openHPI

Woche 2: Hypertext, HTML und XML

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

WWW-Dokumente
Hypertext und Hypermedia (1/2)

WWW ist ein **Hypermedia-System** auch **Hypertext-System**, **WWW-Dokumente** sind durch **Hyperlinks** untereinander verbunden und auf über die Welt verteilten Servern gespeichert. Das WWW wird so zum **verteilten** Hypermedia-System

- Hypermedia-Dokumente überwinden die traditionelle, lineare Dokumentenstruktur – mit dem Link-Mechanismus werden sie Teil eines riesigen Informationsnetzwerks
- Nutzer können ein WWW-Dokument direkt von einem anderen WWW-Dokument aus erreichen, selbst wenn diese auf unterschiedlichen Servern gespeichert sind
- Für die auf verschiedenen Servern gespeicherten WWW-Dokumente ist es schwierig, die Konsistenz von Hyperlinks zu gewährleisten

Hypertext und Hypermedia (2/2)

WWW-Dokumente werden typischerweise mit einer speziellen deskriptiven Sprache beschreiben: der **Hypertext Markup Language – HTML**

Mit HTML können Autoren:

- **Struktur** eines Dokuments beschreiben, also z.B. Unterteilung in (Unter-)Überschriften und Absätze, Listen, Tabellen, …
- Hyperlinks in die Dokumente einbetten,
- Multimedia-Komponenten einbinden und
- (bis zu einem gewissen Grad) die graphische Aufbereitung beeinflussen

Als WWW-Dokumente werden aber nicht nur HTML-Dokumente bezeichnet, sondern alle Dokumente, auf die im WWW zugegriffen werden kann

HTML-Dokumente (1/3)

- HTML-Dokument heißt **Page** (dt. Seite)
- Die Startseite oder Eingangsseite für die Navigation durch ein Informationsangebot im WWW nennt man **Homepage**.
  Zunehmend hat sich dieser Begriff aber auch als Bezeichnung für das gesamte Informationsangebot eines Anbieters (die **Website**) durchgesetzt
- Browser fordern ein HTML-Dokument an, interpretieren es und stellen es dann anhand der dort angegebenen HTML-Elemente dar.
- Jedes HTML-Element ist durch einen **Tag** gekennzeichnet. Tags sind als **Markups** in das Dokument eingebettet

HTML-Dokumente (2/3)

Markup-Sprachen sollten nur für die Beschreibung der Dokumentenstruktur genutzt werden, nicht jedoch für die Beschreibung der Präsentation des Dokuments

- Grundlegende Idee von Markup-Sprachen ist die **Trennung** von **Dokumentenstruktur** und **Dokumentengestaltung**
- Mit **HTML** wird die Dokumentstruktur beschrieben
- **Cascading Stylesheets – CSS** – sind verantwortlich für die Beschreibung der Gestaltung (das Layout) dieser Struktur
  - Wichtigstes Anwendungsgebiete für CSS:
    - exakte Definition des Layouts eines HTML-Dokuments
    - Anpassung an unterschiedliche Ausgabemedien
    - zentrales Management von Layouts

HTML-Dokumente (3/3)

- Mit Hilfe spezieller Tags können Referenzen als **Links** in HTML-Dokumente eingebaut werden, die dort als passive „Wegweiser" zu anderen Dokumenten dienen
- Die global eindeutige Identifikation der verlinkten Dokumente erfolgt mit Hilfe von **URIs** (Uniform Resource Identifiers)
- URIs können als Zeichenketten („Strings") kodiert werden und geben in der Ausprägung als **URLs** den „Ort" (Server) an, an dem ein Dokument gespeichert ist:
  - Name des Zugriffsprotokolls (z.B. http oder https),
  - Name des Servers (meist Domain Name),
  - Pfad (in der Verzeichnisstruktur des Servers)
  - und (Datei-)Name des Dokuments

Dynamisches HTML

- **DHTML – Dynamisches HTML:** Ein HTML-Dokument kann während der Anzeige durch Benutzerinteraktion (oder andere Ereignisse) verändert werden, z.B. Ausklappen eines Menüs, Runterzählen eines Countdowns, ...
- Typische Technik dahinter ist das → **DOM-Skripting** (→ DOM – Document Object Model). Das sind Programme (Skripte), die im Browser ausgeführt werden und auf bestimmte Ereignisse (z.B. Maus-Clicks) reagierend das HTML-Dokument manipulieren
- Während der Anzeige eines Dokuments können Browser und Server mittels → **XMLHttpRequest** kommunizieren, bekannt z.B. aus: AJAX = **A**synchronous **J**avaScript **a**nd **X**ML

Werden uns in Kurswochen 3 und 4 detaillierter mit DOM-Skripting und AJAX bzw. allgemeiner mit der **Client-seitigen Web-Programmierung** befassen

XML

- Hauptkritikpunkt an HTML ist eingeschränkte Flexibilität aufgrund der in ihrer Bedeutung unveränderlich festgelegten Markups
- Mit der Einführung von **XML** – **Extensible Markup Language** – als Meta-Markupsprache ist es möglich, alle nur erdenklichen Dokumenttypen mit einer eigenen, speziellen Syntax zu beschreiben
- XML ist Ausgangspunkt für Reihe Anwendungs-spezifischer Markupsprachen, die genau auf die jeweiligen Anwendungsdomänen oder spezialisierte Ausgabegeräte zugeschnitten sind. Beispiele:
  - **MathML** – Mathematical Markup Language
  - **SVG** – Scalable Vector Graphics
  - **ODF** – Open Document Format for Office Applications
  - ...

Themen der zweiten Kurswoche (1/2)

HTML – Hypertext Markup Language

- **HTML-Dokumente:** Wie funktioniert HTML grundlegend und wie sind HTML-Dokumente aufgebaut?
- **HTML-Syntax und guter Stil:** Welche Regeln sind beim Aufbau von HTML-Dokumenten zu beachten und welche Konventionen gibt es?
- **HTML-Elemente:** Welche Tags gibt es für welche HTML-Elemente, (z.B. Listen, Bilder, Überschriften) und wie funktionieren Attribute?
- **HTML5:** Welche Neuerungen bringt die aktuelle HTML-Version – HTML5 – und welche Tags sollte man nicht mehr verwenden?

Themen der zweiten Kurswoche (2/2)

CSS – Cascading Stylesheets

- **CSS:** Wie funktioniert die Gestaltung von HTML-Dokumenten mit Cascading Stylesheets und wie baut man diese in HTML ein?
- **CSS Box Modell:** Wie funktioniert die Positionierung von HTML-Elementen mit CSS?
- **Responsive Web Design:** Wie kann man mit HTML und CSS Webseiten für verschiedene Endgeräte (z.B. Desktop und Smartphone) bauen?
- **CSS Präprozessoren:** Wie kann das syntaktisch etwas umständliche CSS eleganter formuliert werden?

XML – Extensible Markup Language

- **XML:** Wie kann man mit XML spezielle Dokumententypen für den generellen Austausch von Daten zwischen Computersystemen definieren?

openHPI

HTML-Dokumente

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

Erinnerung

Erfolg des WWW – World Wide Web – basiert im Wesentlichen auf seinen Inhalten, den verlinkten **Hypermedia-Dokumenten**

HTML – **Hypertext Markup Language** – ist die einfache Markupsprache zur Beschreibung dieser Dokumente

- HTML ist neben URIs und HTTP dritte Basiskomponente des WWW
- HTML gibt weder Format noch Layout eines Hypermedia-Dokuments vor, sondern beschreibt im Wesentlichen die **Struktur** der jeweiligen Inhalte
- Einfache Beschaffenheit von HTML ist wichtiger Grund für den Erfolg des WWW
- HTML baut ursprünglich auf der komplexeren Markupsprache **SGML – Standard Generalized Markup Language** auf, deren Prinzipien noch heute gelten

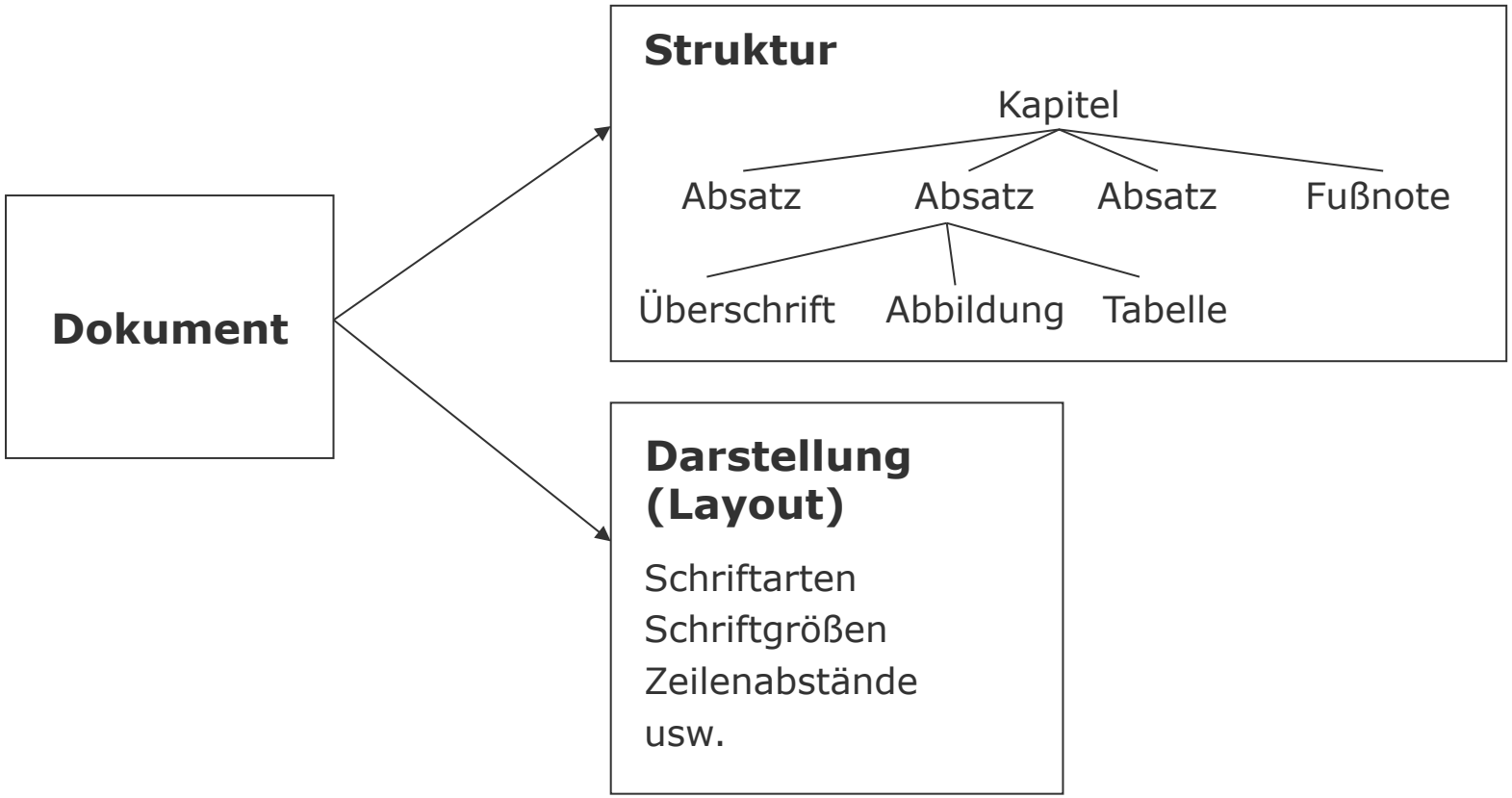
SGML (1/3)

SGML – **Standard Generalized Markup Language** – wurde in den 1960er entwickelt, als moderne Datenverarbeitung ihren Anfang nahm

- SGML sollte den Austausch strukturierter Daten ermöglichen, unabhängig von Herstellern, Geräten oder Plattformen
- SGML ist eine Meta-Markupsprache, in der sich unterschiedliche Datenformate (genauer: Markupsprachen für Datenformate) für alle möglichen (Text-)Dokumente beschreiben lassen
- **Grundlegende Idee**: Trennung von
  - **Dokumentstruktur** – logische Unterteilung und Abhängigkeiten der einzelnen Teile eines Dokuments – und
  - **Dokumentendarstellung** – Formatierung und Layout
- **Vorteil:** Aus einem SGML-Dokument könnten viele verschiedene Layout-Varianten generiert werden, z.B. für Druck im Magazin oder als Buch, für die Anzeige auf Computer oder Smartphone, ...

SGML (2/3)

Trennung zwischen Struktur und Darstellung



SGML (3/3)

- Strukturgebende Elemente werden vermittels sogenannter **Markups** mit unterschiedlichen **Tags** gebildet
  - Markups sind durch Kombinationen spezieller Zeichen gekennzeichnet und werden innerhalb des eigentlichen Dokuments gespeichert
    - Markup-Begrenzer – „**<...>**”, z.B. <kapitel> … </kapitel>
    - Interpreter können so die Tags leicht als solche erkennen
- Beziehungen zwischen den strukturellen Elementen eines SGML-Dokumentes werden durch den SGML **Dokumentenbaum** beschrieben
- SGML erlaubt als Metasprache die Definition eigener Dokumentklassen, z.B. für ein „Buch“, um Beschreibung von Instanzen dieser Klasse zu vereinheitlichen und **automatische Syntaxprüfung** zu ermöglichen
  - Dokumentenklassen werden mittels von **DTDs – Document Type Definitions** – festgelegt

Grundlegende Struktur eines HTML-Dokuments (1/6)

- **HTML** ist eine Markupsprache, die auf SGML basiert
- HTML ist **einfacher** als SGML, da es keine Meta-Markupsprache ist – HTML ist eine Sprache, die „in SGML formuliert" ist …
- HTML führt das Hyperlink-Konzept des WWW ein
- **HTML-Dokumente** werden im Browser dargestellt; dieser muss dazu einen HTML-Interpreter besitzen, der die einzelnen strukturgebenden Elemente im Dokument identifizieren und für die sachgerechte Anzeige vorbereiten kann

Grundlegende Struktur eines HTML-Dokuments (2/6)

- Historisch bedingt wird bei HTML die Trennung von Struktur und Darstellung nicht strikt verwirklicht, die SGML eigentlich fordert, sondern HTML bietet auch Reihe von Möglichkeiten zur Formatierung des Inhalts
- Dennoch können (und sollten) Autoren in HTML keine direkten Formatierungsanweisungen verwenden und stattdessen die Darstellung des Dokuments mittels separater Formatierungsanweisungen beschreiben
  → **Cascading Stylesheets (CSS)**
- Viele der ursprünglichen Formatierungsmöglichkeiten in HTML gelten heute als „ausgemustert"; insbesondere mit Einführung von HTML5 wurde eine ganze Reihe dieser alten Elemente gestrichen

Grundlegende Struktur eines HTML-Dokuments (3/6)

- Ein **HTML-Dokument** beginnt immer mit der Angabe der verwendeten HTML-Version, z.B.
  - `<DOCTYPE html>` für HTML5-Dokumente
- Ein HTML-Dokument besteht immer aus einem **Header** und einem **Body**
- Der **Header** enthält **Informationen über das Dokument**, die nicht zum eigentlichen Inhalt gehören, z.B.
  - Titel, Autor, Keywords, Kodierung, Sprache, …
- Der **Body** enthält **eigentlichen Inhalt** des Dokuments, z.B.
  - Überschriften, Absätze, Listen, Tabellen, Formulare, Referenzen zu Bildern, Hyperlinks, usw.

