Webentwicklung

Frontend: Website-Gestaltung

Inhalt dieser Einheit

1. **Strukturierung von Webseiten**:
   Was kommt wie wohin? Tabellen, Frames und CSS
2. **Semantisches Markup**:
   Was, nicht *Wie* – mit HTML5
3. **Optische Aufwertung**:
   ohne CSS-Kenntnisse
4. **Responsiveness**:
   Anpassen an die Nutzer-Situation
5. **Debugging**:
   Ausprobieren und Defekte ausmerzen
6. **Standard-Konformität**:
   Validieren geht über Probieren

Wdh.: HTML-Eigenschaften

- HTML-Dokumente sollten sein:
  - **wohlgeformt**, d.h. syntaktisch korrekt
    - Tags, Attribute, Tag-Klammerung, Sonderzeichen
  - **valide**, d.h. korrekte Element- und Attribut-Namen, sowie deren Zuordnung und Schachtelung
    - HTML-Spezifikation
  - deklarativ bzw. **semantisch**, d.h. die Rolle/Funktion der Elemente wird ausgezeichnet
    - nicht deren Darstellung
  - **minimal**, d.h. keine unnötigen Elemente/Attribute
    - weniger Schreibaufwand, leichter verständlich für Mensch und Maschine

Wie macht man das nun

in der **Praxis**?

Strukturierung von Webseiten

Was kommt wie wohin?

Moment: Website oder Webseite?

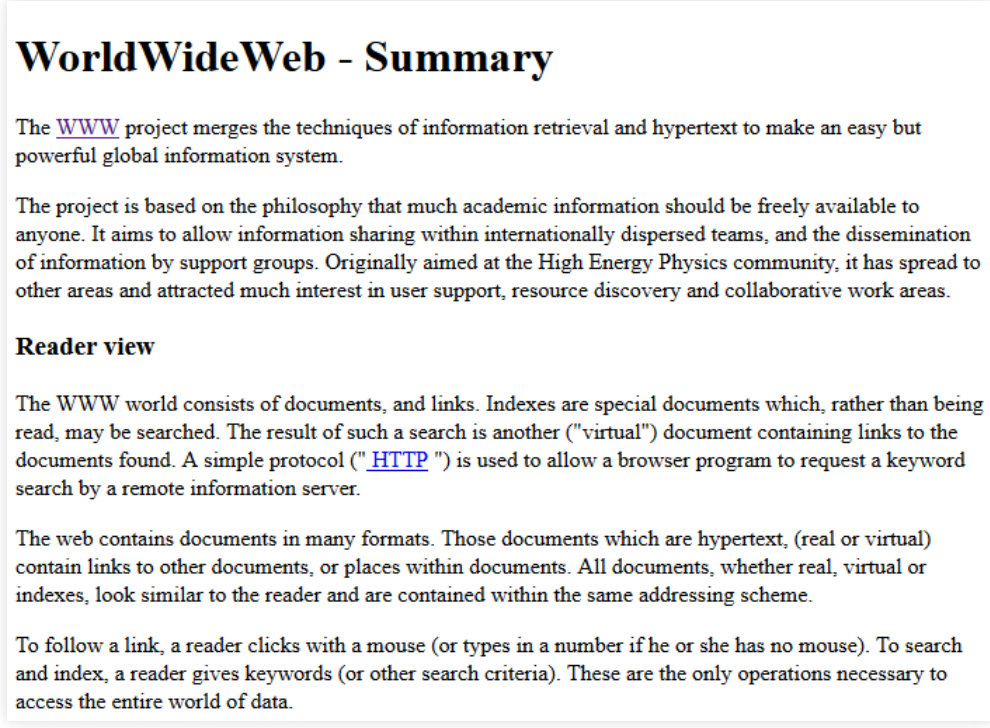
- dt. *Webseite*: engl. “Web page”
  - ein einzelnes HTML-Dokument
  - auch “Seite” oder “Homepage” bzw. “Unterseite”
- engl. *Website*: dt. “Webauftritt” oder “Webpräsenz”
  - engl. *site* = dt. “Ort”
  - eine Sammlung von untereinander verbundenen Webseiten

```
<!-- index.html -->
<a href="faq.html">FAQ</a>

<!-- faq.html -->
<a href="index.html">zur Startseite</a>
```

Erste Websites: Dokumente & Links

- frühes Web: lose verknüpfte Dokumente
  - Inhalte im Mittelpunkt
  - Navigation über Hyperlinks in den Dokumenten
    - keine zentrale Form der Navigation
    - “browsing the web”
- kaum Gestaltungsbedarf
- HTML-Code: pure Semantik



WorldWideWeb - Summary

The WWW project merges the techniques of information retrieval and hypertext to make an easy but powerful global information system.

