

Multimedia Engineering II

02 HTML CSS JS

Johannes Konert



BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences



Agenda

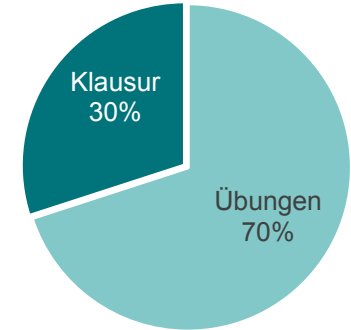
- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

Wiederholung vom letzten Mal

- Übungen sind bestanden, wenn mind 50% der Pflichtpunkte
- Bearbeitung in Gruppen mit persönlicher Abnahme



- Klausur im PZR1:
 - bisher Do, 26.01. 12:00 Uhr,
 - **Ggf. 13 Uhr?**
- **Moodle -Umfrage**
 - Verlängert bis Di, 18.10. 20 Uhr
 - Auswertung zusammen mit Karten nächstes Mal

	Thema	Dauer (Wochen)	Punkte Pflicht- teil	Punkte Bonus- teil
Ü1	Client-Website (miniYouTube)	2	5	2
Ü2	Server mit node.js	2	10	5
Ü3	API mit node.js	2	10	5
Ü4	REST-API mit node.js	3	15	5
Ü5	mongoDB	2	10	5
Ü6	Backbone.js	2	10	5

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

HTML5, CSS3, ECMAScript5: Zuständigkeiten

- Ordnen Sie die folgenden Elemente den 3 rechts zu.
 - 1) **Alleine** in Stillarbeit durchgehen (2min)
 - 2) Mit 2-3 Nachbarn diskutieren und begründen (2min)
- Unklare Fälle klären wir anschließend mit allen

Asynchrones Laden	Layout
Responsive Design	Video-Einbindung
Animation	Canvas
Semantik	Interaktionssteuerung



HTML5

CSS3

ECMAScript5

HTML5, CSS3, ECMAScript5: Zuständigkeiten

- Ordnen Sie die folgenden Elemente den 3 rechts zu.
 - 1) **Alleine** in Stillarbeit durchgehen (2min)
 - 2) Mit 2-3 Nachbarn diskutieren und begründen (2min)
- Unklare Fälle klären wir anschließend mit allen

HTML5

Semantik

Canvas

Video-Einbindung

CSS3

Responsive Design

Layout

Animation

Interaktionssteuerung

ECMAScript5

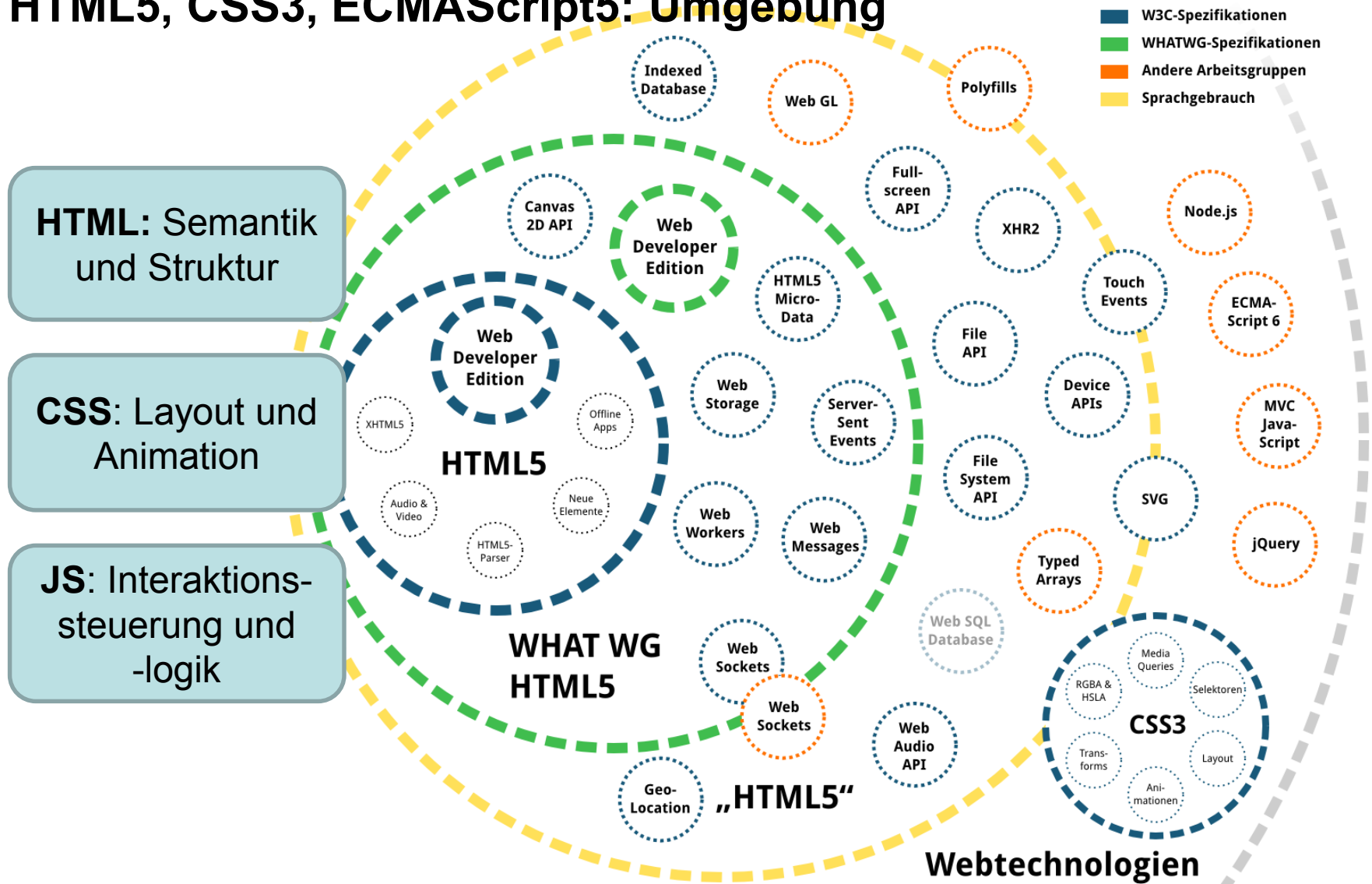
Interaktionssteuerung

Asynchrones Laden

Canvas-Steuerung

Animation

HTML5, CSS3, ECMAScript5: Umgebung



Agenda

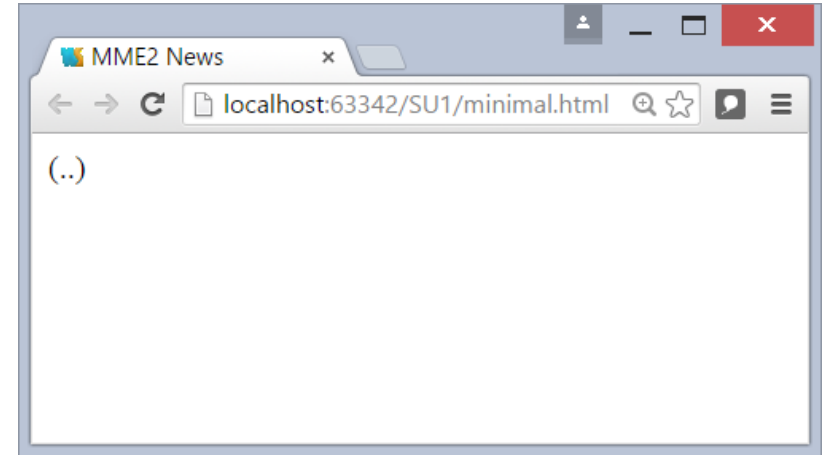
- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

HTML5 Tags und Struktur

```
<!DOCTYPE html>
<html>
<head lang="de">
  <meta charset="utf-8">
  <title>MME2 News</title>
  <link href="base.css" rel="stylesheet" type="text/css">
  <script src="app.js"></script>
</head>
<body>

(..)

</body>
</html>
```



HTML5

■ Welches Tag wofür?