Grundlegende Struktur eines HTML-Dokuments (4/6)

Grundgerüst eines HTML-Dokuments

```
<!DOCTYPE html>
<html>
   <head>
      <title>Über openHPI</title>
      ...
   </head>
   <body>
      <h1>openHPI – die MOOC-Plattform des HPI</h1>
      <p>MOOCs – Massive Open Online Courses – bieten
         eine der innovativsten Lernformen: Frei über
         das Internet können Lernende auf multimediale
         Kursmaterialien zugreifen...</p>
      ...
   </body>
</html>
```

Header

Body

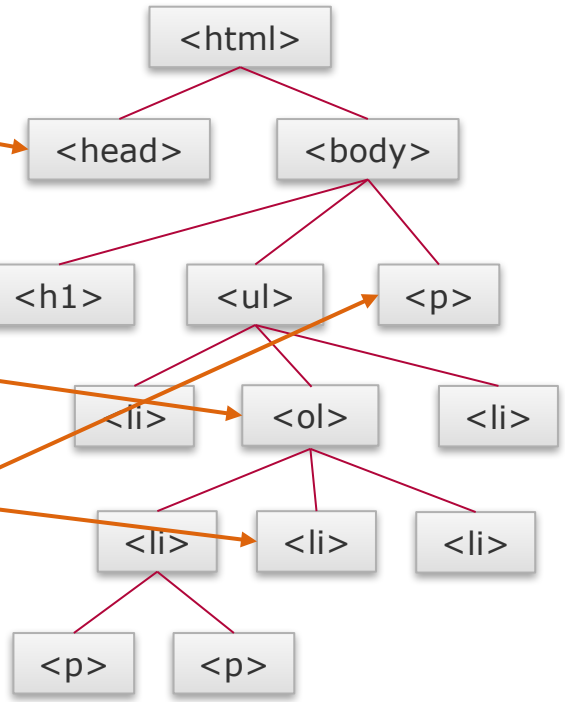
Grundlegende Struktur eines HTML-Dokuments (5/6)

- HTML-Elemente werden – wie in SGML – durch **Markup** deklariert
- Markup wird durch Markup-Bezeichner – sogenannte **Tags** – gekennzeichnet, die jeweils von den Markup-Begrenzern **<** und **>** umschlossen werden
- Strukturelle Elemente des HTML-Dokuments werden jeweils von einen **Start-Tag** und einem **End-Tag** eingefasst, z.B.
  - `<p> Absatz </p>` (engl. „Paragraph“)
- **End-Tags** können in einigen Fällen weggelassen werden
- Tags können **Attribute** für zusätzliche Informationen enthalten, z.B. `<p id="absatz1">...</p>`
- Die aktuelle HTML-Version – HTML5 – kennt etwa 120 verschiedene Tags

Grundlegende Struktur eines HTML-Dokuments (6/6)

- HTML-Elemente werde immer sequenziell angeordnet oder ineinander verschachtelt (niemals „über Kreuz“)
  ➔ dadurch ergibt sich Baumstruktur: **HTML-Dokumentenbaum**

```
<html>
  <head>
    ...
  </head>
  <body>
    <h1>Überschrift</h1>
    <ul>
      <li>Ein Listeneintrag</li>
      <ol>
        <li>
          <p>Erster Absatz</p>
          <p>Zweiter Absatz</p>
        </li>
        <li>Listeneintrag</li>
        <li>Listeneintrag</li>
      </ol>
      <li>Noch ein Listeneintrag</li>
    </ul>
    <p>Ein weiterer Absatz</p>
  </body>
</html>
```



Document Object Model – DOM

- **Document Object Model – DOM** – ist standardisierte Schnittstelle für **programmatischen Zugriff** auf Elemente eines HTML-Dokuments
- Zugriff auf DOM erfolgt mit Client-seitiger Programmierung (JavaScript)
- DOM erlaubt
  - Navigation durch Dokumentenbaum („Auswählen“ von einzelnen oder mehreren HTML-Elementen)
  - Hinzufügen, Löschen oder Verschieben von HTML-Elementen
  - Auslesen oder Ändern von Element-Inhalten (Text)
  - Auslesen oder Ändern von Attributen
- Beispiele:
  - `document.findElementsByTagname("p")` – selektiert alle <p>-Tags
  - `document.findElementById("Liste").children` – selektiert alle Unterelemente („Kindknoten“) des Elements mit der Id „Liste“

openHPI

HTML Syntax und guter Stil

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

HTML Syntax und guter Stil (1/4)

Neben den bereits vorgestellten Regeln zum HTML-Grundgerüst

- Angabe eines DOCTYPEs
- Kapselung des Dokuments in **<html>**-Tags
- Verwendung von Header und Body

gibt es weitere Regeln (**Syntax**), die HTML-Autoren beachten müssen oder Konventionen („**Coding Conventions**"), die sie beachten sollten:

- **Alle Tags schließen:** obwohl Browser recht „nachsichtig" bei der Interpretation von HTML sind und viele Autorenfehler automatisch beheben, sollten geöffneten Tags immer auch geschlossen werden – **„wohlgeformtes HTML"**
  - gut: `<div><p>Das ist ein Absatz.</p></div>`
  - schlecht: `<div><p>Das ist ein Absatz.</div>`
- ...

HTML Syntax und guter Stil (2/4)

- **Leere Elemente schließen:** Manche HTML-Elemente bestehen nicht aus Start-und End-Tag, da sie keine Inhalte haben können, z.B. der **
**-Tag für einen Zeilenumbruch
  - HTML5 fordert nicht, dass leere Elemente geschlossen werden, andere HTML-Versionen aber schon
  - gut: `<br />`
  - nicht ganz so gut: `<br>`
- **Keine kreuzweise Verschachtelung** von HTML-Tags: Enthält ein HTML-Element weitere Elemente, *müssen* diese korrekt verschachtelt sein
  - richtig: `<p><i>Kursiver Text</i></p>`
  - falsch: `<p><i>Kursiver Text</p></i>`
- ...

HTML Syntax und guter Stil (3/4)

- Einheitlich **Kleinbuchstaben** verwenden für **Tag-Namen**
  - gut: `<p>Das ist ein Absatz.</p>`
  - schlecht: `<P>Das ist ein Absatz</P>`
  - schlecht: `<P>Das auch.</p>`
- Einheitlich **Kleinbuchstaben** verwenden für **Attributnamen**
  - gut: `<p class="info">`
  - schlecht: `<p CLASS="info">`
- **Werte von Attributen in Anführungszeichen**
  - gut: `<p title="Ein Absatz">`
  - schlecht: `<p title=Absatz>`
  - ganz schlecht: `<p title=Ein Absatz>`

    Wert („Ein Absatz“) enthält ein Leerzeichen, „Absatz“ würde daher als neues Attribut interpretiert werden

HTML Syntax und guter Stil (4/4)

- **Keine Leerzeichen in Attributzuweisungen**
  - gut: `<p class="info">`
  - schlecht: `<p class = "info">`
- **Keine unnötigen Leerzeichen und Leerzeilen**
  - keine leeren Zeilen ohne triftigen Grund
  - Leerzeilen nur zwischen größeren logischen Blöcken
  - nicht jedes Element einrücken
  - Einrückung nicht mit Tabulator-Taste, sondern mit 2 Leerzeichen
- **Dateinamen**
  - Dateinamen in Kleinbuchstaben mit Unterstrichen (`html_syntax.html`)
  - Dateiendung für HTML: **.html** oder **.htm**
  - Dateiendung für CSS: **.css**
  - Dateiendung für JavaScript: **.js**

openHPI

Grundlegende HTML-Elemente

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

Dokumentenkopf und Meta-Tags (1/2)

- Kopf (Header) eines HTML-Dokuments enthält *Informationen über das Dokument* – **Meta-Informationen**, z.B.
  - Titel, Name des Autors, Beschreibung, etc.
- Meta-Information sind wichtig für automatische Verarbeitung des Dokuments, z.B. für dessen Aufnahme in den Suchindex einer Suchmaschine
- Kopf eines HTML-Dokuments wird umschlossen von **`<head>`**…**`</head>`**
- Folgende Tags können im HTML-Header verwendet werden:
  - **title:** Titel der Seite zur Anzeige in der Browser-Kopfleiste
  - **base:** Basis-URL für alle URLs in der HTML-Datei; alle *relativen URLs* werden dann (vorne) ergänzt um diese Basis-URL
  - **style:** Definition von CSS-Styles für das HTML-Dokument
  - **link:** Einbettung externer Ressourcen, z.B. CSS-Dateien
  - ...

Dokumentenkopf und Meta-Tags (2/2)

- Folgende Tags können in einem HTML-Header verwendet werden:
  - **script:** → JavaScript für das Dokument oder Einbettung einer externen JavaScript-Datei
  - **noscript:** alternativer Seiteninhalt für Browser, die kein JavaScript ausführen können
  - **meta:** Metainformationen zum Dokument, z.B.
    - `<meta name="author" content="Christoph Meinel">`
    - `<meta name="keywords" content="HTML, CSS, XML, ...">`
    - `<meta charset="UTF-8">` (Zeichenkodierung des Inhalts)
    - `<meta http-equiv="refresh" content="10">`
      (Lädt nach 5 Sekunden die Seite neu)

HTML-Elemente für Inhalte: Einführung

- Eigentlicher Inhalt eines HTML-Dokuments befindet sich im **Body**
- Body wird eingefasst von **<body>…</body>**
- HTML unterscheidet zwei Kategorien struktureller Elemente:
  - **Block-Elemente** – bestimmen Block-basierte Strukturen im Body des Dokuments
    - können weitere Block-Elemente oder Inline-Elemente enthalten
    - zwei aufeinanderfolgende Block-Elemente werden grundsätzlich **untereinander** angeordnet und nehmen die **volle Breite** ein
  - **Inline-Elemente** – befinden sich im Fließtext
    - können weitere Inline-Elemente enthalten
    - können am Zeilenende umgebrochen werden
  - Mittels CSS kann für viele Elemente festgelegt werden, ob diese als Block- oder Inline-Element behandelt werden sollen

HTML-Elemente für Inhalte: Text- und Struktur-Elemente

Die wichtigsten **grundlegenden Text- und Strukturelemente** sind:

- **<p>** – Paragraph, Absatz
- ** ** – Division, Abschnitt
- **<h1>** bis **<h6>** – (Zwischen-)Überschriften absteigender Wichtigkeit
- **** – Abschnitte in Fließtext (Inline-Element)
- **<blockquote>** – Zitate
- **<i>, ** – italics, emphasis, Hervorhebungen (Inline-Element, meist als schräggestellter Text dargestellt)
- **** – wichtiger Text (Inline-Element, meist fettgedruckter Text)

Die folgenden Tags kommen grundsätzlich **ohne schließenden End-Tag** aus:

- **<hr>** – horizontal rule, horizontale Linie
- **
** – break, Zeilenumbruch

HTML-Elemente für Inhalte: Hyperlinks (1/3)

Herzstück des WWW sind **Hyperlinks**

- Hyperlinks können voneinander getrennte, unabhängige Informationsressourcen miteinander verbinden
- HTML kennt verschiedene Typen von Hyperlinks:
  - **Unsichtbare Links**, z.B. zwischen HTML-Dokumenten und CSS-Stylesheets, werden im Hintergrund automatisch vom Browser bearbeitet – **<link>**
  - **Sichtbare Links** werden durch einen Mausklick des Nutzers aktiviert und führen zu anderen Informationsressourcen im Web, die über ihre URIs identifiziert werden – **<a>**

HTML-Elemente für Inhalte: Hyperlinks (2/3)

<a>-Element kann für verschiedene Zwecke genutzt werden:

- Markierungen (Ankerpunkte) – `<a name="ziel"></a>`
  `ziel` definiert einen eindeutigen Bezeichner für einen Ankerpunkt innerhalb eines Dokuments, zu dem ein Browser springen kann
- Sprung zu einem Ankerpunkt – `<a href="ziel">Verlinkter Text</a>`
  Klick auf den Link Text springt zu einem Ankerpunkt mit dem Namen `ziel`
- Sprung zu einem externen Ziel – `<a href="URL">Verlinkter Text</a>`
  - URI kann absolut oder relativ angegeben werden, z.B.
    - Sprung zu anderem Server mit absolutem URI:
      `<a href="https://www.google.de">Google-Suche</a>`
    - Sprung zu HTML-Dokument auf gleichem Server mit relativem URI:
      `<a href="/pages/imprint">Impressum</a>`

HTML-Elemente für Inhalte: Hyperlinks (3/3)

<a>-Element kann für verschiedene Zwecke genutzt werden:

- Sprung zu externer Ressource und zu einem Ankerpunkt kann auch kombiniert werden, z.B.

```
<a href="https://open.hpi.de/pages/faq#wo-finde-ich-die-kursmaterialien">
    FAQ-Eintrag auf openHPI
</a>
```

  Klick auf den Link öffnet FAQ-Seite und springt zu angegebener Frage

- **<a>-Tags** kennt neben **href** auch weitere **Attribute**, z.B.
  - **target:** bestimmt, wo verlinkte Ressource geöffnet werden soll
    - `<a href="URL" target="_blank">Verlinkter Text</a>`
      Öffnet Link in neuem Fenster oder Tab (je nach Browser-Einstellung)
    - `<a href="URL" target="_self">Verlinkter Text</a>`
      Öffnet Link in gleichem Fenster oder Tab (Default-Target)

Bilder (1/2)

Wichtigste „verlinkte Ressource“ in HTML-Dokumenten sind **Bilder**

- Haben eigenen Tag: **<img>** (ohne Endtag)
- Wichtige Attribute:
  - **src:** URL, wo das Bild liegt (relativ oder absolut)
  - **alt:** alternativer Text, der angezeigt wird, wenn das Bild nicht geladen oder angezeigt werden kann, z.B. wegen schlechter Verbindung oder mit Screen Readern
    ➔ sollte **immer** angegeben werden
  - **height / width:** Höhe und Breite, in der das Bild auf der HTML-Seite angezeigt werden soll
    - Wird nur ein Wert angegeben, wird der andere automatisch entsprechend der Original-Seitenverhältnisses vom Browser berechnet
    - Alternativ: **style="height: Höhe; width: Breite"**

Bilder (2/2)

Beispiel

- Bilder können auch mit einem Link versehen werden

```
<a href="https://open.hpi.de" target="_blank">
  <img src="/img/hpi_logo.png" alt="Hasso-Plattner-Institut" />
</a>
```

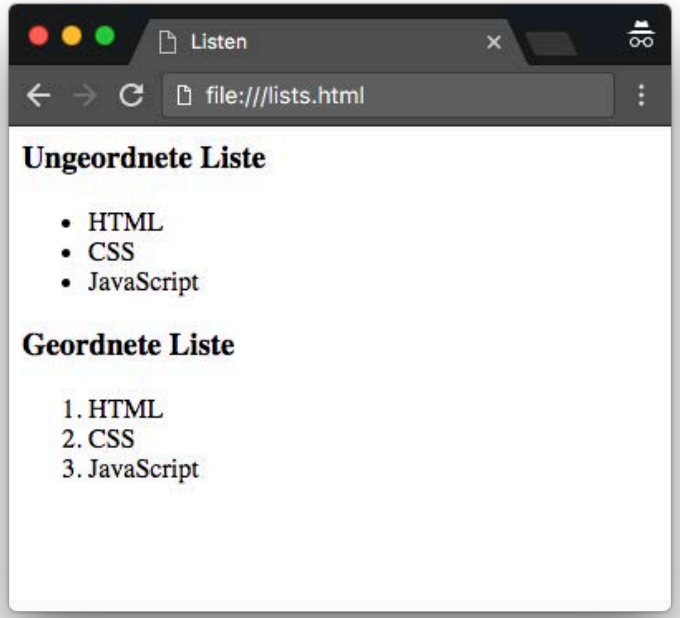
Ein Klick auf das Bild öffnet den Link in einem neuen Fenster