The project is based on the philosophy that much academic information should be freely available to anyone. It aims to allow information sharing within internationally dispersed teams, and the dissemination of information by support groups. Originally aimed at the High Energy Physics community, it has spread to other areas and attracted much interest in user support, resource discovery and collaborative work areas.

Reader view

The WWW world consists of documents, and links. Indexes are special documents which, rather than being read, may be searched. The result of such a search is another ("virtual") document containing links to the documents found. A simple protocol (" HTTP ") is used to allow a browser program to request a keyword search by a remote information server.

The web contains documents in many formats. Those documents which are hypertext, (real or virtual) contain links to other documents, or places within documents. All documents, whether real, virtual or indexes, look similar to the reader and are contained within the same addressing scheme.

To follow a link, a reader clicks with a mouse (or types in a number if he or she has no mouse). To search and index, a reader gives keywords (or other search criteria). These are the only operations necessary to access the entire world of data.

Quelle: http://info.cern.ch/hypertext/WWW/Summary.html

Erste Webauftritte, bis Mitte 1990er

- Unternehmen wollen sich präsentieren
- neue Bedürfnisse:
  - Individualität, Wiedererkennungswert
  - hierarchische Website-Struktur
- Ergebnis:
  - Umrahmung der Inhalte mit Bannern (Logos, etc.) & Navigationsmenüs
- “Layout”: eindimensional

Quelle: https://web.archive.org/web/19961022175210/http://www.ibm.com:80/

Ab Mitte 1990er: Webdesign

- Neu: zweidimensionale Struktur
  - Vorbild: Gestaltungsraster (wie etwa Tageszeitung)
- Verschiedene Ansätze
  1. Tabellen
  2. Frames
  3. Div-Container & CSS-Frameworks

Quelle: https://www.webpagefx.com/blog/web-design/the-evolution-of-web-design/

Running Example

Navigation

- Startseite
- Über mich
- Kontakt

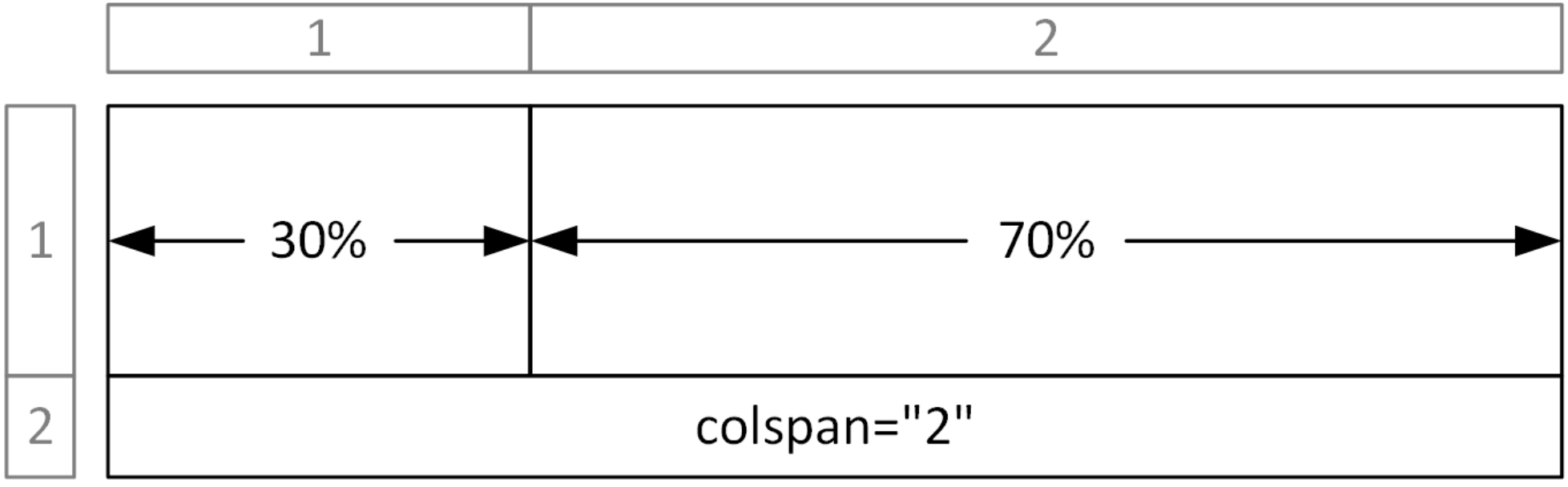
Startseite

Willkommen auf meiner Homepage!

