
Winterbotham, 1974">1</footnote>. Er nahm sich das Leben am
<date><day>7</day><month>Juni</month><year>1954</year></date> nachdem
er der Homosexualit&auml;t f&uuml;r schuldig befunden wurde und
gezwungen wurde weibliche Hormone einzunehmen<footnote source="Alan
Turing: the Enigma, Andrew Hodges, 1983">2</footnote>.</biography>
```

Namensräume (namespaces)

- XML-Namen gehören Namensräumen (Syntaxbeschreibung) an
- Präfixe unterscheiden identische Elemente verschiedener Namensräume
  → Elemente verschiedener DTDs müssen unterschieden werden
  - Beispiel: `svg:img` ist verschieden von `html:img`
  - Namensräume werden bei Elementen durch das allgemein gültige Attribut `xmlns` deklariert:

```
<svg xmlns="http://www.w3.org/2000/svg"
   width="12cm" height="10cm">
   <ellipse rx="110" ry="130"/>
</svg>
<html xmlns:svg="http://www.w3.org/2000/svg">
   <svg:ellipse rx="110" ry="130"/>
</html>
```

  - Die Bindung gilt für darin verschachtelte Elemente

Beispiel mit drei Namensräumen

```
<?xml version="1.0"?>
<html
   xmlns="http://www.w3.org/1999/xhtml"
   xmlns:xlink="http://www.w3.org/1999/xlink">
<head><title>Drei Namespaces</title></head>
<body>
   <h1 align="center">Eine Ellipse und ein Rechteck</h1>
   <svg xmlns="http://www.w3.org/2000/svg"
      width="12cm" height="10cm">
      <ellipse rx="110" ry="130"/>
      <rect x="4cm" y="1cm" width="3cm" height="6cm"/>
   </svg>
   <p xlink:type="simple" xlink:href="ellipses.html">
      mehr zu Ellipsen
   </p>
   <p xlink:type="simple" xlink:href="rectangles.html">
      mehr zu Rechtecken
   </p>
   <hr />
</body>
</html>
```

DTD für Dokumente: Text Encoding Initiative (TEI)

- Spezifikation unter www.tei-c.org
- DTD für Textverarbeitung
- verwendet SGML Merkmale
- einsetzbar für Theaterstücke, Gedichte, und Literatur

DTD für Dokumente: DocBook

- Spezifikation unter www.oasis-open.org/docbook
- Verbreitet für Dokumentation von Informatikinhalten
- wurde vom Verlag O'Reilly für Buchherstellung verwendet
- wird in Linux eingesetzt
- aktuelle Version 5.0
    - DTD für SGML
    - DTD für XML

DTD für Dokumente: Open Ebook (OEB)

- Spezifikation unter www.idpf.org/
- wird in mobilen Geräten (eBook-Reader, Kindle, iPad, …) verwendet
- aktuelle Version: Open eBook Publication Structure (OEBPS) 2.0
- DTD für XML, genauer: Untermenge von XHTML, CSS und DublinCore
- Dokumente liegen mit einem Manifest in einem ZIP-Archiv, aber nicht vergleichbar zu PDF oder ODF

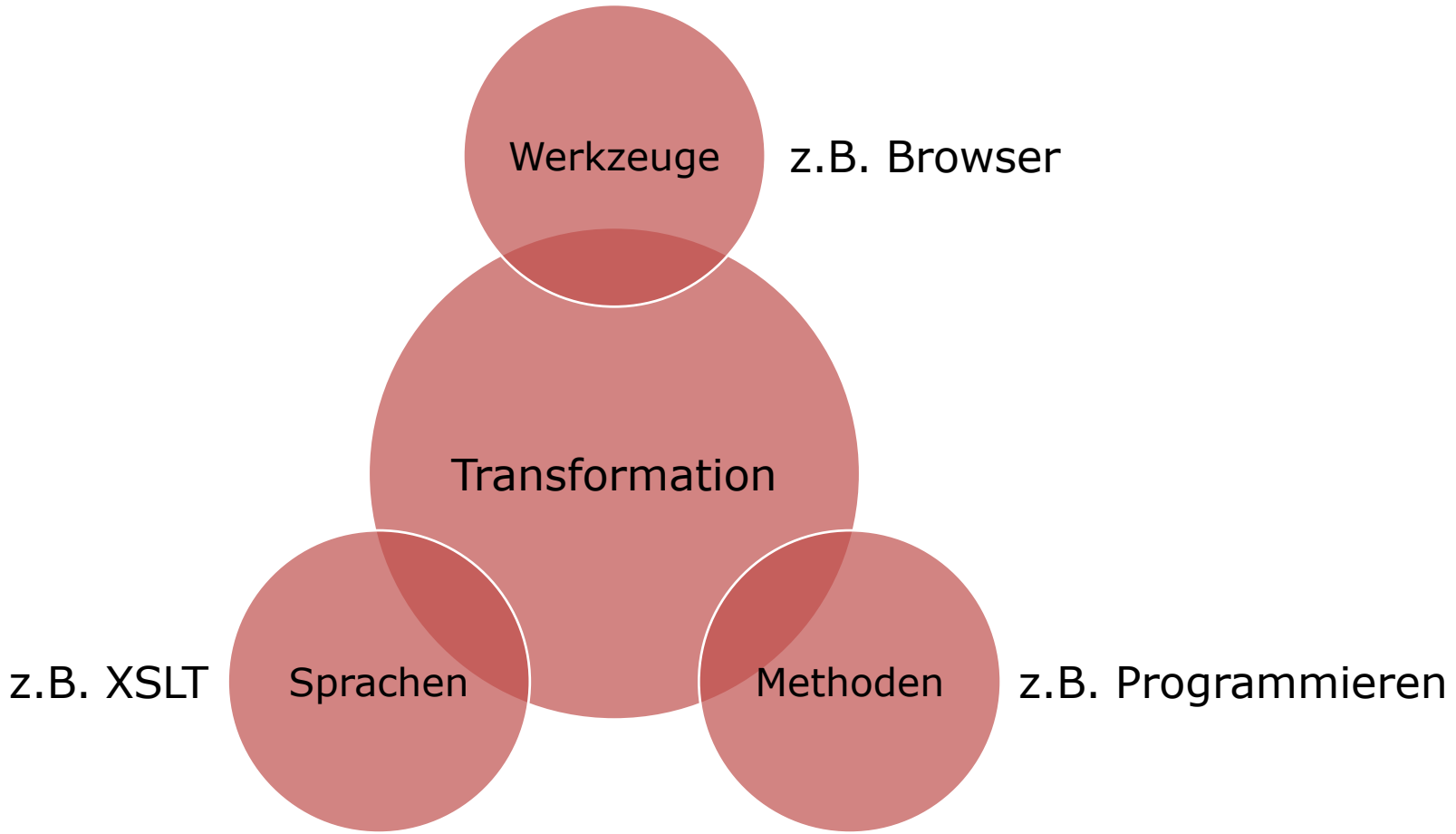
Grenzen der DTDs

- keine Datentypen für Elemente und Attribute; nicht möglich, folgendes zu unterscheiden:
  - Postleitzahlen sind CharData
  - Postleitzahlen haben 5 Ziffern
  - Postleitzahlen können führende 0en haben
  - Postleitzahlen besitzen kein Vorzeichen
- keine Modularisierung
- XML Schema sind selbst XML Dokumente, DTDs nicht

```
<xsd:schema xmlns:xsd=
"http://www.w3.org/1999/XMLSchema">
   <xsd:annotation>
      <xsd:documentation>
         schema for a book
      </xsd:documentation>
   </xsd:anotation>
   <xsd:element name="book"
      type="bookT" />
   <xsd:complexType name="bookT">
      <xsd:element name="title"
         type="xsd:string" />
      <xsd:element
         name="information"
         type="informationT" />
   </xsd:complexType>
</xsd:schema>
```

Extensible Stylesheet Language (XSL) und XSL Transformation (XSLT)

Konzepte der Informatik



XSL und XSLT

- XML ist geeignet, XML-Dokumente in XML zu verarbeiten
- Spezifikation dazu: http://www.w3.org/TR/xslt

Ziele

- Daten zwischen Anwendungen austauschen
- Daten von der Präsentation trennen
- Teilweise wird in mehreren Schritten vorgegangen
  - erste Verarbeitung: Transformation der Struktur
  - zweite Verarbeitung: Formatierung der Ausgabe (HTML, PDF, …)

„Hallo Welt!“ in XSLT – XML-Quelle und HTML-Ausgabe

helloworld.xml:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="helloworld.xsl"?>
<greeting>Hallo Welt!</greeting>
```

Ausgabe:

```
<html>
<head>
<title>Wort zum Tag</title>
</head>
<body>
<p>Hallo Welt!</p>
</body>
</html>
```

„Hallo Welt!“ in XSLT – XSL-Deklaration

helloworld.xsl verwendet 2 XML Sprachen: XSL und HTML

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
    <html>
        <head>
            <title>Wort zum Tag</title>
        </head>
        <body>
            <p><xsl:value-of select="greeting" /></p>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Merkmale von XSL

Verwenden von XML-Syntax

keine Seiteneffekte, alles muss deklariert werden, Sequenz muss nicht immer eingehalten werden

Verarbeitung basiert auf Templates

XSLT 1.1

- mehrfache Ausgabedokumente
- mehrere Bäume
- Unterstützung von Java und EcmaScript

XSLT 2.0

- verwendet XSchema

TRANSFORMATIONEN

Templates anwenden lassen

- `<xsl:apply-templates>`
- bearbeitet Elemente durch Anwendung der entsprechenden *Template-Regeln*

Textzeichen („Wert“) aus einem Element holen

- `<xsl:value-of>`
- extrahiert *alle Informationen des Elements*

Templates wiederholt auf alle identischen Elemente anwenden

- `<xsl:for-each>`
- *jedes Element* wird verarbeitet

Vorlagenregeln (Template Rules)

match-Attribut

- benennt (addressiert) ein Element (mittels der XPath Sprache)
- Ausdruck `/` adressiert Wurzel
- Ausdruck `title` adressiert ein Element
- Ausdruck `chapter/title` adressiert Element innerhalb eines anderen

Weitere Regelelemente

- `xsl:message` gibt eine Nachricht beim transformieren aus
- `xsl:comment` kommentiert eine Regel
- `xsl:text (xml:space)` erzwingt Texte und Leerzeichen
- andere Elemente oder Textzeichen

Verschachtelte Templates

```
<xsl:template match="/">
   <xsl:message>Started!</xsl:message>
   <xsl:comment>generated from XSLT</xsl:comment>
   <html>
      <head>
         <title>Die erste Seite</title>
      </head>
      <body>
         <xsl:apply-templates/>
      </body>
   </html>
</xsl:template>
```

Selektiere alle Nachfolger des aktuellen Knotens im Dokumentenbaum, und wende die entsprechende Templateregel an.