HTML-Elemente für Inhalte: Listen (1/2)

HTML kennt verschiedene Typen von Listen

- **Ungeordnete Listen**
  - Nicht-nummeriert mit Aufzählungszeichen
  - Werden durch **<ul>**-Tags erzeugt
  - Listenelemente stehen in **<li>**-Tags
- **Geordnete Listen**
  - Engl. „ordered list“, nummeriert
  - Werden durch **<ol>**-Tags erzeugt
  - Listenelemente wieder in **<li>**-Tags
- ...

```
<h3>Ungeordnete Liste</h3>
<ul>
  <li>HTML</li>
  <li>CSS</li>
  <li>JavaScript</li>
</ul>
```

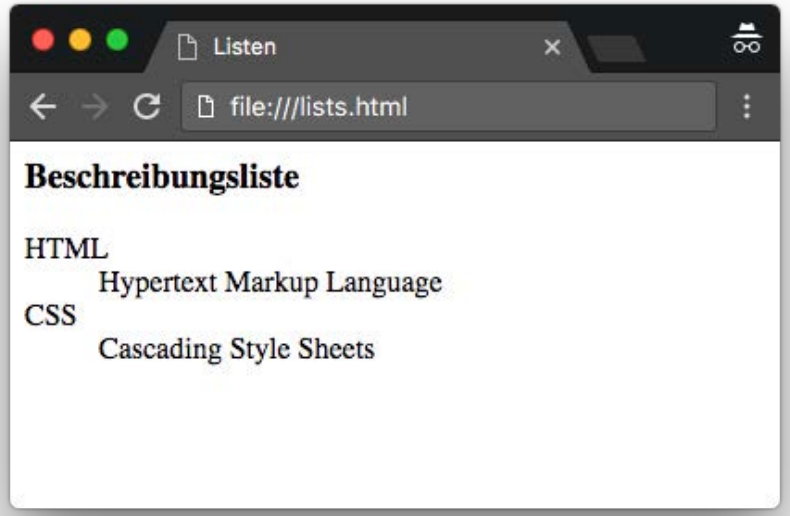


HTML-Elemente für Inhalte: Listen (2/2)

HTML kennt verschiedene Typen von Listen

- **Beschreibungslisten**
  - Engl. „description lists“
  - Werden durch **<dl>-**Tag erzeugt
  - Listenelemente bestehen aus
    - Begriffen (terms) – **<dt>**
    - Erläuterungen (descriptions) – **<dd>**
- Bei allen Listentypen kann die Darstellung mit CSS angepasst werden, z.B.
  - Form des Aufzählungszeichens
  - Art der Nummerierung

```
<h3>Beschreibungsliste</h3>
<dl>
  <dt>HTML</dt>
  <dd>Hypertext Markup Language</dd>
  <dt>CSS</dt>
  <dd>Cascading Style Sheets</dd>
</dl>
```



HTML-Elemente für Inhalte: Tabellen (1/3)

- Tabellen in HTML werden durch die Tags **`<table>...</table>`** erzeugt
- Tabellen bestehen aus rechteckigen Zellen, die in Zeilen und Spalten organisiert sind
  - **<tr>** erzeugt eine neue Zeile (table row) in einer Tabelle
  - **<td>** erzeugt eine neue Zelle in einer Tabellenzeile
  - **<th>** erlaubt die Auszeichnung spezieller Zellen für den Tabellenkopf (table header)
- Innerhalb einer Tabelle können Tabellenkopf und Tabellenfuß auch explizit festgelegt werden
  - **<thead>**, **<tbody>** und **<tfoot>** bezeichnen dann die einzelnen Bereiche innerhalb der Tabelle
  - Bereiche enthalten dann jeweils Zeilen und Spalten
  - Unterteilung erlaubt differenziertere Behandlung bei Skripting und Formatierung

HTML-Elemente für Inhalte: Tabellen (2/3)

- **Benachbarte Zellen** können mit speziellen Attributen für <td> **zusammengefasst** werden
  - Nebeneinanderliegende Zellen: **colspan**

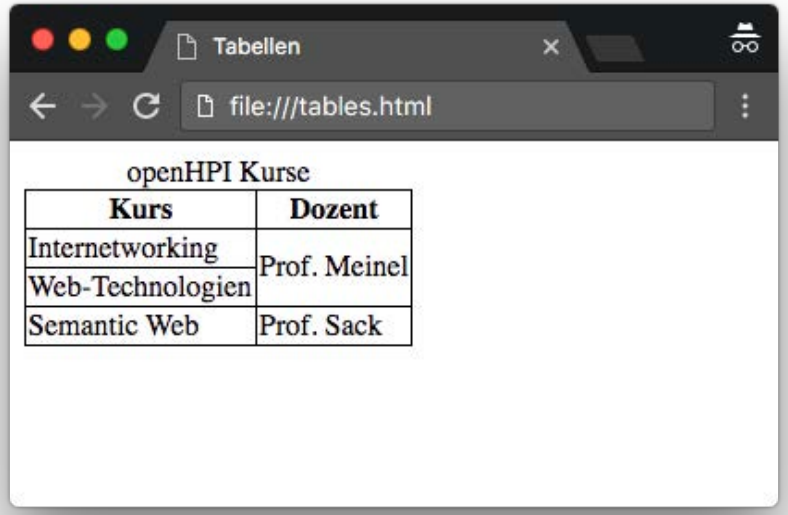
    ```
    <td colspan="2">Zelle geht über 2 Spalten</td>
    ```
  - Übereinanderliegende Zellen: **rowspan**

    ```
    <td rowspan="2">Zelle geht über 2 Zeilen</td>
    ```
- Tabellen können mit Beschriftung versehen werden
  - **<caption>**-Tag wird direkt als erstes innerhalb der Tabelle eingefügt
  - Ohne weitere Formatierung wird Caption oberhalb der Tabelle angezeigt
- Formatierung der Tabelle (Rahmen, Abstände, Ausrichtung, usw.) erfolgt mittels CSS

HTML-Elemente für Inhalte: Tabellen (3/3)

Beispiel

```
<table>
  <caption>openHPI Kurse</caption>
  <tr>
    <th>Kurs</th>
    <th>Dozent</th>
  </tr>
  <tr>
    <td>Internetworking</td>
    <td rowspan="2">Prof. Meinel</td>
  </tr>
  <tr>
    <td>Web-Technologien</td>
  </tr>
  <tr>
    <td>Semantic Web</td>
    <td>Prof. Sack</td>
  </tr>
</table>
```



Etwas CSS für bessere Lesbarkeit:

```
<style>
  table, td, th {
    border: 1px solid black;
    border-collapse: collapse;
  }
</style>
```

HTML-Elemente für Inhalte: Formulare (1/3)

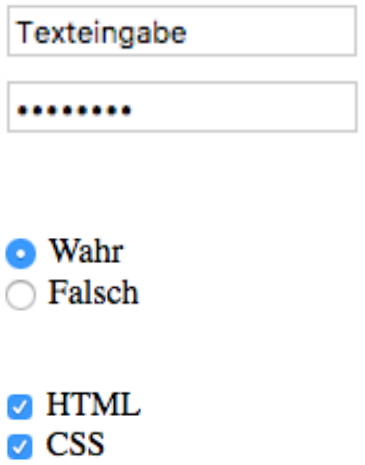
- Zur Übertragung von Daten vom Browser zum Server, die über bloße Anforderung von Dokumenten hinaus gehen, gibt es HTML **Formulare**
- Formulare können für unterschiedliche Zwecke eingesetzt werden:
  - Anforderung ähnlich strukturierter Informationen, z.B.
    - Filter für Daten, Paginierung, …
  - Spezifikation von Suchanfragen in großen Datenmengen
  - Übermittlung eigener Datensätze an einen Server
  - Individuelle Nutzerinteraktion, z.B. in Online Shops
  - Übermittlung von Dateien an einen Server
  - …

HTML-Elemente für Inhalte: Formulare (2/3)

- Formulare werden mit dem **<form>**-Tag erzeugt:

  `<form action="/login" method="POST">...</form>`

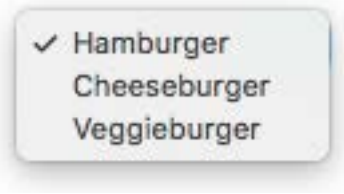
  - **action:** Gibt das Ziel des Formulars an – also eine URL, an die die Formulardaten mit HTTP gesendet werden
  - **method:** Gibt die HTTP-Methode an, mit der das Formular gesendet wird – also z.B. GET oder POST
- Es gibt eine **Reihe grundlegender Eingabefelder**, z.B.
  - **<input type="text">** – Texteingabefeld
  - **<input type="password">** – Texteingabefeld für Passwörter, Eingabe wird nicht angezeigt
  - **<input type="radio">** – „Radio Buttons", runde Knöpfe für die Auswahl einer von mehreren Optionen
  - **<input type="checkbox">** – Boxen mit Häkchen für Mehrfachauswahl von Optionen



HTML-Elemente für Inhalte: Formulare (3/3)

- Es gibt eine **Reihe grundlegender Eingabefelder**, z.B.
  - Auswahlfeld zum Ausklappen („Dropdown-Liste“)

```
<select name="bestellung">
    <option>Hamburger</option>
    <option>Cheeseburger</option>
    <option>Veggieburger</option>
</select>
```



  - Textfeld für mehrzeiligen Text

```
<textarea rows="5" cols="20"></textarea>
```

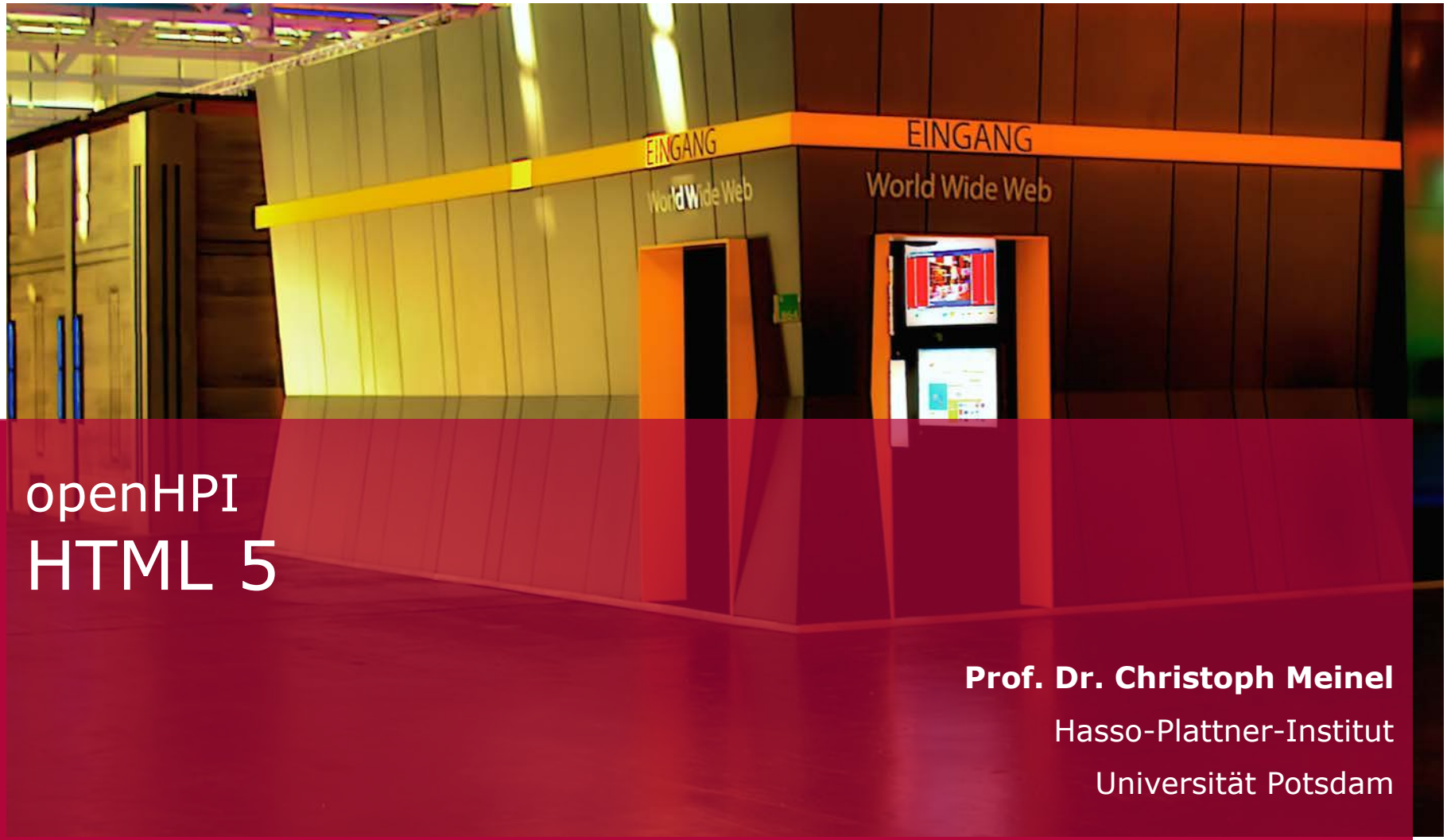
  - Button zum Absenden des Formulars

```
<input type="submit" value="Senden" />
```

➔ Führen später weitere Formularfelder ein; auch im Kontext der Webprogrammierung geht es um Verarbeitung von Formulardaten

HTML-Entitäten

- **Entitäten** werden bei Markup-Sprachen verwendet, um häufig vorkommende Zeichenketten zu verwalten und wiederzuverwenden
- HTML-Entitäten haben das Format **&Kennung;**
- Entitäten werden zum einen für Sonderzeichen verwendet, die für das Markup vorgesehen sind, z.B.
  - < – < bzw. > – >
  - & – &
- Außerdem werden Entitäten für sprachspezifische Sonderzeichen genutzt, z.B.
  - Deutsche Umlaute: ä – ä / Ä – Ä / ß - ß / ...
  - Accents: Á – &Aacut; / ê – ê / ò – ò / ...
  - Griechisches Alphabet: π – π / Ω – Ω / ...
  - Andere, z.B. © – © / ½ – ½ / € – € / ...