Copyright 2017

1. Ansatz: Tabellenlayout

- Idee:
  - Tabelle mit expliziten Zellen-Abmessungen (Höhe/Breite)
  - Zellen verbinden, wenn nötig



Beispiel: Als Tabellenlayout

```
<table>
  <tr>
    <td width="30%"> <!-- Navigation -->
      <h2>Navigation</h2>
      <ul>
        <li>Startseite</li>
        <li>Über mich</li>
        <li>Kontakt</li>
      </ul>
    </td> <!-- Ende: Navigation -->
    <td width="70%"> <!-- Hauptteil -->
      <h1>Startseite</h1>
      <p>Willkommen auf meiner Homepage!</p>
    </td> <!-- Ende: Hauptteil -->
  </tr>
  <tr>
    <td colspan="2">Copyright 1998</td>
  </tr>
</table>
```

1. Ansatz: Tabellenlayout

- tatsächlich: große und geschachtelte Tabellen
  - dutzende Zeilen und Spalten
  - viel, schwer wartbarer HTML-Code
- Techniken
  - Spacer-GIFs
    - Positionierung von Inhalten
    - gestreckte 1x1-Pixel-Grafiken
  - Slicing
    - Zerlegung einer Grafikvorlage in Teilbilder

Quelle: https://www.webpagefx.com/blog/web-design/the-evolution-of-web-design/

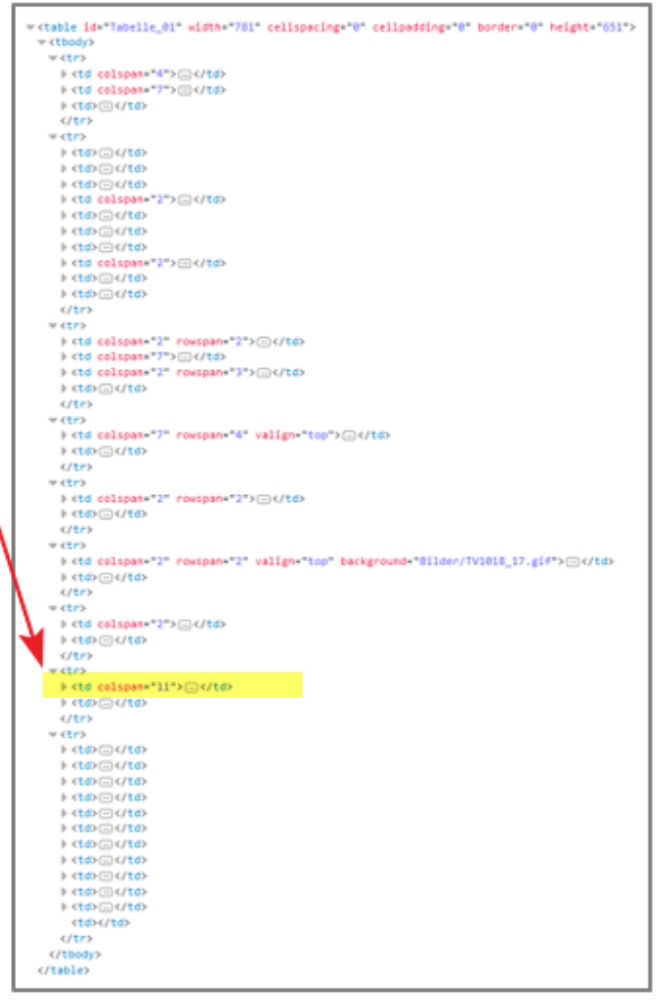
Slicing: Beispiel

Quelle: https://ekiwi.de/workshops/photoshop/slicing/slicing.htm

Slicing: Beispiel

19 Bild-Dateien

HTML:
- **9 Zeilen**
- **12 Spalten**
- **39 Zellen, davon 1 für Inhalt**



Quelle: https://ekiwi.de/workshops/photoshop/slicing/slicing.htm

1. Ansatz: Tabellenlayout

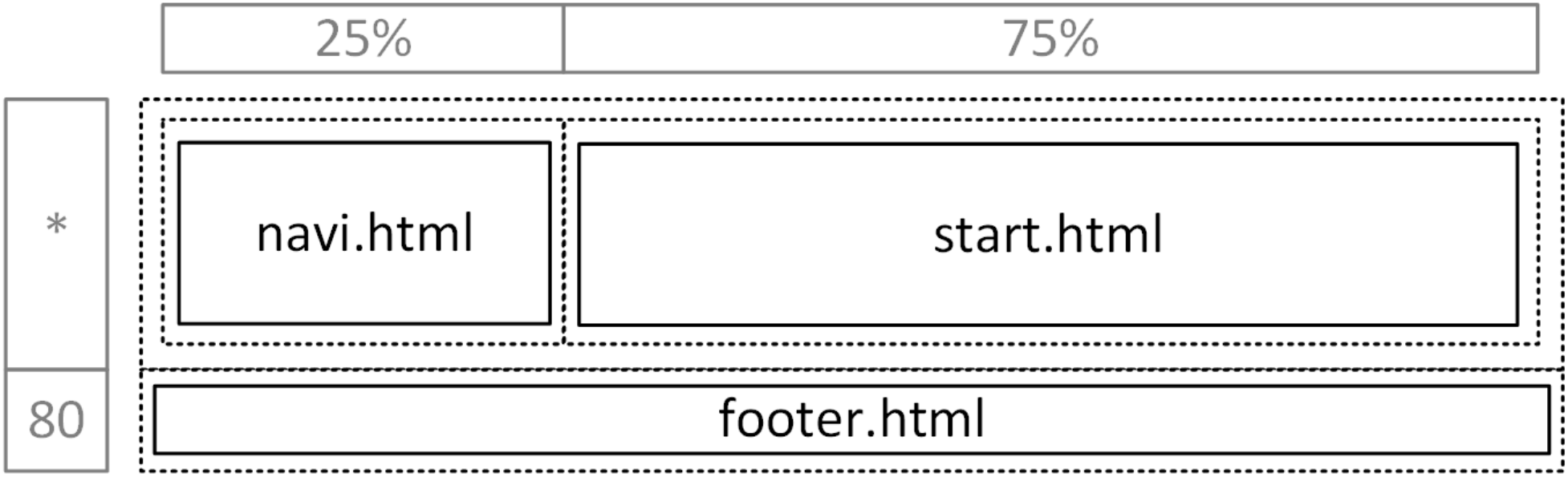
- **Vorteile**
  - gestalterische Freiheit, “pixelgenaue Umsetzung” von Vorlagen
- **Nachteile**
  - unübersichtlicher HTML-Quellcode
    - keine Trennung von Inhalt und Darstellung
    - *valide* und *wohlgeformt*, aber nicht *semantisch*
  - unflexibel bei Änderungen