- Bestimmen Sie in Teams (2-3)
für die Tags einen Verwendungszweck (3min)
- Achtung: Einige Tags sind nicht zu verwenden



1. `<td>`
2. `<a>`
3. `<aside>`
4. ``
5. `<body>`
6. `<c>`
7. `<center>`
8. `<canvas>`
9. `<div>`
10. `<input>`
11. `<main>`
12. `<nav>`
13. `<output>`
14. `<section>`
15. ``
16. `<video>`

HTML5

1. `<td>`
2. `<a>`
3. `<aside>`
4. ~~``~~
5. `<body>`
6. ~~`<c>`~~
7. ~~`<center>`~~
8. `<canvas>`
9. `<div>`
10. `<input>`
11. `<main>`
12. `<nav>`
13. `<output>`
14. `<section>`
15. ``
16. `<video>`

1. Tabellenzelle
2. Verlinkung (Hypertext)
3. Zusatzinformation mit Bezug zur Umgebung
4. Ungültig! (früher mal für Fettdruck)
5. Definiert den sichtbaren Dokumentbereich im Browser
6. Ungültig!
7. Ungültig! (früher mal für Zentrierung)
8. Zeichenfläche
9. Blockelement
10. Eingabeelement
11. Hauptteil des Dokumentes (Einzigartig pro URL)
12. Navigationselemente der Seite/App
13. Zur Ausgabe von Berechnungen in `<form>` Elementen
14. Abschnitt der Seite (geschachtelt, ggf. mit `<article>`)
15. Inline-Element
16. Video-Einbettung (optional mit Steuerelementen)

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5

- Tags

Struktur

- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

HTML5 Tags und Struktur

```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="utf-8">
  <meta name="description" content="..">
  <meta name="keywords" content="MME2">
  <meta name="author" content="J.Konert">
  <title>MME2 News</title>
  <link href="base.css" rel="stylesheet" type="text/css">
  <!--[if lt IE 9]>
    <script
src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
  <![endif]-->
  <script src="app.js"></script>
</head>
<body>
<nav>
  <a href="#home">Willkommen</a>
  <a href="#news">Neuste Meldungen</a>
  <a href="#imprint">Impressum</a>
</nav>
<section id="home"><h1>Willkommen</h1> (..)</section>
<main>
  <section id="news">
    <article>(..)</article>
  </section>
</main>
(..)
</body>
</html>
```

HTML5 Tags und Struktur






```
<!DOCTYPE html>
<html>
  (...)
  <body>
    <nav>
      <a href="#home">Willkommen</a>
      <a href="#news">Neuste Meldungen</a>
      <a href="#imprint">Impressum</a>
    </nav>
    <section id="home"><h1>Willkommen</h1> (...)</section>
    <main>
      <section id="news">
        <article>
          <h1>AngularJS Version 2.0 erschienen</h1>
          <p>Dieses Semester...</p>
          <details>Erschienen:<time>15.09.2016 14:23 Uhr</time></details>
          <aside>
            <h2>Schon gewusst?</h2>
            <p>Normalerweise...</p>
          </aside>
        </article>
      </section>
    </main>
    <section id="imprint"></section>
  </body>
</html>
```

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick



HTML5 Video - Kompatibilität (erst seit 2015!)

					
H.264					
OggTheora					
WebM					

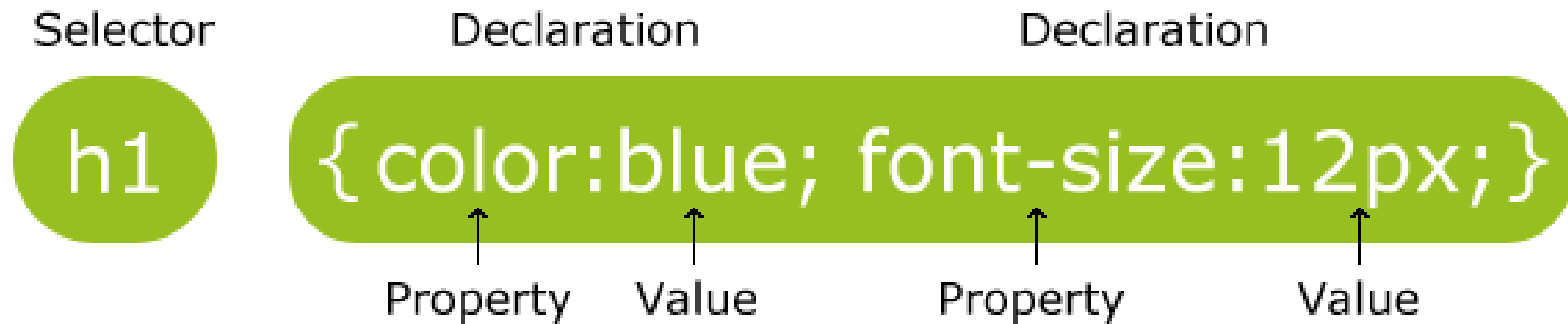
HTML5 Video

```
<!DOCTYPE html>
<html>
<head lang="de">
  <meta charset="utf-8">
  <title>MME2 Video Example</title>
  <link href="base.css" rel="stylesheet" type="text/css">
  <script src="app.js"></script>
</head>
<body>
(..)
<main>
  <section id="topvideo">
    <video controls>
      <source src="http://.../small.mp4" type="video/mp4">
    </video>
  </section>
</main>
(..)
</body>
</html>
```

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

CSS3



- Selektor: welche DOM-Elemente mit Stil versehen werden
- Deklaration: wie der Stil aussehen soll

CSS3 Selektor-Varianten

- Es geht fast alles. Vieles (zu kompliziertes) ist nicht sinnvoll.
- Komplette Vielfalt im Web http://www.w3schools.com/cssref/css_selectors.asp

Einfachselektoren

Syntax	Beispiel	Wirkung
*	* { }	Alle Elemente
.Klasse	.intro	Alle Elemente mit class=„intro“
#id	#navi	Das Element mit der id=„navi“
Element	h1	Alle Überschriften <h1>
[Attribut]	[target]	Alle Elemente die ein Attribut <...target=„...“> haben
[Attribut*="Wert"]	[title*="Beispiel"]	Alle Elemente, deren title die Zeichen „Beispiel“ enthält
Selektor, Selektor	h1, h2	Sowohl alle Überschriften <h1> als auch <h2>

Kombination:

```
div#navi a.beuth-intern[href*="fb6."] {font-weight: bold}
```

CSS3 Selektor-Varianten

- Es geht fast alles. Vieles (zu kompliziertes) ist nicht sinnvoll.
- Komplette Vielfalt im Web http://www.w3schools.com/cssref/css_selectors.asp

Einfachselektoren

Syntax	Beispiel	Wirkung
*	* { }	Alle Elemente
.Klasse	auch: [class="intro"]	Alle Elemente mit class=„intro“
#id	auch: [id="navi"]	Das Element mit der id=„navi“
Element	h1	Alle Überschriften <h1>
[Attribut]	[target]	Alle Elemente die ein Attribut <...target=„...“> haben
[Attribut*="Wert"]	[title*="Beispiel"]	Alle Elemente, deren title die Zeichen „Beispiel“ enthält
Selektor, Selektor	h1, h2	Sowohl alle Überschriften <h1> als auch <h2>