Push-Verarbeitung: XSL-Definition

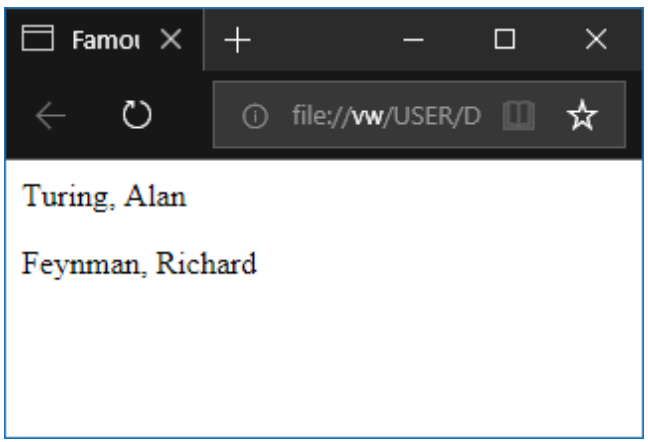
```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="people">
      <html>
         <head>
            <title>Famous Scientists</title>
         </head>
         <body>
            <xsl:apply-templates/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="name">
      <p>
         <xsl:value-of select="surname"/>,
         <xsl:value-of select="birthname"/>
      </p>
   </xsl:template>
   <xsl:template match="person">
      <xsl:apply-templates select="name"/>
   </xsl:template>
</xsl:stylesheet>
```

Push-Verarbeitung: XML-Dokument

```
<?xml version="1.0"?>
<people>
   <person born="1912" died="1954">
      <name>
         <birthname>Alan</birthname>
         <surname>Turing</surname>
      </name>
      <profession>computer scientist</profession>
      <profession>mathematician</profession>
      <profession>cryptographer</profession>
   </person>
   <person born="1918" died="1988">
      <name>
         <birthname>Richard</birthname>
         <initial>M</initial>
         <surname>Feynman</surname>
      </name>
      <profession>physicist</profession>
      <hobby>Playing the bongoes</hobby>
   </person>
</people>
```

Push-Verarbeitung: Ausgabe

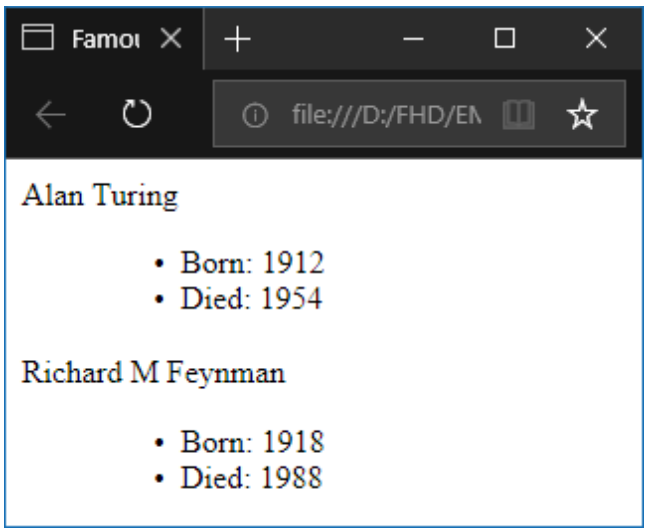
```
<html>
   <head>
      <title>Famous Scientists</title>
   </head>
   <body>
      <p>Turing, Alan</p>
      <p>Feynman, Richard</p>
   </body>
</html>
```



📖 **Rezept:**
Vorlagenregel für jedes Element, Ausgabe festlegen und Vorlagen steuern

ATTRIBUTE LESEN

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="people">
      <html>
         <head>
            <title>Famous Scientists</title>
         </head>
         <body>
            <dl><xsl:apply-templates /></dl>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="person">
      <dt><xsl:apply-templates select="name" /></dt>
      <dd>
         <ul>
            <li>Born: <xsl:apply-templates select="@born" /></li>
            <li>Died: <xsl:apply-templates select="@died"/></li>
         </ul>
      </dd>
      <br/>
   </xsl:template>
</xsl:stylesheet>
```



Schnellübung:

Wie müsste die zugehörige `personattrib.xml` aussehen?

Pull-Verarbeitung

Für unregelmäßige/unvollständige Strukturen werden Vorlagenregeln für jeden Anwendungsfall geschrieben

- im Beispiel: falls kein zweiter Vorname existiert
- Verarbeitung nicht aller aber einiger gezielt ausgewählter Nachfolger eines Elements: `<xsl:apply-templates select="name">`

Beispiele für Anwendungsfälle bei Textverarbeitung sind

- Inhaltsverzeichnis
- Inhalt

Das Attribut `mode` erlaubt Anwendungsfälle zu unterscheiden und ist für `xsl:template` sowie `xsl:apply-templates` verwendbar

Falls kein Modus zutrifft, wird die Regel ohne Modusvorgabe verwendet

```
<xsl:template match="*|/" mode="toc">
<xsl:apply-templates mode="toc">
</xsl:template>
```

Stylesheet mit mehreren Modi

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="people">
        <html>
            <head><title>Famous Scientists</title></head>
            <body>
                <ul><xsl:apply-templates select="person" mode="toc"/></ul>
                <xsl:apply-templates select="person"/>
            </body>
        </html>
    </xsl:template>
    <!-- Table of Contents Mode Templates -->
    <xsl:template match="person" mode="toc">
        <xsl:apply-templates select="name" mode="toc"/>
    </xsl:template>
    <xsl:template match="name" mode="toc">
        <li><xsl:value-of select="surname"/>, <xsl:value-of select="birthname"/></li>
    </xsl:template>
    <!-- Normal Mode Templates -->
    <xsl:template match="person">
        <p><xsl:apply-templates/></p>
    </xsl:template>
</xsl:stylesheet>
```

XSLT und Namensräume

Wenn Namensräume in der Ausgabe benötigt werden, ist die Definition einer Namespace-ID notwendig:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:pe="http://namespaces.oreilly.com/people">
<xsl:template match="pe:people">
   <html>
      <head>
         <title>Famous Scientists</title>
      </head>
      <body>
         <!-- … -->
```

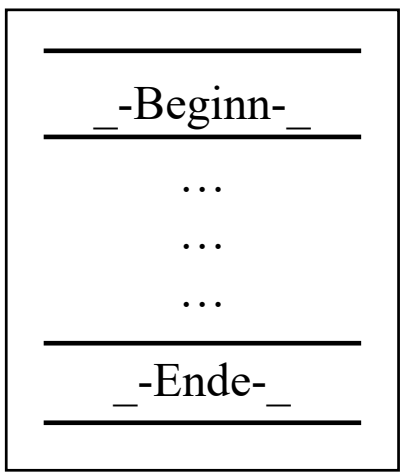
VARIABLEN

- Zuweisung: `<xsl:variable name="width" select="50"/>`
- globale Gültigkeit
- diverse Datentypen möglich
  - `string` jegliche Unicode Zeichen
  - `number` analog zu Java float
  - `boolean` 0..1 oder `false`...`true`
  - Element-Menge (set of nodes in source tree)
  - externe Objekte (z.B. Java Objekt, ...)
- Datentypen werden konvertiert
- Ausdrücke entstehen meist beim Adressieren eines Elements (siehe XPATH)

KONTROLLSTRUKTUR: <IF>

Bedingungen im Attribut match und mittels <if>

```
<xsl:template match="para">
<xsl:if test="position()=1">
   <hr/>_-Beginn-_<hr/>
</xsl:if>
<xsl:apply-templates/>
<xsl:if test="last()=1">
   <hr/>_-Ende-_<hr/>
</xsl:if>
</xsl:template>
```



KONTROLLSTRUKTUREN: <FOR-EACH>

- Wiederholung für alle Elemente desselben Namens
- Beispiel:

```
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xsl:version="1.0">
<head><title> a list of books </title></head>
<body>
   <table>
      <xsl:for-each select="//book">
         <xsl:sort select="author"/>
         <tr>
            <td><xsl:value-of select="author"/></td>
            <td><xsl:value-of select="@category"/></td>
            <td><xsl:value-of select="price"/></td>
         </tr>
      </xsl:for-each>
   </table>
</body>
</html>
```

Behandlung von Leerräumen

- Steuerung der Verwendung von Leerzeichen, Zeilenwechsel, Tabulator, usw. durch die XSL-Elemente
  - `<xml:space>`
  - `<xsl:strip-space>`
- Festlegung welche Leerräume aus dem Stylesheet und welche aus dem XML-Dokument für die Ausgabe übernommen werden
- Besonderheiten je nach Implementierung (z.B. Microsofts MSXML3)

Adressierung mit XPath

- einzelne Ortspfade
  - Wurzel: `/`
  - Nachfolger: `birthname`
  - Attributlokation: `@born`
  - weitere Knotenarten nach Funktion:
    - `comment()`
    - `text()`
    - `processing-instruction()`
- jegliches Element: `*`
- jegliches Attribut: `@*`
- auch mit Namespace-Präfix möglich:
  - `svg:*`
  - `@xlink:*`

ZUSAMMENGESETZTE ORTSPFADE

- Verbinden einzelner Ortspfade mit /
- Sonderformen sind
  - / die Wurzel (XML Wurzel ist deren Nachfolger)
  - // alle Nachfolger
  - .. direkter Vorgänger
  - . aktuelles Element

Beispiel:

```
<xsl:template match="name">
<strong><xsl:value-of select="."></strong>
</xsl:template>
```

Bedingungen

- Erweiterungsoperatoren für Vergleiche: `=`, `!=`, `<`, `>`, `<=`, `>=`
  Vorsicht: `<` und `>` müssen als Literale mit Entitäten (`<` und `>`) umschrieben werden!

- Elemente mit einem bestimmten Wert:

```
match="//profession[.='physicist']"
```

- Finde Elemente mit einem bestimmten Attribut

```
match="//person[@id='p4567']"
```

- Finde Elemente mit einem unbestimmten Attribut

```
match="//name[middle_initial]"
```

- Dazu gelten die logischen Operatoren: `and`, `or`

Allgemeine Ausdrücke in XPath

- mathematische Operatoren: `+`, `-`, `*`, `div`, `mod`

```
<xsl:template match="person">
<xsl:value-of select="@id" div 10/>
</xsl:template>
```

- Zeichenketten (Strings)
- Bool'sche Werte und Ausdrücke (`true` und `false`)
- diverse Zugriffs- und Filterfunktionen:
  - `position()`, `first()`, `count()`
  - `starts-with(last_name, 'T')`, `substring-after()`, `substring()`
  - `string-length()`, `round()`, `floor()`, `ceiling()`

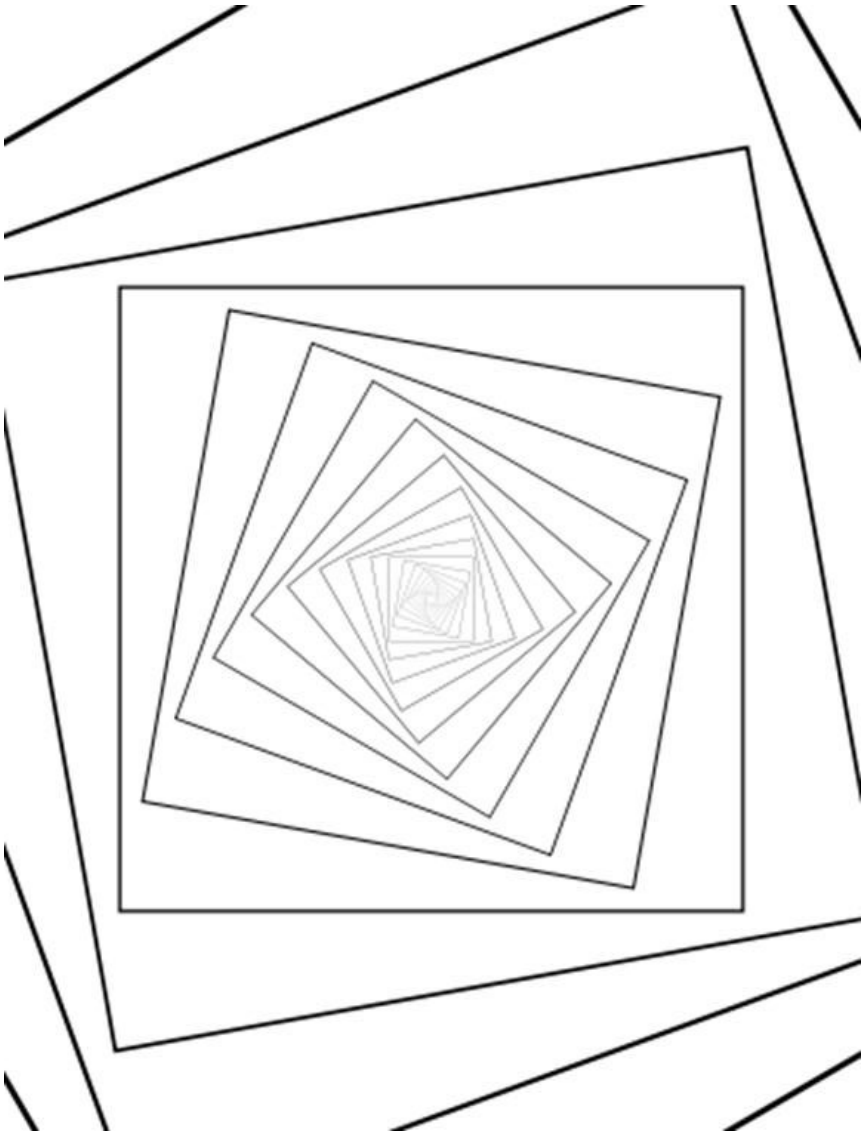
SVG MIT XSL ERZEUGEN

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="" type="text/xsl"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:svg="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
    version="1.0">
    <xsl:template match="/">
        <svg:svg viewBox="-40 -40 80 80">
            <svg:g id="spiral">
                <xsl:call-template name="draw-primitive">
                    <xsl:with-param name="depth" select="30"/>
                </xsl:call-template>
            </svg:g>
        </svg:svg>
    </xsl:template>
    <xsl:template name="draw-primitive">
        <xsl:param name="depth"/>
        <xsl:if test="$depth > 0">
            <svg:g transform="scale(.8, .8) rotate(10)">
                <svg:rect x="-100" y="-100" width="200" height="200" stroke="red"/>
                <xsl:call-template name="draw-primitive">
                    <xsl:with-param name="depth" select="$depth - 1"/>
                </xsl:call-template>
            </svg:g>
        </xsl:if>
    </xsl:template>
</xsl:stylesheet>
```

Rekursion in der XSL-Templateregel

Rekursion:
Die Regel `draw-primitive`
verwendet sich selbst erneut.

- zu Beginn `depth=30`
- falls `$depth > 0` :
  - `depth` um 1 dekrementieren
  - `draw-primitive` aufrufen
- sonst:
  - Abbruch der Rekursion



BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Hypertext Markup Language (HTML)

Konzepte des World Wide Web

Hypertext Transfer Protocol (HTTP)

- Spezifikation der Kommunikation zwischen Web-Server und Web-Client
  - Versand der HTML-Dateien vom Web-Server an den Web-Browser
  - Versand der Formulareingaben vom Web-Browser an den Web-Server
- Spezifikation der Nutzung weiterer Technologien
  - TCP Connection Keep Alive
  - TLS-Verschlüsselung
  - WebSockets
  - …

Konzepte des World Wide Web

Universal Resource Identifier (URI)

- Bezeichnung und Adresierung beliebiger Ressourcen im Web
- URIs unterteilen sich in URL (Uniform Resource Locator) und URN (Uniform Resource Name)

Hypertext Markup Language (HTML)

- Markup-Sprache zur Beschreibung von Web-Dokumenten
- definiert den Inhalt und die Gliederung von (Text-)Ressourcen

Cascading Style Sheets (CSS)

- Auszeichnungssprache zur Beschreibung von Gestaltungsmustern
- definiert die Gestaltung von (HTML-)Ressourcen

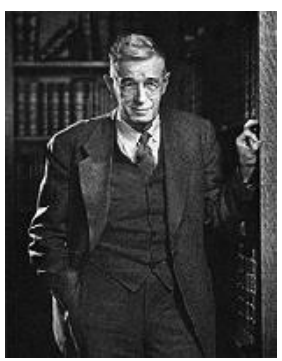
HYPERTEXT

Hypertext ist ein nichtlineares Medium zur Präsentation von Textinhalten. Die Nichtlinearität wird durch Hyperlinks erzielt, die verschiedene Einheiten verknüpfen.

Bush entwickelte 1945 eine technisch-wissenschaftliche Utopie (Memex), die darauf basiert, dass Denken durch Assoziationen gefördert wird.

Das Konzept sieht vor, dass riesige Informationsmengen gespeichert werden können, aber auch dass Benutzer Spuren anlegen, verwandte Texte verknüpfen und die wiederum für den zukünftigen Einsatz verwendet werden können.

Der Begriff *Hypertext* wurde 1965 vom Autor und Philosophen Ted Nelson entwickelt. (siehe auch http://ted.hyperland.com/)



Hypertext Markup Language (HTML)

- Grundlage des WWW
- Einteilung in
  - darzustellende Inhalte
  - zusätzliche Metainformationen
- verwandt mit XML
  - HTML 4.01 (XHTML) vollständig kompatibel mit XML
  - HTML 5 ersetzt XML-ähnliches HTML und XHTML durch XML-valide Beschreibung
- visuelle Darstellung soll nicht in HTML erfolgen
  (sondern durch den Web-Browser und Gestaltungsvorlagen wie CSS)

→ In dieser Vorlesungsreihe: HTML 4, XHTML und HTML 5

- HTML ist eine Auszeichnungssprache, keine Programmiersprache
- ähnliches, um Textsatz für Printmedium ergänztes Konzept: LaTeX

SYNTAX (1/2)

- textbasierte Auszeichnungssprache zur semantischen Strukturierung elektronischer Dokumente
  - Texte mit Hyperlinks
  - Medieninhalte
- Auszeichnungen (Markup) von Textteilen zur Strukturierung
- Markups werden durch Tags markiert (Einzel-Tag oder Tag-Paar)
  - Tags beginnen mit < und enden mit >
  - End-Tags beginnen mit </
  - Einzel-Tags beginnen mit < und enden mit />
- Tags enthalten den Elementnamen und eine optionale Attributliste
- zusammengehörenden Start- und End-Tags bilden mit dem dazwischenliegenden Inhalt ein Element
- Elemente sind verschachtelbar (siehe DTD)

Syntax (2/2)

