

Web-Applikationen

Yvonne Jung
Hochschule Fulda

Hochschule Fulda
University of Applied Sciences



Formale Kriterien

- Web-Applikationen: 4 SWS, 5 CP
72h + 45h + 18h + 15h
Vorlesung + Übung
- Prüfung
 - Klausur
 - Erlaubte Hilfsmittel: ein einseitig handschriftlich beschriebenes DIN A4 Blatt
- Voraussetzung zur Vergabe der ECTS-Pkte.
 - Erfolgreiche, aktive Teilnahme am Praktikum
 - → Erfordert mind. 85% der Übungen



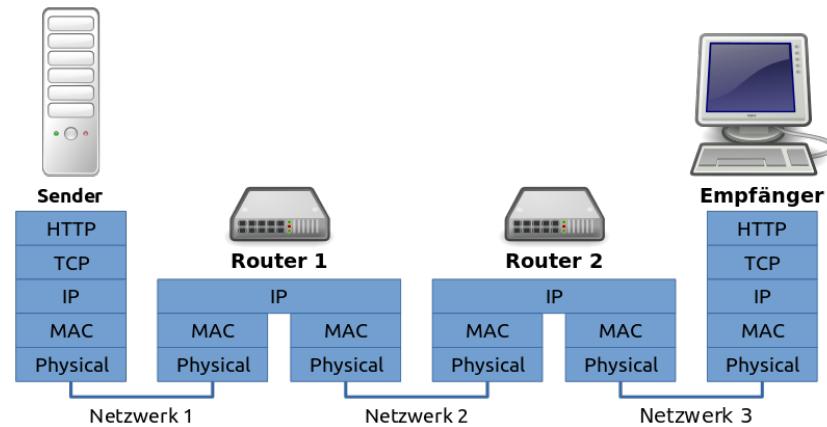
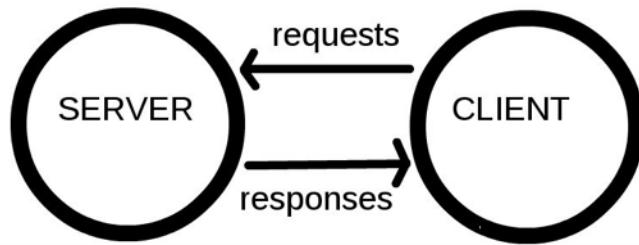
Literaturvorschläge

- <https://developer.mozilla.org/de/docs/Learn>
- <https://developers.google.com/web/fundamentals/>
- <https://www.w3schools.com/js/default.asp>
- Philip Ackermann: *JavaScript: Das umfassende Handbuch. JavaScript lernen und verstehen. Inkl. objektorientierter und funktionaler Programmierung.* 2. Auflage. Rheinwerk Computing, 2018
- Kai Günster: *Schrödinger lernt HTML5, CSS3 und JavaScript.* Galileo Press GmbH, 2013
- Robin Nixon: *Learning PHP, MySQL, JavaScript, and CSS.* Second Edition. O'Reilly Media, 2012
- Douglas Crockford: *JavaScript: The Good Parts.* O'Reilly Media, 2008



Backend

Frontend



Grundlagen: Kommunikation im Internet



Uniform Resource Locator (URL)

- Webadresse
 - Spezieller „Uniform Resource Identifier“ (URI)
 - Identifiziert Ressource im Netz über Ort und Zugriffsmethode (Kommunikationsprotokoll)
- Aufbau einer URL

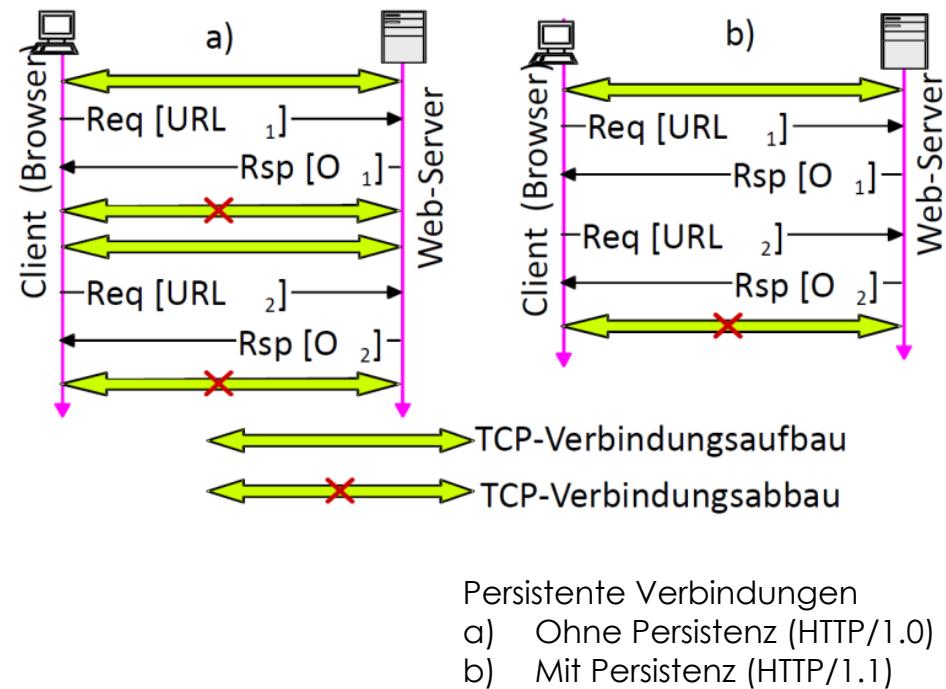
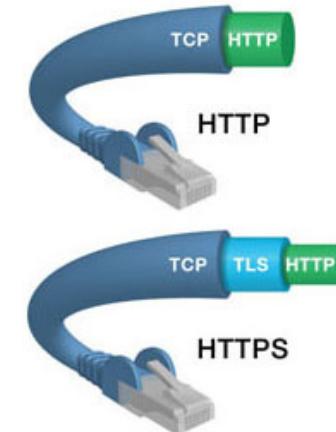
Protokolltyp://[Domäne[:Port]][/Pfad]Dateiname
[?Variable1=Wert1&...][#Fragment]

- Zugriffsmethode, z.B. http, https, ftp, file, mailto
- Host und Domain bzw. IP-Adresse
- Port für geg. Protokoll (443 für https)
- Verweis auf Dienst oder Dokument
- Sog. Query String für Parameter
- I.d.R. Anker auf HTML-Seite (#)



Hypertext Transfer Protocol (HTTP)

- Zustandslose Nachrichtenübermittlung über Transport-Protokoll TCP (standardmäßig Port 80)
 - Client (Browser / User Agent) sendet Request
 - Client initiiert TCP-Verbindung
 - Spezifiziert Anfrage mit URL
 - Server antwortet mit Response
 - Antwort enthält Web-Ressource (bzw. Fehlermeldung)
 - Server schließt TCP-Verbindung



Typen von HTTP-Requests / Methoden

- GET: verlangt Dokument, das durch angegebene URL spezifiziert wird

Request Line

GET /index.php?id=3701 HTTP/1.1

Query String

Request Header

{ Host: www.hs-fulda.de

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64)

- HEAD: ähnlich zu GET, allerdings wird nur Header zurückgeschickt und Daten (Body) weggelassen

Status Line

HTTP/1.1 200 OK

Statuscode

Response Header

{ Server: Apache

Content-Type: text/html; charset=utf-8

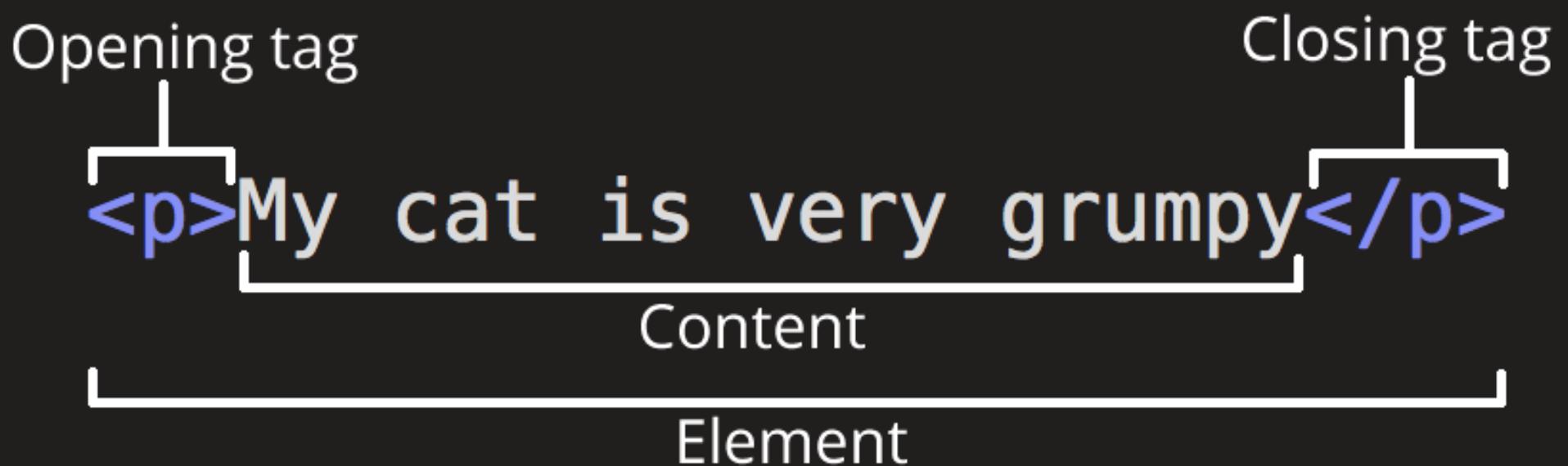
MIME Type

- POST: Übertragung von Daten im Message-Body
 - Z.B. bei Formularen
- Seit HTTP/1.1: PUT, DELETE, OPTIONS, ...
- Empfehlenswertes Tool: curl (<http://curl.haxx.se/>)
 - curl -v --head https://www.hs-fulda.de/index.php?id=3701

Angaben in Headern

- Statuscodes
 - 1xx – Informational
 - Bearbeitung der Anfrage dauert noch an
 - 2xx – Success
 - Anfrage wurde erfolgreich empfangen und akzeptiert
 - 200 OK: Anfrage erfolgreich bearbeitet u. Ergebnis wird übertragen
 - 3xx – Redirection
 - Weitere Schritte seitens Client erforderlich
 - 304 Not Modified: Inhalt hat sich nicht verändert, wird aus Cache geholt
 - 4xx – Client Error
 - Ursache des Scheiterns liegt eher bei Client (z.B. falsche Syntax)
 - 404 Not Found: Angeforderte Ressource nicht gefunden (toter Link)
 - 5xx – Server Error
 - Ursache des Scheiterns eher beim Server
- Content Negotiation
 - Client und Server können aushandeln, welche Variante einer Ressource die je beste ist (z.B. durch „Accept“ im Header)





HTML(5) zur Strukturierung



Hypertext Markup Language (HTML)

- Formale Sprache zur Beschreibung der Struktur verlinkter Webseiten (Hypertext)
 - Mittels sog. Auszeichnungen (Markup), z.B. <p>



Überschrift: "Hallo, ich heiße Bailey"

Absatz:

- Bild

Absatz:

- "Hallo, ich heiße ..."
- **Verweis:** "Kaiserslautern"
- ". Ich bin ein ..."

Absatz:

- "Wenn ich erwachsen ..."
- **Verweis:** "Wikipedia"
- "an. Ihr werdet ..."

Absatz:

- "Ich kann aber ..."
- **Verweis:** "nächste Seite"
- "sehen"



HTML Grundgerüst

- Webseite beinhaltet mindestens...
 - Angabe des Dokumententyps (DOCTYPE)
 - <html>-Element, das gesamtes Dokument umschließt (sog. Wurzelement)
- <html> hat folgende Kindelemente
 - <head>: Informationen über Webseite (wird z.B. von Suchmaschinen ausgewertet)
 - Jedes HTML-Dokument muss Titel haben
 - <title> wird auf Registerkarte des Browsers gezeigt
 - <body>: die auf Webseite sichtbaren Infos



HTML Grundgerüst

Dokumenttyp-
Deklaration

Zeichen-
Codierung,
damit 'ä', 'ö'
usw. richtig
dargestellt
werden

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  
  
<!DOCTYPE html>  
<html>  
  <head lang="de">  
    <title>Meine erste Seite</title>  
    <meta charset="UTF-8">  
  </head>  
  <body>  
    <h1>Eine Überschrift</h1>  
    <div>Ein Testabschnitt</div>  
  </body>  
</html>
```

Header

Body

<!-- Kommentar -->



index.html



HTML: Elemente und Attribute

- Weitere wichtige HTML-Elemente

<h1>Eine Überschrift</h1>

Benanntes Zeichen: Ü

<p>Das ist ein Absatz...</p>

Link

Attribut

 Ein Listeneintrag

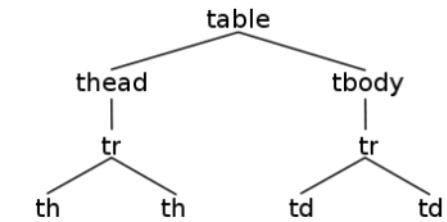
Nummerierte Liste mit

<pre>Vorformatierter Text</pre>

- Tabellen:

```
<table>
  <thead>
    <tr>
      <th>Spalte 1</th><th>Spalte 2</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>1.1</td><td>1.2</td>
    </tr>
    <tr>
      <td>1.1</td><td>1.2</td>
    </tr>
  </tbody>
</table>
```

Spalte 1 Spalte 2	
1.1	1.2
1.1	1.2



HTML: Elemente und Attribute

- div und span
 - <div> ist Blockelement (ähnlich p oder h1)
 - Container-Element zur Gruppierung von HTML-Elementen
 - ist Inline-Element (wie z.B. a)
 - Im Gegensatz zu Blockelementen kein Zeilenumbruch
- In HTML5 neu hinzugekommene Elemente
 - <section>, <article>, <address>, <header>, <footer>, <nav>, ...
- Wichtige Attribute
 - id: Eindeutiger Identifikator für ein Element
 - style: eingebettetes CSS
 - class: verweist auf Style-Klasse in Stylesheet
 - Beispiel:

```
<div id="nav" class="navigation">
    <span>Menu</span><br>
</div>
```

Zeilenumbruch
 - data-*: neu in HTML5, um weitere Daten zu speichern



Formulare in HTML

- Beispielformular

```
<form action="" method="get">  
    <input type="text" name="ort" size="30">  
    <input type="password" name="pwd">  
    <input type="checkbox" value="ok" name="cb" checked>  
    <input type="checkbox" value="nok" name="cb">  
    <select name="obst" multiple size="2">  
        <option>Äpfel</option>  
        <option selected>Birnen</option>  
    </select>  
    <input type="submit">  
    <input type="reset">  
</form>
```

Gleicher Name gruppiert
(wichtig für type="radio")

Absenden

Abbrechen

<form> Tag nie
verschachteln!

- Querystring (GET)

– <http://localhost/MyApp/index.html?ort=Fulda&pwd=4711&cb=ok>

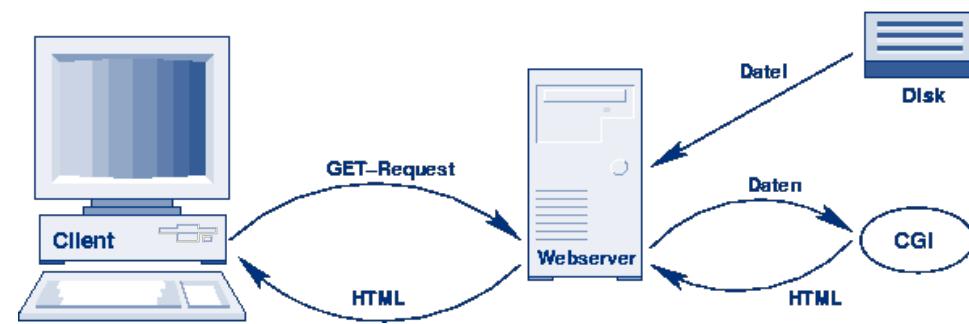
Formulardaten übertragen

```
<form action="auswerten.php" method="get">
    <input type="text" name="ort" size="30">
    <input type="password" name="pwd">
    <input type="checkbox" value="ok" name="cb" checked>
    <select name="obst">
        <option>Äpfel</option>
        <option selected>Birnen</option>
    </select>
    <input type="submit" value="Senden">
    <input type="reset">
</form>
```

```
action="auswerten.php"
method="post" enctype=
"application/x-www-form-
urlencoded"
```

```
action="mailto:foo@bar.de"
method="post"
enctype="text/plain"
```

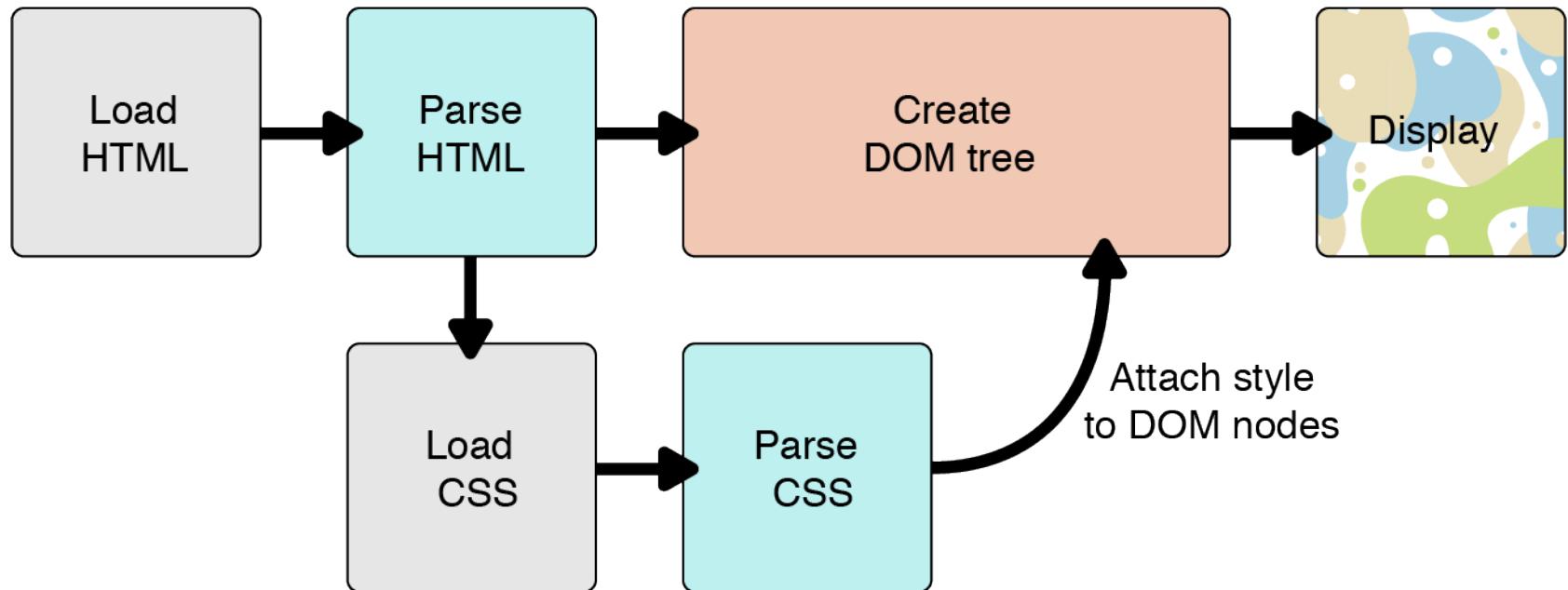
- Querystring (GET)
 - <http://localhost/auswerten.php?ort=Fulda&pwd=4711&cb=ok>



Weitere Formularelemente

- Mehrzeilige Texteingabe
 - <textarea rows="8" cols="20">...</textarea>
- <label> verbindet Formularelement u. Beschriftung
 - Attribut *for* dazu auf *id* des Elements setzen
 - Vergrößert damit klickbaren Elementbereich
- <fieldset> gruppiert Formularelemente
 - Kindelement <legend> zur Beschriftung des Formularbereichs
- <input>-Element ist vielseitig
 - Einfacher Button mit type="button"
 - Hidden Form Field mit type="hidden"
 - Slider mit type="range" (sowie min, max, step)





CSS(3) zum Stylen



Cascading Style Sheets (CSS)

- Zur Trennung von strukturiertem Inhalt von Darstellung bzw. medienspez. Layout
- Syntax:
**selector {
 property: value;
 }**
- Beispiel:

```
h1 { color: red; }  

/* Deklaration */
```

<http://www.milkaddict.com/wp-content/uploads/2009/11/css-cheat-sheet-v2.png>

Selectors	Box Model	Boxes
*	All elements	margin x margin-top margin-right margin-bottom margin-left padding x padding-top padding-right padding-bottom padding-left border x border-top x border-bottom x border-right x border-left x
div	<div>	border-color x border-top-color border-right-color border-bottom-color border-left-color
div *	All elements within <div>	border-style x border-top-style border-right-style border-bottom-style
div span	 within <div>	border-width x border-top-width border-bottom-width border-right-width border-left-width
div, span	<div> and 	
div > span	 with parent <div>	
div + span	 preceded by <div>	
.class	Elements of class "class"	
div.class	<div> of class "class"	
#itemid	Element with id "itemid"	
div#itemid	<div> with id "itemid"	
[attr]	<a> with attribute "attr"	
[attr='x']	<a> when "attr" is "x"	
[class~='x']	<a> when class is a list containing 'x'	
a[lang='en']	<a> when lang begins "en"	
Positioning		
display	clear	
position	z-index	
top	direction +	
right	unicode-bidi	
bottom	overflow	
left	clip	
:active	float	
:focus	visibility	
:link		
:visited		
:lang(var)		
:before		
:after		
Dimensions		
width	min-height	
min-width	max-height	
max-width	vertical-align	
height		
Sizes and Colours		
0	0 requires no unit	
Relative Sizes		
em	1em equal to font size of parent (same as 100%)	background-repeat background-image background-color background-position background-attachment
ex	Height of lower case "x"	
%	Percentage	
Absolute Sizes		
px	Pixels	
cm	Centimeters	
mm	Millimeters	
in	Inches	
pt	1pt = 1/72in	
pc	1pc = 12pt	
Colours		
#789abc	RGB Hex Notation	
#acf	Equates to "#aaccff"	
rgb(0,25,50)	Value of each of red, green, and blue. 0 to 255, may be swapped for percentages.	
Note	Shorthand properties are marked x Properties that inherit are marked +	
		Available free from www.AddedBytes.com

Grundlagen CSS

- Hintergrund- und Textfarbe setzen

- Bsp. hier für ganzes Dokument (inkl. Font)