Kombination:

```
div#navi a.beuth-intern[href*="fb6."] {font-weight: bold}
```

CSS3 Selektor-Varianten

- Es geht fast alles. Vieles (zu kompliziertes) ist nicht sinnvoll.
- Komplette Vielfalt im Web http://www.w3schools.com/cssref/css_selectors.asp

Kombinationsselektoren

Syntax	Beispiel	Wirkung
Selektor Selektor	main p	Alle Paragraphen die (beliebig tief) als Kindelemente des <main>-Tags vorkommen
Selektor > Selektor	ol > li	Alle Listenelemente die direkte (!) Kindelemente einer sortierten Liste sind
Selektor + Selektor	h1 + p	Jeder Absatz der direkt(!) als nachfolgender Knoten nach einer H1 Überschrift kommt

CSS3 Selektor-Varianten

- Es geht fast alles. Vieles (zu kompliziertes) ist nicht sinnvoll.
- Komplette Vielfalt im Web http://www.w3schools.com/cssref/css_selectors.asp

Pseudoklassenselektoren

Syntax	Beispiel	Wirkung
<code>:active</code>	<code>a:active</code>	Aktuell angeklickter Link
<code>:hover</code>	<code>a:hover</code>	Aktueller Link mit mouseover
<code>:focus</code>	<code>input:focus</code>	Aktuelles Eingabefeld mit Fokus (Tab-Steuerung)
<code>:target</code>	<code>a:target</code>	Element, welches das Anker-Ziel des Links ist (Nützlich bei <code></code>)
<code>:disabled</code>	<code>input:disabled</code>	Deaktivierte Eingabeelemente
<code>:required</code>	<code>form *:required</code>	Alle Elemente innerhalb einer Form die Attribut „required“ gesetzt haben.
<code>:not (Selektor)</code>	<code>a:not (.beuth-intern)</code>	Alle Links, die nicht die Klasse beuth-intern haben

CSS3 Selektor-Varianten

- Es geht fast alles. Vieles (zu kompliziertes) ist nicht sinnvoll.
- Komplette Vielfalt im Web http://www.w3schools.com/cssref/css_selectors.asp

Pseudo~~element~~selektoren (Elemente, die nicht im HTML vorhanden sind)

Syntax	Beispiel	Wirkung
<code>::before</code> <code>::after</code>	<code>h1::before {content: url(bild.png) }</code>	Setzt vor jede Überschrift H1 das bild.png davor
<code>::first-letter</code>	<code>p::first-letter</code>	Der erste Buchstabe eines jeden Paragraphen

CSS3 Kaskade

- Treffen auf ein HTML-Element mehrere Selektoren zu, dann werden alle ausgeführt.
- Bestimmte, kaskadierende Reihenfolge, damit es eindeutig ist
 1. Selektor-Spezifität (wie genau passt es)
gibt es mehrere, dann:
 2. Nähe zum Element (wo ist es definiert)

1. Selektor-Spezifität

```
main h1[class*=„long“]:hover { color: red }
```

ist spezifischer als ↓

```
main h1 { color: blue }
```

ist spezifischer als ↓

```
h1 { color: green; font-size: 2em }
```

- **Achtung:** Schriftgröße 2em bleibt beim Zusammenführen auch bestehen

CSS3 Kaskade

- Treffen auf ein HTML-Element mehrere Selektoren zu, dann werden alle ausgeführt.
- Bestimmte, kaskadierende Reihenfolge, damit es eindeutig ist
 1. Selektor-Spezifität (wie genau passt es)
gibt es mehrere, dann:
 2. Nähe zum Element (wo ist es definiert)

1. Selektor-Spezifität (Allgemein)

#id

ist spezifischer als ↓

[Attribut] (bspw. .Klasse)

ist spezifischer als ↓

Element




ist spezifischer als ↓

(geerbte Eigenschaften)

CSS3 Kaskade

- Treffen auf ein HTML-Element mehrere Selektoren zu, dann werden alle ausgeführt.
- Bestimmte, kaskadierende Reihenfolge, damit es eindeutig ist
 1. Selektor-Spezifität (wie genau passt es)
gibt es mehrere, dann:
 2. Nähe zum Element (wo ist es definiert)

2. Nähe zum Element

- a. **Inline Style:** Direkt am Element schlägt alles (wegen Spezifität!)
 ist näher als
 - b. **Internal Style:** Im <head> eingebundener Stil
 ist näher als
 - c. **External Style:** In extra Datei <link ..> xyz.css definierter Stil
 ist näher als
 - d. **Browser Style:** Vom Browser(hersteller) vordefinierter Stil (schwächster)
- (gibt es mehrere vom gleicher Nähe, dann zählt die Ladereihenfolge im Dokument.
Später überschreibt früher.)

CSS3 Kaskade - Beispiel

■ Welche Farbe hat die Überschrift des Artikels?

```
<head lang="de">
  <meta charset="UTF-8">
  <title>MME2 CSS3 Beispiele</title>
  <link href="base.css" rel="stylesheet" type="text/css">
  <style type="text/css">
    #first-head {color: brown}
    body main article h1 {color: blue}
    h1           {color: yellow}
    .art-head    {color: red}
  </style>
</head>
<body>
<main>
  <article id="article" class="first-art">
    <h1 id="first-head" class="art-head" style="color: pink">Newsmeldung...</h1>
  </article>
</main>
</body>
```

base.css

```
main h1 {color: aliceblue}
#first-head {color: purple}
```



CSS3 Kaskade - Beispiel

■ Welche Farbe hat die Überschrift des Artikels?

base.css

```
main h1 {color: aliceblue}  
#first-head {color: purple}
```

```
<head lang="de">  
  <meta charset="UTF-8">  
  <title>MME2 CSS3 Beispiele</title>  
  <link href="base.css" rel="stylesheet" type="text/css">  
  <style type="text/css">  
    #first-head {color: brown}  
    body main article h1 {color:blue}  
    h1           {color: yellow}  
    .art-head    {color: red}  
  </style>  
</head>  
<body>  
<main>  
  <article id="article" class="first-art">  
    <h1 id="first-head" class="art-head" style="color:pink">Newsmeldung..</h1>  
  </article>  
</main>  
</body>
```

Lösung (Reihenfolge der Prio): pink, brown, purple, red, blue, aliceblue, yellow, (black)

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
- Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

CSS Layouts

Nutzung von fertigen Grid-Layouts

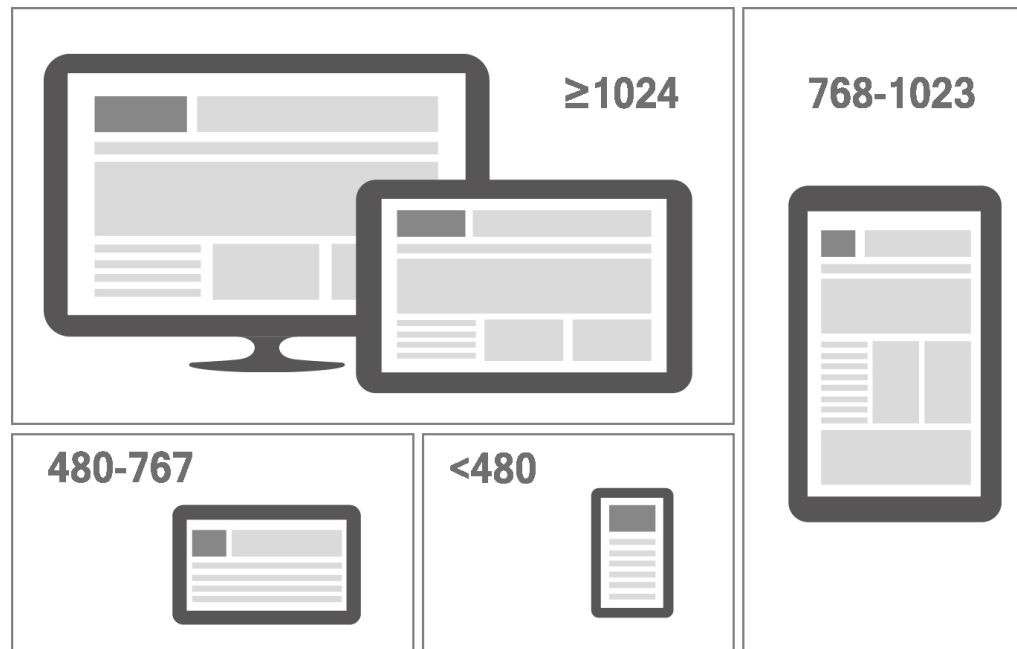
- Bootstrap
- Skeleton
- Initializr



CSS Layouts

Responsive Design

■ Warum?