    - z.B. “Größere Schrift in der Navigationsleiste”
  - Wiederholung von Quellcode (z.B. für Banner und Navigation) für jede Seite
- **Fazit**: Nicht mehr benutzen!

Gestaltungsansätze

1. ~~*Tabellen*~~
2. Frames
3. Div-Container & CSS-Frameworks

2. Ansatz: Frames

- Idee:
  - Aufteilung in einzelne HTML-Dokumente
    - wiederkehrende Elemente (z.B. Navigation) als eigene HTML-Dokumente
  - Zusammenfügung der Teile in einem zentralen Dokument
    - `frameset`-Element (statt `body`), mit `frame`-Kindern
  - horizontale und vertikale Aufteilung, wenn nötig geschachtelt



2. Ansatz: Frames

```
<!-- index.html --->
<frameset rows="*,80"> <!-- statt body-Element -->
  <frameset cols="25%,75%">
    <frame src="navi.html" name="navi">
    <frame src="start.html" name="main">
  </frameset>
  <frame src="footer.html" name="footer">
</frameset>
```

```
<!-- navi.html --->
<h2>Navigation</h2>
<ul><li><a href="kontakt.html" target="main">Kontakt</a></li></ul>
```

```
<!-- start.html --->
<h1>Startseite</h1>
<p>...</p>
```

```
<!-- footer.html --->
<p>Copyright 2000</p>
```

2. Ansatz: Frames

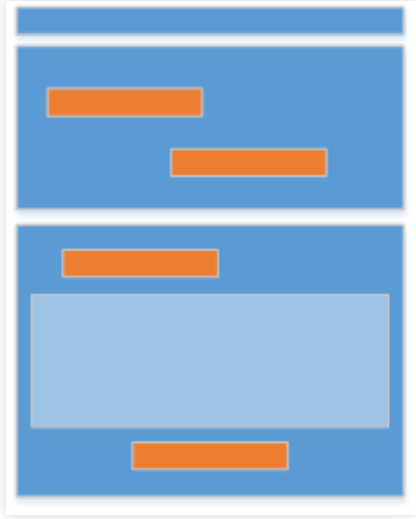
- **Vorteile**
  - wiederverwendbarer Code, dadurch übersichtlicher
- **Nachteile**
  - Frames sind nicht semantisch
  - Verlinkung von Inhalten?
    - auf `index.html` (d.h. immer “Homepage” statt Unterseite) oder
    - auf Unterseite, z.B. `kontakt.html` (d.h. es fehlt das umgebende Layout)
  - angezeigter Titel ist statisch
    - denn der kommt vom Frameset-Dokument `index.html`
- **Fazit**
  - veraltet, nicht mehr verwenden
    - in HTML5: *nicht konformes Feature*, darf nicht benutzt werden
    - Ausnahme: `iframe`