```
body {  
    background-color: #000000;  
    color: #00ff00;  
    font-size: 1em;  
    font-family: Courier, monospace;  
}
```

- Größenangaben auch in Pixel oder Prozent möglich

- Farbangaben mit Farbname oder Hexadezimal

- Schema:

#000000

Rot-Anteil Grün-Anteil Blau-Anteil



HTML stylen mit CSS

- Drei Möglichkeiten
 1. Inline über style Attribut: <p style="color:blue; "> 😞
 2. Intern über <style> Element im <head>
 3. Über im Head verlinktes externes Stylesheet: ☺
`<link rel="stylesheet" type="text/css" href="style.css">`
- Beispiele für CSS-Selektoren
 - Selektoren wichtig für Möglichkeiten 2 und 3
 - .foo (alle Elemente mit class="foo")
 - #bar (das Element mit id="bar")
 - p (selektiert alle <p> Elemente)

Bsp.: p, div { font-family: Verdana, Arial, sans-serif; }



HTML stylen mit CSS

- Wenn Elemente mehrmals vorkommen können:
 - Style-Klassen verwenden
 - Bzw. alle Elemente eines Typs selektieren
 - Oder Attributselektoren nutzen
 - Z.B. alle Elemente mit Attribut "href" auswählen via [href]
 - ID-Selektor (#) nur, wenn Element einmalig vorkommt
- (Elementspezifischer) Klassenselektor

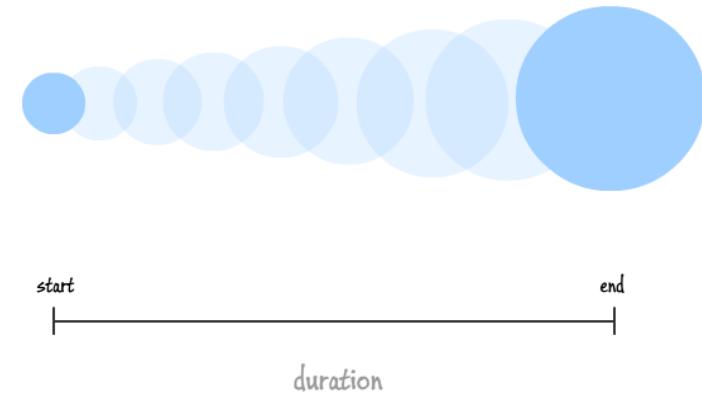
```
<h1 class="wichtig">Willkommen!</h1>
<h1 class="normal">Pferdepflege</h1>
h1.wichtig { color: red; background-color: silver; }
h1.normal { color: navy; }
```
- Styles an Nachfahren im Dokumentbaum vererbt
 - Kaskadierung: bei gleichen Stilvorlagen ist Reihenfolge im Dokument ausschlaggebend (letzte gewinnt)



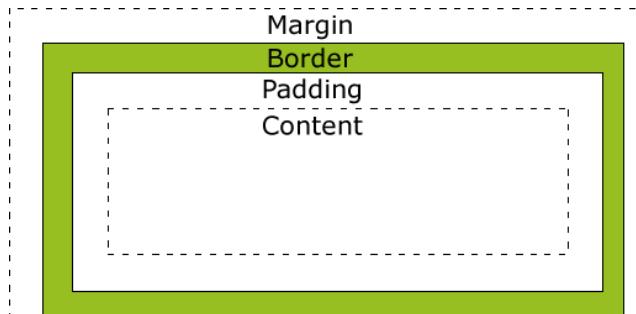
Pseudoklassen und Übergänge

- Sog. Pseudo-Klassen i.d.R. für Effekte zuständig
 - Bsp.: a:hover (Link, der unter Mauszeiger ist)
 - :first-child → Selektiert erstes Kindelement
- Pseudoelemente
 - ::before → *before* Element
 - ::after → *after* Element
 - Beispiel

p::after { content: "Das kommt nach jedem p"; }
- CSS-Transitionen
 - transition: property duration;
 - Beispiele
 - transition: all 0.5s;
 - transition: width 2s, height 4s;



CSS-Boxmodell



```
width: 250px;  
height: 150px;  
margin: 10px;  
padding: 10px;  
border: 1px solid black;
```

- Box-Modell bezeichnet Box um HTML-Elemente
 - Besteht aus Margin, Padding, Border und Inhalt
 - Gesamtbreite im Bsp.: $250\text{px}+20\text{px}+20\text{px}+2\text{px} = 292\text{px}$
- Sichtbarkeit
 - `visibility: hidden` macht Element unsichtbar, beeinflusst Layout
 - Mit `display: none` verschwindet Element
 - Über `display: block` bzw. `inline` lässt sich Blockverhalten steuern
- Textumfluss
 - Bei `float: left` wandert Element soweit wie möglich nach links, nachfolgende Elemente „fließen“ um es herum



CSS-Position

- Mögliche Werte

position: static;

→ Default, folgt normalem Seitenfluss

position: relative;

→ Relativ zur normalen static Position

position: absolute;

→ Relativ zum nächsten positionierten (nicht static) Elternelement

position: fixed;

→ Bleibt immer an der gleichen Position (relativ zu Viewport)

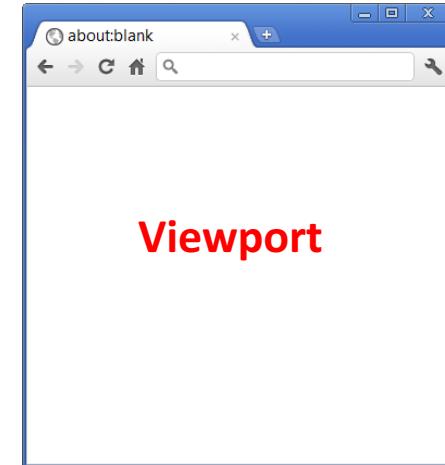
position: sticky;

→ Mischung aus relative und fixed, abhängig von Scroll-Position

- Zum Weiterlesen

<http://de.learnlayout.com/position.html>

https://www.w3schools.com/css/css_positioning.asp



Layout

- Statisches Layout 😞
 - Definition von Koordinaten
 - Größe und Position der Elemente in Pixeln absolut festgelegt
 - Bildschirrmaske für bestimmte Gesamtgröße gemacht
- Dynamisches Layout 😊
 - Definition von Regeln
 - Größe und Position der Elemente u.a. von Größe des Browsers bzw. Auflösung des Bildschirms abhängig
 - Bildschirmunabhängig → Responsive Webdesign



Responsive Web Design

Deutsche Adaption von Rüdiger Klampfl

Home

HOT POTATOES™

Desktop: 1366 x 768 px

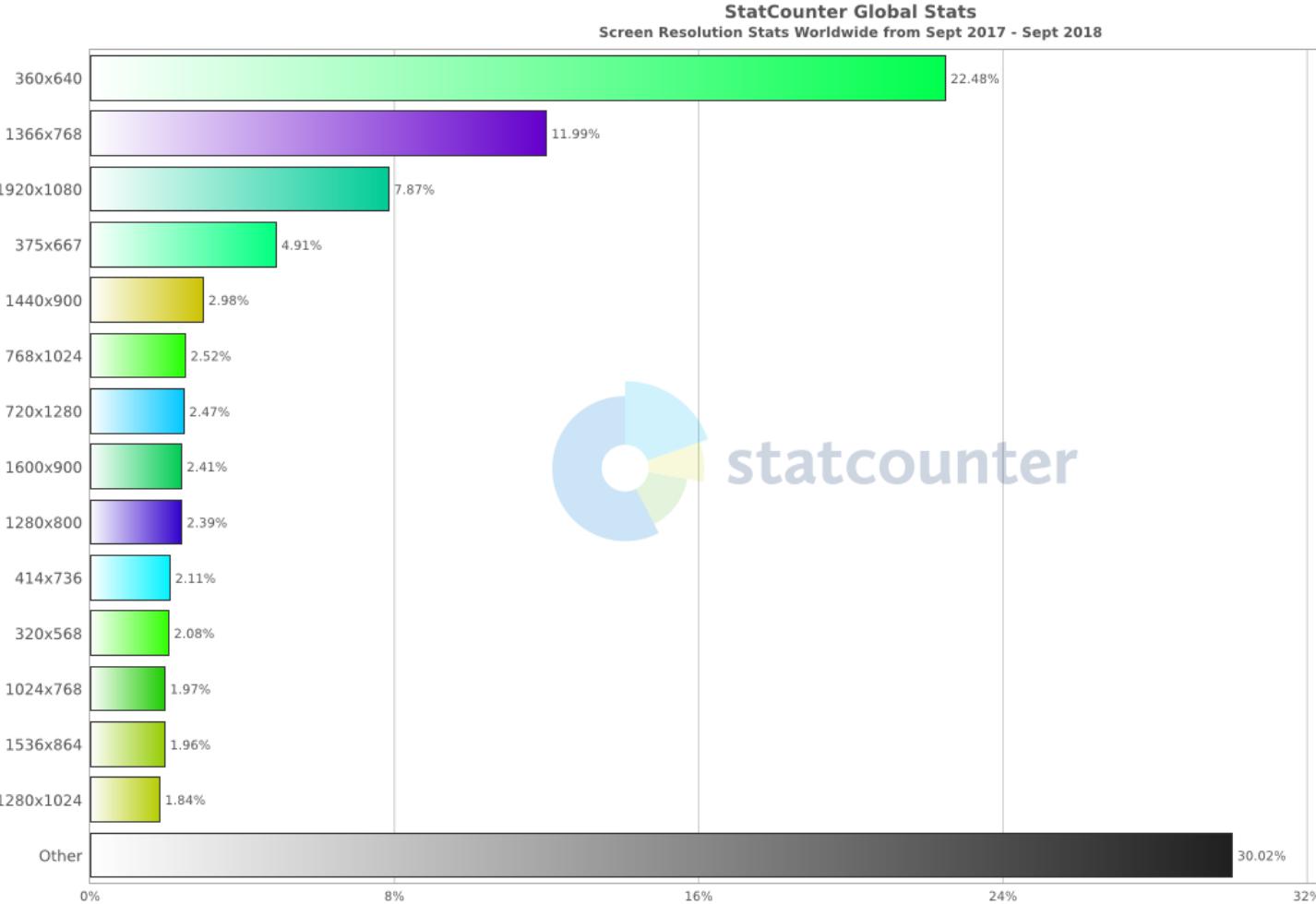
Tablet: 768 x 1024 px

Smartphone: 320 x 568 px

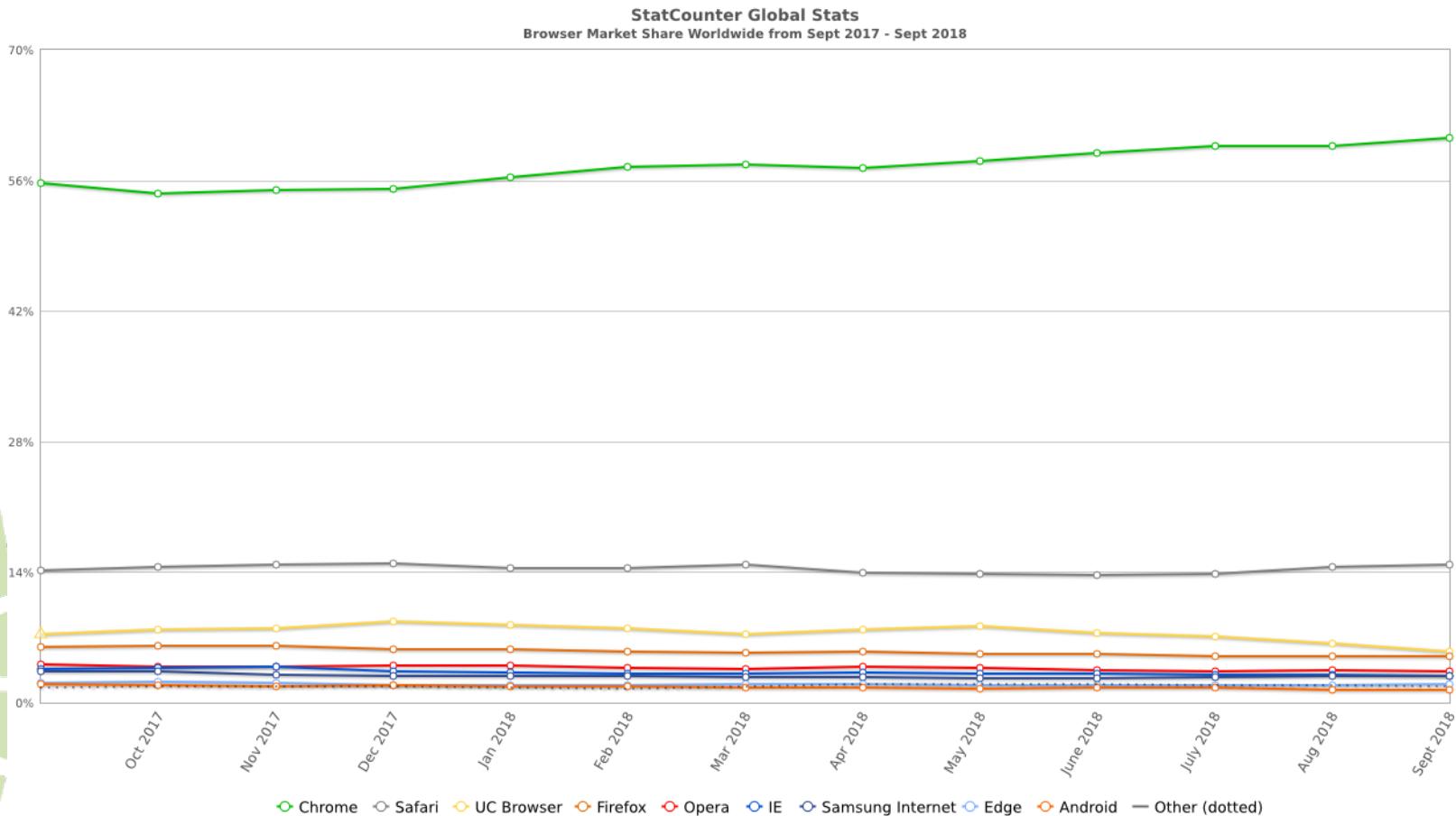
Montag, 29 Juni, 2015

- Produktion von Lernmaterialien

Gängige Bildschirmauflösungen

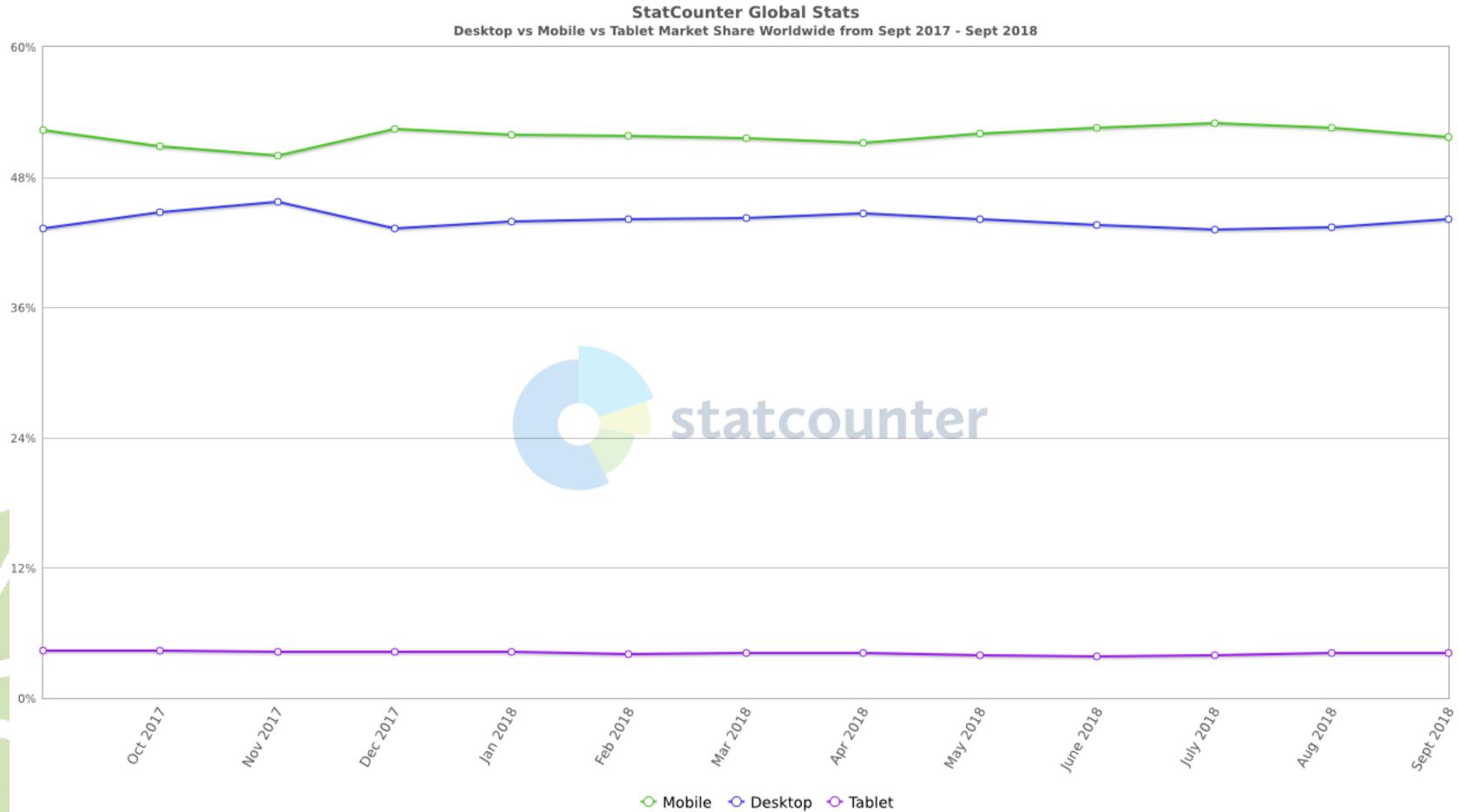


Browser-Verteilung



Note: UC Browser is Web Browser for mobile devices mostly used in China and India

Verteilung Desktop vs. Mobil



Media Queries

- Responsive Websites werden unterschiedlich auf verschiedenen Endgeräten angezeigt
 - Mobile First Ansatz
- Media Queries erlauben es Styles abhängig vom Endgerät anzuwenden

```
@media only screen and (min-width: 30em) {  
    div {  
        width: 50%;  
    }  
}  
  
@media only print {  
    div {  
        color: red;  
    }  
}
```

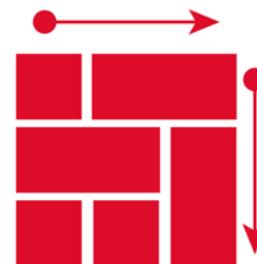
- Einbinden von CSS-Datei für versch. Medien
`<link rel="stylesheet" type="text/css" href="test.css" media="print">`

Flex-Boxen und Grids

- Flex-Box: eindimensionale Positionierung
 - Wachsen, Verkleinern, Richtung
 - <https://css-tricks.com/snippets/css/a-guide-to-flexbox>
https://www.w3schools.com/css/css3_flexbox.asp
 - ```
.flex-container {
 display: flex;
 flex-wrap: wrap;
 flex-direction: row-reverse;
}
.flexItem{
 order: 6;
 flex-grow: 5;
}
```
- Zweidimensionale Positionierung z.B. für Layouts
  - Zellen-Aufteilung / Größen bei Grids frei gestaltbar
  - <http://maddesigns.de/css-grid-layout-2764.html>  
[https://www.w3schools.com/css/css\\_grid.asp](https://www.w3schools.com/css/css_grid.asp)



Flexbox  
ONE DIMENSION



CSS Grids  
TWO DIMENSIONS



# Exkurs: Bootstrap

---

- Freies Open-Source CSS-Framework
  - Insbes. auch für mobile Geräte (Tablet usw.):  
<http://holdirbootstrap.de/>
  - Z.B. durch Klassen, mit denen über Media-Queries Inhalte pro Gerät ein- bzw. ausblendbar sind

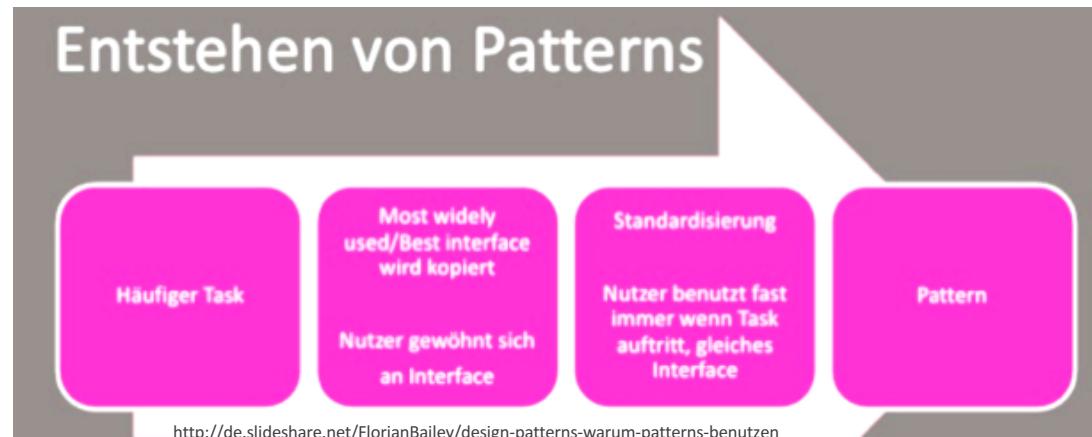
|                            | Extra-kleine Geräte<br>Smartphones (<768px) | Kleine Geräte<br>Tablets ( $\geq 768px$ ) | Mittlere Geräte<br>Desktop-PCs ( $\geq 992px$ ) | Große Geräte<br>Desktop-PCs ( $\geq 1200px$ ) |
|----------------------------|---------------------------------------------|-------------------------------------------|-------------------------------------------------|-----------------------------------------------|
| <code>.visible-xs-*</code> | Sichtbar                                    | Verborgen                                 | Verborgen                                       | Verborgen                                     |
| <code>.visible-sm-*</code> | Verborgen                                   | Sichtbar                                  | Verborgen                                       | Verborgen                                     |
| <code>.visible-md-*</code> | Verborgen                                   | Verborgen                                 | Sichtbar                                        | Verborgen                                     |
| <code>.visible-lg-*</code> | Verborgen                                   | Verborgen                                 | Verborgen                                       | Sichtbar                                      |
| <code>.hidden-xs</code>    | Verborgen                                   | Sichtbar                                  | Sichtbar                                        | Sichtbar                                      |
| <code>.hidden-sm</code>    | Sichtbar                                    | Verborgen                                 | Sichtbar                                        | Sichtbar                                      |
| <code>.hidden-md</code>    | Sichtbar                                    | Sichtbar                                  | Verborgen                                       | Sichtbar                                      |
| <code>.hidden-lg</code>    | Sichtbar                                    | Sichtbar                                  | Sichtbar                                        | Verborgen                                     |



# Interaction Design Patterns

---

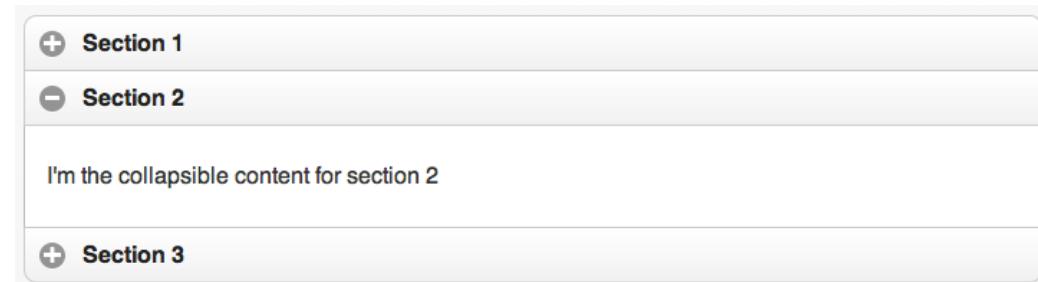
- Erfahrung im Entwurf von Benutzerschnittstellen vermitteln
  - Bestehende Designlösungen wiederverwendbar machen
  - Standardisierte Interaktionsformen (z.B. Tool Tips usw.)
- Gehen über Guidelines hinaus
  - Hinweise zur Gestaltung mit allgemeinen Designrichtlinien
  - Style Guides zur Definition des Look & Feels (z.B. bei iOS)



# Beispiele von UI Patterns

---

- Accordion
  - Ähnlich zu Tabs wird ein Panel angezeigt und Rest kollabiert, z.B. für Settings
- Hauptnavigation und Breadcrumbs
  - Benutzer müssen wissen, wo sie sind und wie sie in der Hierarchie zurück navigieren können



- [http://www.mit.edu/~jtidwell/interaction\\_patterns.html](http://www.mit.edu/~jtidwell/interaction_patterns.html)

# Barrierefreiheit

---

- Zugänglichkeit und Nutzbarkeit von Webseiten für Menschen mit Behinderungen sicherstellen
- Web Content Accessibility Guidelines (WCAG)
  - Informationen und UI müssen so präsentiert werden, dass Benutzer sie wahrnehmen können
    - Textalternativen für Nicht-Text-Inhalte bereitstellen
    - Layout nicht über Tabellen und Frames, da Braillezeilen u. Screenreader Struktur nicht richtig wiedergeben
  - UI und Navigation müssen bedienbar sein
  - Inhalte und Bedienung müssen verständlich sein
    - Z.B. einstellbare Sprache (inkl. sog. Einfacher Sprache)
  - Inhalte müssen so robust sein, dass sie zuverlässig von assistierender Techniken interpretiert werden können



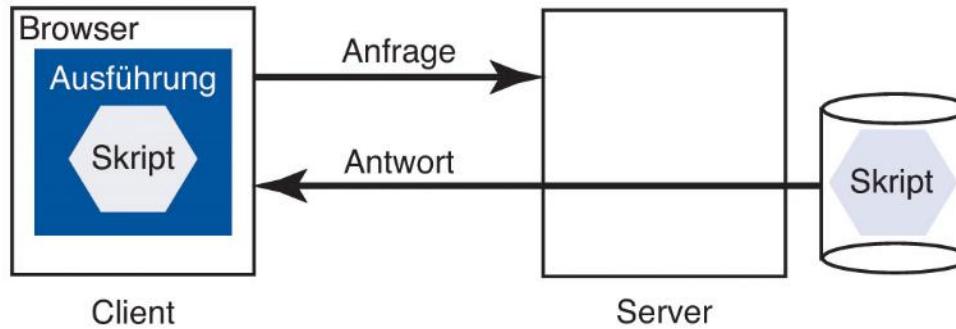


HTML(5) ± CSS(3) ± JS

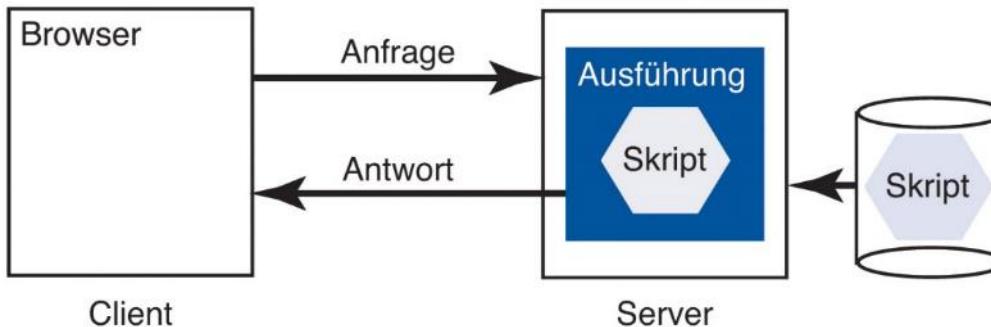


# Dynamische Webseiten

- Clientseitige Skriptausführung



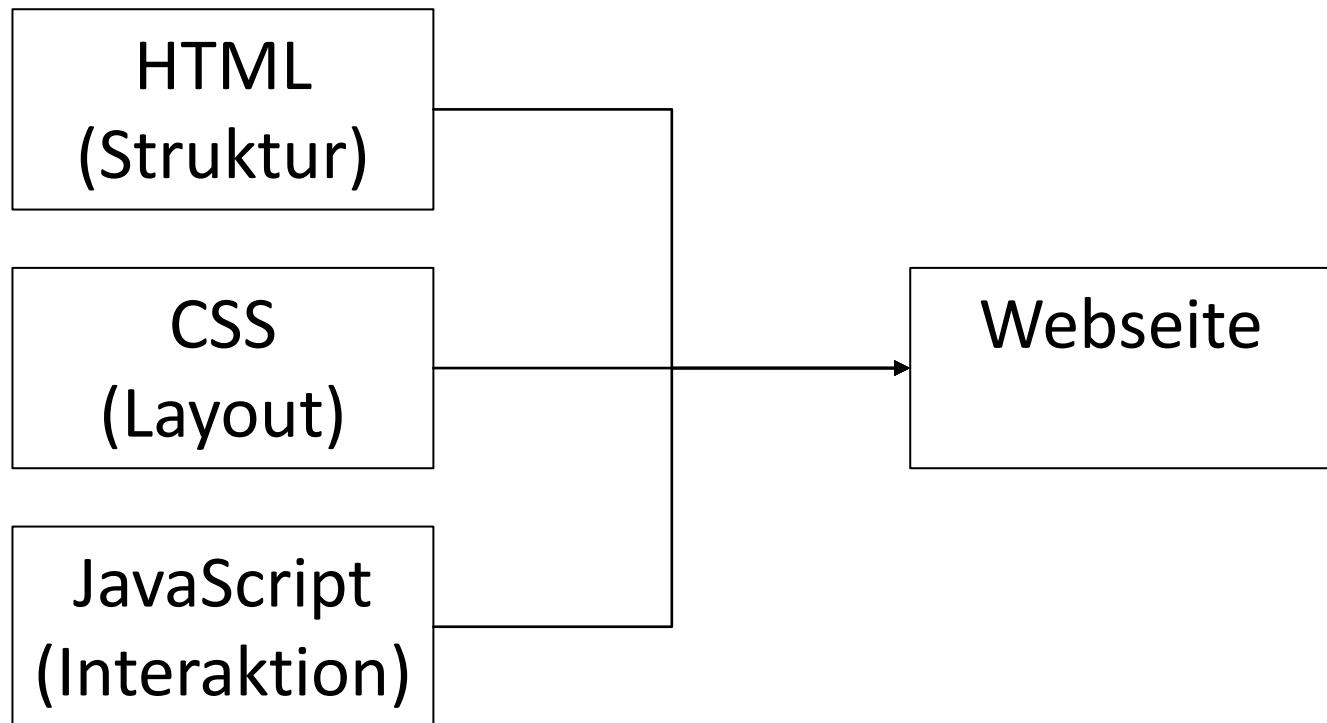
- Serverseitige Skriptausführung



# Dynamische Webseiten

---

- Trennung in Inhalt (HTML), Darstellung (CSS) und Verhalten (JavaScript)
  - HTML-, CSS-, JS-Code in separaten Dateien schreiben



# JavaScript

---

- (Programmier- bzw.) Skriptsprache
  - Zur dynamischen Manipulation von Webseiten
    - Z.B. wenn ein Ereignis auftritt wie Button geklickt
  - Dynamisch typisiert
    - Bei Variablen Deklaration wird kein Datentyp angegeben
  - Prototypenbasiert
    - Objektinstanzen erstellt auf Basis eines Prototyp-Objekts
  - Achtung: JavaScript ≠ Java !!!
- Läuft in sog. „Sandbox“
  - Verhindert, beliebig Dateien zu lesen oder zu schreiben
  - Quelltext wird durch Interpreter ausgewertet
    - Teil der JavaScript-Runtime des Browsers
  - Aktueller Standard: ECMAScript 6 (bzw. inzwischen 9)
    - Vorgänger ES5 aber noch weit verbreitet



# Unterschiede Java – JavaScript

---

## *Java*

- Programmiersprache
- Objektorientiert
- Wird auf virtueller Maschine ausgeführt
  - Laufzeitumgebung: JRE
- Code wird kompiliert
  - Zeigt dabei Syntaxfehler
- Statisch sowie stark typisiert

## *JavaScript*

- Skriptsprache
- Prototypenbasiert (unterstützt OOP)
- Läuft im Browser
  - Laufzeitumgebung: Browser, Node.js
- Code wird interpretiert
  - Fehler erst zur Laufzeit
- Dynamisch typisiert



# JavaScript in HTML einbinden

---

- Über <script> Tag (in Head und/oder Body)
  - In HTML-Dokument eingebettet mit <script>-Tag (nicht empfehlenswert)
  - Oder als Datei(en) ausgelagert:  
`<script type="text/javascript" src="main.js"></script>`
- Laden von Scripten:



Verarbeiten von HTML

Herunterladen  
von JS

Ausführen  
von JS

Verarbeiten von HTML

Zugriff auf erst hier  
geladene HTML-  
Elemente nicht möglich



# Blockieren des Renderings

---

- Laden von JS u. CSS blockiert Darstellung
  - Browser laden während Parsen synchron nach
- Lade externes JS asynchron
  - JS wird ausgeführt, sobald JS-File geladen