CSS Layouts

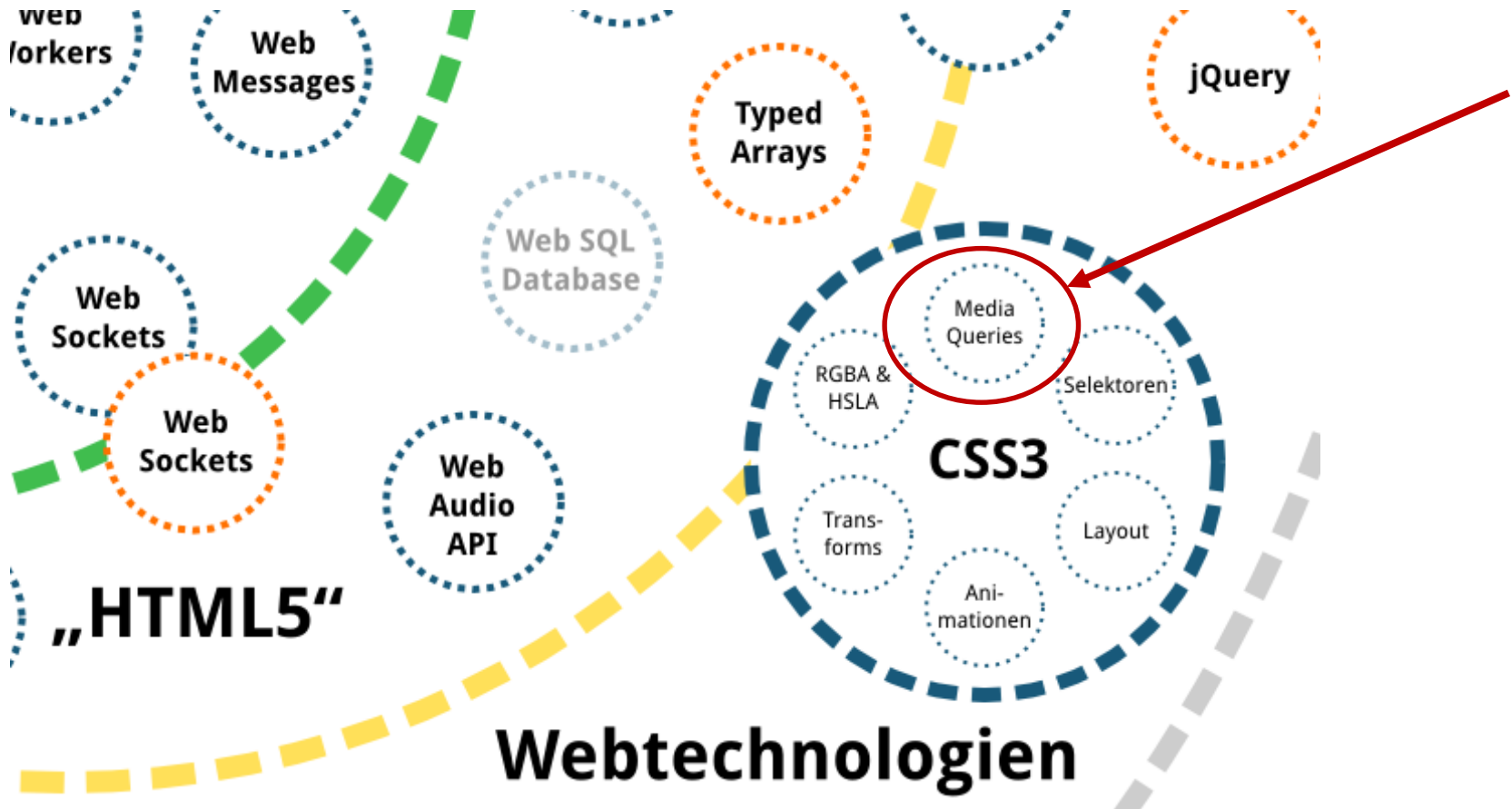
Responsive Design

- Ziel: Anzeige von Inhalten in Abhängigkeit der Ausgabe
 - z.B. Breite des Bildschirms
 - Browser, Betriebssystem
 - uvm.