Gestaltungsansätze

1. ~~*Tabellen*~~
2. ~~*Frames*~~
3. Div-Container & CSS-Frameworks

Elemente ohne Semantik

- semantikfreie HTML-Elemente:
  - `div`: für Blöcke (volle Breite)
  - `span`: für Elemente im Textfluss (“inline”)
- Idee:
  - Unterteilung der Webseiten in `div`-**Container** (*division*)
    - (ähnlich wie die lineare Struktur früherer Webseiten)
  - Layout (d.h. Anordnung und Größe) über sog. **CSS-Regeln**
    - (Details dazu in der nächsten Einheit)



3. Ansatz: CSS-Frameworks

- Idee ab Mitte 2000er:
  - wiederkehrende Aufgaben (u.a. Layout)
  - → wiederverwendbare CSS-Regeln
  - gebündelt in **CSS-Frameworks**
    - kein CSS selbst schreiben müssen
    - sondern einfach als Blackbox verwenden
  - für Layout wichtiger Teil davon: **Grid-Systeme**
- frühe Vertreter
  - 2008: 960 Grid System (Nathan Smith)
  - 2011: Twitter veröffentlicht **Bootstrap** 1.0
  - aktuell: **Bootstrap 3.3**
    - (4.0 ist bald fertig)

3. Ansatz: CSS-Frameworks

- Verwendung:
  - Einbindung eines CSS-Stylesheets im `head`-Element

```
<link rel="stylesheet" href="bootstrap.min.css">
```

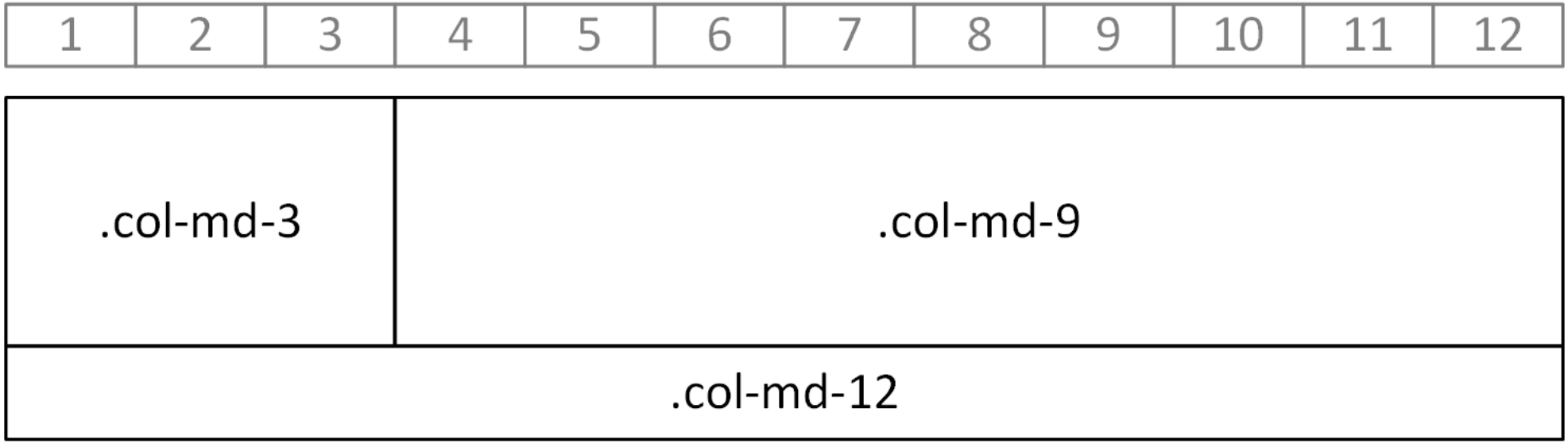
  - Elemente im `body`: vordefinierte Werte im `class`-Attribut
    - (Details zu all dem gibt's in der nächsten Einheit)
- Grid:
  - virtuelles Raster mit (z.B.) 12 Spalten
  - Elemente (`div`-Container) sind *n* Spalten breit
  - Angabe der Breite: über `class`-Attribut

```
<div class="col-md-3">...</div> <!-- 3 Spalten breit -->
```

  - Inhalte werden automatisch auf dem Raster positioniert

3. Ansatz: CSS-Frameworks

```
<div class="container">
  <div class="col-md-3" id="navi"> <!-- Navigation --> </div>
  <div class="col-md-9" id="content"> <!-- Inhalt  --> </div>
  <div class="col-md-12" id="footer"> <!-- Footer  --> </div>
</div>
```