openHPI

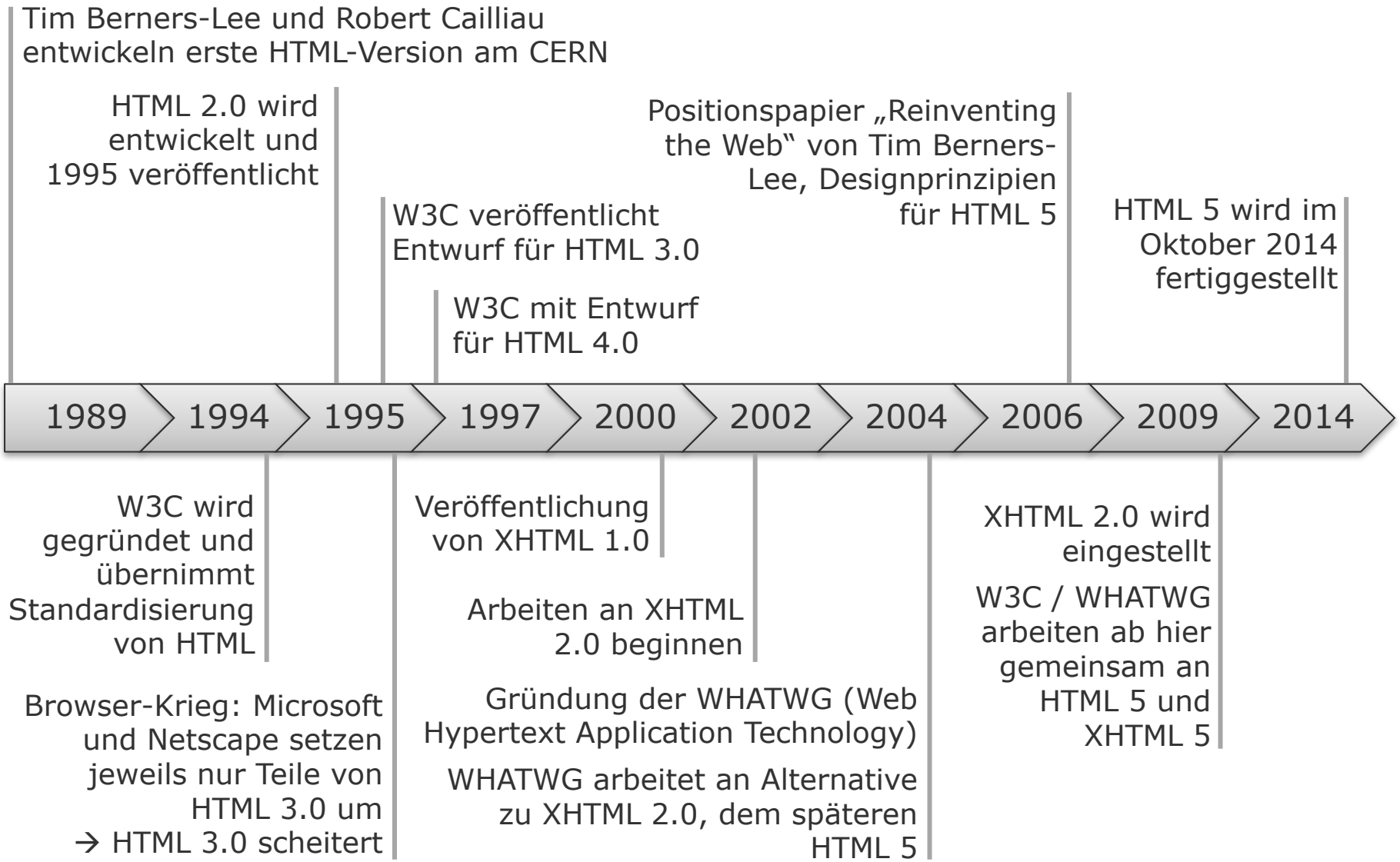
HTML 5

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

Kurze Geschichte von HTML: von HTML 1.0 über XHTML zu HTML 5



HTML 5 – Ziele und Paradigmen

Nachdem HTML-Arbeitsgruppe des W3C zerbrach, konzentrierte sich die WHATWG auf die Entwicklung eines praxisorientierten Standards

- Neuer Standard sollte gemeinsam von Nutzern, Autoren, Designern, Browser-Herstellern, … entwickelt werden
- **Entwicklungsparadigma:** „Nutzer vor Autoren vor Entwicklern vor Spezifikationsdesignern vor „reiner Lehre"
- HTML 5 solle
  - Stück für Stück entwickelt werden
  - abwärtskompatibel sein
  - einfach zu benutzen sein
  - noch stärker auf Trennung zwischen Struktur und Layout achten

HTML 5 – Was ist neu?

- HTML 5 basiert nicht länger auf SGML
  - Keine Document Type Definition (DTD)
  - Automatische Validierung (Syntaxprüfung) mit bestehenden Tools nicht mehr möglich, spezielle Werkzeuge werden benötigt
- HTML 5 führt Reihe **„sprechender" Strukturelemente** ein, die eine bessere Strukturierung erlaubt
- HTML 5 bietet Reihe neuer Elemente für **Multimedia-Inhalte**
- Für **Formulare** gibt es viele neue Tags und Attribute
- **<canvas>**-Element ist Zeichenfläche für Animationen
- SVG (Scalable Vector Graphics) und MathML können direkt in HTML-Dokumenten verwendet werden
- Viele Elemente und Attribute für die Formatierung werden nicht mehr unterstützt

Semantische Elemente (1/2)

- Meistverwendetes strukturierendes Block-Element in HTML 4.0 ist -Tag
- HTML 5 führt dafür eine Reihe „sprechender Strukturelemente“, sogenannte **semantische Elemente** (Elemente mit „Bedeutung“) ein, z.B.
  - <section>
  - <article>
  - <header>, <footer>
  - <aside>
  - <nav>
  - ...
- Maschinen (z.B. Suchmaschinen) können so leichter identifizieren, welche Teile des Dokuments wichtig sind oder welche für die Navigation verwendet werden

Semantische Elemente (2/2)

Beispiel

```
...
<header>...</header>
<nav>...</nav>
<article id="1">
   <header>...</header>
   <section>...</section>
   <footer>...</footer>
</article>
<article id="2">
   ...
</article>
...
<aside>...</aside>
<footer>...</footer>
...
```

Multimedia-Elemente (1/3)

- Audio und Video konnten bisher nur mittels Browser-Plugins in HTML-Dokumenten verwendet werden, z.B. Windows Media Player, Quicktime, VideoLAN client, Adobe Flash, usw.
- Neue Tags für **<audio>** und **<video>** erlauben es, Multimedia-Dateien direkt mit dem Browser abzuspielen
- Wie alle Elemente sind auch die Multimedia-Tags Teil des
  ➔ Document Object Models (DOM)
  - Bedienelemente (Play, Pause, Lautstärke, ...) können mit CSS gestaltet werden

Chrome


Safari

Firefox

  - Komplexere Multimedia-Steuerung (z.B. Sprungmarken, Playlisten, usw.) können mit JavaScript realisiert werden

Multimedia-Elemente (2/3)

- Betriebssysteme bzw. Browser müssen für Wiedergabe sogenannte **Codecs** (Verfahren für Kodierung/Dekodierung) unterstützen
- Vorgesehene Codecs für HTML 5 sind:
  - Audio: mp3, wma, AAC, Ogg Vorbis
  - Video: H.264, Ogg Theora, WebM
- Alternative Quellen (z.B. für unterschiedliche Codecs oder Bitraten) können mit **<source>**-Tag angegeben werden
- (Früher) **problematisch**: Standard legt nicht fest, welche Codecs unterstützt werden müssen, Browser-Hersteller handhaben das unterschiedlich, z.B. bei Video-Codecs:
  - Internet Explorer und Safari unterstützen nur H.264
  - Chrome, Firefox und Opera unterstützen alle Codecs
  - Früher mussten Inhalteanbieter mehrere Codecs bereitstellen, heute reicht in der Regel H.264 aus

Multimedia-Elemente (3/3)

Beispiel: <audio>-Tag mit verschiedenen Quellen

```
<audio>
   <source src="/audio/track1.wma" type="audio/x-ms-wma">
   <source src="/audio/track1.mp3" type="audio/mpeg">
   <p>Wenn Sie diesen Text sehen, unterstützt Ihr Browser
      den HTML5 Audio-Tag nicht</p>
</audio>
```

- Der Browser spielt das erste <source>-Element ab, dessen Codec er versteht
- Unbekannte Tags ignoriert ein Browser, daher ist es wichtig, einen Hinweis für ältere Browser hinzuzufügen

Formulare (1/4)

- HTML 5-Formulare basieren auf Standard Web Forms 2.0
  - Abwärtskompatibel zu Web Forms 1.0 (gehört zu HTML 4.0)
- Gegenüber bisherigen HTML-Formularen gibt es einige Neuerungen, z.B.
  - Input-Elemente können auch außerhalb von Formularen stehen und diesen mit dem **form**-Attribut zugeordnet werden
    - `<input type="checkbox" form="registrierung"> AGB akzeptiert`
  - Neue Elemente: <datalist>, <keygen>, <output>
  - Neue Input-Typen, z.B. für Zahlen, Zeit und Datum, Farbwerte
  - Neue Attribute für Formularelemente, z.B.
    - für Eingabevalidierung: min, max, required, ...
    - mit Platzhalter
    - ...

Formulare (2/4)

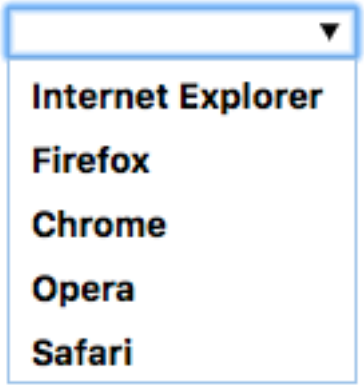
Beispiele für neue Features

- Zahlenbereiche können mit Slider ausgewählt werden

```
<input type="range" min="0" max="10">
```

- Mit dem <datalist>-Element können Vorschläge (mit Auto-Vervollständigung) zu Texteinabe-Feldern ergänzt werden

```
<input type="text" list="browsers">
<datalist name="browsers">
  <option>Internet Explorer</option>
  <option>Firefox</option>
  ...
</datalist>
```



- Eingabefelder nur für n ch-/Runter-Pfeilen)

Vorbelegung von Formularfeldern mit dem **value**-Attribute

```
<input type="number" value="100">
```

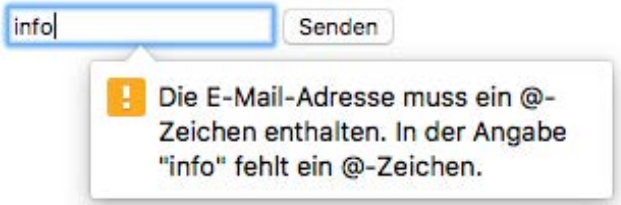
Formulare (3/4)

Beispiele für neue Features

- Eingabefelder mit Eingabevalidierung, z.B. für Email-Adressen

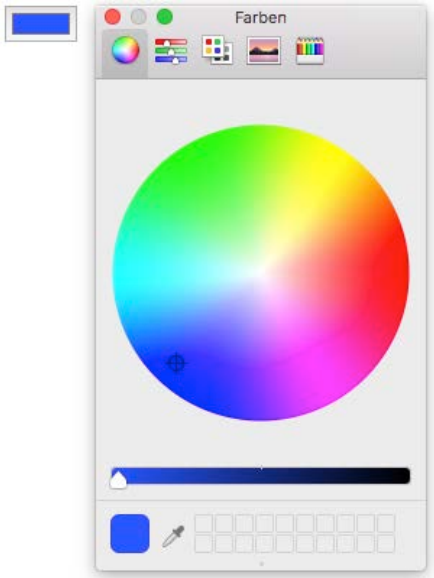
  ```
  <input type="email">
  <input type="url">
  ```

  Bei fehlerhaften Eingaben wird Formular nicht gesendet



- Farbauswahl mit „Color Picker“ (nicht für jeden Browser)

  ```
  <input type="color">
  ```



Formulare (4/4)

Beispiele für neue Features

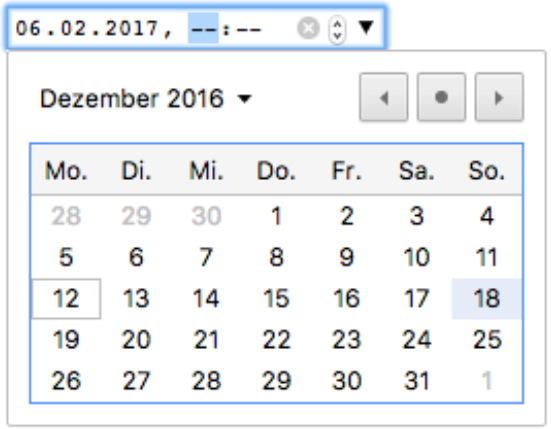
- Eingabefelder für Zeit und Datum, z.B.

  ```
  <input type="datetime-local">
  ```

  - analog: date, time, month, week
  - Auswahl-Dialog für manche Browser
- Attribute für **Eingaberestriktionen**
  - max, min: höchste/niedrigste Werte
  - required: Feld muss ausgefüllt werden
  - pattern: regulärer Ausdruck für Eingabemuster
- Weitere neue Attribute (Auswahl)
  - placeholder: Hinweis auf erwartete Eingabe

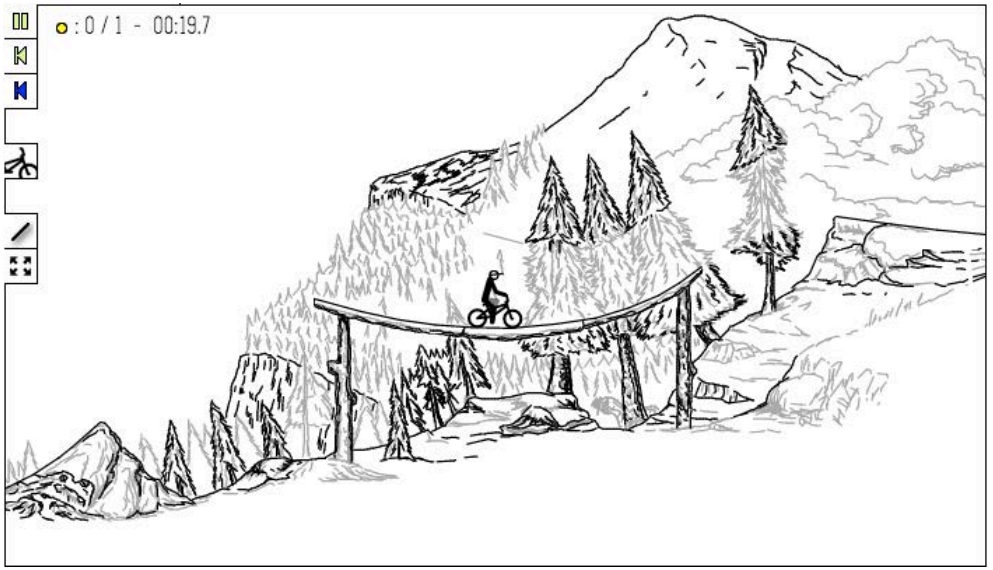
    ```
    <input type="email" placeholder="Ihre E-Mail-Adresse">
    ```

  - multiple: Auswahl mehrerer Dateien für Formular-Upload



Zeichenfläche – Canvas (1/2)