```
<script async src="javascript.js"></script>
```
- Führe JS erst aus, wenn Dokument geladen

```
<script defer src="javascript.js"></script>
```
- CSS nachladen (ein Ansatz)
  - Nach Laden des Dokuments **media="all"** setzen

```
<link rel="stylesheet" type="text/css"
 href="mystyle.css" media="ungueltiger_wert">
```



# JavaScript in Aktion

---

- Beispiel-Skript (zur BMI-Berechnung)

```
<script>
 "use strict";
 var weight = 60.0, height = 1.70;
 var bmi = weight / (height * height);

 document.write("Mit " + weight + " kg und " + height + " m " +
 "haben Sie einen Body Mass Index von " + bmi + ".
");
 if (bmi < 20) {
 document.write("Damit sind Sie zu d&uumlnn!
");
 }
 else if (bmi > 24) {
 document.write("Damit sind Sie evtl. zu dick!
");
 }
 else {
 document.write("Damit haben Sie Normalgewicht.
");
 }
</script>
```



Macht aus Tippfehlern Syntaxfehler, erleichtert so Debugging!



# Variablen-deklaration

---

- Früher (bis ECMAScript 5)
  - Über Schlüsselwort **var**
    - Bsp.: `var i = 0;`
  - Ohne Deklaration sind Variablen u. Funktionen global
    - Und damit Teil des sog. „window“ Objekts!
- Heute (d.h. seit ECMAScript 6)
  - Auch über Schlüsselwörter **let** und **const**
- Datentypen
  - Ermittelbar mit „typeof“
    - Bsp.: `var res = (typeof name === "string");`
  - Primitives: boolean, number, string
  - Rest: undefined, function, object (inkl. null)
- Dynamisch typisiert
  - Wurde Variable nicht initialisiert, hat sie Wert *undefined*



# Window-Objekt

---

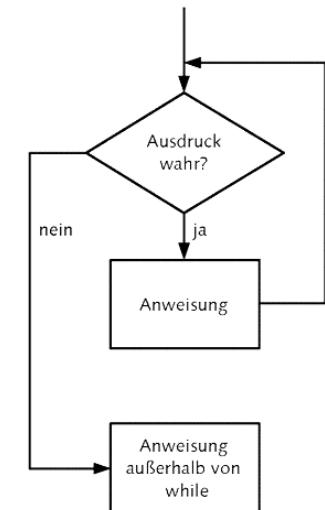
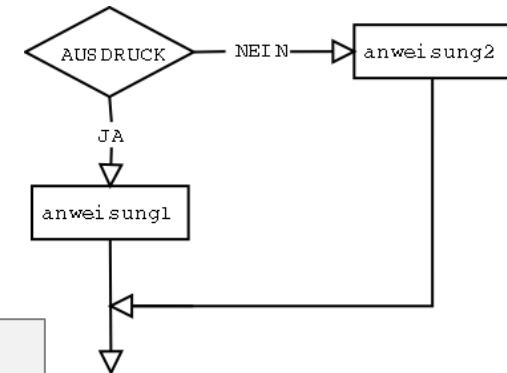
- Repräsentiert Browser-Fenster (bzw. -Tab)
  - window.location liefert Informationen über URL
    - Seite laden: window.location.href = "http://www.google.de/";
  - window.navigator bietet Informationen über Webbrowser
    - navigator.userAgent identifiziert (nur theoretisch) Browser
  - window.screen beinhaltet Informationen über Bildschirm
- Browserfenster per Script öffnen
  - let win = window.open("mypage.html", "Fensternname", "width=400,height=400,location=yes,scrollbars,status");
  - Wieder schließen: win.close();
- Weitere Dialoge: alert(), confirm(), prompt()
- Alle globalen Variablen und Funktionen automatisch Member des Window-Objekts
  - Ist im Browser sog. globales Objekt



# Kontrollkonstrukte

- Fallunterscheidungen
  - if (*Bedingung erfüllt*) { ... }
  - else { ... }
  - switch(*Wert*) {  
    case "foo": ... break;  
    case "bar": ... break;  
    default: ... break;  
}
- Schleifen
  - while (*Bedingung erfüllt*) { ... }
  - do { ... } while (*Bedingung erfüllt*)
  - for (i=0; i<n; i++) { ... }
  - Zusätzliche Kontrollmöglichkeiten
    - break beendet Schleife sofort
    - continue erzwingt nächsten Schleifendurchlauf
- Exceptions
  - try { ... } catch(e) { ... }

Ähnlich zu C: alles, was von  
false / 0 / null / undefined  
verschieden ist, gilt als wahr!



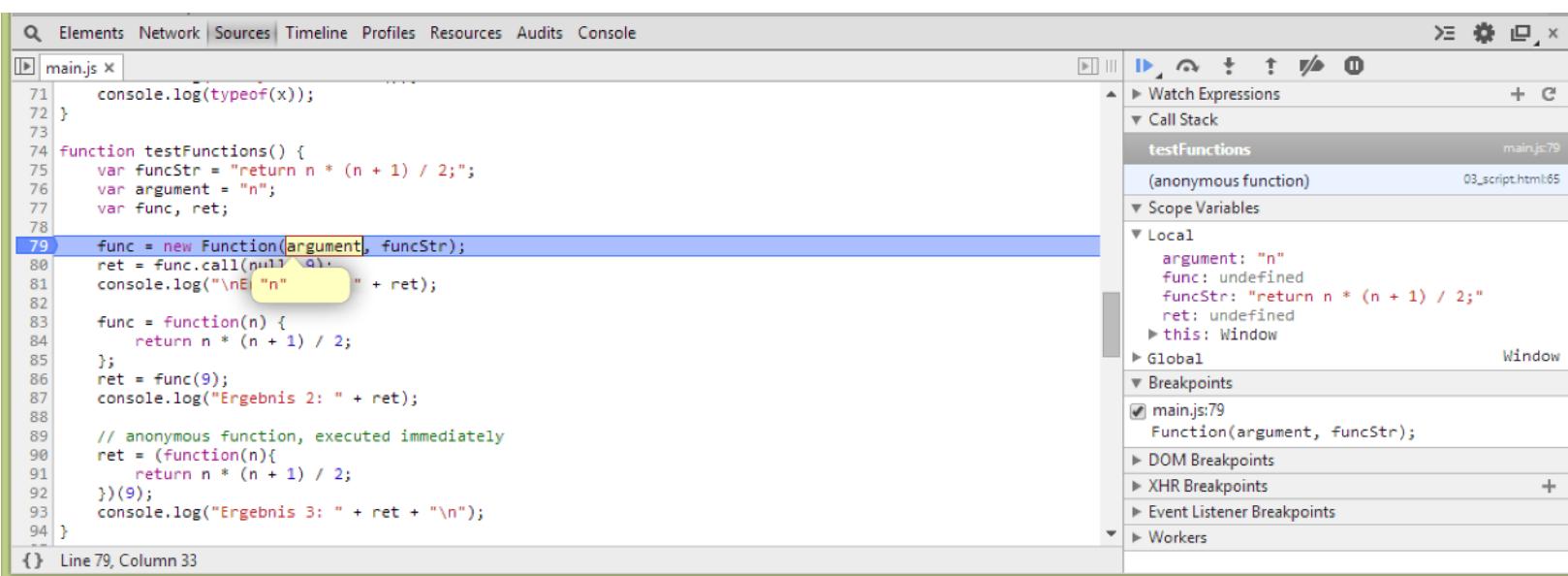
# Operatoren

---

- Arithmetische Operatoren  
+, -, \*, /, %, \*\* (Potenzierung, neu in ES7 für *Math.pow()*)
- Bitweise Operatoren  
&, |, ^, ~, <<, >>
- Zuweisungsoperatoren  
=, += usw. (auch wie in Java oder C/C++)
- Vergleichsoperatoren  
Problem: `42 == "42"` liefert *true* wegen Typumwandlung  
Daher zusätzlich zu den aus Java bzw. C/C++ bekannten:  
`==` Nur True, wenn Wert und Typ gleich sind  
`!=` True, wenn Werte oder Typ ungleich sind
- Ternärer Operator  
Syntax: *Bedingung* ? *Ausdruck1* : *Ausdruck2*

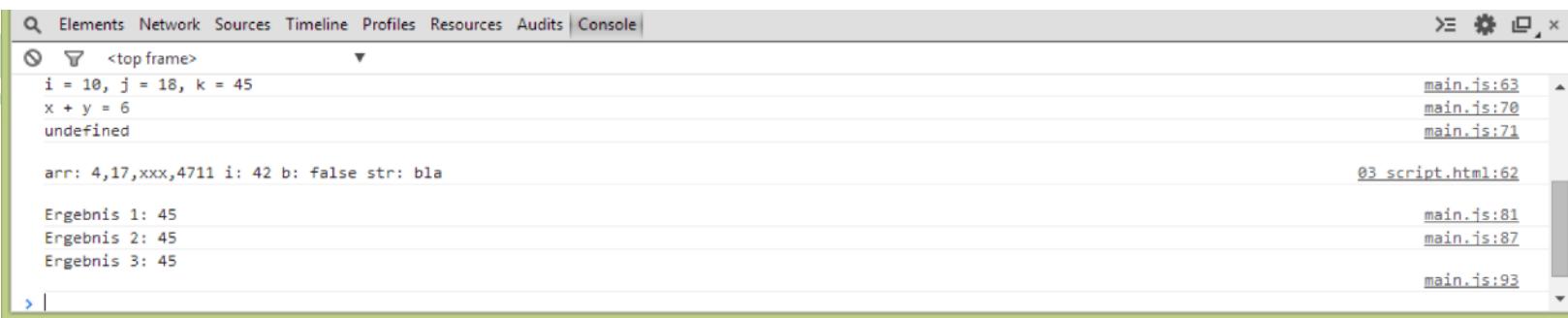


# Debugging im Browser



The screenshot shows the Chrome DevTools interface. The left pane displays the code for `main.js`. A yellow box highlights the line `ret = func.call(null, n);`. The right pane shows the `Call Stack` for the function `testFunctions`, which has an anonymous function as its return value. The `Local` variables pane shows `argument: "n"`, `func: undefined`, `funcStr: "return n * (n + 1) / 2;"`, and `ret: undefined`. The `Global` pane shows `this: Window`. The `Breakpoints` pane has a checked checkbox for `main.js:79`.

- Oft hilfreicher: `console.log()`



The screenshot shows the Chrome DevTools Console tab. It displays several log statements:  
`i = 10, j = 18, k = 45`  
`x + y = 6`  
`undefined`  
`arr: 4,17,xxx,4711 i: 42 b: false str: bla`  
`Ergebnis 1: 45`  
`Ergebnis 2: 45`  
`Ergebnis 3: 45`

On the right side, the console output is mapped to the source code with file names and line numbers:  
`main.js:63`  
`main.js:70`  
`main.js:71`  
`03_script.html:62`  
`main.js:81`  
`main.js:87`  
`main.js:93`



# Funktionen

---

- Sind in JavaScript vollwertige Objekte
  - Bsp.:

```
function addieren(a, b) { return a + b; }
```
  - Können in Variablen gespeichert werden
  - Können als Funktionsargumente übergeben werden
  - Können als Return-Werte zurückgegeben werden
  - Können sogar zur Laufzeit erzeugt werden:

```
var funcStr = "return n * (n + 1) / 2;", arg = "n";
var func = new Function(arg, funcStr);
var retVal = func.call(null, 9);
```
- Parameterübergabe
  - Objekttypen werden als Referenz übergeben (wie bei Java Call-by-Reference)
  - Nur bei primitiven Datentypen wird Wert übergeben (also Call-by-Value)



# Funktionen

---

- Verschiedene Möglichkeiten, Funktionen zu definieren

```
var fn1 = function(a) {
 return a * a;
};
function fn2(a) {
 return a * a;
}
```

Defaultwerte für Parameter  
seit ES6 möglich:

```
function fn(a, n=2) {
 return n * a;
}
```

```
// New ES6 Arrow functions
const fn3 = (a) => {
 return a * a;
};
const fn4 = a => a * a;
const fn5 = (a, n) => n * a;
const fn6 = () => console.log("no Params");
```



# Sichtbarkeitsbereich (Scope)

---

- ES5 ( $\rightarrow$  var)
  - Variablen haben sog. Function Level Scope
    - Im Gegensatz zum Block Scope bei Java oder C/C++
    - In Funktion mit `var` deklarierte Variablen nur dort sichtbar, ermöglichen hier Datenkapselung
  - Hoisting
    - Schiebt Deklarationen zum Anfang des Scopes
      - Initialisierungen werden aber nicht verschoben!
    - Variablen so nutzbar, **bevor** sie deklariert sind
      - Gilt auch für Funktionsdeklarationen
      - Sogar mehrfaches Deklarieren erlaubt (führt leicht zu Fehlern)
      - Ausweg: sog. Strict Mode ( $\rightarrow$  erstes Statement: "use strict";)
        - » Fehler werden dann in Konsole angezeigt



# Sichtbarkeitsbereich (Scope)

---

- ES6 (→ let / const)
  - Variablen haben Block Scope (wie Java oder C/C++)
    - Mit `let` deklarierte Variablen bzw. mit `const` deklarierte Konstanten sind nur im jeweiligen Block sichtbar
    - Const-Deklarationen müssen direkt initialisiert werden
      - Bsp.: `const PI = 3.14159265;`
  - Keine Hoisting-Probleme
    - Nicht so fehleranfällig
      - Muss vor Verwendung erst deklariert werden
      - Keine mehrfache Deklarierung erlaubt
      - Fehler werden in Konsole angezeigt
    - Sollten trotzdem zu Beginn des Scopes deklariert werden
      - ...und möglichst auch initialisiert



# Objekte

---

- Beispiel Objektdefinition (über sog. Objektliteral)

```
var person = {
 name: "Mustermann",
 vorname: "Max",
 alter: 50,
 hobbys: ["Joggen", "Lesen"]
};
```

- Kann auch direkt mit Methoden angelegt werden

```
var anotherPerson = {
 name: "Musterfrau",
 ...
 greet: function() { ... }
};
```

- Zunächst nur leeres Objekt anlegen: **var obj = {};**
- Test, ob Objekt von bestimmtem Typ ist, mit *instanceof*



# Objekte

---

- Besitzen Eigenschaften und Methoden
  - Bsp. für Eigenschaft (property):  
person.name
  - Beispiel für Methode (method):  
person.greet = function() { alert("Hello!"); };
  - Memberzugriff mit Punkt- u. Klammernnotation:  
person["name"] = "Moritz";  
person.greet() oder alternativ: person["greet"]()
- Über Eigenschaften und Methoden iterieren
  - Mit For-in-Schleife möglich

```
for (var key in person) {
 console.log(key + ': ' + person[key]);
}
```



# Konstruktorfunktionen

---

- Falls mehrere Instanzen erzeugt werden sollen
  - Funktion wird zu Konstruktorfunktion, indem man sie mit `new` aufruft
    - Sollte per Konvention mit Großbuchstaben beginnen
  - Kein `return` nötig, stattdessen wird neue Objektinstanz zurückgegeben
    - Darauf in Funktion mit `this` zugreifbar
- Beispiel

```
function Person(name, vorname, alter) {
 this.name = name;
 this.vorname = vorname;
 this.alter = alter;
 this.greet = function() { alert('Hallo'); };
}
```

- Aufruf: `let p = new Person('Meier', 'Horst', 85);`



# Felder (Arrays)

---

- Können Elemente beliebigen Typs enthalten
  - Beginnen bei Index 0
  - Feldgröße über (schreibbare) Eigenschaft „length“
- Array erstellen:
  - `var a1 = ["Mo", "Di", "Mi", "Do", "Fr", "Sa", "So"];`
  - `var a2 = new Array(12); // Instanz mit 12 Werten`
  - Leeres Feld: `var arr = [];`
- Mehrdimensionale Arrays
  - Arrays, die Arrays (die Arrays...) als Elemente enthalten
  - Beispiel:

```
var mat = [[1, 0], [2, 3]]; }
var elem = mat[1][1]; }
2 Zeilen
2 Spalten
```

$$\begin{pmatrix} 1 & 0 \\ 2 & 3 \end{pmatrix}$$


# Mit Feldern arbeiten

---

- Element #*i* aus Feld *arr* löschen
  - let removedElem = arr.splice(i, 1);
- Element einfügen an Stelle *i*:
  - arr.splice(i, 0, *elem*)
- Element suchen (Rückgabe Index)
  - let pos = arr.indexOf(*elem*);
- Feld sortieren (Parameter ist Vergleichsfunktion)
  - arr.sort( function(a, b) { return a - b; } );
- Letztes (bzw. erstes) Feldelement entfernen
  - pop() / shift()
- Neues Element hinten (bzw. vorne) einfügen
  - push(*elem*) / unshift(*elem*)
- In Zeichenkette (mit Separator, z.B. '-') umwandeln
  - let str = arr.join('-');



# Vordefinierte Objekte u. Funktionen

---

- Objektunabhängige Funktionen
  - parseInt(), parseFloat(): Umwandeln in Integer bzw. Float
    - Nützlich bei Formularfeldern u.ä., deren Werte Strings sind
  - eval(expr): Wertet Ausdruck aus (→ Achtung: *eval is evil!*)
- Zeitlich gebundene Anweisungen
  - setTimeout(callback, timeInMs)
  - setInterval(callback, timeInMs)  
`var id = setInterval(function() /*do stuff*/, 1000);`
- Math: Hilfsobjekt mit mathematischen Funktionen
  - Bsp.: var res = Math.floor(Math.PI \* 100);
  - Zufallszahlen in [0, 1[ mittels: Math.random()
- Date: basiert auf Zeit in Millisekunden seit 1.1.1970
  - Bsp.: var today = new Date();



# Strings

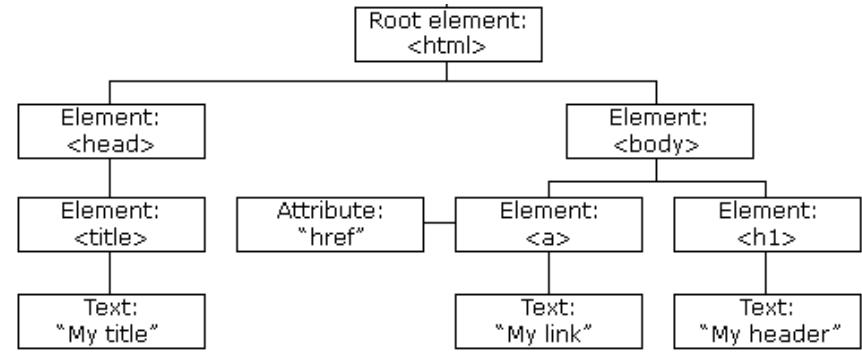
---

- Objekt zur Handhabung von Zeichenketten
  - Strings auf verschiedene Arten anlegbar
    - let s1 = 'single-quoted', s2 = "double-quoted";
    - let s3 = `backticks`;
      - Neu seit ES6, ermöglichen mehrzeilige Strings
      - Können Ausdrücke beinhalten: console.log(`1+1=\${1+1}`);
  - String-Konkatenation mit Operator +
- Ausgewählte Methoden
  - Character-Position: let pos = str.indexOf('@');
    - Zeichen an geg. Position: let c = str.charAt(pos);
  - Kleinbuchstaben: let erg = str.toLowerCase();
  - Teilstring: let sub = str.substring(startInd, endInd);
    - Stringlänge ermittelbar mit Eigenschaft *length*
  - Whitespace am Rand entfernen: let s = str.trim();





DOM



# Baum (Tree)

---

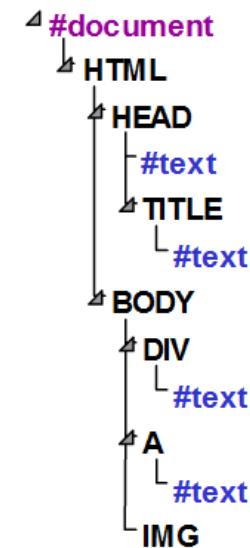
- Datenstruktur mit folgenden Eigenschaften
  - Es gibt genau eine Wurzel (root)
  - Jeder Knoten (node), bis auf Wurzel, hat genau eine Verbindung (Kante) zu einem Elternknoten (parent)
    - Element wird dann auch als Kind (child) bezeichnet
    - In Baum hat Element/Knoten also höchstens einen Vorgänger, aber evtl. mehrere Nachfolger
    - Beziehung zwischen Elementen/Knoten entspricht oft der Beziehung "ist-Teil-von" oder "ist-Art-von"
  - Jeder Knoten ohne Kinder heißt Blatt (leaf)
    - Alle anderen Knoten sind innere Knoten
- Geordneter Baum
  - Unter Kindern gibt es bestimmte Reihenfolge



# Document Object Model (DOM)

---

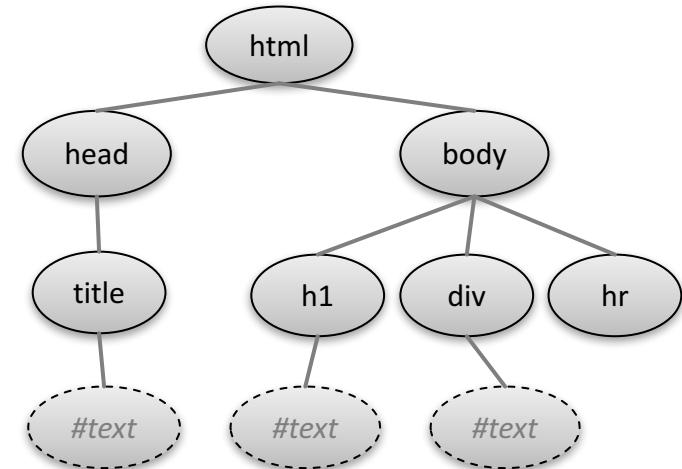
- Vom W3C standardisierte Schnittstelle
  - Erlaubt es, Inhalt, Struktur u. Layout von HTML- bzw. XML-Dokumenten zu ändern
- HTML-Dokument aufgebaut als Baum (Tree) mit Knoten (Nodes)
  - Wurzel/Root in HTML ist `<html>` Tag
  - Ermöglicht Navigation zwischen Knoten
    - `parentNode`, `childNodes[pos]`, `firstChild`, `nextSibling`, ...
- Knoten bzw. Elemente und Attribute können hinzugefügt, entfernt und verändert werden (i.d.R. über JS)
  - Auch CSS-Styles sind so modifizierbar



DOM-Struktur  
(Beispiel)

# DOM-Tree (Beispiel)

```
<html>
 <head>
 <title>Beispiel</title>
 </head>
 <body>
 <h1>Bla Blubb</h1>
 <div>
 Mehr Text im Absatz
 </div>
 <hr>
 </body>
</html>
```



Beliebig viele Kinder möglich,  
aber Reihenfolge ist wichtig

- Jeder Knoten im DOM ist ein Objekt
  - HTML-Elemente sind Element-Knoten, können Kinder haben
  - Textinhalte sind Text-Knoten, haben keine Kinder (Blattknoten)
  - Eigenschaft *nodeType* liefert Zahl, die Knotentyp repräsentiert



# DOM-Scripting: Select Nodes

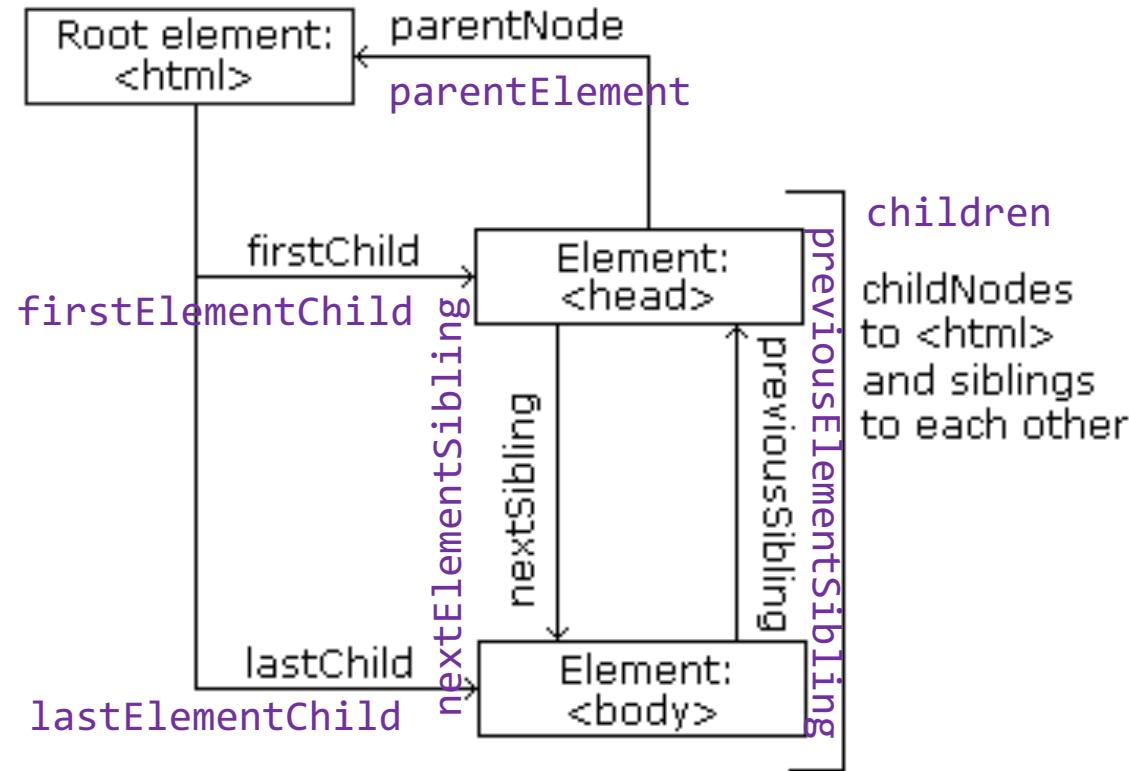
---

- Webseite repräsentiert durch *document* Objekt
  - Ausgangsobjekt (Wurzel) für Elementbaum
    - Z.B. über `document.forms` Liste aller Formulare
    - Wurzelement (`<html>`-Tag): `document.documentElement`
  - Knoten im DOM sind Element-Objekte (d.h. HTML-Elemente), Text, Comments, ...
- Auf Elemente zugreifen
  - `var elem = document.getElementById("myID");`
  - `document.getElementsByTagName("div")`
    - Liefert aktive NodeList mit HTML-Elementen
  - `document.querySelector("myCssSelector")`
    - Liefert erstes Element, dass geg. CSS-Selector entspricht



# Knotenbeziehungen im DOM

- Beispiel: Zugriff auf Elternknoten von Knoten `,elem'` geschieht über Eigenschaft `elem.parentNode`
  - Hinweis: Textknoten sind keine (HTML-) Elementknoten

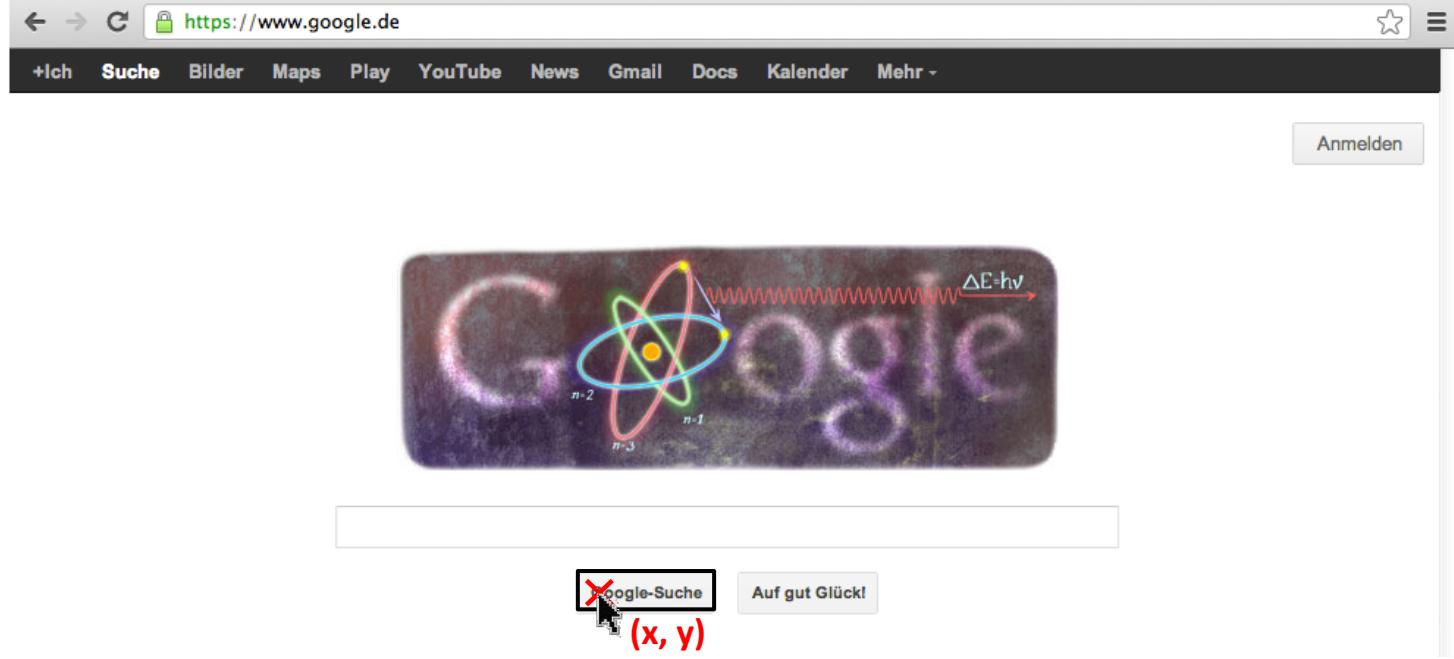


# DOM-Scripting: Manipulate Nodes

---

- Erzeugen und hinzufügen von Elementen
  - var node = document.createElement("div");
    - elem.innerHTML = "<div>Neuer HTML-String</div>";
  - var text = document.createTextNode("bla bla");
    - btn.firstChild.nodeValue = "Klick mich";
  - elem.appendChild(node)
    - Aus Performancegründen kompletten Subtree einhängen!
  - elem.removeChild(node)
- Verändern von Attributen
  - elem.setAttribute("href", "www.google.de");
    - Auch möglich: elem.href = "http://www.hs-fulda.de/"
  - var value = elem.getAttribute("href");
    - Auch möglich: var value = elem.href;





# Event-Behandlung



# Ereignisgesteuerte Programmierung

---

- Ereignisauslöser sind meist User-Interface-Komponenten
  - Bsp.: Klick bei auf Button oder Menüauswahl, aber auch asynchrones Laden (z.B. von Bildern oder via Ajax)
  - Benutzereingaben sind Ereignisse, auf die Programm geeignet reagieren muss
  - Bei interaktiven UIs ist Reihenfolge der Eingaben variierbar
- Dadurch wird von UI-Komponente Ereignis ausgelöst
  - Ereignisse (Events) sind weder Objekte noch Methoden
  - Werden meist, direkt oder indirekt, vom Benutzer ausgelöst
- Ereignis kommt in Event-Queue (Warteschlange)
  - In Ereignisbehandlungsschleife geprüft, ob neues Event anliegt
  - Falls ja, wird Event an registrierten Eventhandler weitergeleitet
  - Eventhandler sind Funktionen, mit denen man als Entwickler geeignet auf Events reagieren kann
- Event-Objekte sind ereignisspezifische Daten
  - Werden zur Ereignisbehandlung als Zusatzinformation dem Eventhandler als Parameter übergeben (z.B. Mausposition)



# Ereignisbehandlung

---

- Viele Elemente (wie z.B. <img>, <a>, <div>, <span>) erzeugen UI-Events (z.B. ,click' oder ,mouseover')
  - Ereignisbehandlung wird von Ereignisquelle (z.B. <button>) delegiert an dort registrierten Eventhandler / Event-Listener
  - Dieser wird bei Ereignis (z.B. click) benachrichtigt
- Eventhandler zu Event hinzufügen (z.B. click)
  1. Über HTML-Attribut: <button onclick="alert('Hi');"> ☹
    - Unübersichtliches Markup, Handler schlecht entfernbar
  2. Implizit im Script
    - Sauberes Markup, aber pro Ereignis nur ein Eventhandler

```
document.getElementById(id).onclick = function(evt) { ... };
```
  3. Über DOM Level 2 Event Listener ☺
    - Sauberes Markup u. beliebig viele (deaktivierbare) Handler

```
elem.addEventListener("click", handlerFunc, useCapture);
elem.removeEventListener("click", handlerFunc, useCapture);
```



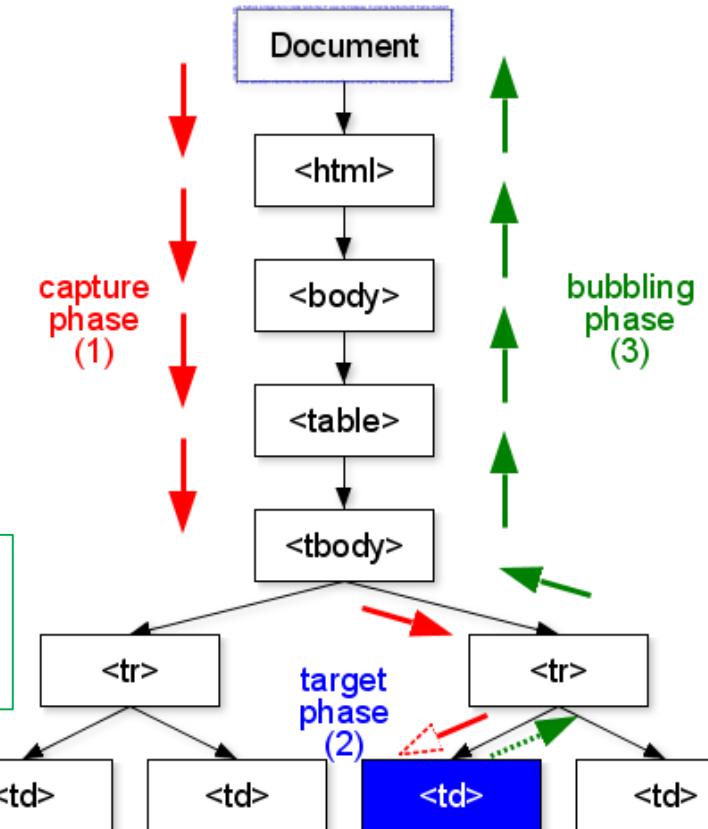
# Eventfluss und Event-Objekte

```
interface MouseEvent : UIEvent {
 readonly attribute DOMString type;
 readonly attribute EventTarget target;
 readonly attribute long screenX;
 readonly attribute long screenY;
 readonly attribute long clientX;
 readonly attribute long clientY;
 readonly attribute boolean ctrlKey;
 readonly attribute boolean shiftKey;
 readonly attribute boolean altKey;
 readonly attribute unsigned short button;
 void stopPropagation();
 void preventDefault();
 [...]
};
```

Verhindert Standardverhalten des Browsers (z.B. Formulare werden nicht gesendet)

Eventfluss unterbrechen:  
event.stopPropagation();  
Internet Explorer < 9:  
event.cancelBubble = true;

- Bubbling
  - Innerste Handler zuerst ausführen (`useCapture = false`)
- Capturing
  - Ereignis wird von Wurzelement bis Ziel behandelt



# Formulare revisited

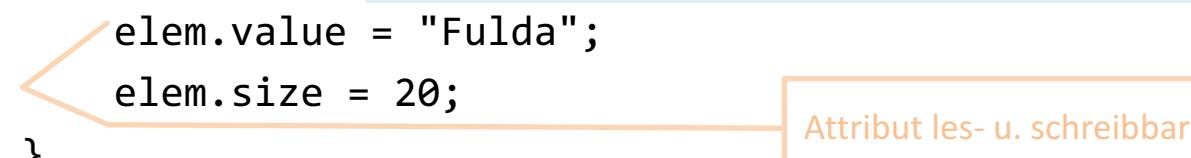
```
<form action="auswerten.php" method="get" onsubmit="return check();">
 <input type="text" name="ort" size="30" id="orteingabe">
 <input type="password" name="pwd">
 <input type="checkbox" value="ok" name="cb">
 <input type="submit" value="Senden">
 <input type="reset">
 <input type="button" value="Ort ändern" onclick="aendereOrt();">
</form>
```



http://localhost/auswerten.php  
?ort=Fulda&pwd=4711&cb=ok

- Mit JavaScript auf Element zugreifen und ändern

```
function aendereOrt() {
 var elem = document.getElementById("orteingabe");
 elem.value = "Fulda";
 elem.size = 20;
}
```



Attribut les- u. schreibbar

- Radiobuttons über Name holen und Liste durchlaufen



# Wichtige Events

---

- Ressourcen sind geladen
  - load (z.B. bei HTML-Seite: window.onload)
  - DOMContentLoaded (nur DOM-Tree geladen)
- Submit-Button von Formular gedrückt
  - submit (ohne Behandlung erfolgt Reload der Seite)
- Änderung bei <input>, <select>, <textarea>
  - change (Verlassen des Elements abfangen: blur), focus
- Tastatur-Events
  - keydown, keyup (Taste gedrückt bzw. losgelassen)
- Maus-Events
  - mousedown, mouseup, mousemove, click, dblclick, wheel, mouseover, mouseout, contextMenu
- Touch-Events
  - touchstart, touchmove, touchend

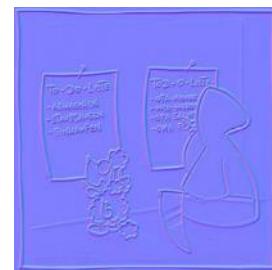


# (Rollover-) Effekte mit JS

```

<script>
 var img = document.getElementById("myImage");
 var colormap = new Image();
 colormap.src = img.src;
 var normalmap = new Image();
 normalmap.src = "img/todoNM.jpg";

 img.addEventListener("mouseover", function(evt) {
 img.src = normalmap.src;
 }, false);
 img.addEventListener("mouseout", function(evt) {
 img.src = colormap.src;
 }, false);
</script>
```



```
img.addEventListener("mouseover", function(evt) {
 img.style.border = "2px solid black";
}, false);
img.addEventListener("mouseout", function(evt) {
 img.style.border = "none";
}, false);
```

Attributname zum Ändern der Klasse:  
*className*  
Alle Klassen über:  
*classList*



# Bilder und Geschwindigkeit

---

- Bilder bzw. Medien sind größtes Bottleneck für Ladegeschwindigkeit von Webpages
  - Sind groß und erfordern viele HTTP-Requests
- Lösungsmöglichkeiten
  - Content Delivery Network (CDN)
    - Zur Dateiübertragung Nutzung weiterer Domains
    - Ermöglicht mehr parallele Verbindungen, entlastet Server, geringere Latenz bei geografischer Nähe
  - Komprimieren (evtl. niedrigere Bildqualität, kleineres Bild, unnötige Header löschen)
    - Format geschickt wählen, z.B. SVG
  - Erst dann nachladen, wenn sie benötigt werden
    - Nach Laden der Seite, wenn man an Position scrollt



# Lazy Loading

---

## Beispiel für JS-Aufbau

1. Eintreten eines Events (z.B. load oder bestimmte Scrollposition) abwarten
  - Nicht sichtbare Bilder müssen nicht geladen werden
  - Keine höhere Auflösung verwenden, als nötig
2. a) (Platzhalter-) Image in HTML einfügen  
b) src Attribute verändern (z.B. von data-src)

## Beispiel für Image-Tag

```

```



# Neues Event in HTML5: Drag and Drop

---

- Mit Attribut „draggable“ Element ziehbar machen  

```

<div id="ziel" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
```
- Drag
  - In *ondragstart* Typ + Wert der gezogenen Daten setzen
    - event.dataTransfer.setData("text/plain", event.target.id);
- Drop
  - Erlauben durch Abschalten des Default-Verhaltens bei *ondragover* mit event.preventDefault();
  - In *ondrop* gezogenes Element bearbeiten
    - var data = event.dataTransfer.getData("text/plain");
    - event.target.appendChild(document.getElementById(data));
    - event.preventDefault();



# Validierung von Formularen

---

- Formulare bereits clientseitig überprüfen
  - Zur Reduktion übertragener Daten, Entlastung des Servers und für mehr Interaktivität
    - Mit Hilfe von String-Methoden oder RegExp
  - Aus Sicherheitsgründen auch auf Server prüfen
    - Z.B. wegen Cross-Site-Scripting-Angriffen (XSS)
- Reguläre Ausdrücke
  - RegExp-Objekt für Pattern Matching (z.B. `/@/`)
    - Zum Ausprobieren: <https://regex101.com/#javascript>
  - Zwei Kategorien
    - Normale Zeichen (stellen sich selbst dar)
    - Metazeichen (haben besondere Bedeutung):  
`\ | ()[]{}^$ * + ? .`



# Reguläre Ausdrücke

---

- Zeichenklassen
  - Ziffern (\d), entspricht Bereich [0-9]
  - Alphanumerische Zeichen (\w), entspr. [A-Za-z0-9\_]
  - Komplementäre Auswahl: [^abc]
  - Alternativen mit: |
  - Whitespace (\s), entspricht [ \r\t\n\f]
  - Beliebiges Zeichen (matcht alle Zeichen): .
  - ^ Anfang, \$ Ende
- Quantoren: {n,m}, {n,}
  - ? kein oder einmaliges Vorkommen
  - + mindestens einmal (z.B. \d+)
  - \* beliebige Anzahl (inkl. 0)

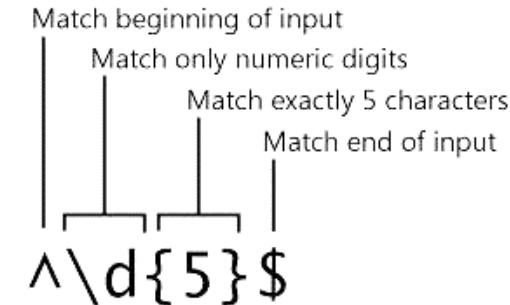


# Reguläre Ausdrücke

---

- RegExp anlegen
  - let regExp = new RegExp('^\d{5}\$');
  - Methode `test(str)` prüft Vorkommen in String `str`
- Beispiele (Stringmethoden)
  - let pos = myStr.search(/[,\s]+/); // Index
  - let f = plz.match(regExp); // Feld mit Vorkommen
  - let f = str.split(','); // trennt String, liefert Feld
  - myStr = myStr.replace(<[^>]+>/g, "");
- HTML5 Pattern Attribut