3. Ansatz: CSS-Frameworks

- **Vorteile**:
  - Übersichtliches, wartbares, semantisches Markup
  - Trennung von Inhalt und Darstellung(sumsetzung)
  - Keine aktiven CSS-Kenntnisse nötig
- **Nachteile**:
  - Aspekte der Darstellung mit Inhalten vermischt
  - etwas umfangreicheres Markup (`div`s)
  - Wiederholung von Quellcode in jedem HTML-Dokument
- **Fazit**
  - für die meisten Layouts eignen sich Grid-Systeme

Gestaltungsansätze

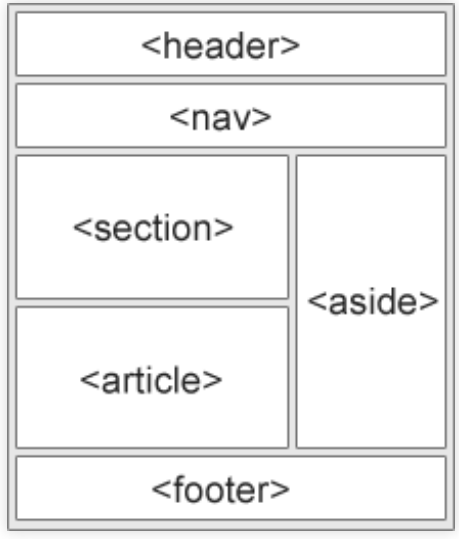
1. ~~*Tabellen*~~
2. ~~*Frames*~~
3. Div-Container & CSS-Frameworks ✓

Semantisches Markup

Was**, nicht **Wie

Semantik?

- Erinnerung: `div`-Elemente haben keine Semantik
  - häufig für wiederkehrende Webseiten-Elemente benutzt
    - Navigation, Titelleiste, …
  - Kennzeichnung der Funktion: über `id` oder `class`
- Problem dabei?
  - Jede Webseite: eigene Bezeichner, erschwert maschinelle Verarbeitung (Suchmaschinen, Lesehilfen, …)
- HTML5: neue Standard-Elemente, z.B.
  - Kopf-, Haupt- und Fußbereich: `header`, `main` und `footer`
  - Navigation: `nav`
  - Inhalt: `article`, unterteilt mit `section`
  - Zusatz-Informationen: `aside`



Unser Bsp. mit HTML5-Elementen

```
<nav>
    <h2>Navigation</h2>
    <ul>
        <li>Startseite</li>
        <li>Über mich</li>
        <li>Kontakt</li>
    </ul>
</nav>

<section>
    <h1>Startseite</h1>
    <p>Willkommen auf meiner Homepage!</p>
</section>

<footer>Copyright 2017</footer>
```

- Darstellung: wie `div`, d.h. ohne Standard-Layout
  - ohne Grid vs. mit Grid (`class="col-md-*"`)
- Weitere Infos: SelfHTML-Tutorial

Fußnote der Geschichte

- HTML 4: schon mal Fokus auf semantisches Markup
  - Früher (HTML 3.2): z.B. `center` und `font`
  - Seit HTML 4: *deprecated*, stattdessen `div` und `span`

```
<!-- alt -->
<center>Inhalt</center>
<font size="3">Big!</font>
```

```
<!-- neu -->
<div style="text-align: center">Inhalt</div>
<span style="font-size: 3em">Big!</span>
```

Aber meine Seiten

sehen so **langweilig** aus!

Optische Aufwertung

ohne CSS-Kenntnisse

Absätze und Typografie

- **Bootstrap** stellt eine Reihe nützlicher Klassen zur Verfügung

```HTML
<p class="lead">Wichtiger Inhalt, den jeder und jede unbedingt
   sehen soll.</p>
<p>Ein ganz normaler Absatz.</p>
<p>Ein weiterer normaler Absatz.</p>
```

Wichtiger Inhalt, den jeder und jede unbedingt sehen soll.

Ein ganz normaler Absatz.

Ein weiterer normaler Absatz.

- Mehr Klassen für typografische Auszeichung:
  - https://getbootstrap.com/docs/3.3/css/#type

Farbliche Hervorhebungen

```HTML
<p class="text-primary">Wichtiger Inhalt, den jeder und jede
   unbedingt sehen soll.</p>
<p>Ein ganz normaler Absatz, farblich unspektakulär.</p>
<p class="bg-info">Ein Info-Text, der aus dem normalen Textfluss
   herausstich.</p>
```

Wichtiger Inhalt, den jeder und jede unbedingt sehen soll.

Ein ganz normaler Absatz, farblich unspektakulär.

Ein Info-Text, der aus dem normalen Textfluss herausstich.