- bestimmte Elemente müssen nicht explizit notiert werden (HTML4)
- einigen Elementen darf End-Tag fehlen (, <li>, <dd>, <br>, …) (HTML4)
- Groß- und Kleinschreibung der Tags irrelevant (HTML4)
- inhaltsleere Elemente möglich, müssen einelementig schließen (
, <hr />, ...) (XHTML, HTML5)
- darstellungsorientierte Attribute möglich (align="…", …) (HTML4, XHTML)
- darstellungsorientierte Attribute verboten (stattdessen style="…" (unerwünscht) oder CSS (bevorzugt)) (HTML5)
- Elemente werden strukturiert ausgezeichnet
  - Textbereichen wird Bedeutung zugeordnet (<h1>Überschrift</h1>, Absatz , betonter Text, ...)
  - HTML gibt Anweisungen für das Parsen
    - Quelltextes einlesen und Informationen verarbeiten: Parsen
    - Aufbereitung für das Ausgabemedium: Rendern

ZEICHENVORRAT

- ursprünglich 7-Bit-ASCII
- ergänzt um zahlreiche, als HTML-Entität kodierte Sonderzeichen (bspw. ä kodiert als `&auml;`, `&#228;`, `&#x00E4;`)
- Unterstützung universeller Zeichensätze erst mit UTF
- abwärtskompatibel auf plattformunabhängige Portierbarkeit angelegt
- Wahl des zugrundeliegenden Zeichenvorrats sollte in Meta-Angaben für Browser hinterlegt sein

Achtung! URLs werden nach MIME-Verfahren in ASCII-Zeichen kodiert (`%20` für Leerzeichen, `%E4` für ä, ...)

Wichtige Tags (1/2)

`<html>…</html>`	Wurzelelement
`<head>…</head>`	Dokumentkopf mit Meta-Infos
`<title>…</title>`	Dokumenttitel
`<meta name="…" content="…" />`	Meta-Information
`<body>…</body>`	Dokumentinhalt
`<h1>…</h1>, ..., <h5>…</h5>`	Überschriften 1. bis 5. Ebene
`<p>…</p>`	Absatz
` `	erzwungener Zeilenumbruch
`<hr />`	Trennlinie
`<img src="…" />`	Bild
`<div>…</div>`	Inhalts-Container
`<script>…</script>`	Client-seitig zu kompilierende und auszuführende Programme

Wichtige Tags (2/2)

`<em>…</em>`, `<strong>…</strong>`	Hervorhebung (kursiv, fett)
`<ul>…</ul>`	ungeordnete Liste
`<ol type="…" >…</ol>`	geordnete Liste (vom Typ 1, A, a, I oder i)
`<li>…</li>`	Listenelement (innerhalb `ul`/`ol`)
`<span>…</span>`	Inhaltsbereich (oft zum Ansteuern und/oder Stylen von Inline-Textpassagen)
`<sup>…</sup>`, `<sub>…</sub>`	hochgestellt, tiefgestellt
`<object data="…">…</object>`	externe Ressource
`<iframe src="…">…</iframe>`	eingebetteter Frame

Anker

- wichtigstes Element in HTML
  → macht aus gewöhnlichem Text Hypertext

- definiert adressierbare Anker (Anchor) im Quelltext
  → entsprechen Positionen im angezeigten Dokument

```
<a name="ankermarke">Anzeigetext</a>
```
(HTML 4, XHTML)

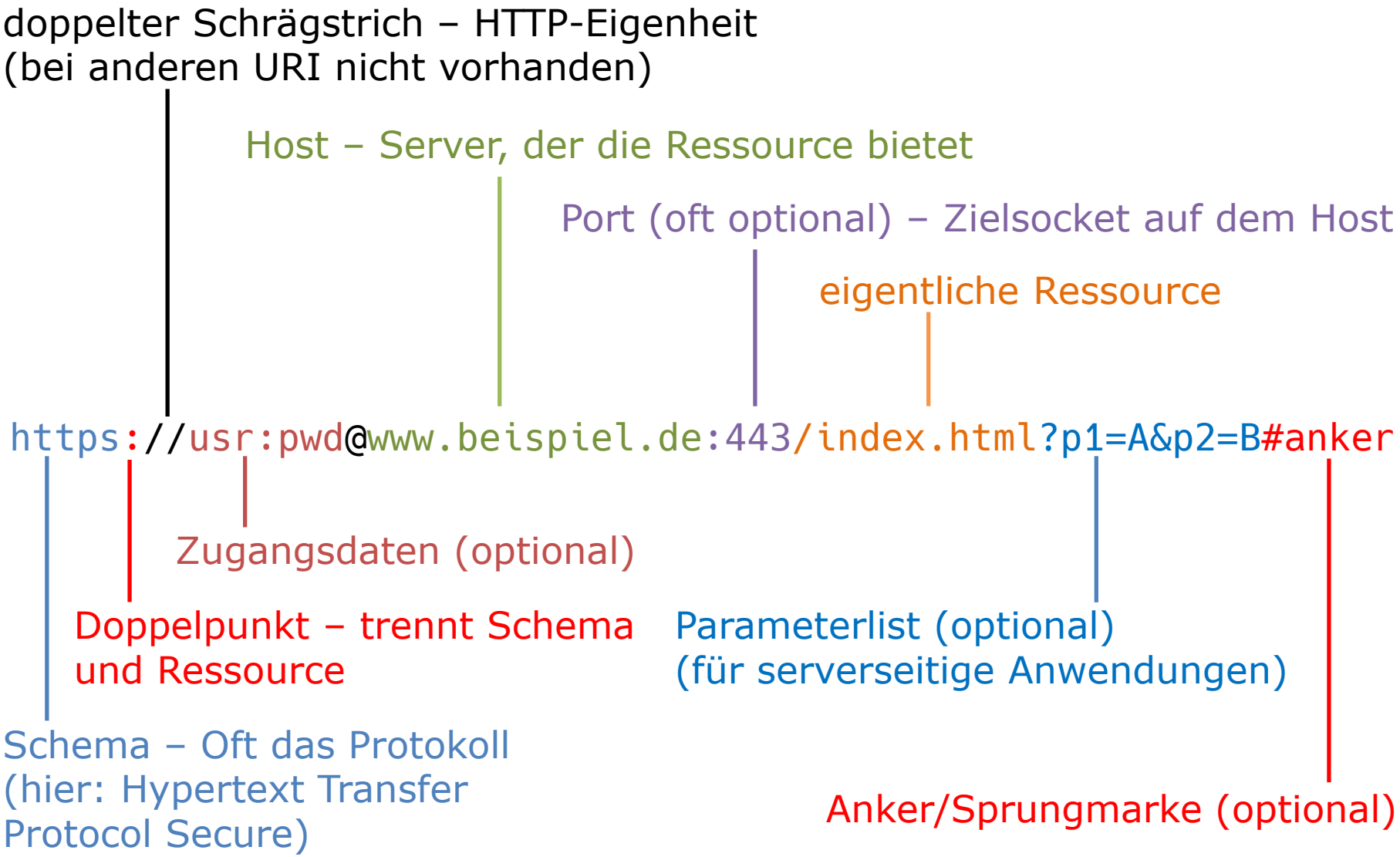
```
<… id="sprungmarke">Anzeigetext</…>
```
(HTML 5)

- setzt Sprungziele (Link) zu anderen Dokumenten oder Ankern
  → oft als Uniform Resource Identifier (URI) oder Locator (URL)

```
<a href="schema:ressource" target="ziel">Anzeigetext</a>
```

  - unterschiedliche Protokolle und Ressourcentypen als Ziel in der Hypertextreferenz (`href`) möglich
  - angeforderte Ressource kann (optional) mittels `target` an ein Ziel gegeben werden (`_self`, `_blank`, `_parent`, `_top`, `Framename`)

URI und URL


doppelter Schrägstrich – HTTP-Eigenheit
(bei anderen URI nicht vorhanden)
Host – Server, der die Ressource bietet
Port (oft optional) – Zielsocket auf dem Host
eigentliche Ressource
https://usr:pwd@www.beispiel.de:443/index.html?p1=A&p2=B#anker
Zugangsdaten (optional)
Doppelpunkt – trennt Schema
und Ressource
Parameterlist (optional)
(für serverseitige Anwendungen)
Schema – Oft das Protokoll
(hier: Hypertext Transfer
Protocol Secure)
Anker/Sprungmarke (optional)

ABSOLUTE UND RELATIVE URL

Beispiele für absolute URL

- `https://de.wikipedia.org/wiki/URL#Beispiele`
- `ftps://pi:raspberry@172.20.209.100:22/~/Downloads/`
- `mailto:tenshi.hara@ba-sachsen.de`
- `file:///F:/Lehre/GLWEP/1%20-%20W3C-Standards.pptx`

Beispiele für relative URL

- `//de.wikipedia.org/wiki/Datei:Vivaldi.png` – gleiches Protokoll
- `/wiki/Vivaldi_(Browser)` – gleiches Protokoll, gleicher Host (entspricht absolutem Pfad)
- `Vivaldi_(Browser)` – gleiche Pfadebene
- `#Versionen` – Anker in der gleichen Ressource
- `#` – die Ressource selbst (oft bei jQuery, da href im a-Tag nicht leer sein soll)

Hallo-Welt-Web-Seite (HTML 5)

```
<!DOCTYPE html>
<html>
   <head>
      <title>Hallo Welt!</title>
      <meta name="author" content="Tenshi Hara" />
      <meta http-equiv="content-language" content="de" />
   </head>
   <body>
      <h1>Hallo Welt!</h1>
      <p>Das ist meine erste Webseite.</p>
      <hr />
   </body>
</html>
```

kann in den gängigen Browsern weggelassen werden
(für absolut konforme Dokumente aber notwendig!)

Hallo-Welt-Web-Seite (XHTML 1.1)

bei XHTML notwendig!