```
<input type="text" id="plz"
placeholder="PLZ" required
pattern="^\d{5}$">
```



# Cookies

---

- Textdateien zur clientseitigen Speicherung sessionspezifischer Daten
  - Speicherplatz weniger als 4KB
  - Vom Server u. Client schreib-/lesbar
  - Mit HTTP-Anfrage mit übermittelt
- Durch Name-Wert-Paare setzen

```
var d = new Date();
d.setTime(d.getTime() + (2*60*1000)); // 2 min
var exp = "; expires=" + d.toUTCString(); // Ablauftermin
document.cookie = "name=Smith" + exp;
document.cookie = "vorname=John" + exp;
```
- Auslesen analog: var keks = document.cookie;
  - Einträge getrennt durch Semikolon (u. ggfs. Space)
  - Zunächst daran splitten, dann Wertpaare an „=“

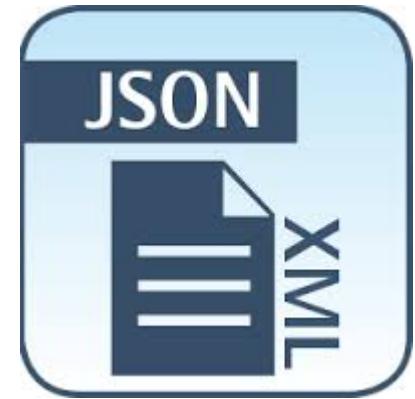


# HTML Web Storage

---

- Clientseitiger Speicher
  - Limit größer als bei Cookies, mindestens 5 MB
  - Sicherer, da Daten nur lokal gespeichert
  - Schlechtere Browser-Unterstützung als Cookies
  - Daten als Schlüssel-Wert-Paare gespeichert
- LocalStorage
  - Speichert Daten pro Browser ohne Ablaufdatum
  - Beispiel: `localStorage.setItem("key", "value");`
- SessionStorage
  - Speichert Daten für eine Sitzung (überlebt Reloads)
  - Daten nach Schließen des Fensters / Tabs verloren





# Datenaustauschformate



# JSON

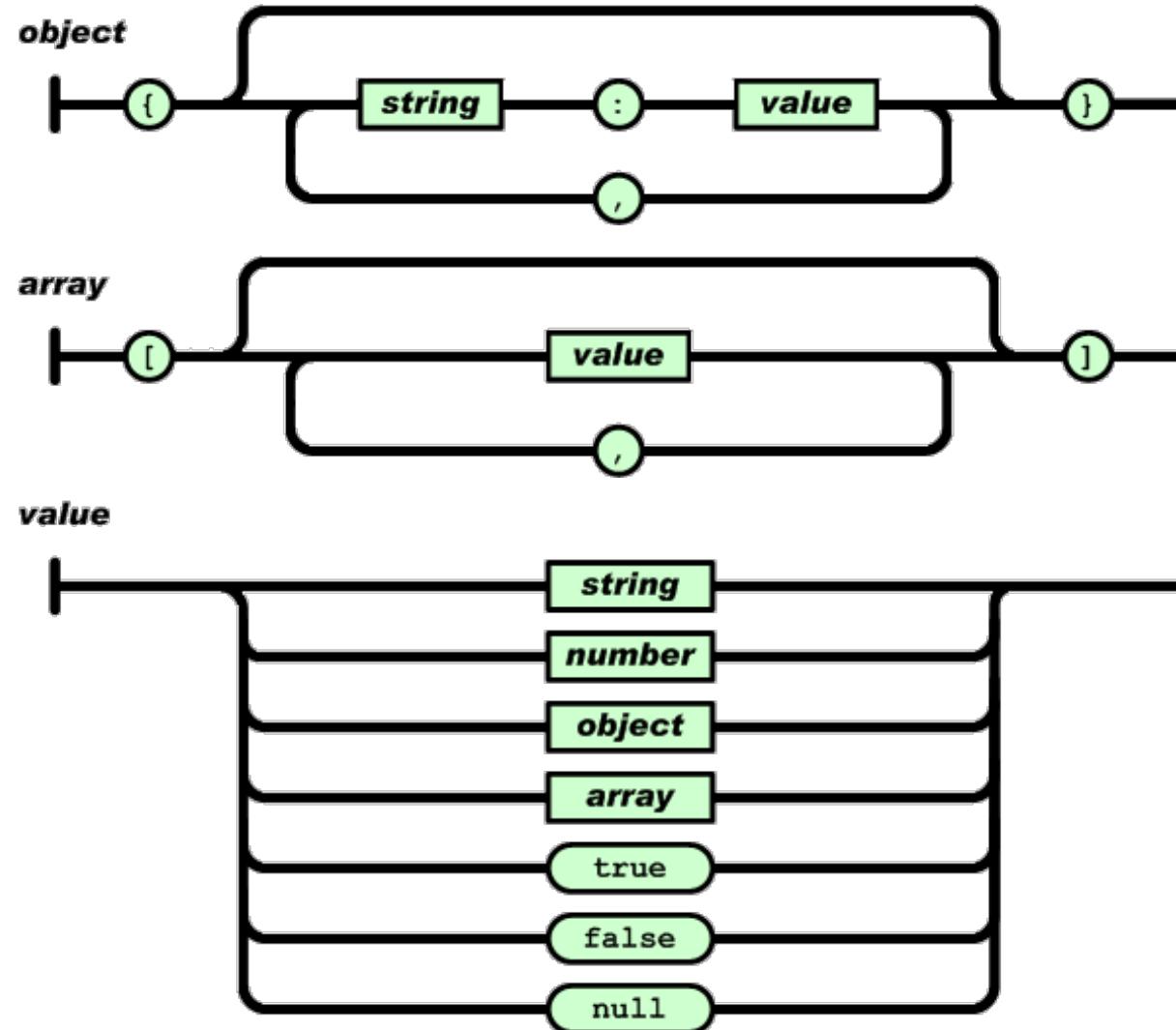
---

- JavaScript Object Notation (JSON)
  - JSON-Dokument muss gültiges JS-Objekt sein
  - Programmiersprachenunabhängig
  - Besteht aus Name-Wert-Paaren und Arrays
  - Beispiel:

```
{
 "name": "Mustermann",
 "vorname": "Max",
 "alter": 42,
 "hobbys": ["Joggen", "Lesen"]
}
```
- Kompaktes Datenaustauschformat
  - Insbes. zw. Client und Server, z.B. mittels Ajax
  - Ressourcenschonender als XML



# JSON-Syntax



# Extensible Markup Language (XML)

---

- Für Beschreibung und Austausch von Daten
  - XML (und HTML) abgeleitet von SGML
  - XML-Dokument mit XSLT über XSLT-Stylesheet in veränderte Ergebnisdokumente transformierbar
- Auszeichnungssprache mit frei wählbaren Tags
  - Metasprache zur Definition von Sprachen
  - Datenstruktur beschrieben durch Document Type Definition (DTD) oder XML Schema (XSD)
- Dokumente beginnen mit XML-Deklaration

```
<?xml version="1.0" encoding="UTF-8"?>
```

  - Bestehen aus Baumstruktur von Elementen mit einem Wurzelement
  - Normaler Text in XML über CDATA-Abschnitt:  
`<![CDATA[ Inhalt ]]>`



# Bsp.: XML-Encoding von X3D

- X3D: Auszeichnungssprache zur deklarativen Beschreibung dynamischer, interaktiver 3D-Welten im Web

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
"http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance'
profile='Full' version='3.0' xsd:noNamespaceSchemaLocation=
'http://www.web3d.org/specifications/x3d-3.0.xsd'>
 <Scene>
 <Transform DEF='trafo' translation='0 0 0'>
 <Shape>
 <Appearance>
 <Material ambientIntensity='0.2' diffuseColor='0.54 0.05 0.25'
shininess='0.83' specularColor='0.81 0.77 0.75'
transparency='0.3' />
 </Appearance>
 <Cone/>
 </Shape>
 </Transform>
 </Scene>
</X3D>
```

<http://www.web3d.org/standards/version/V3.3>



# XML vs. HTML

---

- XML
  - Dokumentbeschreibungssprache
  - Zum Transportieren und Speichern von Daten
  - Alle Elemente müssen geschlossen werden
  - Case-sensitive (lowercase)
  - Bei korrekter Syntax ist Dokument **wohlgeformt**
  - **Gültig**, wenn gegen DTD / XML-Schema validiert wurde
- HTML
  - Zur Darstellung von Daten (Text, Multimedia)
  - Keine self-closing Tags
  - Keine Closing-Tags bei sog. leeren Elementen
  - Nicht case-sensitive (aber meist lowercase)
  - Besteht aus im HTML-Standard definierten Tags
  - XHTML: XML-konformes HTML



# Parsen und Serialisieren

---

- JSON
  - JS-Objekt aus JSON-String erzeugen: `JSON.parse()`
  - JS-Objekt in JSON-String konvertieren: `JSON.stringify()`
  - JSON-File nach Laden direkt in Objekt konvertieren:

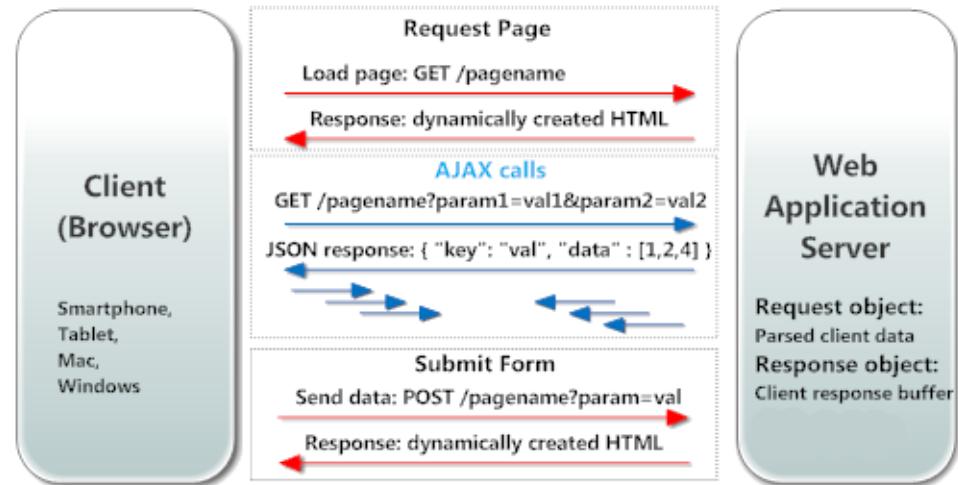
```
var obj = JSON.parse(xhr.responseText);
```
  - Sonderbehandlung mit `eval()` bei uralten Browsern:

```
var obj = eval("(" + xhr.responseText + ")");
```
- XML
  - `var parser = new DOMParser();`  
`var node = parser.parseFromString(xmlStr, "text/xml");`  
Knotename (String): `node.localName`  
Kindknoten (Array): `node.childNodes`
  - `var serializer = new XMLSerializer();`  
`var xmlStr = serializer.serializeToString(node);`

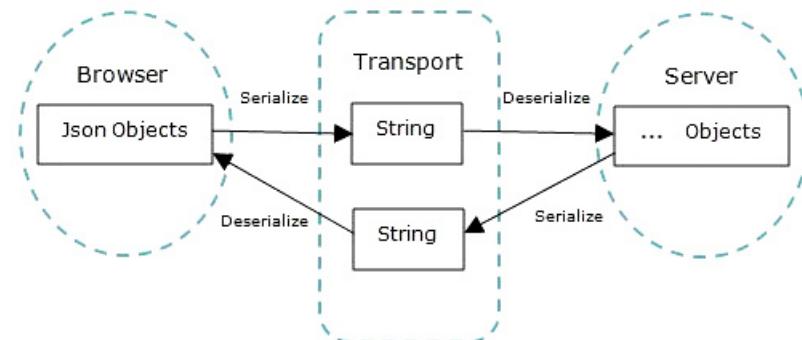


# Client-Server-Kommunikation

- Datenübertragung
  - Verschiedene Varianten:



- Datenaustausch über Strings



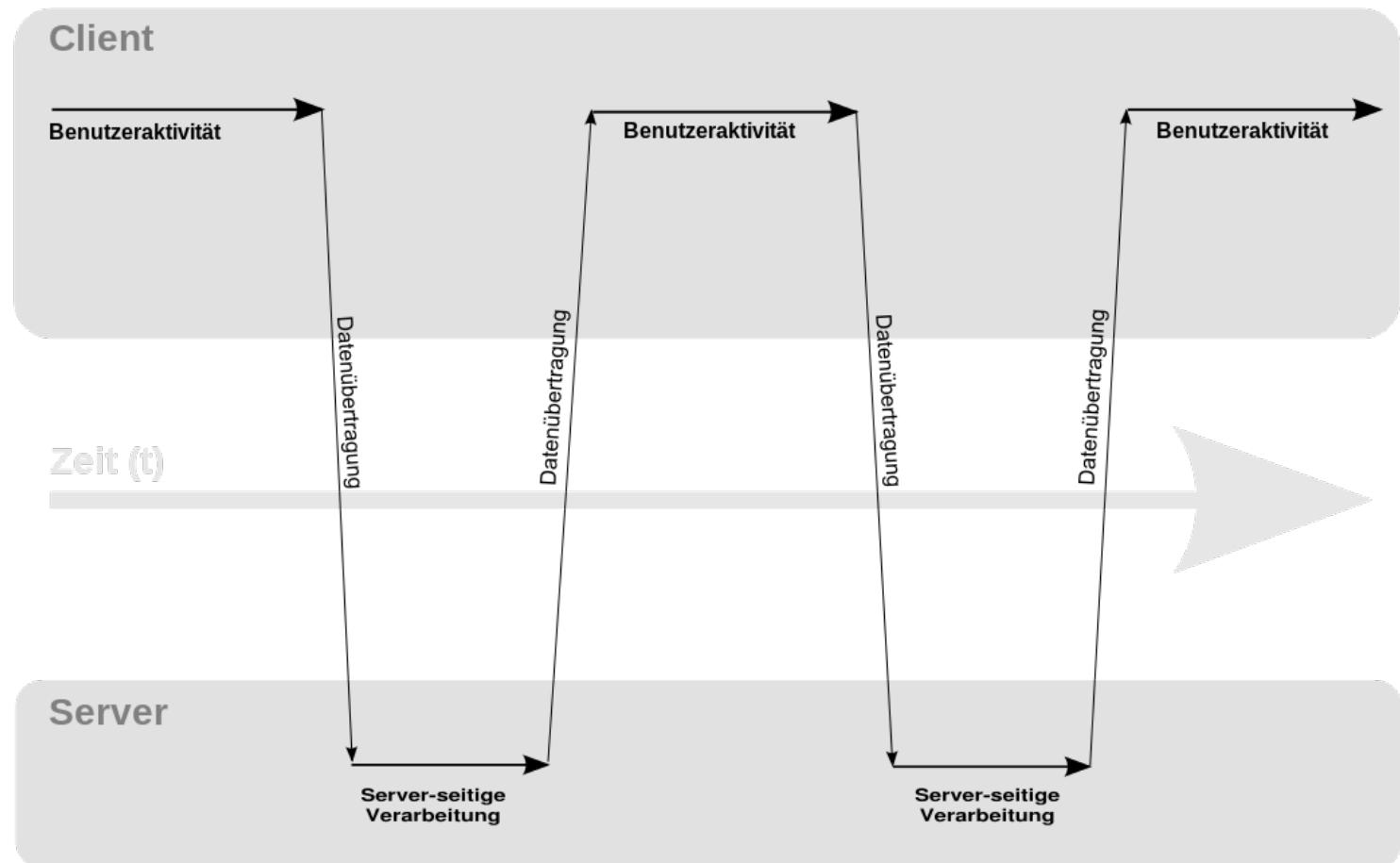


# Ajax: Inhalte nachladen



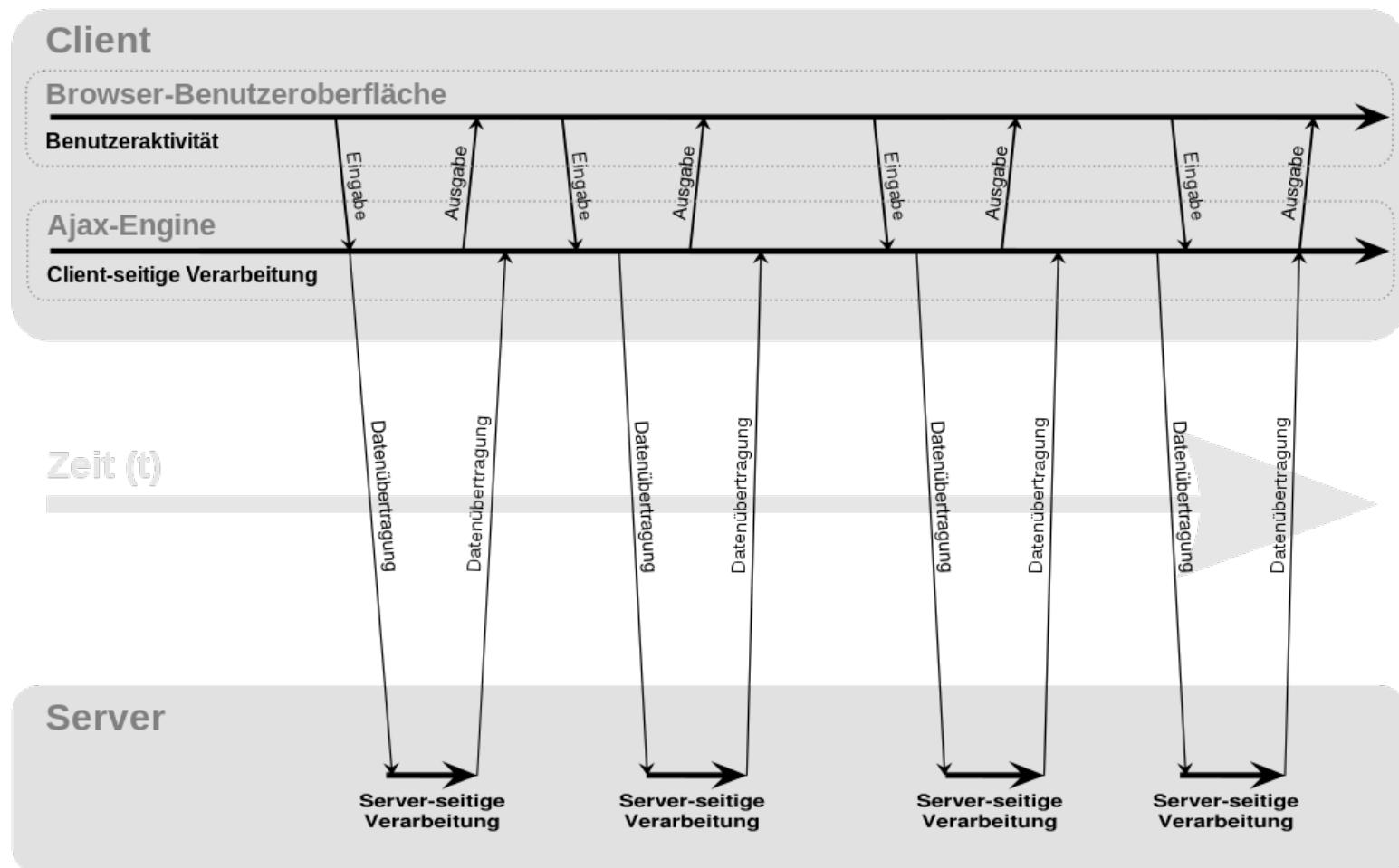
# Synchroner Datenfluss

Klassisches Modell einer Web-Anwendung (synchrone Datenübertragung)



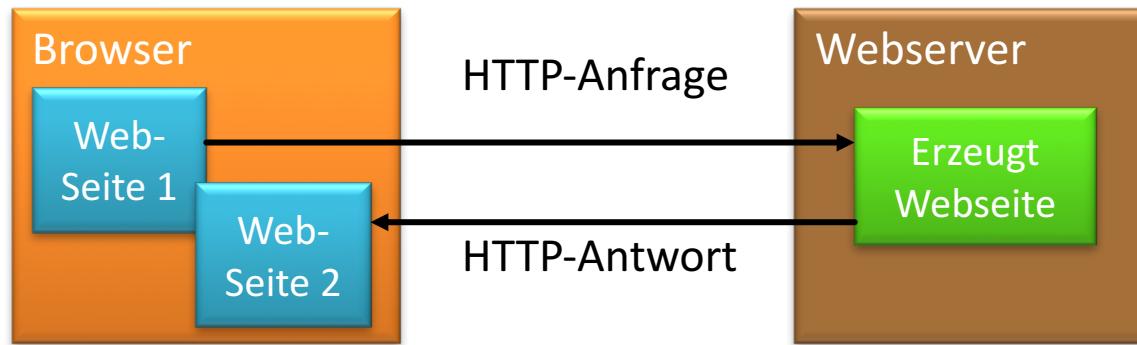
# Asynchroner Datenfluss

Ajax Modell einer Web-Anwendung (asynchrone Datenübertragung)

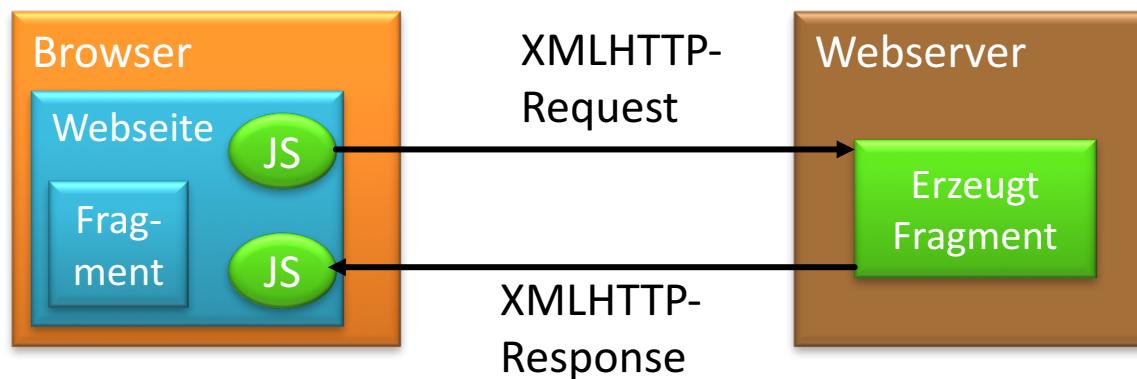


# Web-Anwendungen mit Ajax

- Klassische Webanwendung



- Ajax-Anwendung



# Ajax

---

- Verwendet asynchrone Kommunikation
  - Erlaubt es, neue Anfragen an Server zu senden, während man parallel noch auf Antworten wartet
  - Ermöglicht es, nur Teile einer Webseite vom Server zu laden, ohne alles neu zu laden
    - Z.B. automatisch aktualisierte Suchbegriffsliste beim Googlen oder bei Newsticker
  - Verbessert Nutzerfreundlichkeit und Bedienbarkeit
- Grundlage ist XMLHttpRequest
  - Asynchronous JavaScript *and* XML
  - Zum Übertragen von Daten – nicht nur für XML!
  - Synchron ☹ und asynchron ☺ möglich



# XMLHttpRequest (XHR)

---

- JavaScript-Objekt zum Datenaustausch
- Eigenschaften
  - `onreadystatechange` // Event-Handler
    - Bzw. besser: `onload` und `onerror` verwenden
  - `response` bzw. `responseText` // Antwort oder null
  - `responseType` // "text", "json", etc.
  - `status` bzw. `statusText` // HTTP-Statuscode
- Methoden
  - `open( HTTP-Methode, URL, async )` // `async` sollte true
    - Öffnet Verbindung zum Server
  - `setRequestHeader( header, value )`
    - Bsp.: "Content-type", "application/x-www-form-urlencoded"
  - `send( content )` // bei GET kein Param.