- Viele tolle Beispiele:
<https://www.awwwards.com/websites/responsive-design/>

CSS Media Queries



Media Queries



- Ein Query kann auch mit einem media_type Verknüpft werden
 - `[[only | not]? <media_type> [and <expression>]*`
 - `only` ist im Normalfall implizit, verhindert aber, dass ältere Browser die Regel ausführen (mit `only` also nur für CSS3 Browser)
- media_type: `all | aural | braille | embossed | handheld | print | projection | screen | tty | tv | speech **`

```
@media only screen and (min-width: 700px)
    and (orientation: landscape) {

background-color: #222;

}

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

Übliche Breiten und Höhen von Geräten:
siehe Web*

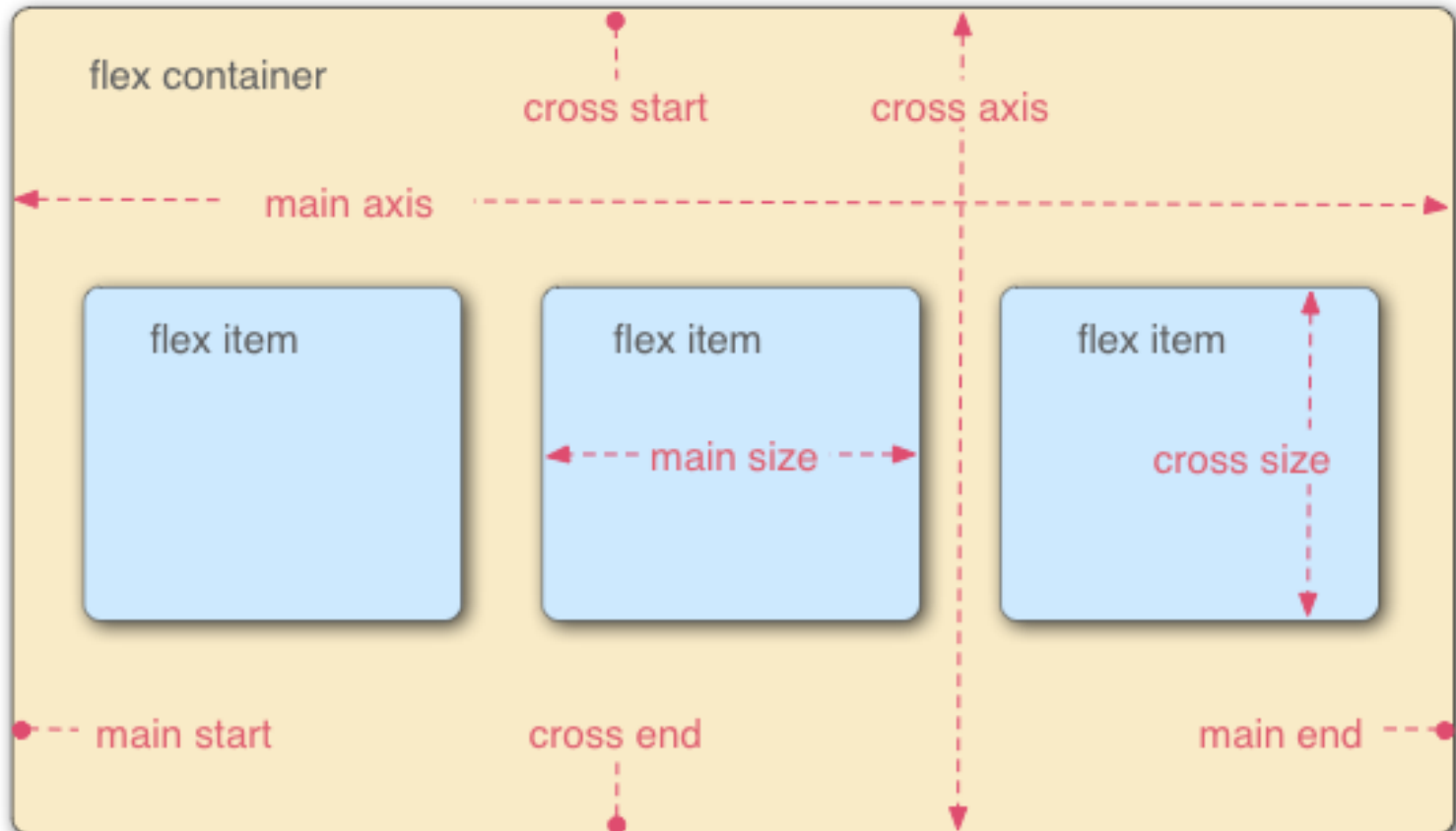
Generell nicht auf pixelgenaue Layouts fixieren,
sondern eher “breakpoints” definieren, ab
denen Sie auf andere Layouts wechseln

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
- Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

CSS Layout: Flexbox Layout

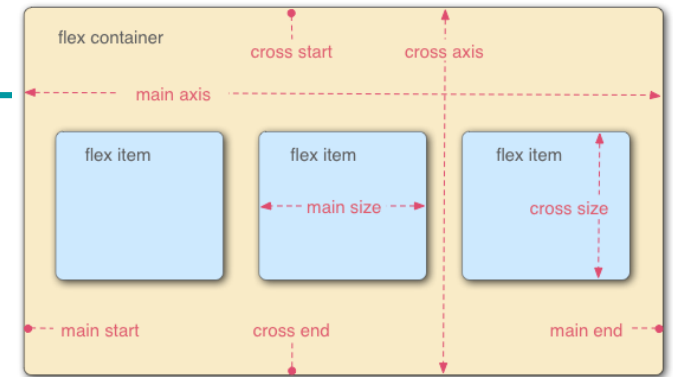
- Flexiblere Alternative zu `float:right` und `display:block` und `clear:both`
- `display:flex` für den sog. flex container



CSS Layout: Flexbox Layout

Container mit display:flex

- **flex-direction**: für die Hauptachse (row)
- **justify-content**: Anordnung auf der Achse (flex-start, center, space-between, ..)
- **align-items**: Wie Elemente angeordnet werden auf der zweiten Achse (flex-start, center, stretch, ..). Relevant bei flex-wrap:nowrap oder wenn nur eine Zeile an Flex Items.
- **align-content**: Wenn Flex Items mehrzeilig werden, dann wie die Zeilen aufgeteilt werden (flex-start, center, space-between, ..)



Flex Item innerhalb des Containers

- **flex-basis** für den Anteil am Container-Platz für dieses Element
- **flex-grow**, **flex-shrink** für das Verhältnis zu den anderen Elemente, wenn mehr Platz da ist (grow) oder zu wenig (shrink)
- **flex**: als Kurzschreibweise *grow*, *shrink* *basis* (bspw. **flex: 1 1 20%**)

CSS Layout: Flexbox

■ <http://flexboxin5.com>

tour 1 / 53

Step right up!

Click the right arrow at the bottom of this box to start an interactive tour of flexbox.

control panel

flex-direction: row

flex-wrap: wrap

justify-content: flex-start

align-items: flex-start

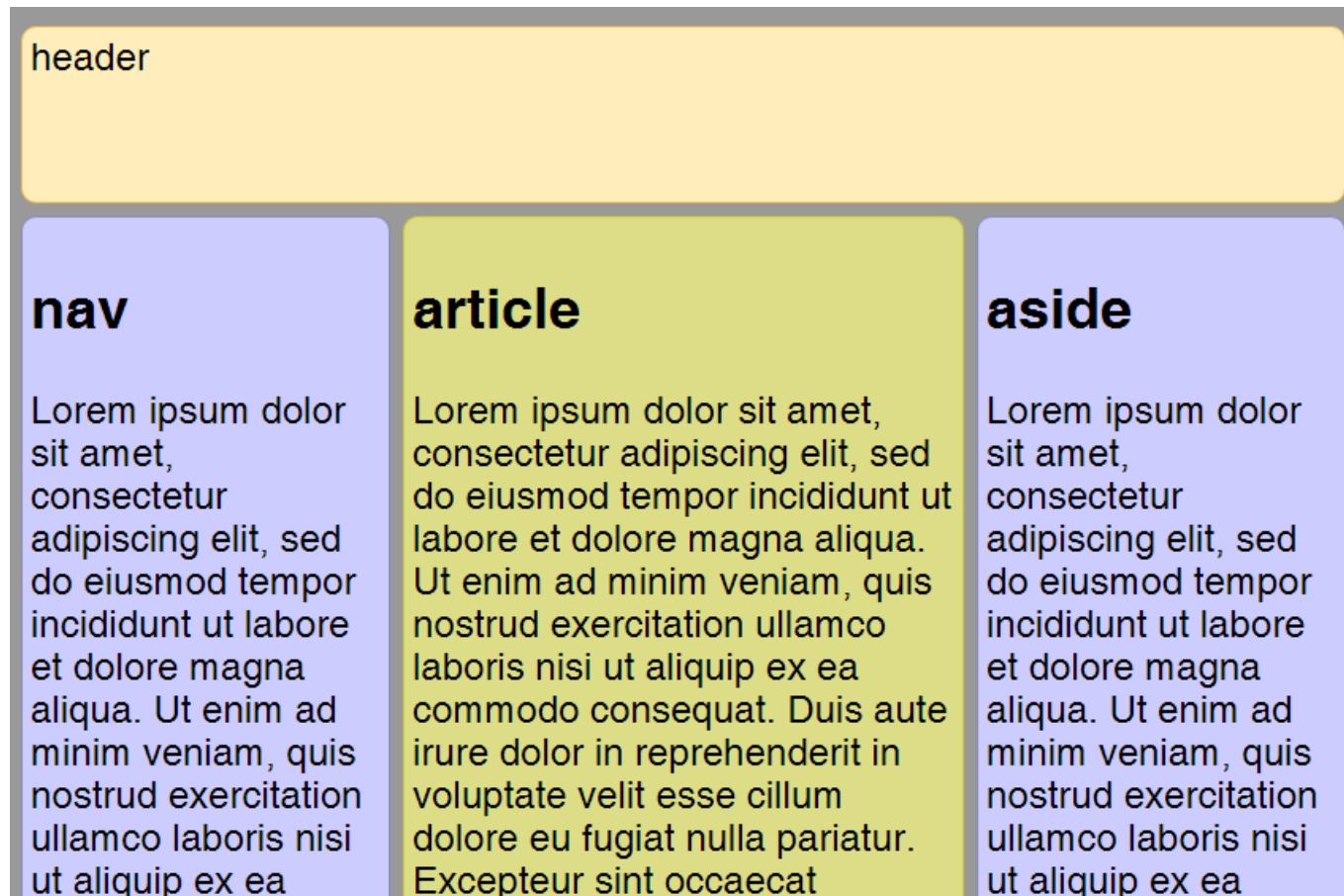
align-content: flex-end

code samples