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
          "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <title>Hallo Welt!</title>
      <meta name="author" content="Tenshi Hara" />
      <meta http-equiv="content-language" content="de" />
   </head>
   <body>
      <h1>Hallo Welt!</h1>
      <p>Das ist meine erste Webseite.</p>
      <hr />
   </body>
</html>
```

FORMULARE

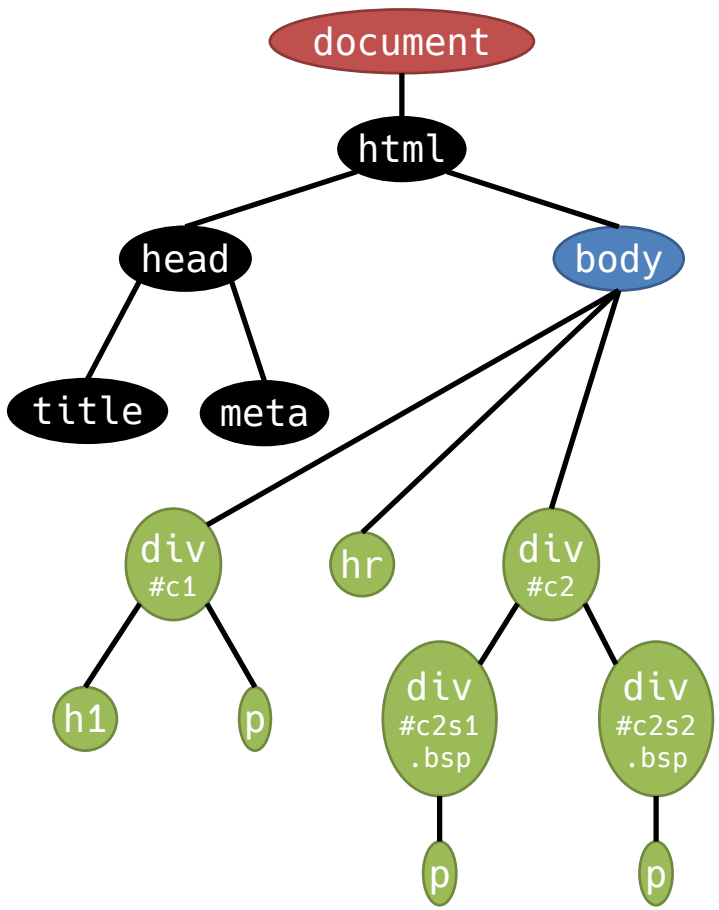
- Schlüsselelement zum Senden von Daten an den Server
- zwei grundlegende Möglichkeiten zum Senden
  - GET Daten werden in der URI als Parameter übertragen
    (z.B. `https://example.com/form.html?p1=Test&p2=null`)
  - POST Daten werden im Header übertragen
    (in Kommunikationsprotokollen erscheinen sie nicht in der URI)
- grundlegender Tag: `<form>`
  - innerhalb jedes Formulars gibt es Eingabe-Elemente (`<input>`) mit verschiedenen Typen (Text, Zahl, Datum, Datei, …)
  - mindestens einen Mechanismus zum Absenden
    (i.d.R. `<input type="submit">` oder JavaScript `form.submit()`)
  - ggf. einen Mechanismus zum Zurücksetzen
    (z.B. `<input type="reset">`)

Document Object Model (DOM)

Document Object Model (DOM)

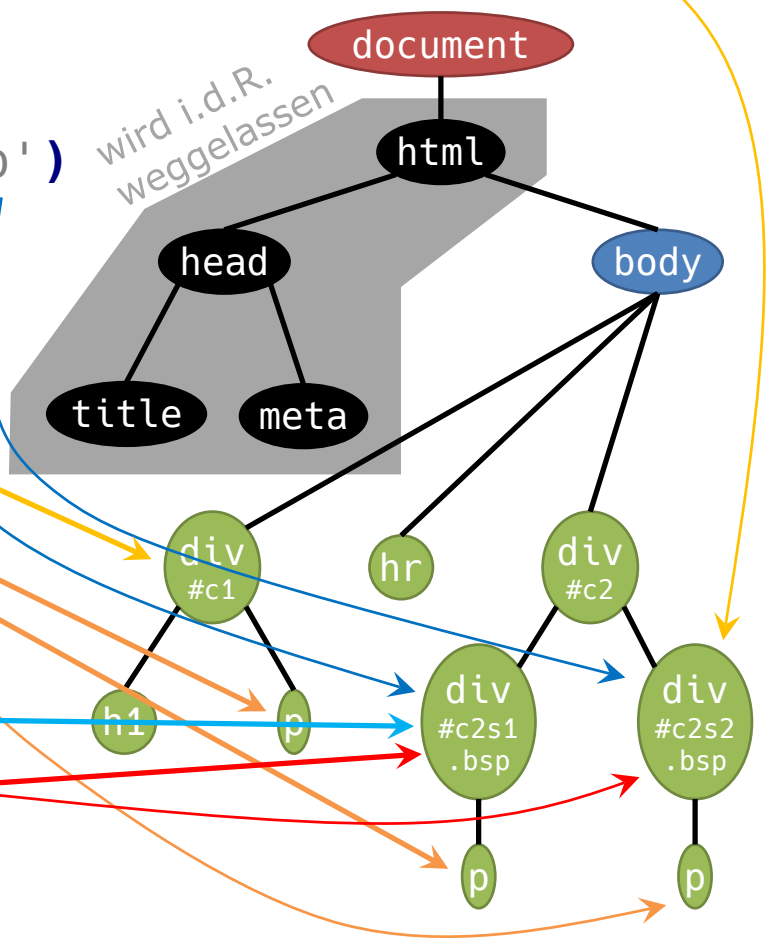
- alle HTML-Dokumente lassen sich anhand ihrer Struktur in einen Baum überführen, nämlich das Objektmodell des Dokuments (DOM)

```
<html>
  <head>
    <title>Hallo Welt! - Version 2</title>
    <meta name="author" content="Tenshi Hara" />
  </head>
  <body>
    <div id="c1">
      <h1>Hallo Welt!</h1>
      <p>Sei gegr&uuml;&szlig;t, liebe Welt.</p>
    </div>
    <hr />
    <div id="c2">
      <div id="c2s1" class="bsp">
        <p>DOM-Beispiel</p>
      </div>
      <div id="c2s2" class="bsp">
        <p>Im Rahmen von 3MI-GLWEB-20</p>
      </div>
    </div>
  </body>
</html>
```



DOM-Zugriffe

- vollständiger Pfad:
  `document.body.div[2].div[2]`
- surjektive Klasse:
  `document.getElementsByClassName('bsp')`
- bijektiver Identifier:
  `document.getElementById('c1')`
- Element-Typ:
  `document.getElementsByTagName('p')`
- Query-Selektoren
  `document.querySelector('#c2s1')`
  `document.querySelectorAll('.bsp')`

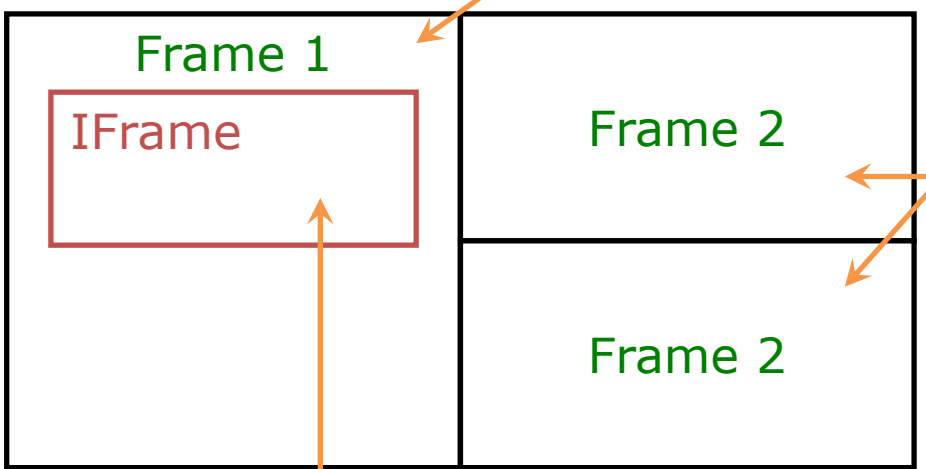


Multiple Dokumente im DOM (Frames) (1/2)

- Web-Seiten können bestehen aus
  - einem einzelnen Dokument (Single Page Application)
  - mehreren Dokumenten, die
    - nacheinander geladen werden
    - gleichzeitig geladen werden (Framed Application)
    - teilweise gleichzeitig, teilweise nacheinander geladen werden (auch das ist eine Framed Application)
- Framed Applications enthalten mehrere definierte Anzeigebereiche (Frames), welche in Framesets organisiert sind
  - Frames können statisch definiert (Frameset) oder dynamisch im Inhalt (IFrame) eingebunden sein
  - Adressierung von Frames erfolgt im DOM mittels
    `document.window.frames.framename`
    → darin dann normaler DOM

Multiple Dokumente im DOM (Frames) (2/2)

```
<!DOCTYPE html>
<html>
<frameset cols="50%,50%">
    <frame src="frame_1.html">
    <frameset rows="50%,50%">
        <frame src="frame_2.html">
        <frame src="frame_2.html">
    </frameset>
</frameset>
</html>
```




```
<!DOCTYPE html>
<!-- iframe.html -->
<html>
<head><style>h1{color:dark-red;}</style></head>
<body><h1>IFrame</h1></body>
</html>
```

```
<!DOCTYPE html>
<!-- frame_1.html -->
<html>
<head>
  <style>
     h1{color:dark-green;text-align:center;}
  </style>
</head>
<body>
  <h1>Frame 1</h1>
  <iframe src="iframe.html"></iframe>
</body>
</html>
```