# Datei laden mit XMLHttpRequest

---

- Callback-Funktion für Antwort definieren

```
function handler() {
 if (xhr.readyState == 4) {
 if (xhr.status == 200)
 console.log(xhr.responseText);
 else
 console.warn(xhr.statusText);
 }
}
```

readyState (0..4) enthält Status der Verbindung:  
4 heißt, Daten wurden vollständig übertragen

status enthält HTTP-Status der Verbindung

- Verbindung herstellen und Anfrage senden

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = handler;
// Modernere Alternative zu onreadystatechange Handler:
// xhr.onload = function() { console.log(xhr.responseText); };
xhr.open("GET", "files/test.txt", true); // ,true' für async
xhr.send();
```



# Webserver

---

- Verschiedene Server verbreitet
  - Apache, nginx, Microsoft IIS, Lighttpd, ...
- XAMPP einfache PHP-Entwicklungsumgebung
  - Download: <https://www.apachefriends.org/>
  - Beinhaltet Apache-Distribution mit PHP u. MySQL
  - Webprojekte im Unterordner „htdocs“ abspeichern
    - Macht z.B. NetBeans automatisch bei Anlegen eines neuen PHP-Projekts
  - Server konfigurieren über  
*apache/conf/httpd.conf*
    - PHP über *php/php.ini*



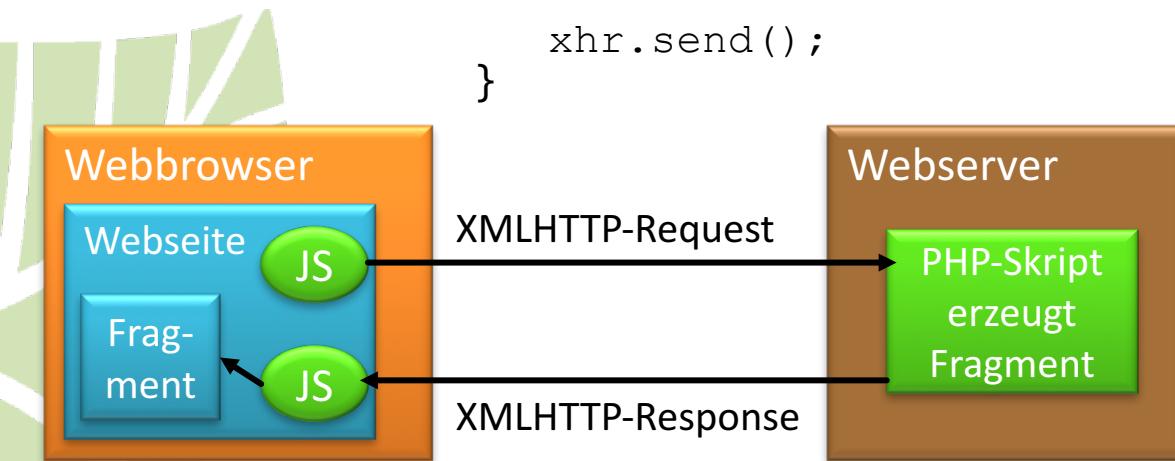
# Ajax und PHP

```
<input type="text" name="uname" id="uname">
<input type="text" name="msg" id="msg">

<input type="button" value="Get Data" onclick="getData()">

<div id="dataDiv">
</div>
```

```
function getData() {
 var username = document.getElementById("uname");
 var message = document.getElementById("msg");
 var queryString = "chat.php?uname=" + username.value
 + "&msg=" + message.value;
 var xhr = new XMLHttpRequest();
 xhr.onload = function() {
 var elem = document.getElementById("dataDiv");
 elem.innerHTML += this.responseText;
 };
 xhr.open("get", queryString, true);
 xhr.send();
}
```

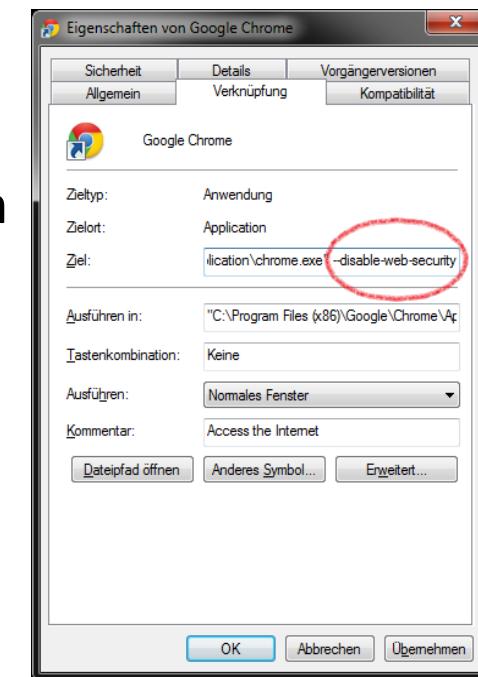
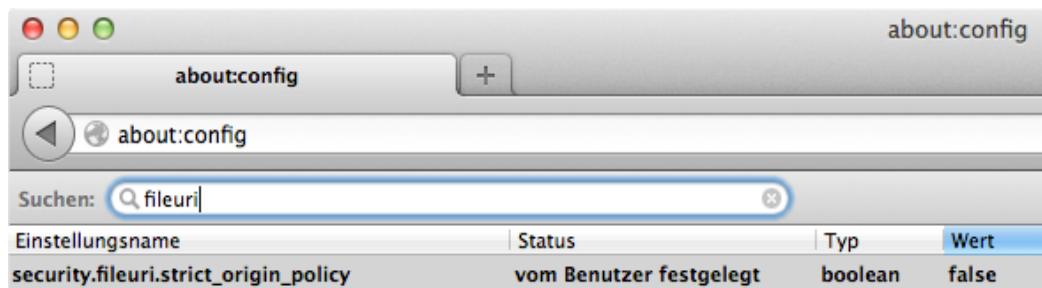


```
<?php
header("Access-Control-Allow-Origin: *");
```

```
<?php
$user = $_GET["uname"];
$msg = $_GET["msg"];
echo $user . ":" .
 $msg . "
";
?>
```

# Von externen Ressourcen laden

- Websicherheitsmechanismen erlauben keinen Zugriff auf andere Domains
  - XHR / Fetch folgen Same-Origin Policy, d.h. HTTP-Ressourcen können nur von gleicher Domain geladen werden
  - Ausnahme: HTTP-Response beinhaltet passenden CORS-Header (Cross-Origin Resource Sharing)
- Schließt Filesystem mit ein (file://...)
  - Webseite von lokalem Webserver starten
  - (...oder Security wie in Screenshots gezeigt zum Testen abschalten ☹)



# Fetch API

---

- Moderne API, um Ressourcen über Netzwerk zu laden
  - Alternativ zu XMLHttpRequest
  - Für asynchrone Anfragen, gibt sog. Promise zurück
    - Einfacher konfigurierbar über Settings-Objekt (z.B. für CORS)
  - Methoden *then()* und *catch()* erlauben es, Callbacks für Event- bzw. Error-Handling zu definieren
    - Wird ausgeführt, wenn Promise festgelegt ist ("settled")
    - An *then()* übergebener Handler erhält Response-Objekt
    - Im Fehlerfall greift *catch()*
- Verwendungsbeispiel
  - `fetch('./test.json', {method: 'get', mode: 'cors'})`  
`.then(response => response.json())`  
`.then(data => console.log(data))`  
`.catch(err => console.error(err));`
  - Hinweis: `json()` erzeugt Stream und liefert wieder Promise



# Asynchrone Programmierung

---

- Wichtiges Prinzip bei Webprogrammierung
  - Ressourcenzugriffe sind zeitaufwendig (und ggfs. blockierend), aber Webseite sollte dabei responsiv bleiben
- Bei Aufruf von asynchroner Funktion übergibt man als Argument Callback-Funktion
  - Wird irgendwann später aufgerufen, wenn Ergebnisse vorliegen (z.B. Daten vom Server geladen)
  - Sehr einfaches Beispiel:

```
setTimeout(function() { console.log('World'); }, 100);
console.log('Hello');
```
  - Ergebniswert (oder evtl. Fehler) wird Callback-Funktion als Argument übergeben
    - Ergebnis wird innerhalb des Callbacks behandelt
  - Code wird unübersichtlich und schwer wartbar, wenn asynchrone Operationen weitere solche anstoßen (sog. Pyramid of Doom)
    - S. <https://alexandernaumov.de/artikel/javascript-promises-asynchrones-programmieren>



# ES6 Promises

---

- Kapseln asynchrone Operationen in einheitlicher API
  - Asynchrone Funktionen können so Ergebniswerte in ähnlicher Weise wie synchrone Fktn. zurückgeben
  - Statt tatsächlichem Wert wird Promise zurückgegeben
    - Platzhalter / Versprechen, wird erst in Zukunft aufgelöst
    - Noch nicht überall unterstützt (insbes. nicht von IE)
  - Nutzen Callbacks je für Erfolgs- und Fehlerfall
- Zustände
  - Pending: Anfangszustand, Ergebnis noch unbestimmt, da asynchroner Vorgang noch nicht abgeschlossen
  - Fulfilled: Operation erfolgreich, Wert liegt vor
  - Rejected: Operation gescheitert (mit Grund)

} Settled



# ES6 Promises

---

- Wichtigste Methoden sind *then()* und *catch()*
  - Können verkettet werden, da Promise zurückgegeben wird
  - Operationen laufen damit quasi nacheinander ab
- Beispiel

```
function asyncFunc() {
 let p = new Promise(function(resolve, reject) {
 setTimeout(function() {
 let res = Math.floor(Math.random() * 100);
 if (res >= 50) resolve(res);
 else reject("Fehler, " + res + " zu klein");
 }, 2000);
 });
 return p;
}
asyncFunc().then(res => console.log(res))
 .catch(err => console.warn(err));
```



# ES7 Async Functions

---

- Von C# entlehnte weitere Vereinfachung asynchroner Programmierung
- Mit *async* markierte Fkt. gibt implizit Promise zurück
- In Async-Function kann man *await* for asynchronen Funktionsaufruf setzen
  - Aufrufende Funktion wartet, bis Ergebniswert feststeht
    - Oder Fehler auftrat → Verwendung von try / catch() möglich
  - Vorteil: ähnlich wie synchrone Funktionen verwendbar
- Beispiel

```
async function loadText(name) {
 let str = await fetch(name);
 let out = document.getElementById("output");
 out.innerHTML = await str.text();
}
loadText("test.txt");
```



# Weitere neue Web-APIs

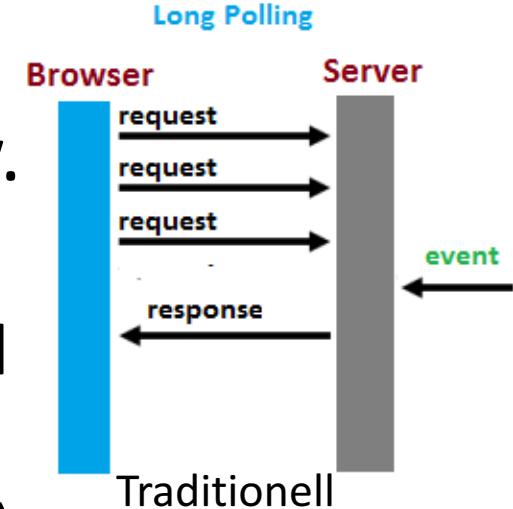
---

- Zugriff auf geographische Position
  - Nur in sicheren Kontexten nutzbar (→ https)
  - Über navigator.geolocation
    - Kombinieren mit Google Maps / OpenStreetMap Service
- Push statt pull
  - WebSockets und Server-Sent Events (ermöglichen es, aktiv vom Server aus Nachrichten an Client zu schicken)
- Web Worker, um Programme parallel auszuführen
- WebRTC
  - Umfasst verschiedene APIs und Protokolle für Echtzeitkommunikation via JS
  - Z.B. Kamerastream mit getUserMedia() abgreifen

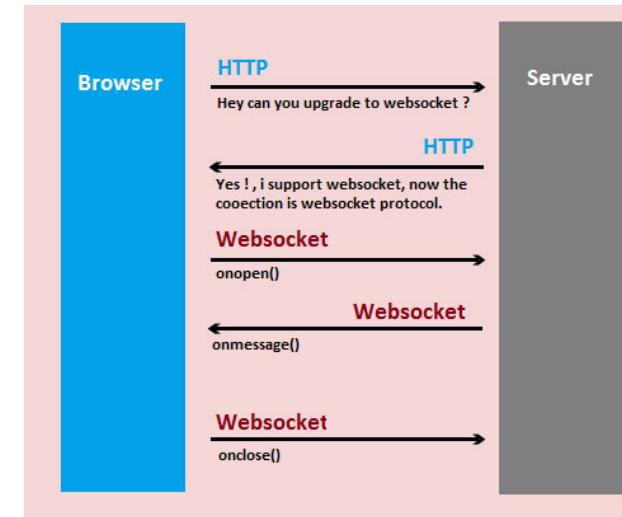


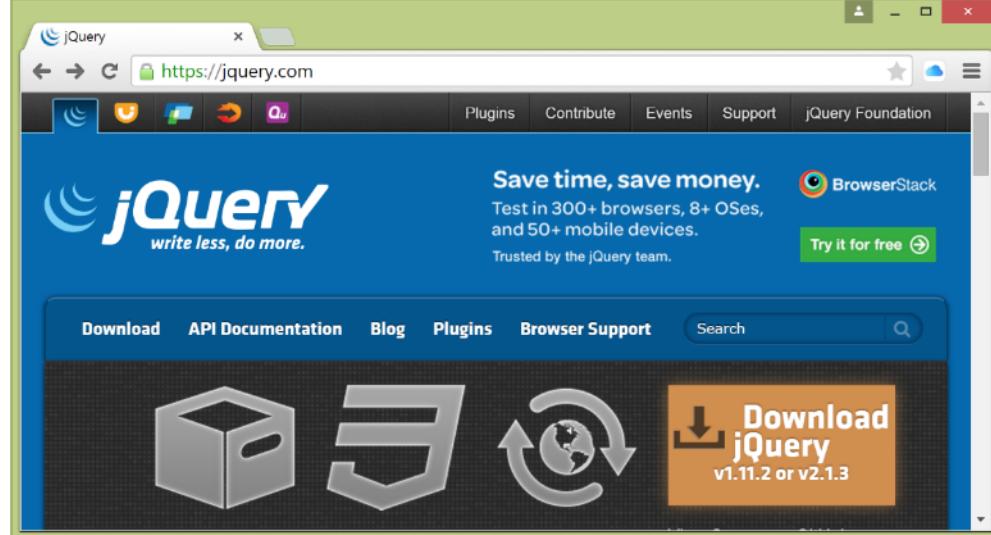
# WebSockets

- Bidirektionale Verbindung zw. Client und WebSocket-Server
  - Reduziert Netzwerk-Traffic und Latenz, erlaubt Streaming
  - Über HTTP bzw. HTTPS (wss://)
- Client öffnet Verbindung
  - Verbindung bleibt offen
  - Server kann Nachrichten jederzeit pushen
  - Client kann auch jederzeit Nachrichten senden



Mit WebSockets:





# Exkurs: JS-Bibliothek **jQuery**



# (Mobile) JavaScript Libraries

- Many different platforms available
  - Desktop: Windows, Mac, Linux
  - Mobile: Android, iOS, ...
- Lots of problems
  - Native apps need to be developed for all systems (→ portability)
  - Both, desktop applications and web sites, need to be developed
- Possible solution
  - Develop web sites for mobile browsers
    - ... with look and feel of native apps
    - No need to install app
    - Normal web developer can create apps
  - JS UI libraries help supporting look and feel
    - Using library often more efficient

The screenshot shows a mobile web application interface. At the top, there's a green header bar with the title 'Nested list'. Below the header, there's a sidebar on the left containing links like 'List views', 'Basic linked list', 'Nested list' (which is highlighted in black), 'Numbered list', 'Split button list', and 'List dividers'. The main content area displays a nested list structure under the heading 'Animals'. It lists items from 'Aardvark' to 'Zebra' with descriptions. Below the animal section, there are sections for 'Colors' and 'Vehicles'.

<http://jquerymobile.com/demos/1.1.0/docs/lists/lists-nested.html>

The screenshot shows a mobile web application interface with a dark theme. At the top, there's a black header bar with the title 'UI Demos'. Below the header, there's a sidebar on the left containing links like 'Home', 'Lists', 'Buttons', and 'Forms'. The main content area displays a list of UI components. Under 'Lists', there are sections for 'Edge to Edge', 'Plastic', and 'Metal'. Under 'Buttons', there are sections for 'Buttons' and 'Forms'. Each section has a link to its details.

<http://www.jqtouch.com/preview/demos/main/#ui>



# jQuery

---

- Very powerful, easy to use and easy theming
- Very well documented with many examples
  - <http://api.jquery.com/>
- Easy to manipulate DOM
  - Especially when supporting older Browsers
  - But slower than pure JS
- Include core library

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js "></script>
```
- Include core library (slim version)

```
<script src=" https://code.jquery.com/jquery-3.3.1.slim.min.js">
</script>
```



# jQuery

---

- Allows for DOM manipulation
  - But does not provide structure to your code!
- jQuery commands start with „jQuery“ or „\$“
  - Commands can be chained:  
e.g. `$ (variable) .find ("something") .each (...)`
- Provides some useful functions
  - Wait till the DOM (page) is loaded to execute code

```
$ (document) .ready(function() {
 // do stuff when DOM is ready
});
```
  - Selectors can be used as in CSS

```
$ ("a") , $ ("#id") , $ (" .class ")
```
  - Search descendants of already selected elements

```
$ ("#list") .find ("li") equal to $ ("#list li")
```
- Events, e.g.: `$ ("a") .click (callbackFunc) ;`



# jQuery

---

- Elements can be created with

```
var p = jQuery('<p>' + contentVar + '</p>');
group.append(p);
```

- Create an element and write attributes

```
var elem = jQuery('<div/>').attr({
 "data-role" : "collapsible",
 "data-collapsed" : "true",
 "data-mini" : "true",
 "data-iconpos" : "right",
 "data-theme" : "b",
 "class" : "metaDataEntry"
});
```

- attr() or prop() functions can be used to read or write attributes



# jQuery UI

- <http://jqueryui.com> (tutorials: <http://learn.jquery.com>)
- Useful for easy creation of GUIs
  - Draggable/Resizable Elements

```
$ ("#myDiv").draggable();
```
  - Slider, Tabs
  - Autocomplete fields
  - ...

Sed vel diam id libero [rutrum convallis](#). Donec aliquet leo vel magna. Phasellus rhoncus faucibus ante. Etiam bibendum, enim faucibus aliquet rhoncus, arcu felis ultricies neque, sit amet auctor elit eros a lectus.

text input

checkbox

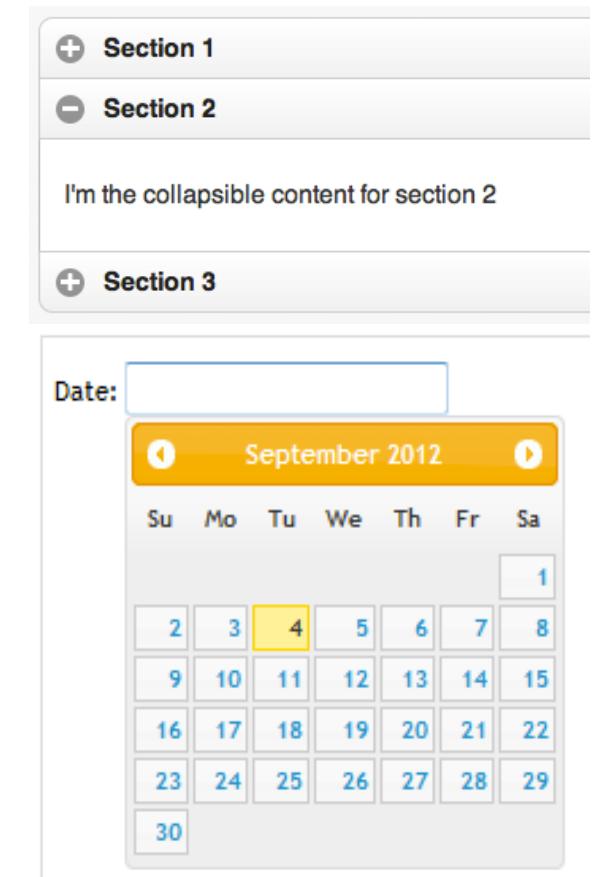
radio

select ▾

textarea

Basic dialog ×

This is the default dialog which is useful for displaying information. The dialog window can be moved, resized and closed with the 'X' icon.



# jQuery Mobile

- Builds upon jQuery and jQueryUI
  - Include jQuery and jQuery Mobile libs
- Loads entire Website, not just currently viewed page
  - No loading lags when switching page because of preloading
- Required page setup
  - jQuery Mobile is all about convention over configuration

```
<div data-role="page">
 <div data-role="header">
 <h1>My Title</h1>
 </div><!-- /header-->
 <div data-role="content">
 <p>Hello world</p>
 </div><!-- /content-->
</div><!-- /page-->
```

Mini flip switch:

Yes



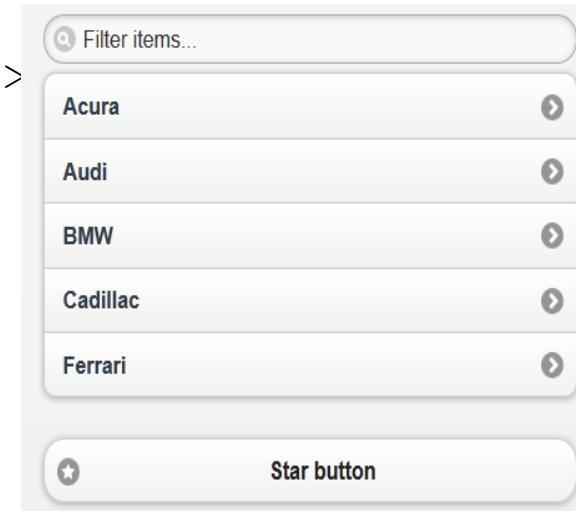
# jQuery Mobile

---

- HTML code has certain commands (or attributes) to tell the framework how to present a section
  - Mostly start with “data-”
  - W3C recommendation for unknown attributes
- Examples

```
<ul data-role="listview"
 data-inset="true" data-filter="true">
 Acura
 Audi
 BMW
 Cadillac
 Ferrari

<a href="#" data-role="button"
 data-icon="star">Star button
```



# jQuery Mobile

---

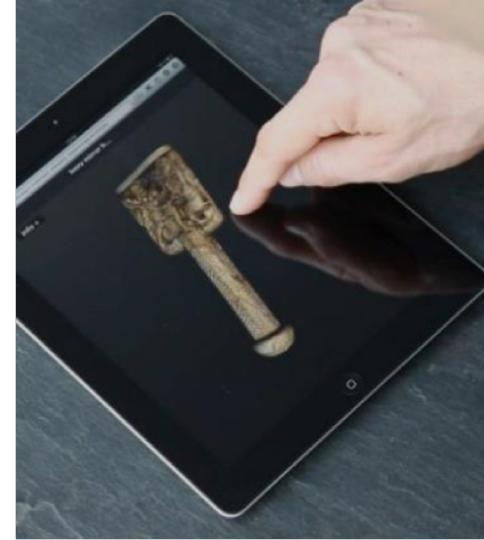
- In dynamic apps HTML is created from XML, JSON or DB
- If HTML is created after DOM is loaded, jQuery Mobile will not convert it automatically
- To force conversion, functions are provided that must be executed after HTML is written into the DOM
- Example: select added element and than convert view

```
jQuery('div[data-role=collapsible]').collapsible();
```



The screenshot shows a mobile interface with a header and a collapsible content area. The header contains two buttons: 'Header swatch A' with a plus sign icon and 'Header swatch B' with a minus sign icon. Below the header is a blue-themed content block with the text 'I'm the collapsible content with a themed content block set to "d".'. The content block has rounded corners and a slight shadow.