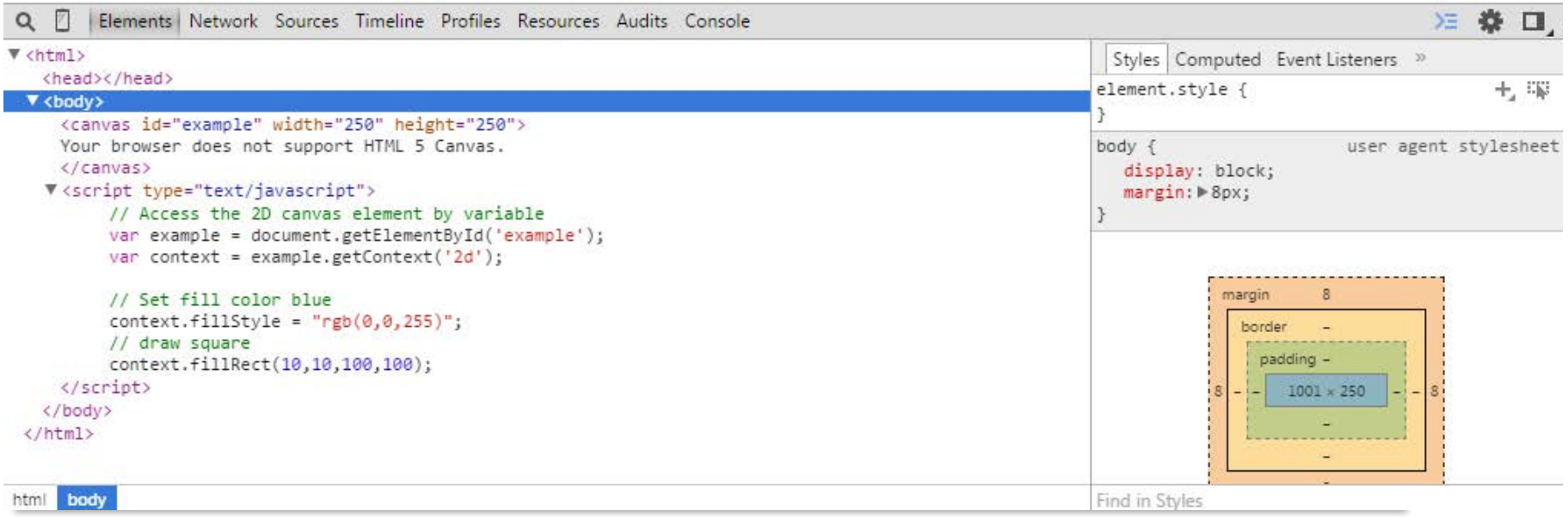
- **<canvas>-Tag** bietet Zeichenfläche für Bitmap-Grafiken
- Ursprünglich von Apple für WebKit-Browser entwickelt, mittlerweile in den WHATWG-Standard integriert
- Zeichnen auf den Canvas mit JavaScript (JS), Animation der Zeichnungen ebenfalls mit JS möglich
- Unterstützte Zeichenelemente z.B.
  - Linien
  - Rechtecke
  - Kreise
  - Bögen
  - Bézier-Kurven
  - Farbverläufe
  - ...



http://canvasrider.com

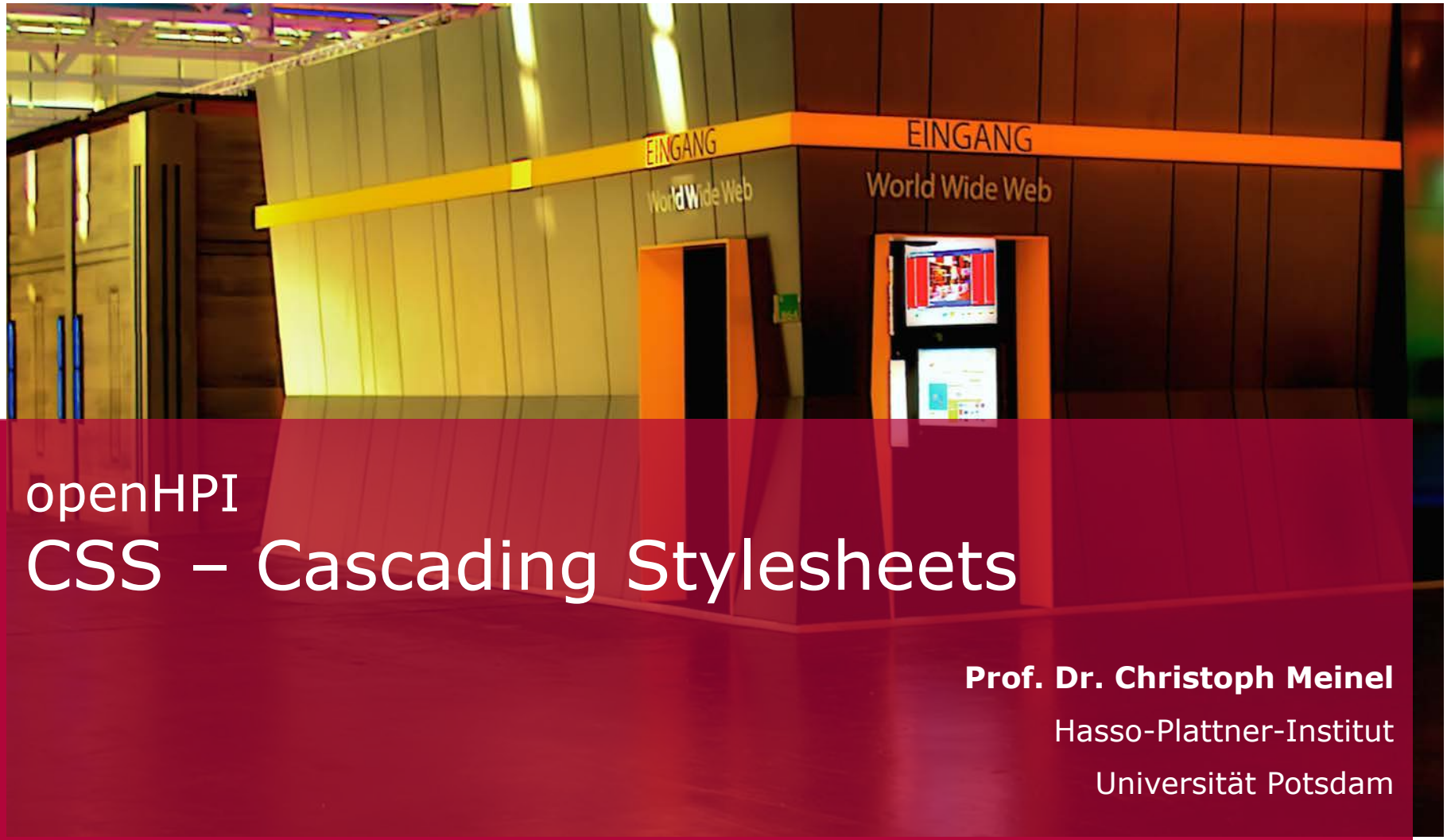
Zeichenfläche – Canvas (2/2)

Beispiel: Zeichnen eines blauen Quadrats



Weitere Features und Browserunterstützung

- HTML 5 führt eine ganze Reihe weitere Features ein, z.B.
  - neue Tags wie z.B. <time>, <meter> oder <progress>
  - komplette Übersicht: http://www.w3schools.com/tags/ oder http://www.selfhtml5.org/
- HTML 5 streicht Reihe von Tags aus HTML 4.01, vor allem im Bereich der Formatierung, z.B.
  - , <font>, <u>, <frame>, <frameset>, …
- Ein großer Teil der HTML 5-Features werden heute von aktuellen Versionen der gängigen Browser unterstützt – aber noch läuft nicht alles überall
  - Übersicht: http://caniuse.com/



openHPI

CSS – Cascading Stylesheets

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

Erinnerung (1/2)

- HTML folgt der grundlegenden Idee von SGML:
  Trennung von Dokumentstruktur und Dokumentpräsentation
  - SGML beschreibt die Dokumentstruktur
  - Formatierungsbeschreibungssprache DSSSL (Document Style Semantics and Specification Language) definiert das Layout
- Da Browser zu Beginn keine grafische Anzeige besaßen, wurde der Layout-Präsentation bei Webdokumenten kaum Beachtung geschenkt und zu HTML anfänglich keine separate Formatierungs-Beschreibungssprache eingeführt
- Mit dem Aufkommen von grafischen Browseroberflächen – 1993 Mosaic Browser – erfolgten die zunehmend erforderlichen Formatierungsanweisungen direkt in HTML

Erinnerung (2/2)

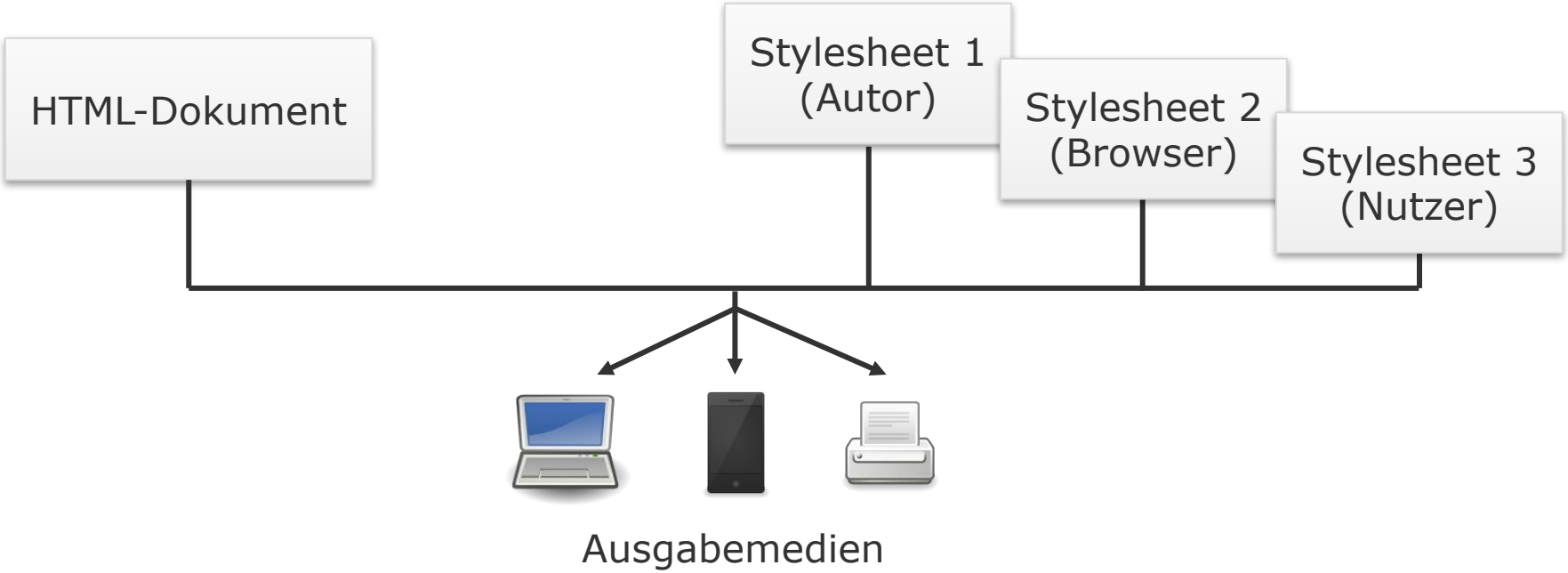
- Rapides Wachstum des WWW führte zum Wunsch, das Dokumenten-Layout so gestalten zu können, das es sicher reproduzierbar und auch an das jeweilige Ausgabemedium angepasst sein sollte
- Mit **Cascading Stylesheets** (**CSS)** wurde eine unabhängige Formatierungsbeschreibungssprache entwickelt, um **Stylesheets** für die Layout-Präsentation von HTML-Dokumenten beschreiben zu können
- Dabei beschreibt HTML die Dokumentstruktur und Stylesheets legen die Layout-Präsentation der einzelnen Strukturelemente fest für die jeweiligen Ausgabemedien

CSS – Cascading Stylesheets (1/3)

Mit CSS erstellte Stylesheets ermöglichen:

- Einheitliches Layout in zugehörigen HTML-Dokumenten
- Entwurf eines Dokuments nach den Wünschen und Bedürfnissen des Autors oder Nutzers
- Nutzung und Koexistenz von kompletten Stylesheet-Hierarchien, welche vom Autor, Browser-Hersteller oder Nutzer erstellt werden können
- Dokumentenübergreifende Wiederverwendung von Layouts

Verwendung von Stylesheets



CSS – Cascading Stylesheets (3/3)

- CSS bietet direkte Ergänzung zu HTML und weist individuellen HTML-Kommandos Ausgabemedien-spezifische Formatierungen zu
- Haupteigenschaften von CSS sind:
  - exakte Definition des Layouts eines HTML-Dokuments
  - Anpassung an verschiedene Ausgabemedien, z.B.
    - Anweisungen für Druck-Layout,
    - künstliche Sprachausgabe,
    - Smartphones, ...
  - zentrales Layout-Management – Layout-Definition und Aktualisierung kann an zentraler Stelle erfolgen

CSS – Kurze Geschichte

- Erste Spezifikation von CSS (1.0) wurde 1996 vom W3C veröffentlicht zusammen mit Anpassung des HTML 4.0-Standards
- **CSS 2.0** (1998) bietet Ausgabe verschiedener Medientypen – Druck- und Sprachausgabe – und Pixel-genaue Positionierung von integrierten Objekten
- Browser arbeiten (bisher) nicht immer Standard-konform
  - 2009 nahm W3C die Zwischenversion CSS 2.1 als „empfohlenen Kandidaten" an, um Unregelmäßigkeiten zu korrigieren
- Gleichzeitig begann Entwicklung von CCS 3, eine Version mit modularer Struktur
  - Die Spezifikationsphase für die meisten Teile ist abgeschlossen und Browserhersteller setzten die meisten der empfohlenen Module der W3C um

CSS – Verbindung mit HTML-Dokumenten

Stylesheets können auf verschiedene Arten mit HTML-Dokumenten verbunden werden:

- **Inline-Definition** in HTML-Elementen mit **style**-Attribut, z.B.
  - `<span style="font-weight: bold">Fetter Text</span>`
- **(Interne) Stylesheets** werden in einem HTML-Dokument definiert mit dem **<style>**-Tag (sollte im Header stehen)
  - **Anmerkung**: Hierbei sind Dokumentenstruktur und Präsentation noch nicht strikt getrennt
- **(Externe) Stylesheets** aus separaten Dateien werden eingebunden mithilfe der **<link>**-Deklaration im Header des HTML-Dokuments
- Wenn kombiniert wird, erhalten Inline-Styles Vorrang vor internen Stylesheets (im Header des Dokuments) und externen Stylesheets (lokale Definition vor globaler Definition)

CSS Syntax – Regeln

- **CSS Stylesheet** ist Ansammlung von Regeln die auf ein HTML-Dokument angewandt werden
- Beispiel einer **CSS Regel**:

```
h1 {
  color: blue;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 14px;
}
```

- **CSS Regel** besteht aus:
  - **Selektor** (`h1`) und
  - **Deklaration(en)** (`{ color: blue; font-family: […] }`)

CSS Syntax – Deklarationen

- **Deklaration** beinhaltet die eigentliche Formatierungsanweisung und weist einer **Eigenschaft** einen oder mehrere **Werte** zu
- Jede CSS-Regel kann **mehrere Deklarationen** enthalten
- Mögliche Formatierungsanweisung beziehen sich z.B. auf die folgenden Elemente:
  - **Schrift** – Schriftart, Schriftgröße, Schriftfarbe
  - **Abstände und Rahmen** – für alle Arten von HTML-Elementen
  - **Position und Größe** – für alle Arten von HTML-Elementen
  - **Farben und Hintergründe** – für alle Arten von HTML-Elementen
  - **Sichtbarkeit und Art der Anzeige** – für alle Arten von HTML-Elementen
  - **Umfließung** – für alle Arten von Block-Elementen
  - ...