```
<!DOCTYPE html>
<!-- frame_2.html -->
<html>
<head>
  <style>
    .absmid{color:dark-green;position:
      relative;top:50%;-webkit-transform:
      translateY(-50%);-ms-transform:
      translateY(-50%);transform:
      translateY(-50%);text-align:center;}
  </style>
</head>
<body>
  <div class="absmid">Frame 2</div>
</body>
</html>
```

Ereignisse

Interaktionen mit dem DOM lösen Ereignisse aus

→ können weiterverarbeitet werden, bspw. durch CSS oder JavaScript

`onLoad`	beim Laden des DOM	insbesondere beim Laden und Verlassen einer Seite
`onUnload`	beim Verlassen des DOM	
`onChange`	bei Veränderung eines DOM-Objekts	
`onMouseOver`	sobald Maus über DOM-Objekt bewegt wird	
`onMouseOut`	sobald Maus DOM-Objekt verlässt	
`onMouseDown`	beim Klickstart auf ein DOM-Objekt	
`onMouseUp`	beim Klickende auf ein DOM-Objekt	
`onClick`	beim Anklicken eines DOM-Objekts (d.h. komplettes Ereignis aus Klickstart und Klickende)	
`onFocus`	beim Fokussieren eines DOM-Objekts	

Ereignisse

DOM kann durch Ereignisse verändert werden

→ können bspw. durch CSS oder JavaScript ausgelöst werden

```
<!-- Ändert Textfarbe bei Berührung mit der Maus. -->
<div style="color:red;"
     onMouseOver="this.style.color='green';"
     onMouseOut="this.style.color='red';">
   Beispiel
</div>

<!-- Ersetzt die Schaltfläche durch statischen Text. -->
<div>
   <button onClick="this.parentElement.innerHTML=Date();">
      Zeitstempel
   </button>
</div>
```

Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS)

- zum Layouten und Designen von Web-Seiten notwendig
  (da HTML nur Struktur des Dokuments angibt)
- kann im HTML-Dokument selbst definiert werden
  - im Header oder Body mittels `style`-Tag und DOM-Referenzen
  - im jeweiligen HTML-Tag mittels `style`-Attribut
- alle Browser haben ein Standard-Stylesheet, welches verwendet wird, wenn keine CSS vorgegeben werden (i.d.R. weißer Hintergrund, schwarzer Text, Schriftgröße des Systems, unterstrichene Links, etc.)

EXTERNE STYLESHEETS

- erhöhen Wiederverwendbarkeit von Styles in vielen HTML-Dokumenten
- werden eingebunden
  - mittels style-Tag im HTML-Header oder HTML-Body

```
<style type="text/css">
   @import url(/my.css);
</style>
```

  - mittels link-Tag im HTML-Header

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

  - mittels xml-stylesheet-Referenz im XHTML-Header

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

Grundlegender Aufbau von CSS

EBNF:

```
(
   Selektor ( "," Selektor )* "{"
      ( Eigenschaft ":" Wert ";" )*
   "}"
)*
```

Beispiel:

```
.box, textarea, input.new, #boxart {
   /*
    * gestaltet alle Elemente der Klasse „box“,
    * alle Textfelder, alle Eingabebereiche der Klasse
    * „new“ und das Element mit der ID „boxart“.
    */
   height: 25%;
   width: 30em;
   border: solid black 2pt;
}
```

CSS-Attribute

- dienen dazu, bestimmte Eigenschaften von Elementen zu verändern
- differenzierbar in
  - visuelle Eigenschaften: verändern das Aussehen
  - platzierende Eigenschaften: verändern die Positionierung

Beispiele:

Attribut	Wertbeispiele	Bedeutung
`width`	`25%`, `120px`, …	Elementbreite
`border`	`2px`, `solid black 1pt`, …	Rahmen um Element
`position`	`absolute`, `fixed`, …	Positionierungsreferenz
`left`	`50rem`, `300`, …	Abstand vom linken Rand
`margin`	`1em`, `0 4px 8px 2px`, …	Abstand zum nächsten Element
`visibility`	`visible`, `hidden`, …	Sichtbarkeit
`display`	`block`, `none`, …	Art der Darstellung

Wertzuweisung

- Elemente erben implizit Attributwerte ihrer Eltern (explizit mit `inherit`)
- Werte können absolut (z.B. `14pt`) oder relativ (z.B. `1.2em`) sein
- Positionierungsangaben kann sich beziehen auf
  - das Dokument (`position:absolute`)
  - das Anzeigefenster (`position:fixed`)
  - das umgebende Elternelement (`position:relative`)
    (innerhalb des Elternelements absolut!)
  - die Angabe im Elternelement (`position:inherit`)
  - die benachbarten Elemente (`float:left`|`right`|`both`)
    (Fließumgebung wird mit `clear:left`|`right`|`both` und ggf. mit `clear:inherit`|`initial` aufgehoben)

Längenangaben

`%`	relativ zur Größe des Elternelements
`mm`, `cm`, `in`	Größe entsprechend definierter Eichskala (in Millimeter, Zentimeter oder Zoll)
`em`	Größe relativ zur aktuell berechneten Schriftgröße (z.B.: Schriftgröße 12px → `1.5em` entsprechen dann 18px)
`ex`	Größe relativ zur Größe des Minuskel-x der aktuellen Schriftart und berechneten Schriftgröße
`px`	Größe in Pixel (wahrnehmbare „echte“ Bildschirmpixel)
`pt`	Größe in Satzpunkten (1 Satzpunkt entspricht $\frac{1}{72}$ Zoll)
`pc`	Größe in Pica (1pc $\stackrel{\text{def}}{=}$ 12pt, nämlich $\frac{1}{6}$ Zoll)

Selektoren und Kombinatoren

Selektor	Auswahl
`*`	jedes Element
`T`	Elemente vom Typ `T`
`.c`	Elemente der Klasse `c`
`#id`	Element mit dem Identifikator `id`
`[attr]`	Elemente mit gesetztem Attribut `attr`
`[attr=val]`	Elemente, deren Attribut `attr` den Wert `val` hat
`[attr~=val]`	Elemente, deren Attribut `attr` das Wort `val` enthalten (d.h. bspw. „`val wal`“, nicht aber „`valwal`“)
`[attr|=val]`	Elemente, deren Attribut `attr` den Wertpräfix `val` hat

Kombinator	Auswahl
`P Q`	Selektor `Q` als Nachfahren des Selektors `P`
`P > Q`	Selektor `Q` als Kind des Selektors `P`
`P + Q`	Selektor `Q` auf gleicher DOM-Ebene mit direktem Vorgänger Selektor `P`

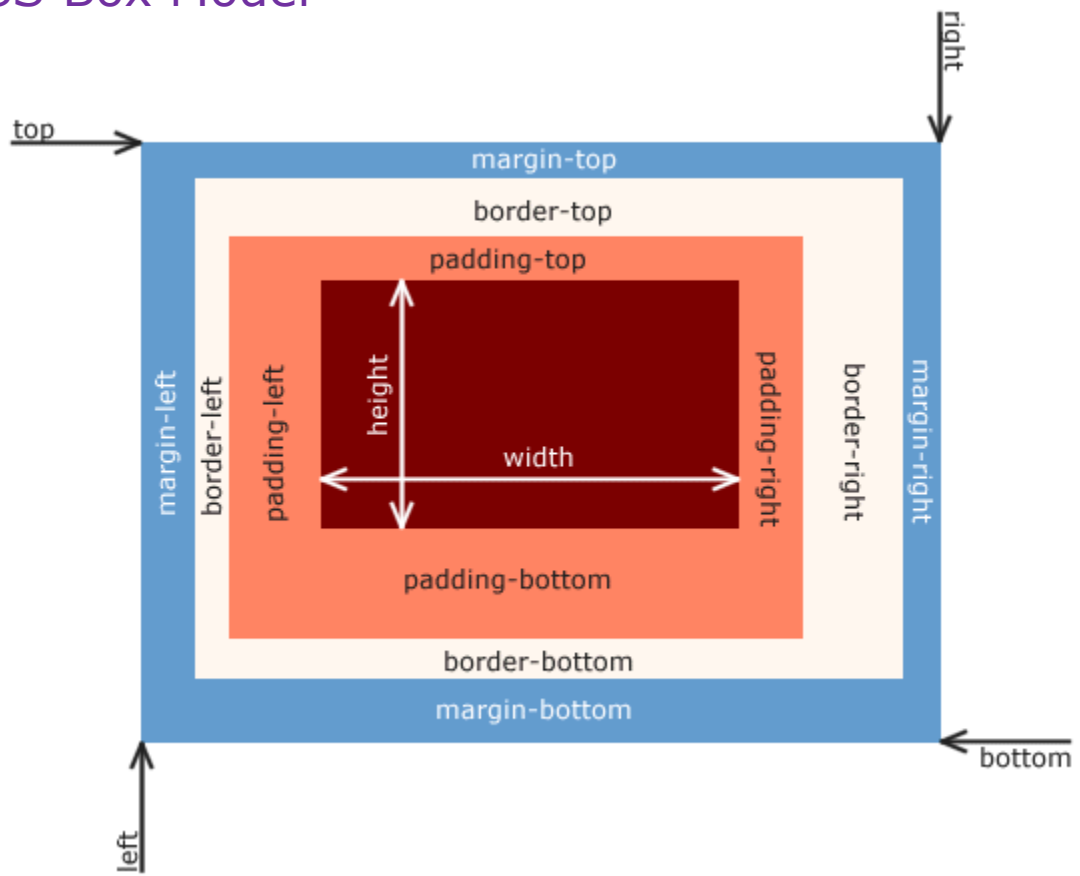
Pseudoklassen und Pseudoelemente

Pseudoklasse	Auswahl
`:link`	noch nicht besuchte Verweise
`:visited`	besuchte Verweise
`:active`	gerade angetipptes/angetaptes Element
`:hover`	vom Mauszeiger/Schwebefinger berührtes Element
`:focus`	mit Fokus belegtes Element
`:lang( )`	Elemente, deren Attribut lang eine Sprache festlegen (z.B. `lang="JavaScript"`)

Pseudoelement	Auswahl
`::first-line`	erste Zeile eines formatierten Textes
`::first-letter`	erstes Zeichen eines formatierten Textes
`Q::before`	erzeugt ein Element direkt vor allen Instanzen des Selektors Q (und wählt dieses aus)
`Q::after`	erzeugt ein Element direkt hinter allen Instanzen des Selektors Q (und wählt dieses aus)

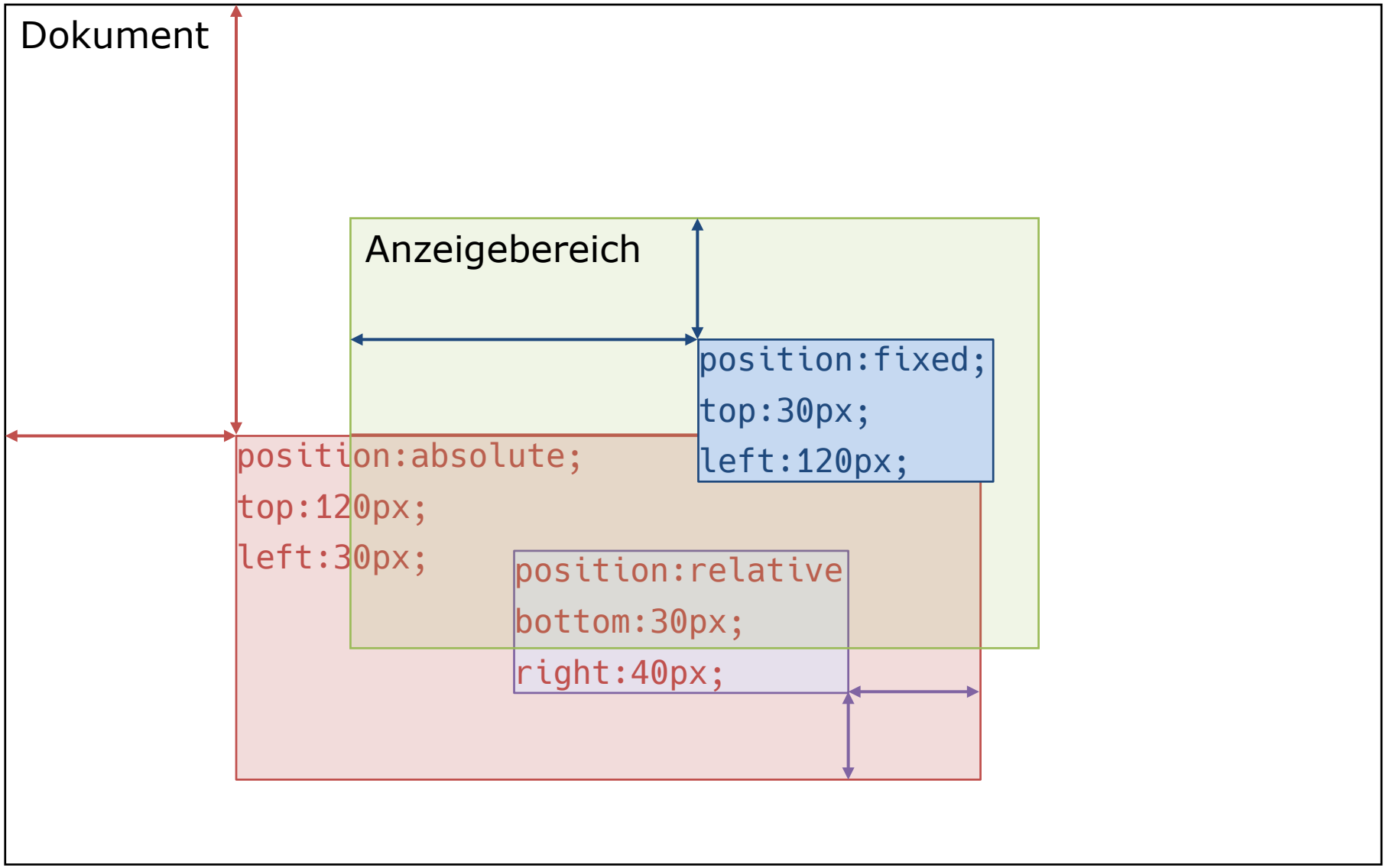
POSITIONIERUNG (1/3)

CSS Box Model

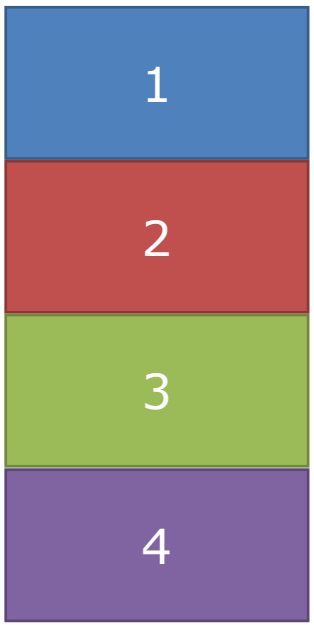


Public Domain, via Wikimedia

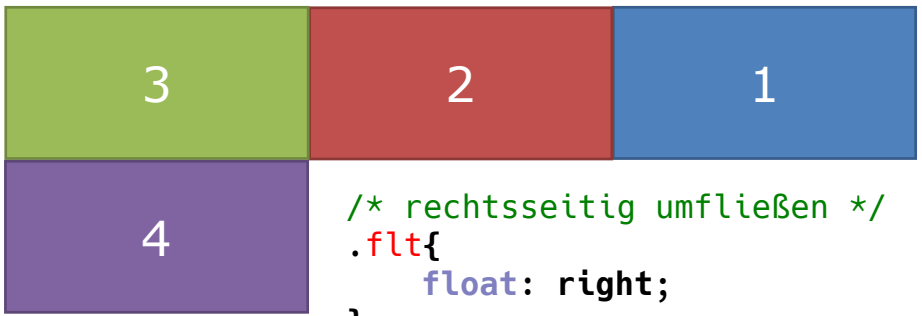
POSITIONIERUNG (2/3)



Positionierung (3/3)