```
.flex-container {  
  display: flex;  
  flex-direction: row;  
  flex-wrap: wrap;  
  justify-content: flex-start;  
  align-content: flex-end;  
  align-items: flex-start;  
}
```


CSS Layout: Flexbox

- Beispielcode mit fester Basis-Größe und flexiblem Wachstum.



CSS Layout: Flexbox

- **order**: Endlich eine Lösung um unabhängig von HTML die Reihenfolge anzugeben
- **flex-grow** u. **flex-basis**:
Wachstum der Elemente untereinander

```
<body>
  <header>header</header>
  <div id='main'>
    <article><h2>article</h2></article>
    <nav><h2>nav</h2></nav>
    <aside><h2>aside</h2></aside>
  </div>
  <footer>footer</footer>
</body>
```

```
#main {
  min-height: 800px;
  display: flex;
  flex-direction: row;
}

#main > article {
  flex: 2 0 300px;
  order: 2;
}

#main > nav {
  flex: 1 0 200px;
  order: 1;
}

#main > aside {
  flex: 1 0 200px;
  order: 3;
}
```

CSS Layout: Flexbox

Was müsste man ändern, damit alle drei Boxen (nav, article, aside)

- untereinander erscheinen und
- alle die volle Breite einnehmen

Besprechen Sie sich
in kl. Gruppen (2min)



```
<body>
  <header>header</header>
  <div id='main'>
    <article><h2>article</h2></article>
    <nav><h2>nav</h2></nav>
    <aside><h2>aside</h2></aside>
  </div>
  <footer>footer</footer>
</body>
```

```
#main {
  min-height: 800px;
  display: flex;
  flex-direction: row;
}

#main > article {
  flex: 2 0 300px;
  order: 2;
}

#main > nav {
  flex: 1 0 200px;
  order: 1;
}

#main > aside {
  flex: 1 0 200px;
  order: 3;
}
```

Css Layout: Flexbox

Was müsste man ändern, damit alle drei Boxen (nav, article, aside)

- untereinander erscheinen und
- alle die volle Breite einnehmen

AW: **flex-direction: column**



```
#main {  
  min-height: 800px;  
  display: flex;  
  flex-direction: column;  
}  
  
#main > article {  
  flex: 2 0 300px;  
  order: 2;  
}
```

```
<body>  
  <header>header</header>  
  <div id='main'>  
    <article><h2>article</h2></article>  
    <nav><h2>nav</h2></nav>  
    <aside><h2>aside</h2></aside>  
  </div>  
  <footer>footer</footer>  
</body>
```

header

nav

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

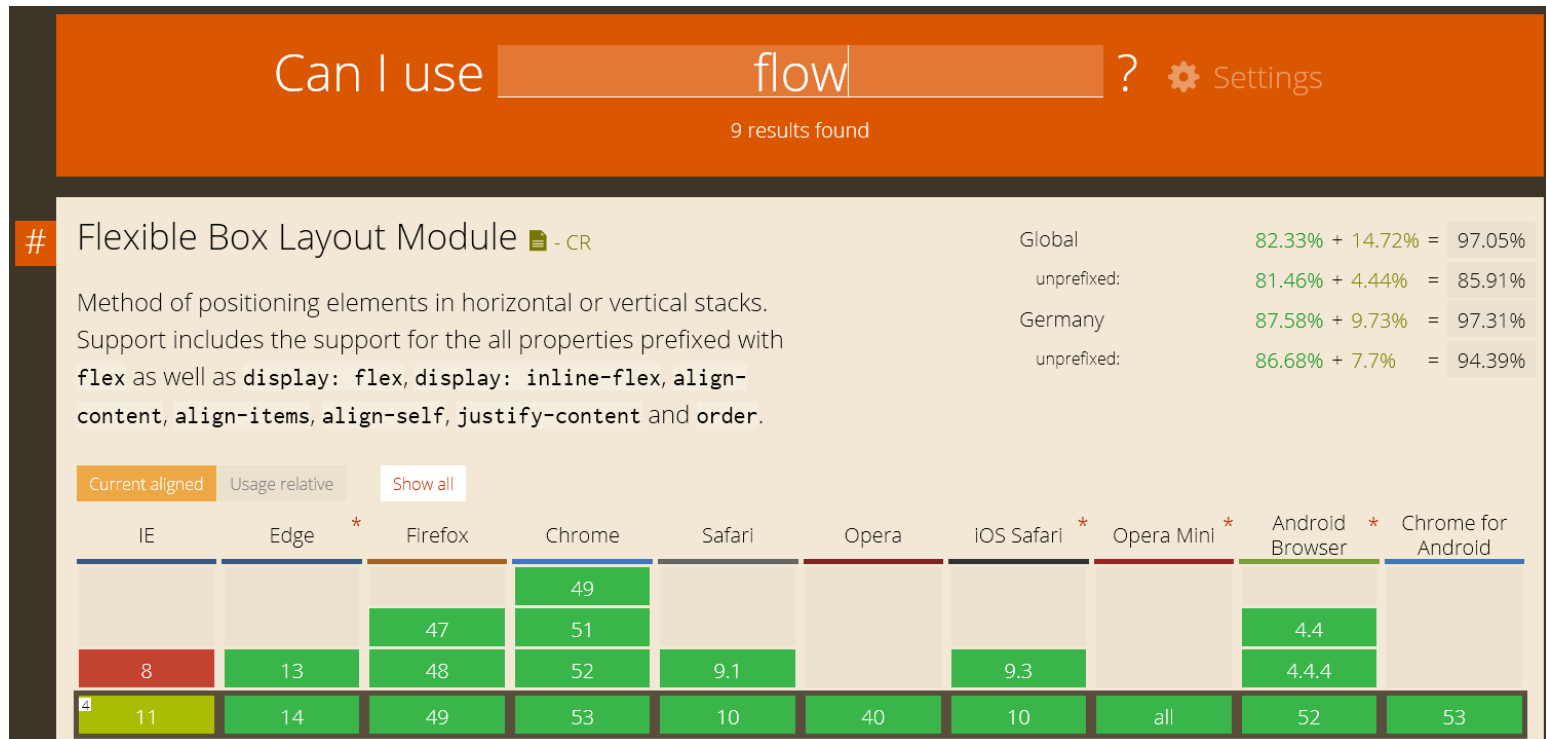
article

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

CSS Layout: caniuse.com

<http://caniuse.com>

- Für HTML, CSS, oder JavaScript schnell prüfen wie verbreitet das von den Browsern unterstützt wird.



Agenda

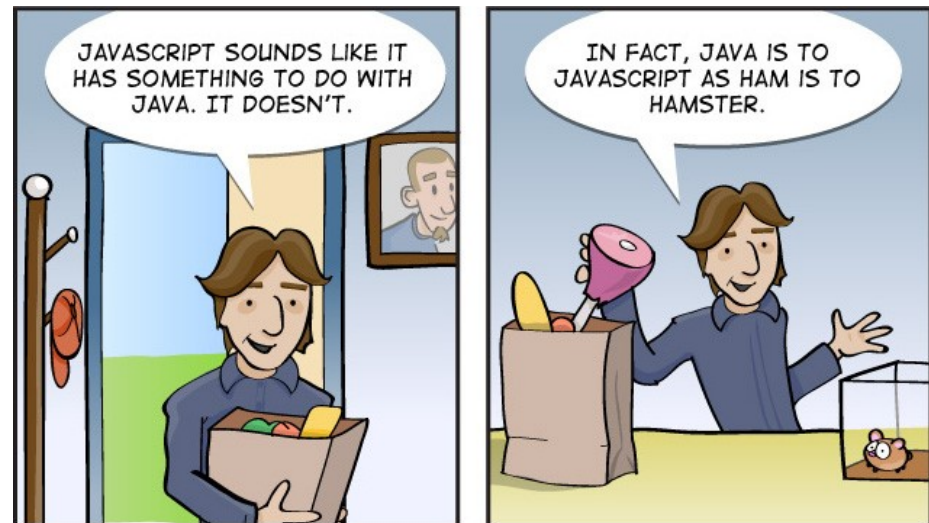
- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- **JavaScript (ECMAScript v5)**
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

JavaScript bzw. ECMAScript 5

- **funktional:** mit Funktionen als First-Class-Citizens, Closures, ...
- **objektorientiert:** aber nicht mittels Klassen, sondern mittels Prototypen.
- **prinzipiell unstrukturiert:** JavaScript gibt keine Strukturen vor; diese müssen mittels Patterns und Disziplin vom Entwickler geschaffen werden.
- **kompromisslos dynamisch:** Objekte können zur Laufzeit um Methoden und Attribute erweitert werden, Quellcode kann zur Laufzeit hinzugefügt werden, ...

Insbesondere:

- Keine statische Typisierung
- Keine Sichtbarkeiten
- Kein Multi-Threading



JavaScript Typisierung

Dynamische und schwache Typisierung ist trickreich