# Audio und Graphik



# Multimedia

---

- Audio
  - Angabe der URL mittels Attribut *src* oder *<source>*-Tag
  - ```
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
  Your browser does not support the audio element.
</audio>
```
- Video
 - Leider keine einheitliche Unterstützung für Videoformate
 - ```
<video width="320" height="240" controls>
 <source src="movie.mp4" type="video/mp4">
 <source src="movie.ogg" type="video/ogg">
 Your browser does not support the video element.
</video>
```
- Mit JavaScript lassen sich Audio & Video gezielt steuern
  - Beide HTML5-Elemente bieten API und neue DOM-Events



# Elemente interaktiv positionieren

---

- DOM-Elementen Interaktionen zuordnen
  - Vereinfacht durch JS-Bibliothek „jQuery UI“
    - Beispielsweise sog. Resizable oder Draggable:  
`$("#myMoveableDiv").draggable();`
  - Umsetzung intern mit HTML, CSS u. JavaScript

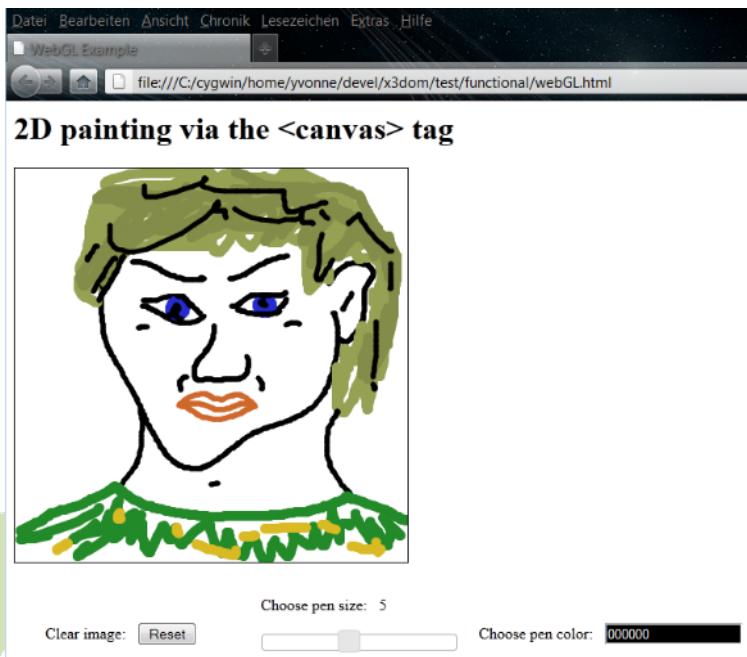


## Übungsaufgabe

- Implementieren Sie ein interaktiv mit der Maus verschiebbares HTML-Element (z.B. ein Bild)
  - Aber ohne Hilfsbibliotheken wie z.B. jQuery UI
- Wie kann man HTML-Elemente positionieren?
  - Folgende CSS-Eigenschaften sind nützlich: position, top, left, z-index
- Auf welche Maus-Ereignisse muss man reagieren?



# DIY-Malprogramm mit <canvas>



## “Zutaten”:

- Canvas-Element als Leinwand  
`<canvas width='400' height='400' id='c'>`
- Mouse-Event-Handler
  - Funktionen die aufgerufen werden:
  - Wenn sich Mauscursor bewegt oder
  - Buttons gedrückt/losgelassen werden
- Zeichenmethoden

```
var canvas = document.getElementById('c');
var context = canvas.getContext('2d');

context.beginPath();
context.moveTo(17, 4);
context.lineTo(47, 11);
context.stroke();
```



# Canvas 2D API

See e.g. [https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Canvas\\_tutorial](https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Canvas_tutorial)

## Canvas element

### Attributes

Name	Type	Default
width	unsigned long	300
height	unsigned long	150

### Methods

Return string	Name toDataURL( [Optional] string type, [Variadic] any args)
Object	getContext(string contextId)

## 2D Context

### Attributes

Name	Type
canvas	HTMLCanvasObject [readonly]

### Methods

Return void	Name save()
void	restore()

## Transformation

### Methods

Return void	Name scale(float x, float y)
void	rotate(float angle)
void	translate(float x, float y)
void	transform( float m11, float m12, float m21, float m22, float dx, float dy)
void	setTransform( float m11, float m12, float m21, float m22, float dx, float dy)

## Image drawing

### Methods

Return void	Name drawImage( Object image, float sx, float sy, float sw, float sh)
void	Argument "image" can be of type HTMLImageElement, HTMLCanvasElement or HTMLVideoElement
void	drawImage( Object image, float sx, float sy, float sw, float sh, float dx, float dy, float dw, float dh)

## Compositing

### Attributes

Name	Type	Default
globalAlpha	float	1.0
globalCompositeOperation	string	source-over

Supports any of the following values:



## Line styles

### Attributes

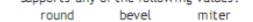
Name	Type	Default
lineWidth	float	1.0
lineCap	string	butt

Supports any of the following values:



### lineJoin

Supports any of the following values:



### miterLimit

Supports any of the following values:



## Colors, styles and shadows

### Attributes

Name	Type	Default
strokeStyle	any	black
fillStyle	any	black
shadowOffsetX	float	0.0
shadowOffsetY	float	0.0
shadowBlur	float	0.0
shadowColor	string	transparent black

### Methods

Return CanvasGradient	Name createLinearGradient( float x0, float y0, float x1, float y1)
CanvasGradient	createRadialGradient( float x0, float y0, float r0, float x1, float y1, float r1)
CanvasPattern	createPattern( Object image, string repetition)

Argument "image" can be of type HTMLImageElement, HTMLCanvasElement or HTMLVideoElement  
"repetition" supports any of the following values:  
[repeat (default), repeat-x, repeat-y, no-repeat]

## CanvasGradient interface

void	addColorStop( float offset, string color)
------	----------------------------------------------

## CanvasPattern interface

No attributes or methods.

## Paths

### Methods

Return void	Name beginPath()
void	closePath()
void	fill()
void	stroke()
void	clip()
void	moveTo(float x, float y)
void	lineTo(float x, float y)
void	quadraticCurveTo( float cpx, float cpy, float x, float y)
void	bezierCurveTo( float cp1x, float cp1y, float cp2x, float cp2y, float x, float y)
void	arcTo( float x1, float y1, float x2, float y2, float radius)
void	arc( float x, float y, float radius, float startAngle, float endAngle, boolean anticlockwise)
void	rect(float x, float y, float w, float h)
boolean	isPointInPath(float x, float y)

## Text

### Attributes

Name	Type	Default
font	string	10px sans-serif
textAlign	string	start

Supports any of the following values:

[start, end, left, right, center]

Name	Type	Default
textBaseline	string	alphabetic

Supports any of the following values:

[top, hanging, middle, alphabetic, ideographic, bottom]

### Methods

Return void	Name fillText( string text, float x, float y, [Optional] float maxWidth)
void	strokeText( string text, float x, float y, [Optional] float maxWidth)
TextMetrics	measureText(string text)

## TextMetrics interface

width	float	[readonly]
-------	-------	------------

## Rectangles

### Methods

Return void	Name clearRect( float x, float y, float w, float h)
void	fillRect( float x, float y, float w, float h)
void	strokeRect( float x, float y, float w, float h)

## Pixel manipulation

### Methods

Return ImageData	Name createImageData(float sw, float sh)
ImageData	createImageData(ImageData imagedata,
void	float dx, float dy, [Optional] float dirtyX, float dirtyY, float dirtyWidth, float dirtyHeight)
void	putImageData( ImageData imagedata, float dx, float dy, [Optional] float dirtyX, float dirtyY, float dirtyWidth, float dirtyHeight)

## ImageData interface

width	unsigned long	[readonly]
height	unsigned long	[readonly]
data	CanvasPixelArray	[readonly]

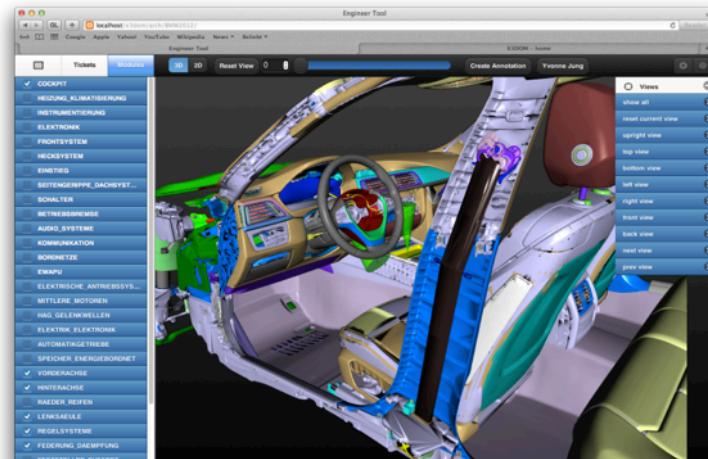
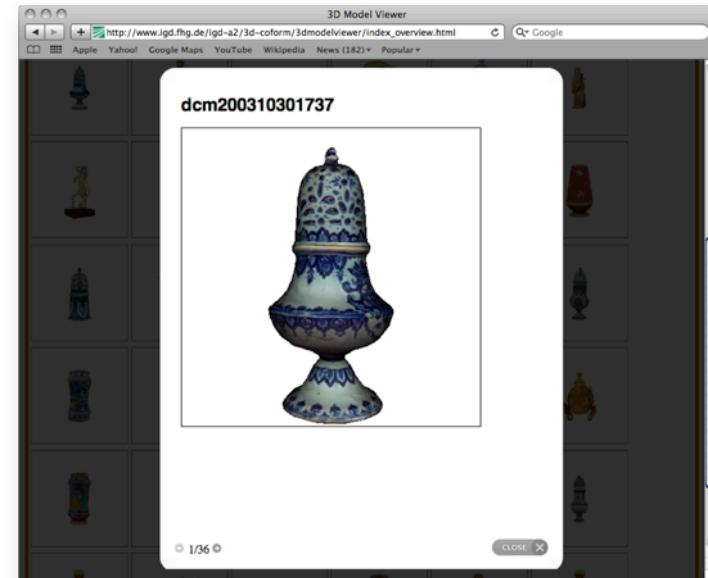
## CanvasPixelArray interface

length	unsigned long	[readonly]
--------	---------------	------------

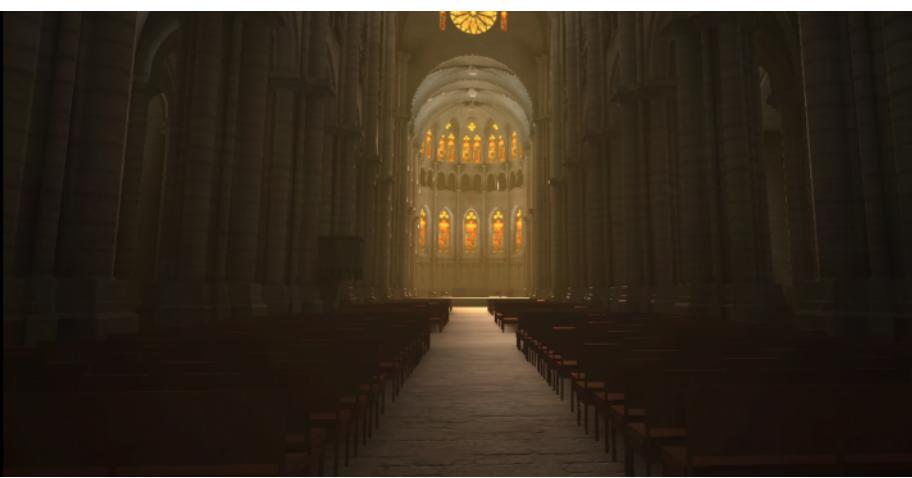


# 3D Information on the Web

- Websites have become Web applications
- Increasing interest in 3D for
  - Games
  - Product presentation
  - Experiencing Cultural Heritage
  - Visualization of abstract information
  - Supporting decision making, e.g. in Virtual Engineering
  - ...
- Enhancing user experience with more sophisticated visualizations
  - Interactive 3D instead of videos
  - Native Web Browser integration via WebGL



# WebGL: OpenGL & GLSL im Web



<http://patapom.com/topics/WebGL/cathedral/index.html>



<http://www.unrealengine.com/html5/>

- JavaScript-Binding für OpenGL ES 2.0 im Webbrower
  - Firefox, Chrome, Safari, Opera, IE11
  - Desktop und mobile Systeme
  - Verwaltet von Khronos Group
- Rein GLSL-Shader basiert, keine sog. Fixed-Function Pipeline mehr
  - Keine Variablen aus GL-State
  - Kein Matrix-Stack, keine Mathe-Lib...
- HTML5 `<canvas>` Element bietet auch 3D-Rendering-Context
  - API-Aufrufe über GL-Objekt:
    - `gl = canvas.getContext('webgl');`
    - `gl.viewport(0, 0, 400, 300);`



# Zugriff auf Binärdaten: TypedArray

<b>Uint8Array</b>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
<b>Uint16Array</b>	0	1	2	3	4	5	6	7								
<b>Uint32Array</b>	0		1		2		3									3
<b>Int16Array</b>	2			16-bit 2's complement signed integer											<b>short</b>	
<b>Uint16Array</b>	2			UNSIGNED SHORT	16 Bit ( $2^{16}$ )	= 65536										<b>short</b>
<b>Int32Array</b>	4			32-bit 2's complement signed integer											<b>int</b>	
<b>Uint32Array</b>	4			UNSIGNED INT	32 Bit ( $2^{32}$ )	= 4294967296										<b>uint32</b>
<b>Float32Array</b>	4			32-bit IEEE floating point											<b>float</b>	
<b>Float64Array</b>	8			64-bit IEEE floating point											<b>double</b>	



# Three.js

---

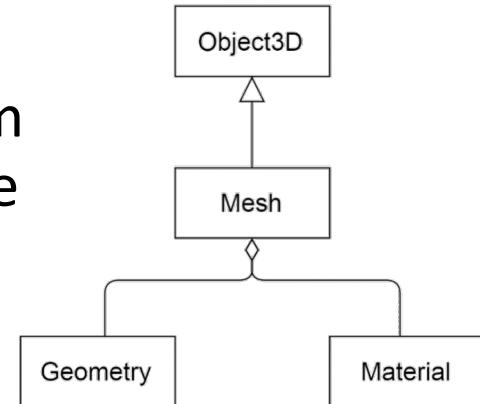
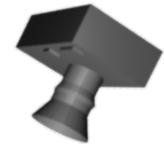
- Open-Source JavaScript-Bibliothek für Echtzeit-3D-Graphik im Webbrowser
  - „[...] for dummies“ [https://github.com/mrdoob/three.js]
  - Einbinden in HTML-Dokument:  
`<script src="http://threejs.org/build/three.min.js">  
</script>`
- Wrapper um WebGL, kapselt low-level WebGL-API
  - Kein langwieriges Initialisieren von Shadern, VBOs etc.
  - Kaum Vektor-/Matrixmathematik nötig
  - Sehr viele Beispiele und Tutorials verfügbar
  - Große User-Community, viele Erweiterungen
  - Schnellere Projektentwicklung als mit reinem WebGL



# Important Elements

---

- Scene
  - Holds all objects that make up 3D world
    - Lights, graphical objects, and cameras
  - Acts as root node for so-called scene graph
    - Hierarchical tree-like structure with nodes
- Camera
  - Object that represents viewpoint from which image of 3D world can be made
  - Represents combination of viewing transformation and projection
- Renderer
  - Creates image: draws to <canvas> element

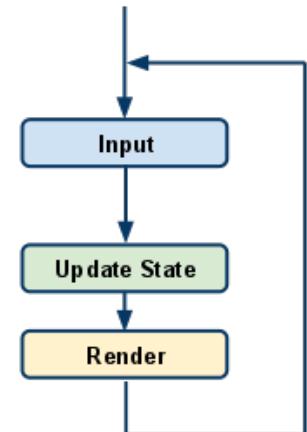


# Animationen

---

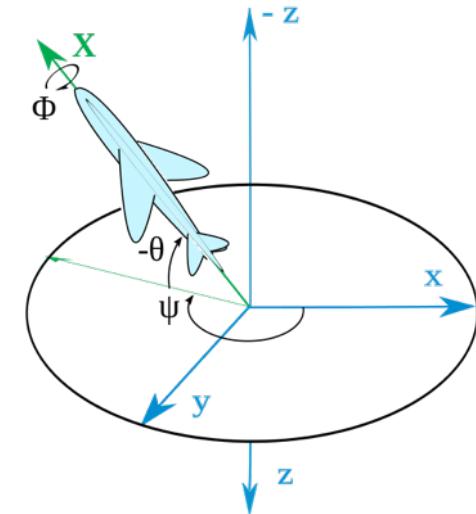
- Rendering-Loop
  - Eingaben verarbeiten (z.B. Mausbewegung)
  - Aktualisiere 2D-/3D-Szene (z.B. Kameraposition)
  - Zeichne Szene neu
- Per-Frame Update via *requestAnimationFrame*

```
(function animate() {
 updateScene();
 renderScene();
 requestAnimationFrame(animate);
})();
```



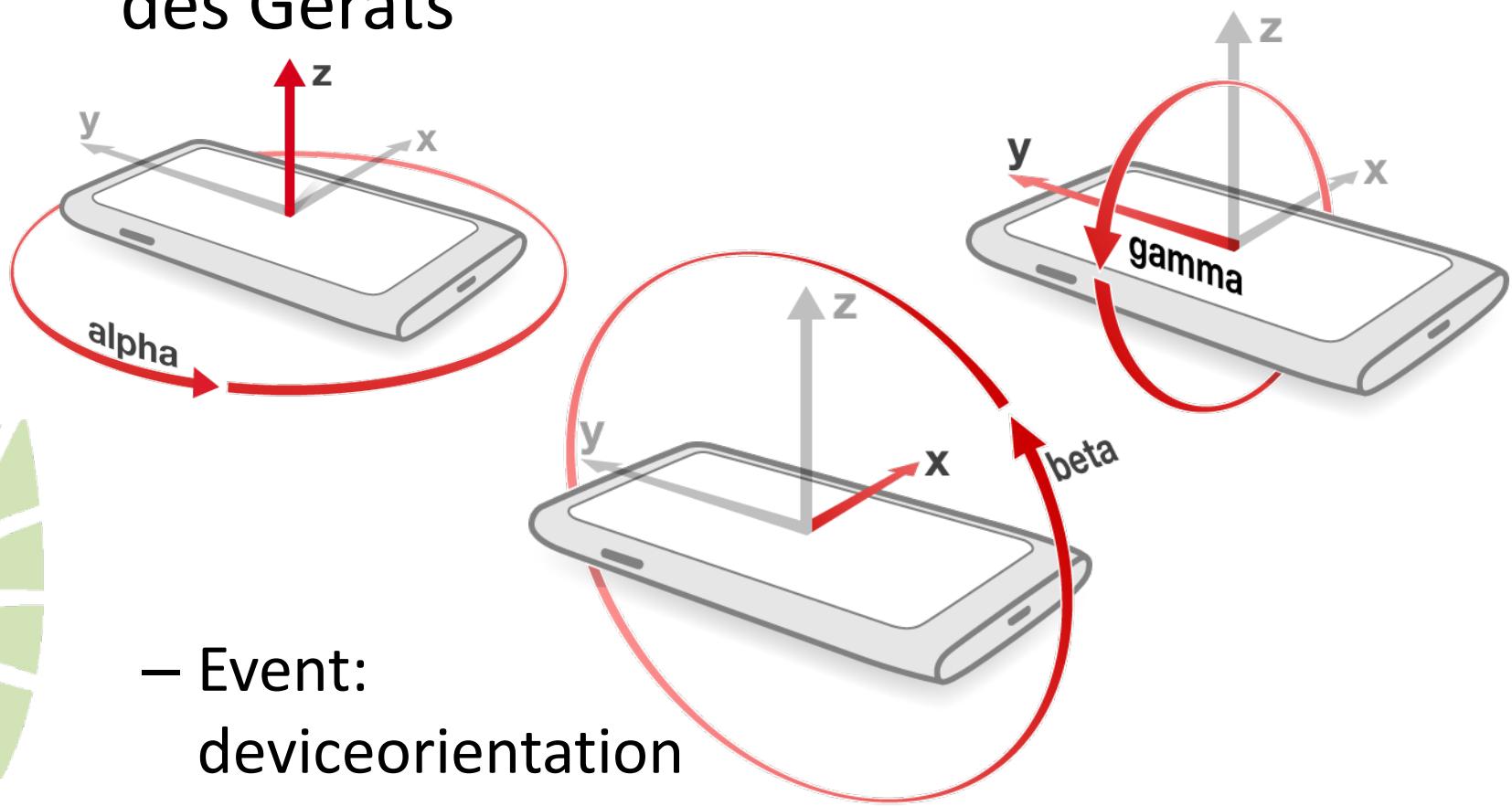
# Mobile Sensoren im Web

- Verschiedene Sensoren in mobilen Geräten
  - Accelerometer (Beschleunigungssensor)
  - Gyroskop (Lagesensor)
  - GPS
  - Magnetfeldsensor
  - Helligkeitssensoren
  - Kamera
  - ...
- Seit HTML5 fast alle in Webanwendungen nutzbar
  - Siehe z.B. [https://developer.mozilla.org/en-US/docs/Web/API/Detecting\\_device\\_orientation](https://developer.mozilla.org/en-US/docs/Web/API/Detecting_device_orientation)



# Gyroscope Basics

- Erfasst Drehungen um die 3 Raumachsen des Geräts



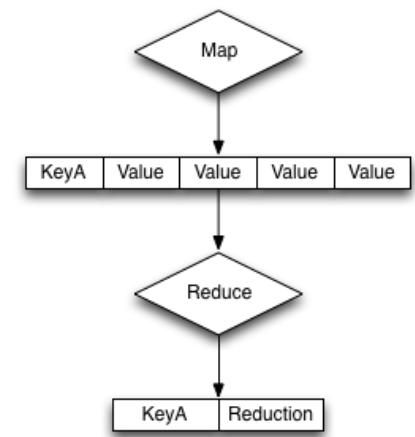
# Accelerometer Basics

- Erfasst Beschleunigung des Geräts an allen 3 Raumachsen
- An z-Achse wirkt immer auch Gravitation mit  $9,81 \text{ m/s}^2$ 
  - Daten liefert mittels Event-Objekt
    - acceleration.{x | y | z}
  - Event: devicemotion



```
// Go on access the sensors
if (window.DeviceMotionEvent) {
 window.addEventListener('devicemotion', devMotionHandler, false);
}

if (window.DeviceOrientationEvent) {
 window.addEventListener('deviceorientation', devOrientationHandler, false);
}
```



# Funktionale Aspekte



# Arrays revisited

---

- Konzepte funktionaler Programmierung
  - Funktionen sind *Objekte erster Klasse*, können also wie Objekte verwendet werden (z.B. Variablen zuweisen)
  - Operationen verändern Datenstrukturen nicht, sondern erzeugen jeweils neue Daten
- Funktionale Methoden
  - Map: gibt Feld mit gleicher Länge zurück

```
let arr = [1, 2, 3, 4];
const res = arr.map(x => x * x);
```
  - Filter: erlaubt es, Werte herauszufiltern
  - Reduce: gibt einen einzelnen Wert zurück
- Iterieren
  - Statt mit herkömmlicher *for*-Schleife bis *length* erreicht:  
`arr.forEach(function(currValue, index) { ... });`



# Closure (Funktionsabschluss)

---

- Objekt, das Funktion und Umgebung, in der Funktion erstellt wurde, verknüpft
  - Funktion, die Zugriff auf Scope ihrer umfassenden Funktion hat, selbst nachdem umschließende Funktion abgeschlossen wurde
  - Variablen innerhalb des Sichtbarkeitsbereichs einer äußeren Funktion können von innerer Funktion aus angesprochen werden
- Umgebung schließt alle lokalen Variablen ein, die zur Erstellungszeit sichtbar waren
  - Häufig verwendet als Callback/Eventhandler
  - Ermöglicht private Eigenschaften und Methoden
  - Vorteile: Datenkapselung, einschränkbarer Zugriff und sauberer (globaler) Namensraum
  - Zum Weiterlesen:  
<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures>



# Closures in Schleifen

---

- Aufgabe: Onclick-Handler, der Position im Array ausgibt
- Lösung:

```
var addHandlers = function(nodes) {
 for (var i = 0; i < nodes.length; i++) {
 nodes[i].onclick = (function(c) {
 return function(evt) { alert(c); };
 })(i);
 }
};
```

- Eventhandler-Funktionen sind an Variable i gebunden, nicht an Wert von i, als Funktion erzeugt wurde
- Zurückgegebene Funktion ist an Wert der übergebenen Variable i gebunden, nicht an Variable i selbst
  - Hat damit je separate Umgebung