Weitere Möglichkeiten mit Boostrap

- Siehe: https://getbootstrap.com/docs/3.3/css
  - (viele Klassen sind nicht streng *semantisch*)
- Tabellen
  - z.B. hervorgehobene Zeilen
- Formulare
- Bilder
  - z.B. abgerundete Ecken

Aufwertung mit Icons

- Unicode (https://unicode-table.com/de/):
  - direkt im Quelltext (z.B. ☂)
  - als Entity (z.B. `&#9730;`)
- Font Awesome (http://fontawesome.io/get-started/)
  - Einbindung eines CSS-Stylesheets

```
<link rel="stylesheet" href="font-awesome.min.css">
```

  - Hunderte Icons
  - Verhalten sich wie Text
- Glyphicons (http://glyphicons.com/)
  - ähnlich wie Font Awesome

Font-Awesome: Beispiele

```HTML
<p class="text-primary">
    <i class="fa fa-birthday-cake"></i> Happy Birthday
</p>

<a class="btn btn-warning" href="#">
    <i class="fa fa-bug"></i> Submit Bug Report
</a>
```

Responsiveness

Anpassen an die Nutzer-Situation

Responsiveness

- Anpassung einer Website an das Anzeigemedium
- hat viele Facetten, zwei davon:
  - **Anordnung** und **Größe** von Inhalten in Abhängigkeit der Bildschirmgröße
    - Desktop-Monitor: viel Platz für Elemente nebeneinander
    - Smartphone: eher vertikales Scrollen
  - **Ausblenden** von Inhalten
- technisch:
  - Webbrowser kennt u.a. die Größe des **Viewports**
  - CSS (und JavaScript) können diese abfragen
    - (Details in den jeweiligen Einheiten)
  - CSS-Frameworks (wie Bootstrap) erleichtern das

Responsiveness mit Bootstrap

- Beispiel von vorhin

```
<div class="container">
    <div class="col-md-3" id="navi"> <!-- Navigation --> </div>
    <div class="col-md-9" id="content"> <!-- Inhalt  --> </div>
    <div class="col-md-12" id="footer"> <!-- Footer  --> </div>
</div>
```

- Das “`md`“ im `class`-Wert steht für “medium”, d.h.
  - Breiten (3, 9, und 12) gelten für mind. mittlere Geräte (≥992px)
  - kleinere Geräte (Tablet, Smartphone): volle Breite
- Demo

Verschiedene Layouts

```
<div class="container">
  <div class="row"> <!-- empfohlen: zur Gruppierung -->
    <div class="col-md-3 col-sm-6" id="navi"> <!-- ...     --> </div>
    <div class="col-md-9 col-sm-6" id="content"> <!-- ... --> </div>
  </div>
  <div class="row">
    <div class="col-xs-12" id="footer"> <!-- ...          --> </div>
  </div>
</div>
```

- Das sind drei verschiedene Layouts auf einmal:
  - Auf Laptop und größer (`md`): wie bisher (3 und 9 Spalten)
  - Auf Tablet (`sm`): Navigation & Inhalt je halber Bildschirm
  - Auf Smartphone (`xs`): alles auf voller Breite
- Demo

Inhalte aus- und einblenden

```
<span class="visible-xs-inline">Hallo, Smartphone.</span>
<span class="visible-sm-inline">Hallo, Tablet.</span>
<span class="visible-md-inline">Hallo, Laptop.</span>
<span class="visible-lg-inline">Hallo, Desktop.</span>

<!-- Positive vs. negative Ausdrücke -->
<span class="visible-xs-inline visible-sm-inline
             visible-md-inline">Hallo, kleines Gerät.</span>
<span class="hidden-lg">Hallo, kleines Gerät.</span>
```