```
var myID = undefined;
if (!myID) {
    myID = 0;
}
// ..later
if (!myID) {
    throw 'myID has not been defined!';
}
```

False Werte in JS

0	Number
NaN	Number
' '	String
false	Boolean
null	Object
undefined	Undefined

Besser:

```
if (myID == undefined) { .. }
```


JavaScript Typisierung

Dynamische und schwache Typisierung ist trickreich

```
( '' == '' );  
( 0 == '' );  
( 0 == '0' );  
( false == '0' );  
( null == false );  
( false == undefined );  
( null == undefined );
```

False Werte in JS

0	Number
NaN	Number
''	String
false	Boolean
null	Object
undefined	Undefined

Aufgabe: Welche zwei Vergleiche sind false?



JavaScript Typisierung

Dynamische und schwache Typisierung ist trickreich

```
( '' == '' );           true
( 0 == '' );           true
( 0 == '0' );          true
( false == '0' );      false
( null == false );     false
( false == undefined ); true
( null == undefined ); true

( '\t\r\n' == 0 );     true
```

False Werte in JS

0	Number
NaN	Number
''	String
false	Boolean
null	Object
undefined	Undefined

Daher noch besser: Typensicher vergleichen

```
if (myID === undefined) { .. }
```

Vergleiche in JS mit !== und === durchführen

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

JavaScript Sichtbarkeiten

- In JavaScript gibt es kein `private` oder `public`
- Doch es gibt den **Ausführungscontext** und **Funktionale Programmierung**

```
var count = (function() {  
    var privateCounter = 0;  
    var changeBy = function (val) {  
        privateCounter += val;  
        return privateCounter;  
    };  
    return changeBy;  
})();
```

Ausführungskontext,
in dem `changeBy` definiert wurde
(Zugriff auf `privateCounter`)

Funktion als Rückgabeobjekt

```
console.log(count(1));  
console.log(count(1));  
console.log(count(-1));
```

JavaScript Sichtbarkeiten

- In JavaScript gibt es kein `private` oder `public`
- Doch es gibt den **Ausführungscontext** und **Funktionale Programmierung**

```
var count = (function() {  
  var privateCounter = 0;  
  var changeBy = function (val) {  
    privateCounter += val;  
    return privateCounter;  
  };  
  return changeBy;  
})();
```

Ausführungskontext,
in dem `changeBy` definiert wurde
(Zugriff auf `privateCounter`)

Funktion als Rückgabeobjekt

```
console.log(count(1)); // 1  
console.log(count(1)); // 2  
console.log(count(-1)); // 1
```

**`changeBy` (und damit `count`) ist eine sog.
Closure-Funktion**

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
- Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

JavaScript: kein Multi-Threading

- JavaScript wird in nur einem Thread ausgeführt*
- Praktisch, weil so kein `synchronized{...}` nötig ist
- Unpraktisch, wenn langsame Aufrufe und Berechnungen das ganze Benutzerinterface lahm legen.

```
<body>
Diese Seite ist offen seit
<time id="time">--</time> Millisekunden.
<script>
  var startTime = new Date();
  var timeElement =
    document.getElementById("time");
  while(true)
  {
    timeElement.innerHTML =
      new Date()-startTime;
  }
</script>
</body>
```



```
<body>
Diese Seite ist offen seit
<time id="time">--</time> Millisekunden.
<script>
  var startTime = new Date();
  var timeElement =
    document.getElementById("time");
  var updateTime = function() {
    timeElement.innerHTML =
      new Date()-startTime;
  };
  window.setInterval(updateTime,1000);
</script>
</body>
```



Lösung:
Funktionen als callback Parameter übergeben

HTML/CSS/JavaScript - Ladereihenfolge

1. `<head>` Elemente der Reihe nach inkl. externer Ressourcen
2. `<body>` Elemente der Reihe nach (inklusive aller synchronen Skripts!)
Dann wird das Event `DOMContentLoaded` ausgelöst
3. `<body>` Element-Inhalte die per `src` und `url` referenziert sind (Bilder usw.)
Dann wird das Event `load (window.load)` ausgelöst


```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>DOMLoading</title>
  <script src="DOMdefineMME2.js"></script>
  <script>
    if (MME2 !== undefined) {
      console.log("MME2 found in HEAD with: "+MME2);
    }
    if (document.body === null) {
      console.log("No access to document.body yet");
    }
    document.addEventListener("DOMContentLoaded", function(event) {
      console.log("DOM fully loaded and parsed (body available)");
    });
    window.addEventListener("load", function(event) {
      console.log("Window fully loaded and parsed (all external resources available like images)");
    });
  </script>
</head>
<body>
  Look at console (F12).
  <script>
    // this is directly executed
    if (document.body !== null) {
      console.log("Access at end of body to document.body element");
    }
  </script>
</body>
</html>
```

DOMdefineMME2.js

```
var MME2 = "MME2 Text";
console.log("External script done");
```

Look at console (F12).