CSS Syntax – Selektoren (1/5)

Selektor begrenzt potenziellen Anwendungsbereich von Deklarationen (Formatierungsanweisungen), kurz,

> ein Selektor **wählt HTML-Elemente aus**, auf die Deklarationen angewendet werden

Es gibt:

- **Klassen- und ID-Selektoren** – begrenzen den gültigen Bereich auf bestimmtes Element bzw. Gruppe von Elementen gleicher Klasse
- **Attribut-Selektoren** – Elemente mit einem bestimmten Attribut bzw. Attribut-Wert
- **Context-Selektoren** – begrenzen Vererbung von Formatierungsanweisungen
- **Externe Selektoren** (Pseudo-Elemente) – trifft auf Elemente zu, die nicht Teil des HTML-Dokuments sind, aber Teil des Kontexts

CSS Syntax – Selektoren (2/5)

Beispiele für verschiedene Selektor-Typen

- Standard-Selektor `p { color: red; }`
  - Beispiel: `<p>roter Text</p>`

- Attribut-Selektor `p[small] { font-size: 8px; }`
  - Beispiel: `<p small>Das ist eine Fußnote …</p>`
  - oder: `a[target="_blank"] { color:red; }`

    `<a target="_blank">roter Link</p>`

- Kontext-Selektor `td p { color:#0000FF; }`
  - Beispiel: `<td><p>blauer Text in Tabelle</p></td>`
  - Erläuterung: Absätze (<p>) in Tabellen-Zelle (<td>) werden blau dargestellt

CSS Syntax – Selektoren (3/5)

Beispiele für verschiedene Selektor-Typen

- Klassen-Selektor `p.footnote { font-size: 8px; }`
  - Beispiel: `<p class="footnote">Dies ist eine Notiz</p>`
  - Hinweis: Ein Element kann mehrere Klassen haben, bspw. **`<h1 class="blue center">...</h1>`**
- ID-Selektor `a#imprint { font-weight: bold; }`
  - Beispiel: `<a id="imprint" href="...">Impressum</a>`
- Hinweis: **class**-Attribut können für **mehrere Elemente** identisch sein, **id** muss in einem Dokument **einzigartig** sein
- Klassen- und ID-Selektoren können ohne Element definiert (und auf verschiedene Elemente angewandt) werden:
  - Beispiele:

```
.red { color: #FF0000; }
#invisible { visibility: hidden; }
```

CSS Syntax – Selektoren (4/5)

Gruppieren von Selektoren

- Will man identische Deklarationen für mehrere Selektoren anwenden, kann man diese zusammenfassen
- ```
  h1        { font-weight: bold; }
  #headline { font-weight: bold; }
  ```
  wird zu:
  ```
  h1, #headline { font-weight: bold; }
  ```

Manipulation von Element-Inhalten

- Mit den Selektoren `::before` und `::after` kann vor oder nach dem Inhalt eines Elements Text und/oder CSS ergänzt werden
- ```
  p.wichtig::before {
    content: "Wichtig: ";
    color: red;
    font-weight: bold;
  }
  ```
  ```
  <p class="wichtig">CSS lernen</p>
  ```
  Wichtig: CSS lernen

CSS Syntax – Selektoren (5/5)

Pseudo-Klassen und Pseudo-Elemente

Einige Selektoren können sich auf bestimmte Zustände von Elementen oder auf bestimmte Untermengen beziehen

- **:hover** – für „Mouseover"-Zustand von Elementen, z.B.
  ```
  a:hover { text-decoration: underline; }
  ```
  (Links werden unterstrichen, wenn Maus darauf zeigt)
- **:focus** – für aktives Eingabefeld in Formularen, z.B.
  ```
  input:focus { border: 1px solid blue; }
  ```
  (blauer Rahmen um aktives Eingabefeld)
- **::first-letter** – für den ersten Buchstaben in einem Element, z.B.
  ```
  p::first-letter { font-size: 24px; font-weight: bold; }
  ```
  („Initial": erster Buchstabe eines Absatzes groß und fett)
- Gibt eine Reihe weitere, z.B. ::first-line, :first-child, :active, :required, ::selection, ...

Dynamische Selektoren in CSS 3

CSS 3 bietet Reihe neuer **dynamischer Selektoren**, z.B.:

- E:nth-child(*n*) *n*-te Kind-Element vom Element *E*
- E:nth-last-child(*n*) *n*-te Kind-Element gezählt vom letzten
- E:nth-of-type(*n*) *n*-te Kind-Element vom Typ
- E:nth-last-of-type(*n*) *n*-te Kind-Element gezählt vom letzten
- Anstelle von Zahlenwert *n* sind auch Formeln möglich, z.B.:

```
li:nth-of-type(2n) { color: red; }
```

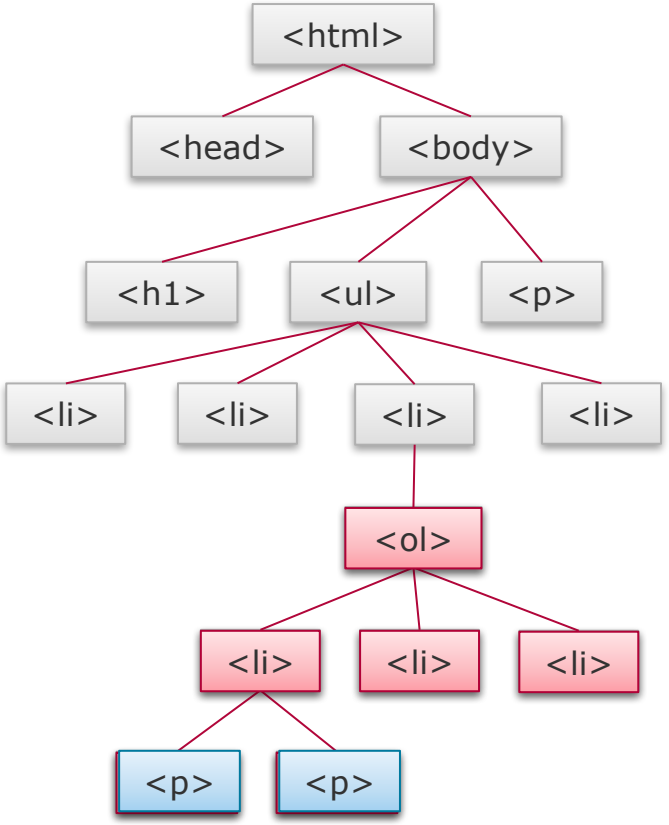
(Färbt jedes zweite <li> rot)

- Außerdem gibt es in CSS 3 den **Negations-Selektor**
  - :not(E) alle Elemente außer Element E
  - `:not(p)` wählt alle Elemente außer `<p>`

CSS Syntax – Vererbung (1/2)

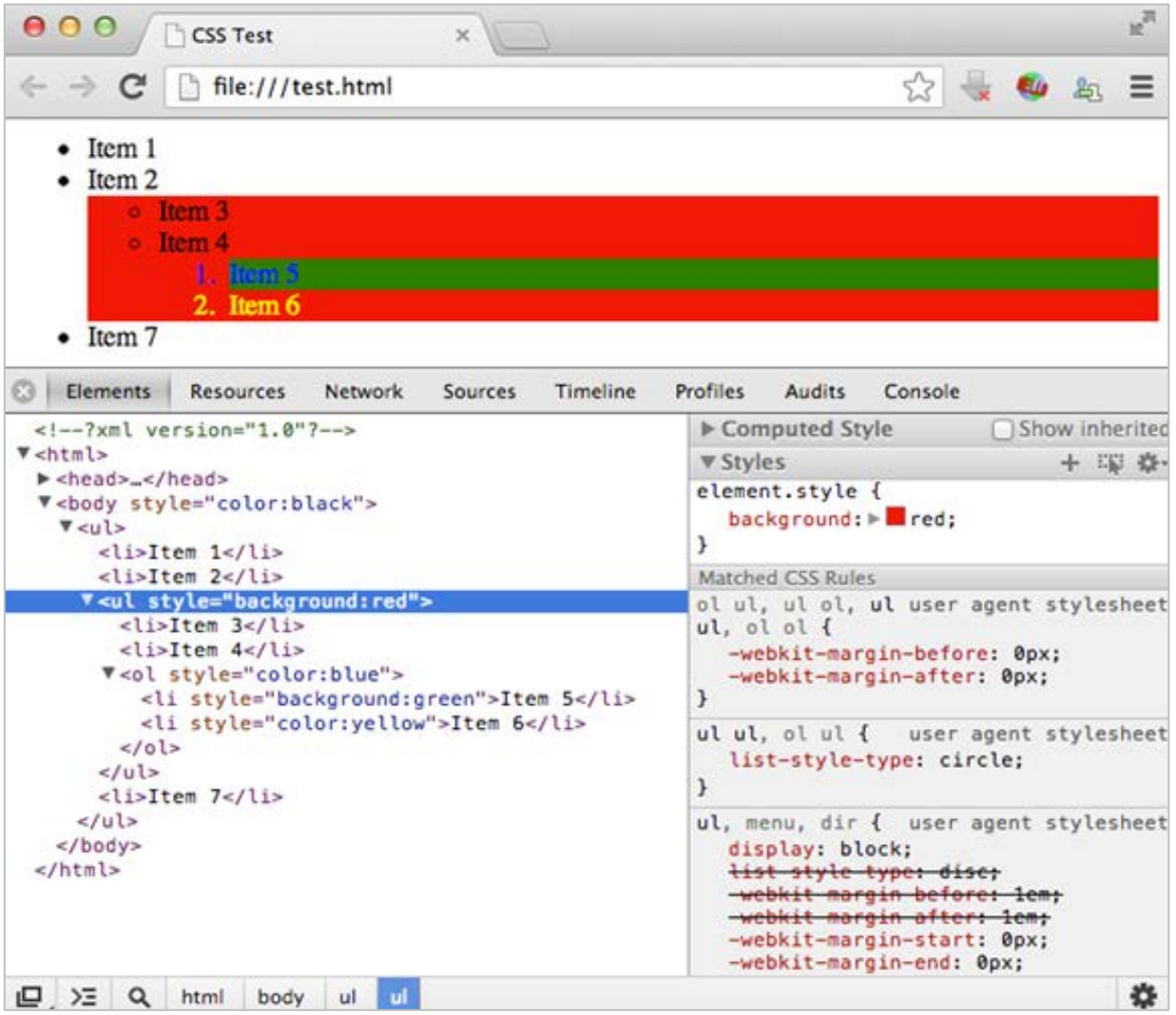
CSS-Formatierungsanweisungen werden im HTML-Dokumentenbaum an alle innerhalb eines Elements liegenden Unterelemente **vererbt**

- Beispiel:
  - `ol { background-color: red; }`
  - Unterbaum von `<ol>` erbt rote Hintergrundfarbe
- Spezielle Selektoren können Elemente **im Unterbaum** eines Elements auswählen:
  - `ol p { background-color: blue; }`
  - Alle `<p>`-Elemente unterhalb von `<ol>` werden blau gefärbt
- Spezifischere Selektoren überschreiben ererbte Eigenschaften (`ol p` ist spezifischer als `ol` → Hintergrund von `<p>` wird blau)



CSS Syntax – Vererbung (2/2)

Beispiel:
CSS Vererbungs-konzept



CSS – Mehrere Stylesheets (1/2)

Eine grundlegende Idee von CSS ist gleichzeitige Nutzung von mehreren Stylesheets. **Mögliche Kaskadierung von Stylesheets** umfasst:

- **Browser-Stylesheet**s – in jeder Dokumentenrepräsentation wird ein Browser-spezifisches Stylesheet genutzt
- **Nutzer-Stylesheets** – Browser bieten Nutzern (begrenzte) Möglichkeiten an, die Dokumentenrepräsentation anzupassen (Schrift, Farbe, etc.). Priorität ist höher als Browser-Stylesheets
- **Autoren-Stylesheets** – Autoren haben großen Freiraum in der Layout-Gestaltung; sie können mehrere Stylesheets definieren:
  - modulare Stylesheet Organisation
  - hierarchische Stylesheet Organisation
  - alternative Stylesheets, z.B. für verschiedene Präsentationsformen

CSS – Mehrere Stylesheets (2/2)

Nutzung von mehreren Stylesheets kann zu **Darstellungsfehlern** führen, wenn einem HTML-Element gegensätzliche Formatierungsanweisungen zugewiesen werden – was auch innerhalb eines Stylesheets möglich ist

CSS definiert Reihe von Regeln zur Konfliktlösung:

1. Finde alle relevanten Regeln
2. Ordne Regeln nach ihrer Wichtigkeit – Autoren können Regeln in CSS-Definitionen das Attribut `!important` geben
3. Ordne Regeln nach ihrem Ursprung: Autoren- vor Nutzer- vor Browser-Stylesheets
4. Ordne Regeln nach ihrem Spezialisierungsgrad: Anzahl von ID- und Klassen-Attributen
5. Ordne verbleibende, konkurrierende Regeln chronologisch nach ihrem Auftreten

openHPI

CSS – Positionierung und Box-Modell

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

CSS Positionierung (1/2)

- Mittels **Positionierung** können Elemente innerhalb eines HTML-Dokumentes angeordnet werden
- Auch festgelegt wird, wie Elemente von Text umflossen werden
- Mit `position` können Elemente beliebig positioniert und aus dem regulären Elementfluss entfernt werden:
  - `absolute`
    - Position bezieht sich auf Vorfahren-Element oder Wurzelelement
    - losgelöst vom Textfluss
    - Position per `left`, `right`, `top` und `bottom`
  - …

CSS Positionierung (2/2)

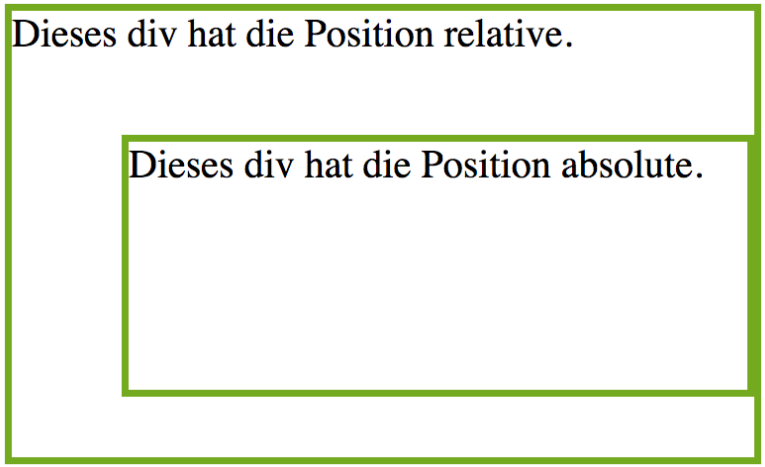
- **`fixed`**
  - Starre Ausrichtung ohne Abhängigkeiten
  - Losgelöst vom Textfluss
  - Position per **`left`**, **`right`**, **`top`** und **`bottom`**
- **`relative`**
  - Bezugspunkt für absolut positionierte Kind-Elemente
  - Bleibt im Textfluss
  - Position per **`left`**, **`right`**, **`top`** und **`bottom`**
- **`static`**
  - Standardwert der Eigenschaft **`position`**
  - Bleibt im Textfluss
  - **`left`**, **`right`**, **`top`** und **`bottom`** werden ignoriert

CSS Positionierung – Beispiel

- **CSS Stylesheet**:

```
div.relative {
  position: relative;
}
div.absolute {
  position: absolute;
  top: 50px;
  right: 0;
}
```

- **Ausgabe**:



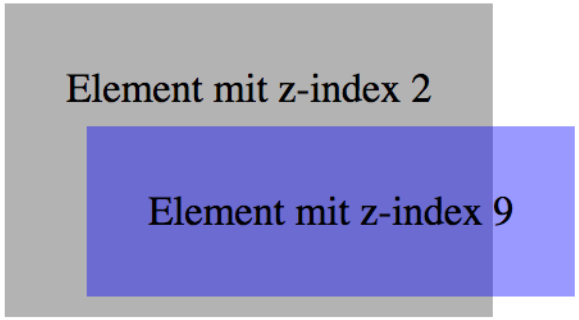
- **HTML**:

```
<div class="relative">Dieses div hat die Position relative.
  <div class="absolute">Dieses div hat die Position absolute.</div>
</div>
```