```
/* Standardverhalten
 * entspricht: */
.flt{
    float: none;
    clear: both;
}
.clr{
    clear: right;
}
/* Kann also weg
 * gelassen werden */
```




```
/* rechtsseitig umfließen */
.flt{
    float: right;
}
.clr{
    clear: right;
}
```


```
/* linkssseitig umfließen */
.flt{
    float: left;
}
.clr{
    clear: left;
}
```

```
<!-- HTML-Quelle -->
<div class="box1 flt">1</div>
<div class="box2 flt">2</div>
<div class="box3 flt">3</div>
<div class="box4 clr">4</div>
```

```
/* allgemeines CSS */
.box1{background-color:blue;border:dark-blue solid 2pt;}
.box2{background-color:red;border:dark-red solid 2pt;}
.box3{background-color:green;border:dark-green solid 2pt;}
.box4{background-color:purple;border:dark-purple solid 2pt;}
.box1,.box2,.box3,.box4{color:white;}
```

Medien

- CSS kann unterschiedliche Zielmedien definieren

`all`	Stylesheet gilt für alle Ausgabegeräte
`speech/aural`	Screen Reader
`braille`	Blindenschriftausgabegeräte
`embossed`	Blindenschrift-Drucker
`handheld`	Handhelds
`print`	Drucker
`projection`	Beamer
`screen`	Bildschirme
`tty`	Fernschreiber und Terminals
`tv`	Fernseher

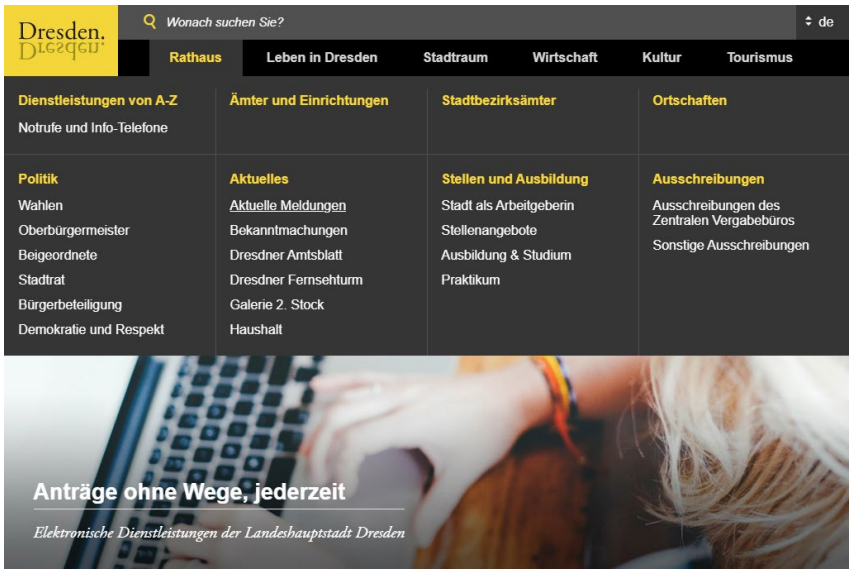
- mehrere Medientypen können in einem CSS definiert werden und sich bei Bedarf überschreiben
- Definition der einzelnen Medientypen mittels `@media` im CSS oder `media`-Attribut im `link`- oder `style`-Tag

Hypertext-Navigation

NAVIGATIONSTECHNIKEN

- für jede Webseite muss ein Navigationskonzept erstellt werden
- „Lost in (Hyper)Space"-Problem soll vermieden werden
  → Keine Sackgassen!
- Benutzer soll vorwärts und rückwärts durch die Liste bisher besuchter Seiten wandern können
- Verwenden von bekannten Metaphern
- Leitfragen von Nivergelt & Nielsen zu Plänen der Benutzenden:
  - Wo bin ich? Was kann ich hier tun?
  - Wo war ich vorher? Wie komme ich dorthin zurück?
  - Wohin kann ich gehen? Wie komme ich dorthin?

NAVIGATIONSKOMPLEXITÄT



Formulare

- Auswahllisten, Suche...

Dynamische Menüs

Bäume

Inhaltsverzeichnis

Index/Glossar

Navigationstechnik – Karten

- Karten (Image Maps) enthalten an bestimmten Region (area) Verweise
- verschiedene Regionen führen zu verschiedenen URIs
- verschiedene Formen für Regionen können verwendet werden
- Datenstruktur für das Bedienelement

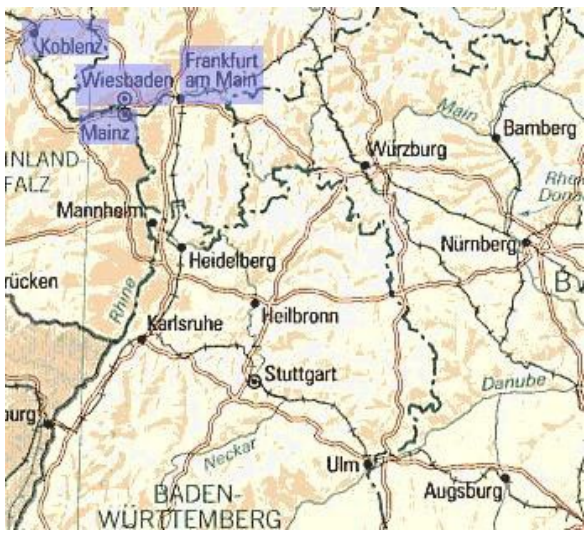
```
<map name="mapName">
<area xx="yy">
```

  - shape
  - coords
  - href
  - alt

- Verwenden der Karte: `<img usemap="#mapName" …>`

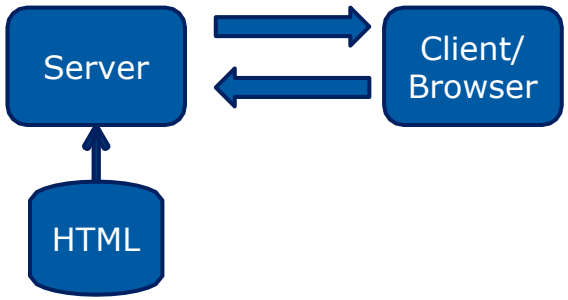
Navigationstechnik: Karten – Beispiel