```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>DOMLoading</title>
  <script src="DOMdefineMME2.js"></script>
  <script>
    if (MME2 !== undefined) {
      console.log("MME2 found in HEAD with: "+MME2);
    }
    if (document.body === null) {
      console.log("No access to document.body yet");
    }
    document.addEventListener("DOMContentLoaded", function(event) {
      console.log("DOM fully loaded and parsed (body available)");
    });
    window.addEventListener("load", function(event) {
      console.log("Window fully loaded and parsed (all external resources available like images)");
    });
  </script>
</head>
<body>
  Look at console (F12).
  <script>
    // this is directly executed
    if (document.body !== null) {
      console.log("Access at end of body to document.body element");
    }
  </script>
</body>
</html>
```

DOMdefineMME2.js

```
var MME2 = "MME2 Text";
console.log("External script done");
```

External script done
MME2 found in HEAD with: MME2 Text
No access to document.body yet
Access at end of body to document.body element
DOM fully loaded and parsed (body available)
Window fully loaded and parsed (all external resources available like images)

DOMdefineMME2... :7:1
DOMLoading.h... :9:13
DOMLoading.... :12:13
DOMLoading.h... :27:9
DOMLoading.... :15:13
DOMLoading.... :18:13

JavaScript und DOM-Anbindung

- Browser bieten JavaScript Zugriff auf das Document Object Model (DOM) = Events und objektorientierte Struktur des HTML-Dokumentes.

index.html

```
<head>
  ...
  <script type="text/javascript" src="js/app.js"></script>
</head>
<body>
  ...
  <video id="video1">
    <source src="http://.../small.mp4" type="video/mp4">
  </video>
  <button id="play1" onclick='play("video1", "play1") '>Play</button>
  ...
</body>
```



app.js

```
function play(videoId, playId) {
  var video = document.getElementById(videoId);
  video.play(); }
```



JavaScript und DOM-Anbindung

- Browser bieten JavaScript Zugriff auf das Document Object Model (DOM) = Events und objektorientierte Struktur des HTML-Dokumentes.

Kein guter Stil:

- ✗ JavaScript direkt an die HTML-Elemente schreiben
- ✗ Globale Funktionen in JavaScript-Dateien definieren

JavaScript und DOM-Anbindung

index.html

```
<head> ...  
  <script type="text/javascript" src="js/app.js"></script>  
</head>  
<body>...  
  <video>  
    <source src="http://.../small.mp4" type="video/"  
  </video>  
  <button name="play" title="Abspielen"></button>
```

app.js

```
1: window.addEventListener("load", function(event)  
2:   var videos = document.getElementsByTagName("video");  
3:   for (var i = 0; i < videos.length; i++) {  
4:     var video = videos[i];  
5:     // ... find button objects and add listener ...  
6:     playButton.addEventListener("click", function (event) {  
7:       video.play();  
8:     }); // ...  
9:   }  
10: });
```

Achtung: Für die Übung 1 müssen Sie noch das Problem lösen, dass var video immer auf das letzte Video zeigt.

JavaScript: Closures

```
var i = 100;  
var globalCounter = function(value) {  
    return ++value;  
};  
i = globalCounter(i); // 101  
// i = i+1;
```

kein Closure

```
var getCounter = function(value) {  
    var i = value;  
    return function() {  
        return ++i;  
    }  
};  
  
var globalCounter = getCounter(100);  
globalCounter() // 101
```

mit Closure

Ein Closure ist eine Funktion, die Elemente Ihres umgebenden Definitionskontextes verwendet, (obwohl dieser bereits beendet wurde).

JavaScript: Closures

```
var getCounter = function(startValue) {  
    var i = startValue;  
    return function() {  
        return ++i;  
    }  
};  
  
var globalCounter = getCounter(100);  
globalCounter()
```

Anwendung von Closures immer dann,

- wenn eine Funktion auf bestimmte Elemente zugreifen soll, die nicht als Parameter übergeben werden
- wenn Variablen oder Hilfsfunktionen vor äußerem Zugriff „versteckt“ werden sollen

Weitere Quellen:

Closures Herleitung über Definitionskontext

<https://developer.mozilla.org/de/docs/Web/JavaScript/Closures>

Demystifying Closures

<http://www.sitepoint.com/demystifying-javascript-closures-callbacks-ifes/>

Why use Closures? (and when)

<http://howtonode.org/why-use-closure>

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML

JSDoc

- Zusammenfassung und Ausblick

JSDoc

@use JSdoc

■ Stark an JavaDoc angelehnte Kommentierungs-Konventionen

```
/**  
 * Represents a book.  
 * @constructor  
 * @param {string} title - The title of the book.  
 * @param {string} author - The author of the book.  
 */  
function Book(title, author) {  
}
```

- **@constructor**: für mit new aufzurufende Konstruktorfunktionen
- **@param {Type} name - description**: für Parameter
- **@return {Type} description**: für Rückgabe
- **@throws {Type} description**: Exceptions der Funktion

Weitere Infos siehe <http://usejsdoc.org/#block-tags>

Agenda

- Wiederholung
- Zuständigkeiten von HTML, CSS und JavaScript
- HTML5
 - Tags
 - Struktur
- CSS3
 - Sektoren und Kaskade
 - Responsive Design und Media Queries
 - Flexbox Layout
- JavaScript
 - Typen und Vergleiche
 - Ausführungskontext und funktionale Programmierung
 - Besonderheiten mit HTML
 - JSDoc
- Zusammenfassung und Ausblick

Zusammenfassung

■ HTML5-Dokumente semantisch strukturieren

- Meta-Tags
- Sprache und Zeichenkodierung

■ CSS Selektoren sparsam einsetzen (“keep it simple”)

- IDs nur, wenn nötig (Sprunganker)
- Media-Queries
- Flexbox Layouts

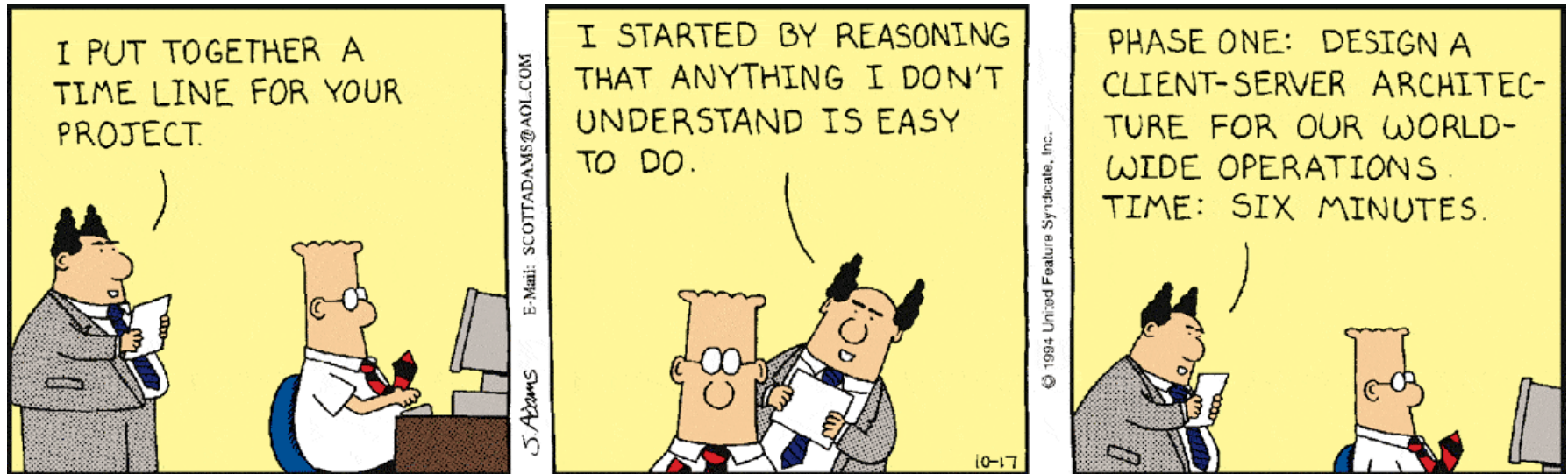
■ JavaScript nicht mit HTML vermischen

- EventListener nutzen
- Closures verwenden
- Typsichere Vergleiche (===)

■ Bonus-Frage: Was ist **Variable Hoisting** und **Function Scope** und welche Probleme können dadurch entstehen?

Nächste Woche

■ Client-Server Architekturen (Stacks)



**Vielen Dank
und
bis zum nächsten Mal**