CSS Positionierung – z-index

- Die `z-index` Eigenschaft kommt zur Anwendung, wenn sich Elemente überlappen
- Je größer der Wert, desto „höher" liegt das Element und überdeckt andere Elemente
- Kann nur für Elemente angegeben werden, die eine `position` definiert haben und nicht `static` sind
- **Beispiel** (CSS):

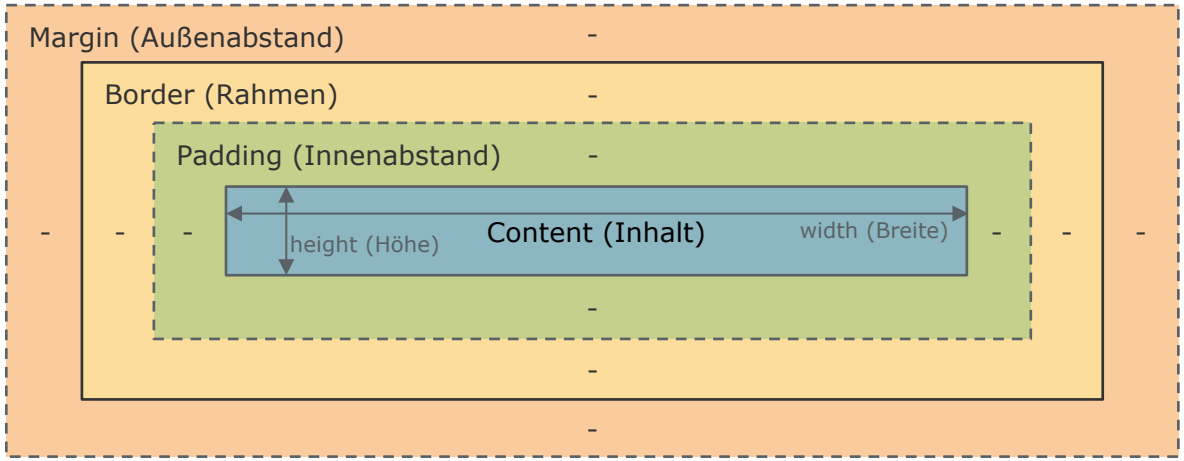
```
#element1 {
  z-index: 2;
}
#element2 {
  z-index: 9;
}
```



Box-Modell – Einführung (1/2)

- HTML-Elemente, wie z.B. , werden durch innerhalb von Rechtecken dargestellt
- Diese Rechtecke werden durch das **Box-Modell** beschrieben
- Bestandteile einer Box sind:
  - **Inhalt** – Texte und Bilder
  - **Innenabstand** (padding)
  - **Rahmen** (border)
  - **Außenabstand** (margin)
- Die Innen- und Außenabstände sowie der Rahmen können für alle vier Seiten der Box individuell festgelegt werden

Box-Modell – Einführung (2/2)



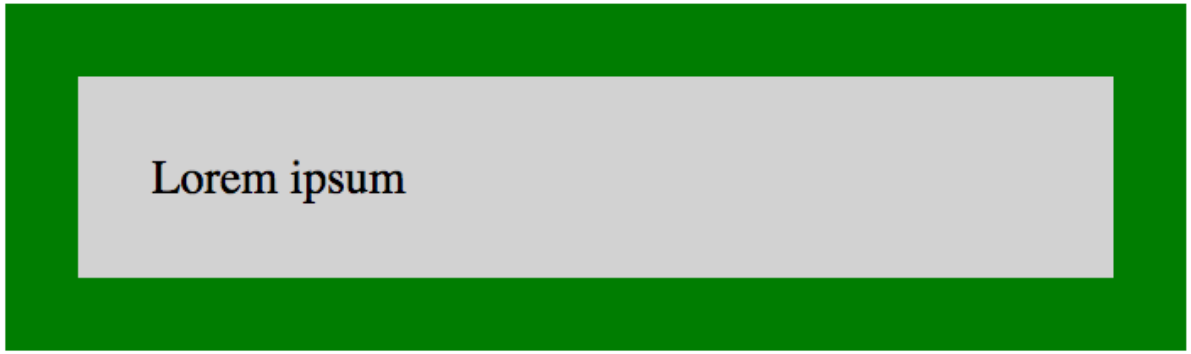
- Die einzelnen Bereiche werden bezeichnet als:
  - Content-Box
  - Padding-Box
  - Border-Box
  - Margin-Box

Box-Modell – Beispiel

- **CSS Stylesheet**:

```
div {
  background-color: lightgrey;
  width: 300px;
  padding: 25px;
  border: 25px solid green;
  margin: 25px;
}
```

- Ausgabe:



Box-Modell – Innen- und Außenabstand

- Abstände einzeln definierbar mit:

```
padding-top          margin-top
padding-right        margin-right
padding-bottom       margin-bottom
padding-left         margin-left
```

- Oder als zusammenfassende Eigenschaft:
  - Bei einer Angabe gilt der Wert für alle vier Seiten

```
padding: 10px;
margin: 10px;
```

  - Zwei bis vier Angaben: 1. Wert für top, 2. Wert für right, 3. Wert für bottom, 4. Wert für left

```
padding: 10px 5px;
margin: 10px 5px 10px 5px;
```

- Als Werte sind nicht negative numerische Längenmaße erlaubt, z.B. Pixel (`px`), Punkt (`pt`), Zentimeter (`cm`), Millimeter (`mm`), ...

Box-Modell – Rahmen

- Eigenschaft **`border`** bestimmt Rahmendicke, Rahmentyp und Rahmenfarbe, **`z.B. 25px solid green;`**
- Alle vier Seiten sind ebenfalls einzeln definierbar, **`z.B.`**
  - **`border-top`** **`border-right`**
    `border-bottom` **`border-left`**
- Rahmendicke: **`border-width`**
  - Numerischer Wert (z.B. **`25px`**) oder
    `thin` (dünn), **`medium`** (mittel), **`thick`** (dick)
- Rahmentyp: **`border-style`**
  - **`none`** (transparent), **`dotted`** (gepunktet), **`dashed`** (gestrichelt), **`solid`** (durchgezogen), **`double`** (doppelt durchgezogen), ...
- Rahmenfarbe: **`border-color`**
  - **`transparent`** oder Farbangabe (z.B. **`#`**

openHPI

Responsive Webdesign

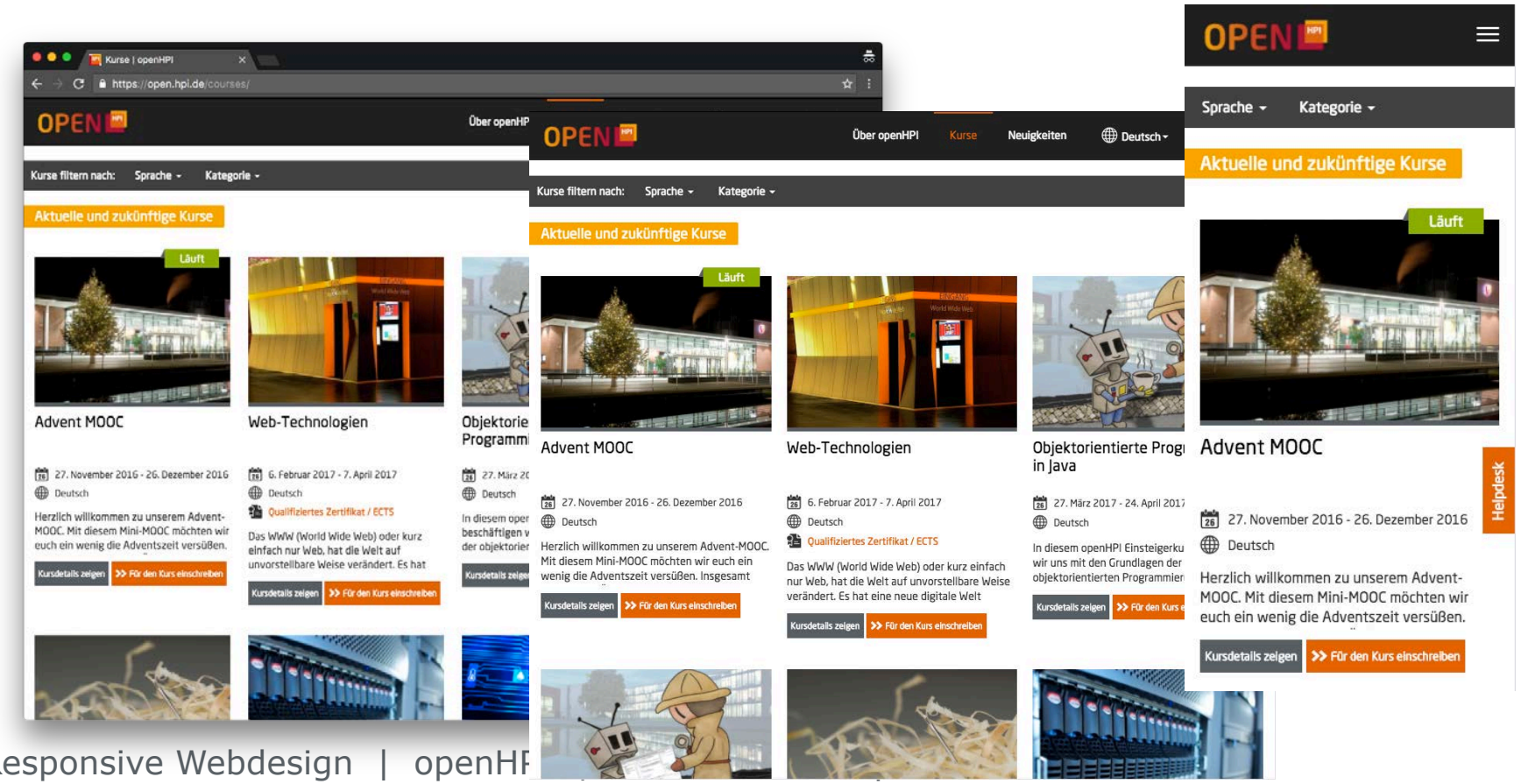
Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

Responsive Webdesign (1/2)

Responsives („reagierendes“) **Webdesign** (RWD) ist **Paradigma** bei Erstellung von Webseiten → Webdesign soll auf Anforderungen verschiedener Endgeräte (Desktop, Tablet, Smartphone) reagieren können



Responsive Webdesign (2/2)

- **Ziel beim Responsive Design**: Webseite erstellen, die für alle Nutzer auf allen Geräten gut aussieht und gut zu bedienen ist
- RWD sollte technisch nur mit HTML und CSS realisiert werden – JavaScript ist in der Regel nicht nötig
- RWD sollte nicht darauf beruhen, auf kleineren Bildschirmen einfach Inhalte wegzulassen, sondern diese anders anzuordnen
- **Beispiel**: openHPI-Kursliste
  - Tablets: Bilder werden zunächst schmaler, bei kleineren Bildschirmen werden 3 statt 4 Kursen pro Zeile angezeigt
  - Smartphones:
    - Navigation hinter „Hamburger-Icon“ verborgen 
    - Kurse untereinander angeordnet
    - Links in Fußleiste werden untereinander angeordnet

Viewport (1/2)

- **Früher**: Webseiten wurden für Desktop-Browser mit fester Breite gestaltet („Dieses Seite ist optimiert für 1024x768 Pixel“)
  - Seiten waren für mobile Endgeräte zu breit
  - Mobile Browser verkleinern gesamte Seite → unlesbare Schrift
- **Einfache Maßnahme**: Angabe des **Viewports**
  - Viewport ist der für Nutzer sichtbare Bereich einer Webseite
  - variiert mit Breite des Bildschirms bzw. Fensters
- HTML 5 erlaubt Autoren, den Viewport selbst festzulegen:

```
<meta name="viewport" content="width=device-width">
```

  - **device-width** ist die Breite des genutzten Gerätes, Elemente werden in der Breite auf Bildschirmbreite begrenzt
- Zusätzlich beachten: Keine Elemente mit großer fester Breite verwenden (also z.B. width="1000px"), diese ragen immer über Viewport hinaus

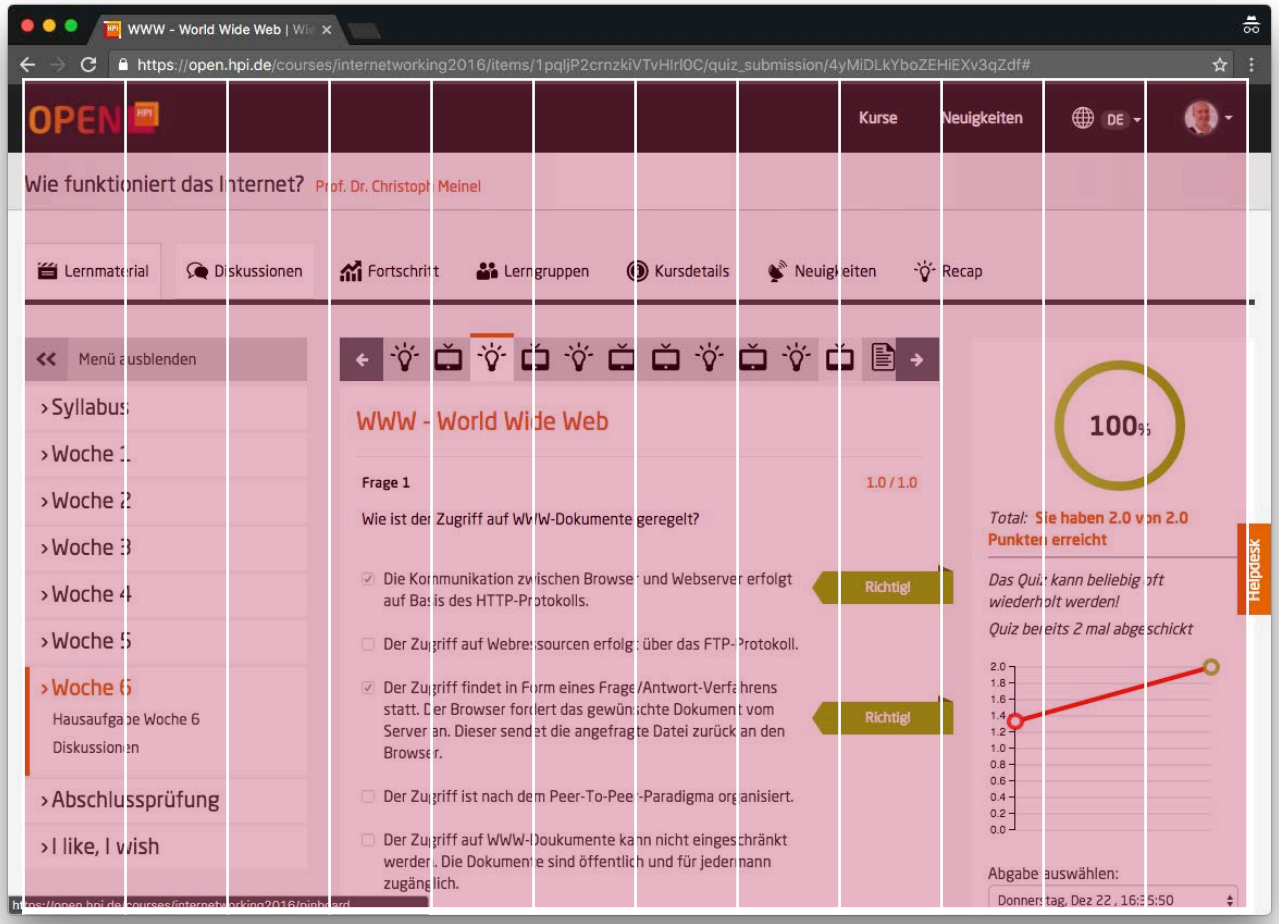
Viewport (2/2)