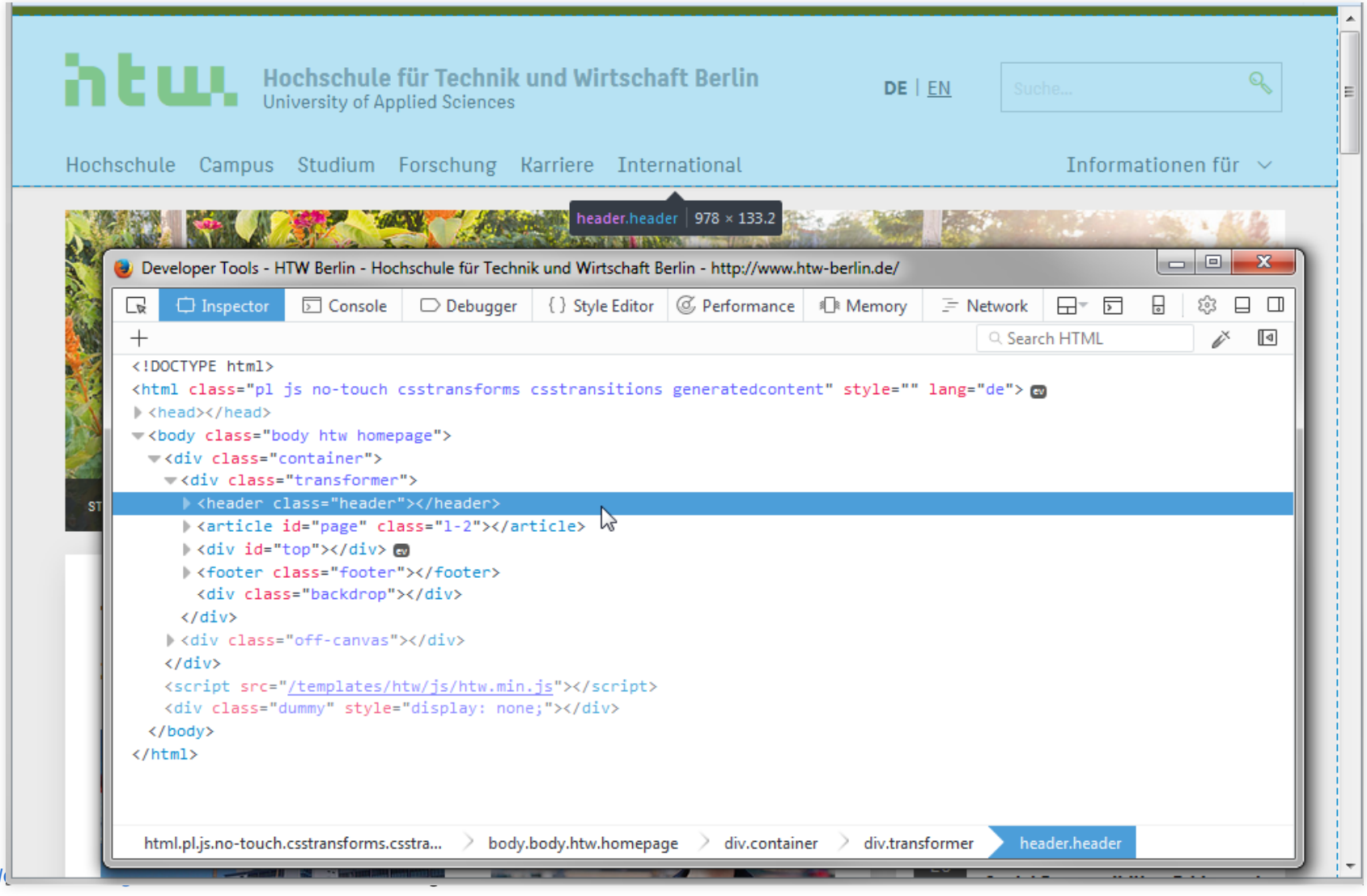
- Alle denkbaren Konfigurationen lassen sich ausdrücken
  - https://getbootstrap.com/docs/3.3/css/#responsive-utilities-classes
- Demo

Debugging

Ausprobieren und Defekte ausmerzen

Debugging-Hinweise

- Moderne Browser: Entwicklertools via `F12`



Developer-Tools

- Für HTML interessant:
  - **Inspector** (Firefox)/**Elements** (Chrome):
    - Zeigt den DOM, also den aus dem HTML-Code generierten Baum
      - (Rechtsklick > “View Source”: nur HTML-Code)
    - Mouse-Over über Knoten im DOM: Hervorhebung in grafischer Anzeige
    - Umgekehrt: Rechtsklick > “Inspect Element”
    - DOM-Knoten und ihre Attribute können direkt verändert werden
  - Geräte-Simulation (**CTRL+SHIFT+M**)
    - Verschiedene Bildschirmgrößen
    - Touch-Navigation mit Maus-Cursor
- Live-Demo:
  - http://ai-bachelor.htw-berlin.de/

Standard-Konformität

Validieren geht über Probieren

W3C-Validator

- https://validator.w3.org/nu/
- Prüft HTML-Dokumente gegen die Spezifikation
  - Meldungen: *Errors*, *Warnings* und *Infos*
  - mit konstruktiven Hinweisen
- Beispiel: Validierung der HTW-Seite

Zusammenfassung: Heutige Inhalte

- wünschenswerte Eigenschaften von HTML-Dokumenten kennen
  - wohlgeformt, valide, semantisch, minimal
  - semantisches Markup (HTML5) einsetzen können
- mit Hilfe von CSS-Frameworks:
  - einfache Layouts umsetzen können
  - gezielt optische Akzente setzen (Farben, Icons, …)
  - Websiten *responsive* gestalten können
- einfache, aber wichtige Werkzeuge einsetzen können
  - Debugging
  - Validierung

Danke!