```
<map name="mapOfGermany">
   <area shape="rect" coords="11,10,59,29"
      href="https://www.koblenz.de/"
      alt="Koblenz" />
   <area shape="rect" coords="42,36,96,57"
      href="https://www.wiesbaden.de/"
      alt="Wiesbaden" />
   <area shape="rect" coords="42,59,78,80"
      href="https://www.mainz.de/"
      alt="Mainz" />
   <area shape="rect" coords="100,26,152,58"
      href="https://www.frankfurt.de/"
      alt="Frankfurt" />
</map>
<p>
   <img src="karte.gif" width="345" height="312"
      border="0" alt="Karte" usemap="#mapOfGermany" />
</p>
```

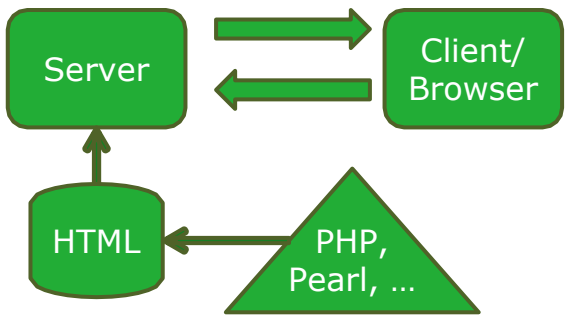


Statisches vs. Dynamisches Web

statisches Web
(Webserver liefert HTML aus)



dynamisches Web, serverseitig
(Webserver startet Programm,
das HTML generiert)



dynamisches Web, clientseitig
(Browser wertet Anweisungen
aus und verändert HTML)

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Exkurs: Barrierefreiheit im Web

Literatur:

Hellbusch: *Ba*
verstehen u
Webstanda
zugängli
Intern

Eingeschränktes Sichtfeld

erzeugen.

die Integration

nen in der Gesellschaft.

uns alltäglich geworden.

vielfältig:

nbehinderte Menschen

hörbehinderte Menschen

körperbehinderte Menschen

enschwache Menschen

alkulie)

geistig behinderte

ersbedingt sein.

Literatur:

Hellbusch: *Ba*
verstehen u
Webstand
zugängl
Inter

Eingeschränkte Farb- und Kontrast wahrnehmung

Medien können Barrieren erzeugen.

Barrieren verhindern die Integration behinderter Menschen in der Gesellschaft.
→ Das Web ist für uns alltäglich geworden.

Behinderung ist vielfältig:

- blinde und sehbehinderte Menschen
- gehörlose und hörbehinderte Menschen
- motorisch und körperbehinderte Menschen
- lese- und rechenschwache Menschen (Dyslexie, Dyskalkulie)
- lern-, sprach- und geistig behinderte Menschen

Behinderungen können altersbedingt sein.

Literatur:

Hellbusch: *Barrierefreiheit verstehen und umsetzen: Webstandards für ein zugängliches und nutzbares Internet*; dpunkt Verlag

Grauer Star

Medien können Barrieren erzeugen.

Barrieren verhindern die Integration behinderter Menschen in der Gesellschaft.

→ Das Web ist für uns alltäglich geworden.

Behinderung ist vielfältig:

- blinde und sehbehinderte Menschen
- gehörlose und hörbehinderte Menschen
- motorisch und körperbehinderte Menschen
- lese- und rechenschwache Menschen (Dyslexie, Dyskalkulie)
- lern-, sprach- und geistig behinderte Menschen

Behinderungen können altersbedingt sein.

Literatur:

Hellbusch: *Barrierefreiheit verstehen und umsetzen: Webstandards für ein zugängliches und nutzbares Internet*; dpunkt Verlag

Barrieren verstehen

Medien können Barrieren erzeugen.

Barrieren verhindern die Integration behinderter Menschen in der Gesellschaft.
→ Das Web ist für uns alltäglich geworden.

Behinderung ist vielfältig:

- blinde und sehbehinderte Menschen
- gehörlose und hörbehinderte Menschen
- motorisch und körperbehinderte Menschen
- lese- und rechenschwache Menschen (Dyslexie, Dyskalkulie)
- lern-, sprach- und geistig behinderte Menschen

Behinderungen können altersbedingt sein.

Literatur:

Hellbusch: *Barrierefreiheit verstehen und umsetzen: Webstandards für ein zugängliches und nutzbares Internet*; dpunkt Verlag

Tutorials zur Barrierefreiheit

https://www.heise.de/developer/artikel/Barrierefreiheit-Stolpersteine-bei-mobilen-Anwendungen-ueberwinden-Teil-1-4476673.html

https://www.heise.de/hintergrund/Accessibility-im-Web-Teil-1-Erstellen-einer-Webanwendung-7368850.html

Barrierefreiheit als Ziel

Von barrierearmen Webseiten profitieren alle Nutzenden!

- leichter bedienbare GUI → ISO 9241-151:2008:
  zurückgezogener Standard, enthält aber viele sinnvolle Tipps und Ideen
- hilft der Usability → ISO 9126 (DIN 66272) und DIN EN ISO 9241:
  enthalten Definitionen und Prinzipien zu
  - Funktionalität
  - Zuverlässigkeit
  - Benutzbarkeit
  - Effizienz
  - Änderbarkeit
  - Übertragbarkeit
  - Aufgabenangemessenheit
  - Selbstbeschreibungsfähigkeit
  - Erwartungskonformität
  - Erlernbarkeit
  - Steuerbarkeit
  - Robustheit gegen Benutzerfehler
  - Benutzerbindung
- befördert gute SEO-Scores

Formularelemente

FORMULARE

- dienen der Übertragung von Informationen vom Client an der Server
- können vom Client mit Inhalten befüllt werden
  → dynamische Informationen ggü. statischer Links
- bilden die Grundlage des Rückkanals im interaktiven Webs

FORMULARELEMENTE

`input`	Eingabeobjekt eines bestimmten Typs (nächste Folie) (kann alle Formularelemente außer `textarea`, `select` und `option` ersetzen)
`button`	Schaltfläche
`checkbox`	Markierungsfeld (Mehrfachauswahl möglich)
`radio`	Auswahlfeld (nur eine Auswahl möglich)
`textarea`	Textfeld (i.d.R. für mehrzeilige Eingaben)
`select`	Ausklappmenü (d.h. Dropdown-Menü)
`option`	Auswahloption in einem Ausklappmenü
`data-*`-Tag	entspricht verborgenem Datenspeicher (ab HTML5) (`<input type="hidden" name="meinKey" value="meinWert" />` wird abgebildet auf `<div data-meinKey="meinWert"></div>`; Abfrage mit JavaScript: `var val=this.getAttribute("data-meinKey");`)

INPUT-ELEMENT

*: mit HTML5 eingeführt

`button`	Schaltfläche
`checkbox`	Markierungsfeld
`color`	Farbauswahldialog*
`date`	Datumsauswahldialog*
`datetime-local`	Datum und Zeit im lokalen Format*
`email`	Textfeld für E-Mail-Adressen*
`file`	Dateiauswahldialog
`hidden`	verborgenes Wertfeld
`image`	Bild
`month`	Monatsauswahldialog*
`number`	Textfeld für Zahlen*
`password`	Textfeld mit verdeckter Eingabe
`radio`	Auswahlfeld
`range`	Schieberegler im def. Wertebereich*
`reset`	Formular zurücksetzen
`search`	Textfeld als Suchfeld*
`submit`	Formular absenden
`tel`	Textfeld für Telefonnummern*
`text`	Textfeld
`time`	Uhrzeitauswahl*
`url`	Textfeld für URLs*
`week`	Wochenauswahldialog*