Ohne Viewport-Angabe

Mit Viewport-Angabe

CSS Raster Layout (1/2)

- Webseiten werden häufig anhand eines horizontalen Rasters (Grid) gestaltet, die für verschiedene Bildschirmbreiten skaliert werden



12-spaltiges Raster bei openHPI

CSS Raster Layout (2/2)

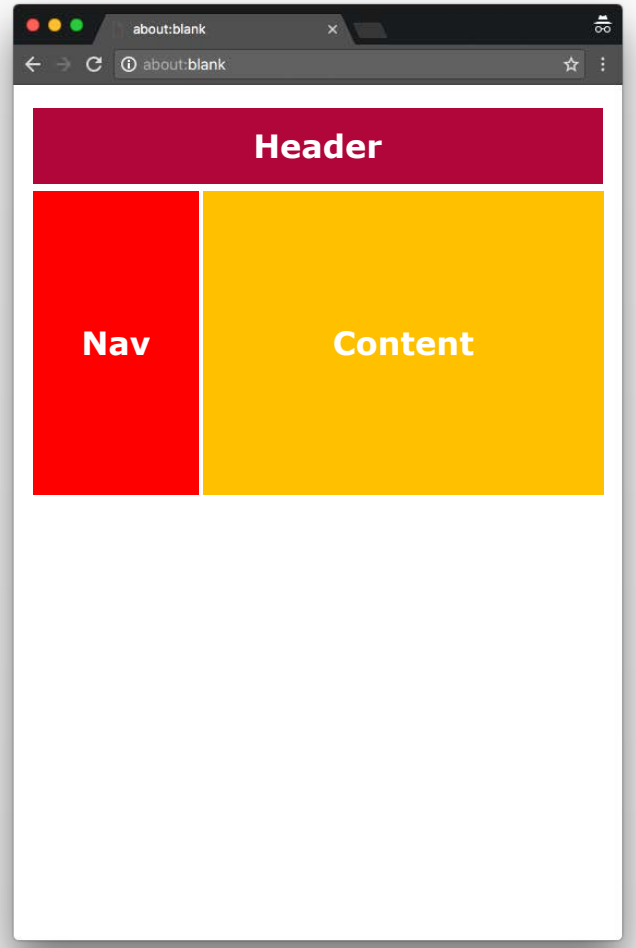
Code-Beispiel für CSS Raster

```
.col-1  {width:
.col-2  {width:
.col-3  {width:
.col-4  {width:
.col-5  {width:
.col-6  {width:
.col-7  {width:
.col-8  {width:
.col-9  {width:
.col-10 {width:
.col-11 {width:
.col-12 {width:

[class*="col-"]
    float: left
}

 .row::after {
    content: ""
    clear: both
}
```

Raster bewirkt, dass alle Elemente der Webseite auf **schmalen** Geräten in der Breite **gleichmäßig skaliert** werden




```
Header</div>

avigation</div>
ontent</div>
```

CSS Media Queries (1/3)

- CSS 3 führt erstmals **Media Queries** ein
- **@media**-Blöcke erlauben Definition von CSS-Regeln, die nur gelten, wenn bestimmte Bedingung gelten
- **Beispiele**:
  - nur für Bildschirme bis maximale Smartphone-Breite

```
@media only screen and (max-width: 768px) {
    img { width: 100% }
}
```

  - nur für Bildschirme im Querformat

```
@media only screen and (orientation: landscape) { ... }
```

  - nur für Druck

```
@media only print { ... }
```

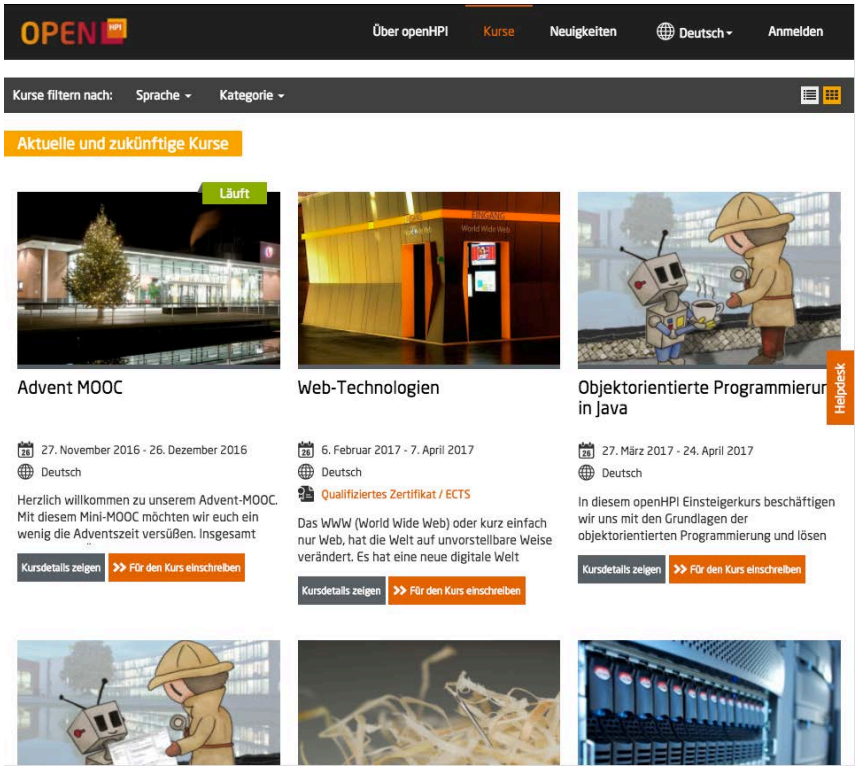
- Media Queries eignen sich auch für Einbinden externer CSS-Dateien:

```
<link rel="stylesheet" href="druck.css" media="print">
```

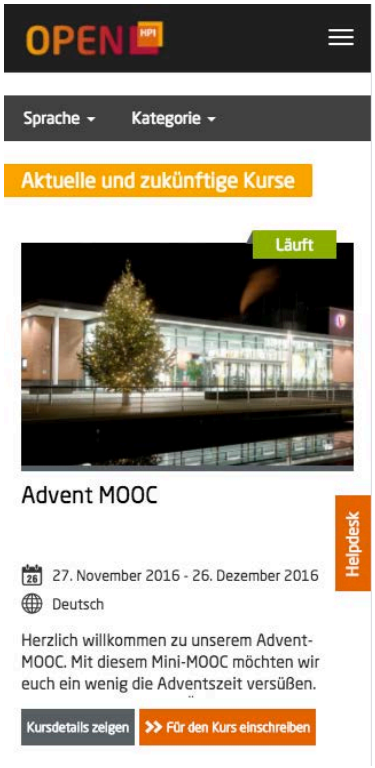
CSS Media Queries (2/3)

Kombination von Media Queries und CSS Raster Layout erlaubt tatsächliches reagierendes Design, z.B. bei openHPI

Spalten nebeneinander



Spalten untereinander



CSS Media Queries (3/3)

Kombination von Media Queries und CSS Raster Layout erlaubt tatsächliches reagierendes Design, z.B. bei openHPI

```
/* Für Desktop / Tablet */
.col-1  {width: 8.33%;}
.col-2  {width: 16.66%;}
.col-3  {width: 25%;}
[...]
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
```

Spalten nebeneinander

```
/* Für Smartphones */
@media only screen and (max-width: 768px) {
  [class*="col-"] {
      width: 100%;
  }
}
```

Spalten untereinander

- Bildschirmbreiten, an denen sich Gestaltung einer Webseite ändert, nennt man **Breakpoints**

CSS Frameworks

- Für Responsive Design und andere wiederkehrende CSS-Anwendungsfälle gibt es **Frameworks** – Sammlungen von vorgefertigten Gestaltungselementen und Hilfsmitteln
- Bekanntes Framework: **Bootstrap** (von Twitter)
  - kommt auch bei openHPI zum Einsatz
  - enthält neben CSS-Elementen auch eine Reihe nützlicher Javascript-Komponenten
  - Tools für Responsive Design:
    - umfassendes Grid-System mit Media Queries
    - erlaubt Ausblenden von Seiten-Elementen bei bestimmten Bildschirmbreiten
    - Hilfsmittel für responsive Images und Videos
  - bietet Unterstützung für → **CSS Präprozessoren**

openHPI

XML – Extensible Markup Language

Prof. Dr. Christoph Meinel

Hasso-Plattner-Institut

Universität Potsdam

Einführung (1/2)

- Eine große Kritik an HTML war immer ein Mangel an Flexibilität
- Der Grund dafür ist die in der **HTML Document Type Definition** (DTD) festgelegte Syntax von HTML
- Autoren von HTML-Dokumenten können HTML-Elementen keine eigene Bedeutung zuweisen oder selbst neue Elemente definieren
  - ist großes Hindernis für die Interaktion mit externen Anwendungen
- Genau hier kommt die **Extensible Markup Language** (**XML)** ins Spiel
  - erlaubt Einführung von neuen und aussagekräftigen Elementen durch die Definition von eigenen DTDs

Einführung (2/2)

- Vorlage für die Entwicklung von XML war, genau wie bei HTML, die komplexe Meta-Markupsprache SGML (Standard Generalized Markup Language), mit der jedes Dokument unterschieden und strukturiert werden kann
- Da XML primär für Webanwendungen entwickelt, wurde nur Teilmenge von SGML zur Definition von XML genutzt, um Interpretation von XML-Dokumenten, Autorenwerkzeuge, Browser, usw. so einfach wie möglich zu halten
- In XML wurden nur absolut notwendige Bestimmungen von SGML umgesetzt, um die vollständige Flexibilität von SGML auch für das WWW zu nutzen

XML – Extensible Markup Language (1/2)

- **XML-Spezifikation** beschreibt nur die Struktur von Dokumenten
- **XML-Dokument** das den XML-Erzeugungsregeln [1] entspricht, ist auch syntaktisch korrekt, z.B.:
  - Dokument hat genau ein Wurzelelement
  - Elemente haben Start- und End-Tag ("<xy></xy>", Kurzform "<xy />")
  - …
- XML-Dokument das der allgemeinen XML-Spezifikation entspricht und die Regeln und Definitionen in der Grammatik einhält, wird **valide** genannt
- Definition der Grammatik kann z.B. durch ein **XML-Schema** oder eine **DTD** festgelegt werden

[1] http://www.w3.org/TR/REC-xml/#sec-well-formed

XML – Extensible Markup Language (2/2)

XML

- Ersetzt HTML nicht, aber ergänzt es in Bereichen, wo HTML Schwächen hat, wie z.B. in der semantischen Unterstützung von Interaktionen mit externen Anwendungen
- Allgemeines Objektmodell erlaubt die Entwicklung von Anwendungen mit maschinellem Zugang und zur Manipulation der Daten
  - **Document Object Model – DOM**
- Bietet offenen Standard zur Beschreibung beliebiger Datenstrukturen, was den Austausch über das WWW vereinfacht und eine integrierte Verbindung zwischen Dokument und Anwendung schafft

XSL – Extensible Stylesheet Language

- XML beschreibt nur die Struktur; die Formatierung eines Dokuments wird durch eine Dokument-Formatbeschreibungssprache in Form von Stylesheets bestimmt ähnlich wie bei HTML
- **Extensible Stylesheet Language – XSL** (eigentlich XSL-FO Formatting Objects) dient zur Beschreibung des Layouts von XML-Dokumenten
- XSL wird oft in enger Verbindung mit **XSL-Transformation** – **XSLT** genutzt
- Typische Verwendung:
  - XML-Dokument wird umgewandelt durch XSLT in ein XSL-Dokument, um vom FO-Prozessor als lesbares/druckbares Dokument dargestellt werden zu können

DTD – Document Type Definition (1/3)

- XML bietet Autoren Möglichkeit zur maßgeschneiderten Definition ihrer Dokumente vermittels eigener Dokumenttypbeschreibung- **Document Type Definitions** – **DTDs**
- Um die **Validität** eines XML-Dokuments überprüfen zu können, wird eine Grammatik benötigt, z.B. definiert als DTD oder XML-Schema
- **XML-Parser** wird benötigt, um XML-Dokumente gemäß zugehöriger DTD zu interpretieren – Deklaration und Regeln der XML-Syntax
- **XML-Schemas** sind komplexer, bieten aber zusätzlich
  - Vererbung,
  - XML-Parsing,
  - ausführlichere Beschreibungen, ...

DTD – Document Type Definition (2/3)

Beispiel für eine DTD – Zeitung

```
<!DOCTYPE newspaper [
        <!ELEMENT newspaper (article+)>
        <!ELEMENT article (title, introduction, text)>
        <!ELEMENT title (#PCDATA)>
        <!ELEMENT introduction (#PCDATA)>
        <!ELEMENT text (#PCDATA)>

        <!ATTLIST article author CDATA #REQUIRED>
        <!ATTLIST article lecturer CDATA #IMPLIED>
        <!ATTLIST article date CDATA #IMPLIED>

        <!ENTITY newspaper "New York Times">
        <!ENTITY publisher "Arthur Ochs Sulzberger Jr.">
]>
```

Hinweis: Entitäten erlauben die Definition von Konstanten (Sonderzeichen oder häufig verwendetet Textteile)

DTD – Document Type Definition (3/3)

Beispiel eines XML-Dokuments entsprechend der newspaper.dtd

```
<?xml version="1.0" ?>
<!DOCTYPE newspaper SYSTEM "./newspaper.dtd">
<newspaper>
   <article author="Hasso Plattner" date="2015-01-07">
     <title>User Oriented</title>
     <introduction>Introduction text …</introduction>
     <text>Main article text</text>
   </article>
   <article author="Christoph Meinel" date="2015-01-09">
     <title>New Student Welcome</title>
     <introduction>Introduction text …</introduction>
     <text>Main article text</text>
   </article>
</newspaper>
```

Ist valides XML-Dokument in Bezug auf newspaper.dtd

XML – Verknüpfungskonzept (1/2)

Erinnerung: HTML Verknüpfungskonzept mit dem Hyperlink-Element <a> ist sehr eingeschränkt:

- HTML-Hyperlink verlinkt nur zwei Ressourcen
- HTML-Hyperlink ist immer unidirektional
- HTML-Hyperlink ist Teil der HTML-Form wo der Link herstammt
- HTML-Hyperlink beeinflusst nicht die Darstellung des Link-Ziels im Browser

XML – Verknüpfungskonzept (2/2)

Für XML wurden zwei separate Sprachstandards entwickelt, um HTML-Verknüpfungskonzept zu verallgemeinern und für die Definition von weiteren Verbindungen zwischen Informationsressourcen:

- **XML Linking Language** – **Xlink**
  - definiert auch multidirektionale Verbindungen
- **XML Pointer Language** – **XPointer**
  - erweitert die XPath-Spezifikation um auf Teile von XML-Dokumenten zu verweisen
  - Beispiel:
    - xlink:href="cds.xml#xpointer(/ABBA/Waterloo)"

Bereichsspezifische Beschreibungssprachen

Mit dem XML-Metasprachkonzept zur Definition eigener DTD ist der Weg für Entwicklung eigener bereichsspezifischer Beschreibungssprachen frei, z.B.:

- Mathematical Markup Language – **MathML**
- Chemical Markup Language – **CML**
- Synchronized Multimedia Integration Language – **SMIL**
- Scalable Vector Graphics – **SVG**
- …

Da XML mehr ein Werkzeug für Spezialisten ist, kommt der Endnutzer nur in Kontakt mit XML bei verschiedenen Anwendungen oder mit XHTML als XML-konformem HTML