# Singleton

---

- Stellt sicher, dass Klasse nur eine Instanz besitzt
  - Verwendbar, um zusammengehörigen Code zu gruppieren, auf den durch eine einzige Variable zugegriffen wird
  - In JavaScript nützlich als „Namespace“, um Anzahl globaler Variablen zu minimieren
- JS-Codegerüst mit Lazy Loading
  - Zugriff auf Singleton-Objekt über ‚statische‘ Methode *the()*

```
MyAPP.MySingleton = (function() {
 var theInstance = null;
 function createInstance() { /* some private construction code */ }
 return {
 the: function() {
 if (!theInstance) { theInstance = createInstance(); }
 return theInstance;
 }
 };
})(); // cause anonymous function to execute and return object
```



# Module Pattern

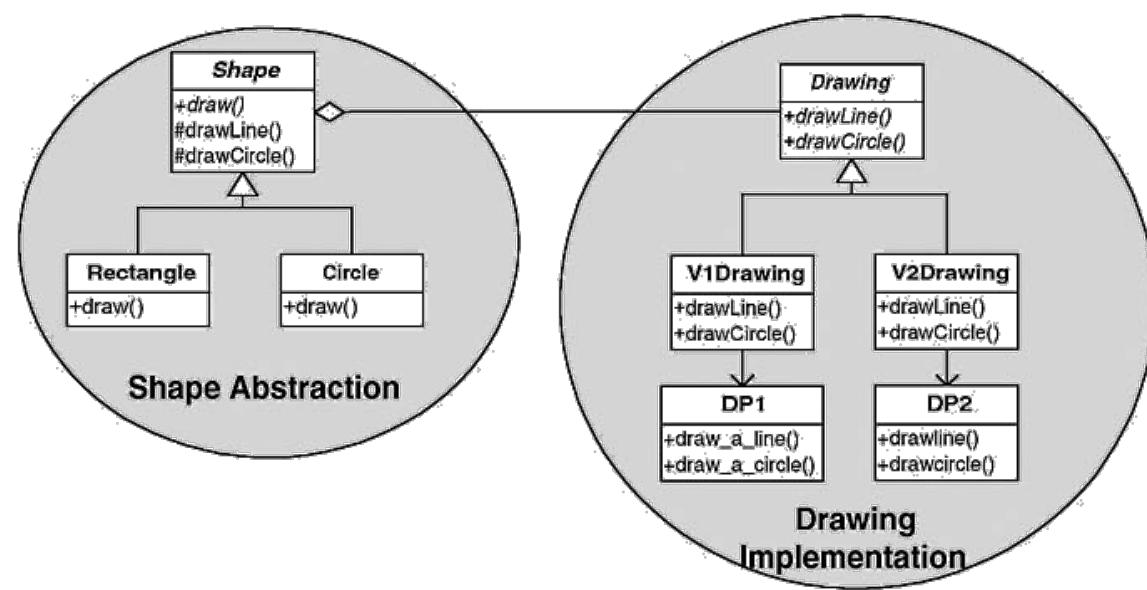
- Ein Modul ist ein Objekt, das Interface bereitstellt, aber internen Zustand (State) und Implementierung verbirgt
  - → Information Hiding statt globaler Variablen
- Beispiel: `var MYAPP = {};`



Private Section {

```
MyAPP.myModule = (function() {
 var myPrivateVar = "Private variable can be accessed only from within myModule";
 var myPrivateMethod = function () {
 console.log("private method can be accessed only from within myModule");
 };
 return {
 myPublicProperty: "Variable accessible as myModule.myPublicProperty",
 myPublicMethod: function () {
 console.log("Method accessible as myModule.myPublicMethod()");
 console.log(this.myPublicProperty); // Access "public" members using this
 // In the following, we access "private" methods and variables
 myPrivateMethod();
 console.log(myPrivateVar);
 }
 };
})(); // Cause anonymous function to execute and return object
```

Public Section }



# Objektorientiertes JavaScript



# Wdh. Objekte

---

- Objektdefinition über sog. Objektliteral

```
var person = {
 name: "Mustermann",
 vorname: "Max",
 alter: 21
};
```

- Leeres Objekt anlegen: var obj = {};

- Haben Eigenschaften und Methoden

- Bsp. für Eigenschaft (property): person.name = "Moritz";

- Bsp. für Methode (method):

- ```
person.greet = function() { alert("Hello!"); };
```

- Memberzugriff mit Punkt- oder Klammernotation:

- ```
person["name"] = "Moritz";
```

- ```
person.greet() oder alternativ: person["greet"]()
```

- Objekt an beliebige Funktion binden: someFunc.**bind**(obj)



Setter und Getter definieren

- Variante 1 (Definition direkt bei Erzeugung)

```
var obj = { a: 7,
            get b() { return this.a + 1; },
            set c(x) { this.a = x / 2; }
        };
console.log(obj.a); // 7
console.log(obj.b); // 8
obj.c = 50;
console.log(obj.a); // 25
```

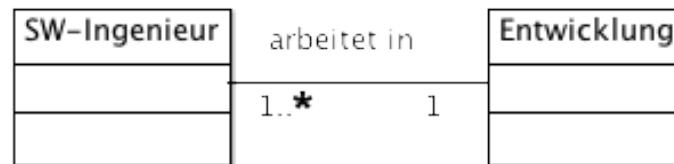
- Variante 2 (Definition erst nach Erzeugung)

```
var obj = { a: 7 };
Object.defineProperties(obj, {
    b: { get: function() { return this.a + 1; } },
    c: { set: function(x) { this.a = x / 2; } }
});
obj.c = 10;           // a := 5
console.log(obj.b); // 6
```

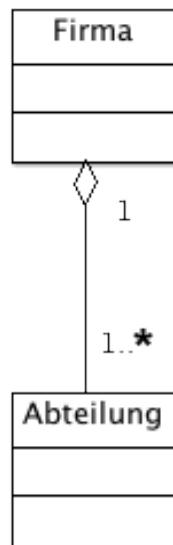


Wdh. Klassendiagramme in UML

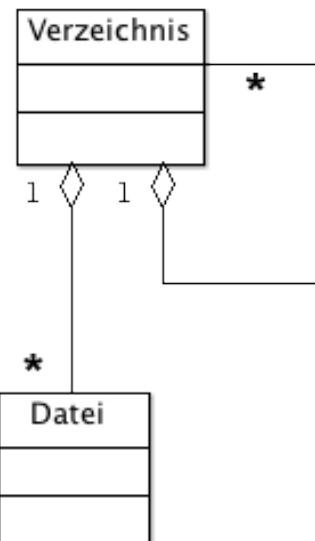
Assoziation:



Aggregation:



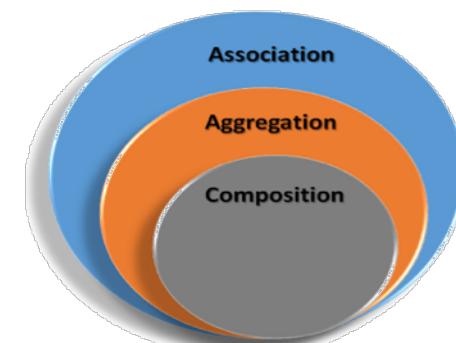
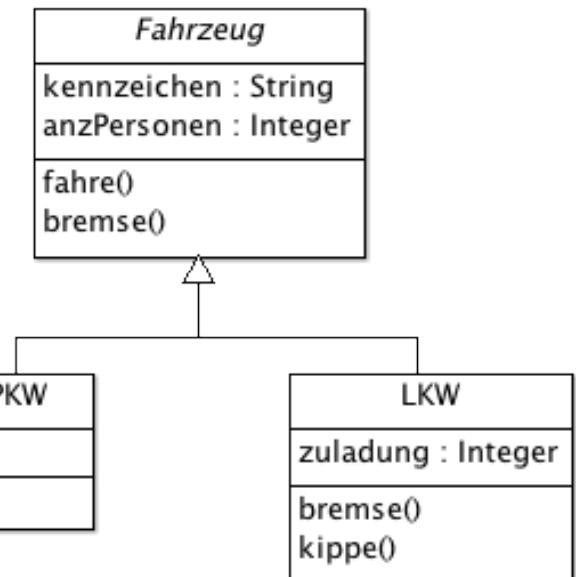
reflexive Aggregation:



Komposition:



Vererbung:



Konstruktorfunktion und Klassen

- Beispiel:

```
var Hund = function(name, besitzer) {  
    this._name = name;  
    this._besitzer = besitzer;  
};  
Hund.prototype.bellen = function() {  
    console.log(this._name + „ sagt: Wau wau!“);  
};  
var waldi = new Hund(„Waldi“, „Krause“);  
waldi.bellen();
```

Klasse Hund

- Memberzugriff erfolgt immer mit „this“!
 - Leider kein „private“, daher Konvention: `_privateVariable`
- Methoden in „prototype“ Eigenschaft definieren
 - Werden damit für Objekte gleichen Typs geshared!
 - Exist. Member in Instanz nicht, sucht JS im Prototype



...this and that

- Objekte zur Laufzeit erweiterbar

```
Hund.prototype.getClub = function() {  
    return this._club;  
};
```

– Alternativ nur eine Instanz erweitern

```
waldi.getClub = function() {  
    return this._club;  
};  
waldi.newParam = "Quick and dirty...";
```

Eigenschaft löschen:
`delete waldi.newParam;`

- Achtung: bei Funktionen innerhalb einer Objektmethode verweist **this** nicht auf Objekt selbst sondern auf *window*

```
Hund.prototype.doSomething = function() {  
    var that = this;  
    function inner() {  
        var myName = that._name;  
        //...  
    }  
    //...  
};
```

Eigenschaft enthalten?
`'newParam' in waldi`



Einschub: ECMAScript 6

- Klassen bisher:

```
var Shape = function (id, x, y) {  
    this.id = id;  
    this.move(x, y);  
};  
  
Shape.prototype.move = function (x, y) {  
    this.x = x;  
    this.y = y;  
};
```

- Jetzt neu:

```
class Shape {  
    constructor (id, x, y) {  
        this.id = id;  
        this.move(x, y);  
    }  
    move (x, y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```



Klasse erstellen

- Mit Schlüsselwort *class*
 - Intern trotzdem prototypische Objektorientierung
 - → Objekte werden von Prototyp-Objekt geclont
- Funktion *constructor* wird beim Erstellen eines Objekts zur Initialisierung aufgerufen
- Zugriff auf Variablen und Methoden mit *this*
- Beispiel

```
class Haustier{  
    constructor(name) {  
        this.name = name; // initialer Wert  
    }  
    hinlegen () {  
        return this.name + ' liegt auf Teppich';  
    }  
}
```



Instanzen und statische Methoden

- Neues Objekt der Klasse mit *new* erstellen
 - Beispiel

```
let waldi = new Haustier('Waldi');
let bello = new Haustier('Bello');
```
 - Auf Eigenschaften und Methoden mit Punkt zugreifen

```
console.log(waldi.name);
console.log(waldi.hinlegen());
```
- Klassenmethoden mit *static*
 - Oft für Helper-Funktionen – Beispiel:

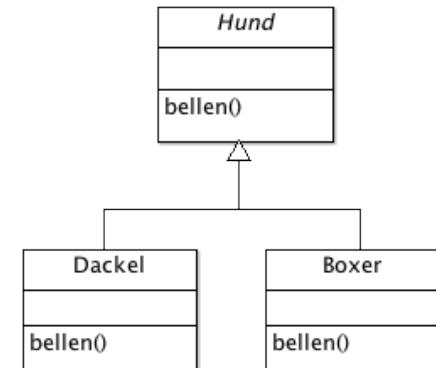
```
class MatheHelper {
    static add(a, b) {
        return a + b;
    }
}
const sum = MatheHelper.add(2, 4);
```



Vererbung (ES5)

- Prototypische Vererbung
 - Objekte erben Eigenschaften und Methoden von Prototyp über sog. Prototypenkette
- Ableitung über Prototypenkette
 - Konstruktorfunktion schreiben
 - Evtl. Parent-Konstruktor aufrufen

```
var Dackel = function(name, besitzer, club) {...};
```
 - Setup der Prototypenkette
 - Vererbung, indem man *prototype* der abgeleiteten Klasse auf Instanz der Parent-Klasse setzt
 - Dann sollte *constructor* Attribut zurückgesetzt werden auf Konstruktor der abgeleiteten Klasse
 - instanceof Operator überprüft Prototypenkette



Vererbung (ES5)

- Konstruktorfunktion schreiben

```
var Dackel = function(name, besitzer, club) {  
    Hund.call(this, name, besitzer);  
    this._club = club;  
};
```

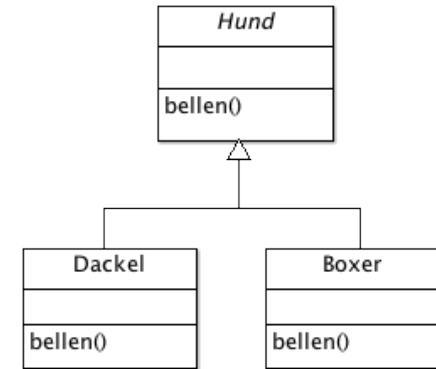
- Setup der Prototypenkette

```
Dackel.prototype = Object.create(Hund.prototype);  
Dackel.prototype.constructor = Dackel;
```

```
Dackel.prototype.getSpruch = function() {  
    console.log(this._besitzer + " sagt: Ordnung muss sein!");  
};
```

- Instanz erzeugen

```
var waldi = new Dackel("Waldi", "Krause", "Teckel e.V.");  
waldi.getSpruch();
```



früher:

Dackel.prototype = new Hund();

Ableitungsprozess wrappen

```
function extend(subClass, superClass) {  
    var F = function() {};  
    F.prototype = superClass.prototype;  
    subClass.prototype = new F();  
    subClass.prototype.constructor = subClass;  
    subClass.parentClass = superClass.prototype;  
}
```

```
subClass.prototype =  
Object.create(F.prototype);
```

Leere Klasse **F** verhindert, dass – bei historischer Vorgehensweise mit *new* – unnötig neue Instanz der Parent-Klasse erzeugt wird
(→ Wegwerf-Instanz)

- Anwendungsbeispiel

```
var Hund = function(name, besitzer) { /* Implementierung */ };
```

```
var Boxer = function(name, besitzer, bissig) {  
    Boxer.parentClass.constructor.call(this, name, besitzer);  
    this._bissig = bissig;  
};  
extend(Boxer, Hund);
```

Hund

```
var nero = new Boxer("Nero", "Meier", true);
```



Einschub: ECMAScript 6

- Vererbung bisher:

```
var Rectangle = function (id, x, y, width, height) {  
    Shape.call(this, id, x, y);  
    this.width = width;  
    this.height = height;  
};  
Rectangle.prototype = Object.create(Shape.prototype);  
Rectangle.prototype.constructor = Rectangle;
```

- Jetzt neu:

```
class Rectangle extends Shape {  
    constructor (id, x, y, width, height) {  
        super(id, x, y);  
        this.width = width;  
        this.height = height;  
    }  
}
```



Vererbung

- Klasse ableiten mit Schlüsselwort *extends*
 - Klasse Hund hat alle Eigenschaften u. Methoden von Klasse Haustier und kann weitere hinzufügen
- Mit *super()* Konstruktor der Parentklasse aufrufen
 - Zugriff auf Variablen und Methoden von Haustier

```
class Hund extends Haustier{  
    constructor(name) {  
        super(name);  
        // ...  
    }  
    bellen() {  
        // Zugriff auf name (von Haustier geerbt)  
        return this.name + ' macht "Wau Wau"';  
    }  
}
```



Setter + Getter revisited

- Methoden zum kontrollierten Zugriff auf Attribute
 - Werden wie Variablen verwendet (ohne Klammern)
 - Beispiel:

```
class Haustier {  
    constructor() {  
        this._fellLaenge = 3;  
    }  
  
    get fellLaenge() {  
        return this._fellLaenge;  
    }  
  
    set fellLaenge(len) {  
        if ("some condition that must be met") {  
            this._fellLaenge = len;  
        }  
    }  
}  
  
let waldi = new Haustier();  
console.log('Felllänge: ' + waldi.fellLaenge);  
waldi.fellLaenge = -1; // sollte im Setter abgefangen werden
```

Private Eigenschaften mit Closures

- Problem: privat gedachter Member `_fellLaenge` kann trotz `set/get` von außen benutzt werden
 - Man muss sich auf guten Willen/Wissen des Entwicklers verlassen
- Lösung: Variablen mit Closure kapseln

```
constructor() {
    let fellLaenge = 3; // private

    this.getFallLaenge = function() {
        return fellLaenge;
    };
    this.setFallLaenge = function(len) {
        fellLaenge = len;
    };
}
```

- Variable `fellLaenge` nur im Konstruktor-Scope definiert
 - Nach Ausführen des Konstruktors kein Zugriff von außen möglich
 - Somit kann `fellLaenge` nur innerhalb genutzt werden
- Methoden `get-/setFallLaenge` können weiter darauf zugreifen
 - Zur Laufzeit im C'tor definiert → gültige Referenz auf `fellLaenge`
 - Sind public und können von außen genutzt werden



Interfaces

- Interface definiert vorhandene Methoden
 - Alle Objekte mit diesem Interface können gleich behandelt werden
- Aber: in JS gibt es weder Interfaces wie in Java noch Mehrfachvererbung wie in C++
 - TypeScript z.B. hat Interfaces und kann zu JS umgewandelt werden
- Emulieren von Interfaces, z.B. via Duck Typing
 - Objekt kann als Instanz einer Klasse aufgefasst werden, wenn es die Methoden implementiert
 - Reflection erlaubt es, geg. Objekt auf vorhandene Methoden und Attribute zu überprüfen
 - Zusätzliche Kommentare zur Dokumentation!



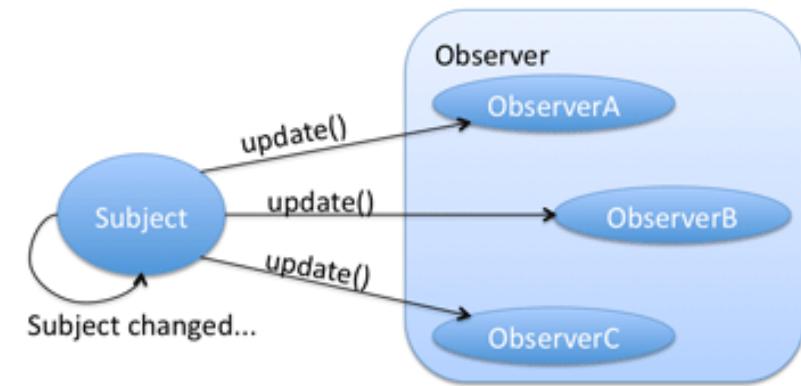
Interfaces und Introspection

```
// Our "Summable" interface
var Summable = [ 'add', 'sub' ];

var obj = {
    // Implements Summable
    add: function(other) { this.value += other.value; },
    sub: function(other) { this.value -= other.value; }
    // Here some more methods and properties
};

for (var i = 0; i < Summable.length; i++) {
    var f = obj[Summable[i]];
    if (!f || typeof f !== "function") {
        console.warn('No method "' + Summable[i] + '" found');
        break;
}
}
```





Exkurs: Observer Pattern



Entwurfsmuster

- Bewährte Lösungsschablonen für wiederkehrende Entwurfsprobleme
- Beschreibung kommunizierender Objekte u. Klassen
 - Wurden angepasst, um allgemeines Entwurfsproblem in bestimmtem Kontext zu lösen
 - Abstraktionen, die über der Ebene einzelner Klassen liegen
- Entwurfsprinzipien
 - OO-Basics: Abstraktion, Trennung von Zuständigkeiten
 - Kapseln veränderbarer Teile der Anwendung
 - Aggregation statt immer nur Vererbung!
 - Auf Schnittstelle programmieren, nicht auf Implementierung
 - Lose Kopplung zwischen und starke Bindung in Modulen



Entwurfsmuster Observer

Observer

Type: Behavioral

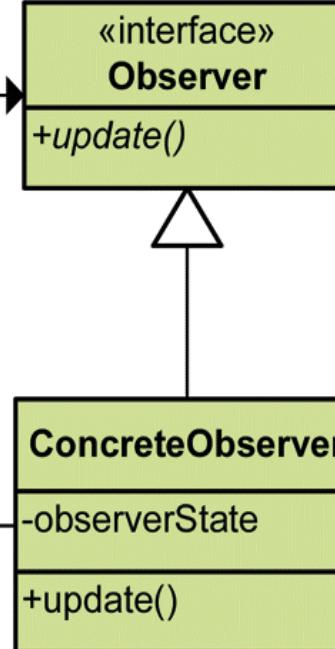
GUI-Komponente
(z.B. Button, informiert
registrierte Observer)

What it is:

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

notifies

observes



EventListener
(Registriert sich bei Subject)



Observer-Pattern

- Definiert 1-zu-n-Abhangigkeit zw. Objekten, so dass Zustandsanderung eines Objekts zur Benachrichtigung aller abhangigen Objekte fuhrt
- Objekte werden uber Zustandsanderungen informiert
 - Moglichst geringe Kopplung zwischen beteiligten Objekten
 - Objekt kennt seine Beobachter nicht
 - Jeder Beobachter entscheidet selbst, wie er mit Informationen umgeht
 - Anwendungsbeispiel
 - Zu Objekt gibt es Views, die uber nderungen informiert werden
 - Ubernehmen Rolle des Beobachters (Observer)
 - Objekt selbst (Observable bzw. Subject) wird beobachtet
- Beobachtetes Objekt hat folgende Methoden
 - attach: fuge Beobachter hinzu
 - detach: entferne Beobachter
 - notify: benachrichtige Beobachter (→ Broadcast)
 - getState: erfrage Objektzustand
 - setState: setze Objektzustand
- Beobachtendes Objekt hat Aktualisierungsmethode: update



Umsetzung in JavaScript

- Wie kann man Observer-Pattern konkret nutzen?
 - Wie können Objekte (z.B. Formularelemente) überwacht werden?
 - Wie kann man Registrieren von Beobachtern implementieren?
 - Wie können Ereignisse an Beobachter gemeldet werden?



```
var subject = new Subject();
var observer = new Observer();

subject.attach(observer);
subject.setState(42);

var Observer = function() {
    this.update = function (evt) {
        document.write("New value is " + evt.value + "<br>");
    };
};
```

Umsetzung in JavaScript

```
var Subject = function() {
    this.state = { value: 0 };      // simple example
    this.observers = [];           // holds observer

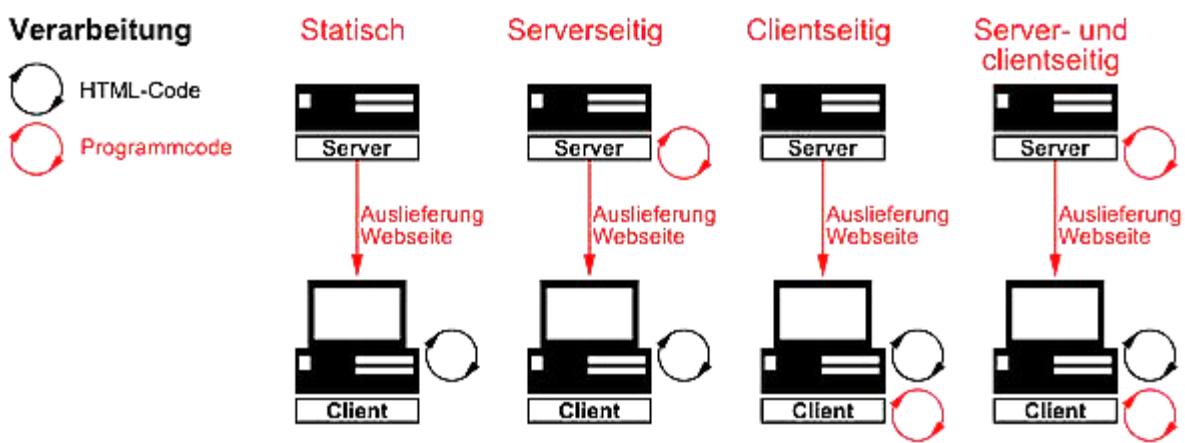
    this.setState = function (value) {
        this.state.value = value;
        this.notify();
    };

    this.notify = function () {
        for (var i=this.observers.length-1; i>=0; i--) {
            if (this.observers[i].update)
                this.observers[i].update(this.state);
            } else if (typeof(this.observers[i]) == "function") {
                this.observers[i].call(null, this.state);
            }
        };

    this.attach = function (observer) {
        this.observers.push(observer);
    };

    this.detach = function (observer) {
        for (var i=this.observers.length-1; i>=0; i--) {
            if (this.observers[i] === observer)
                this.observers.splice(i, 1);
        }
    };
};
```





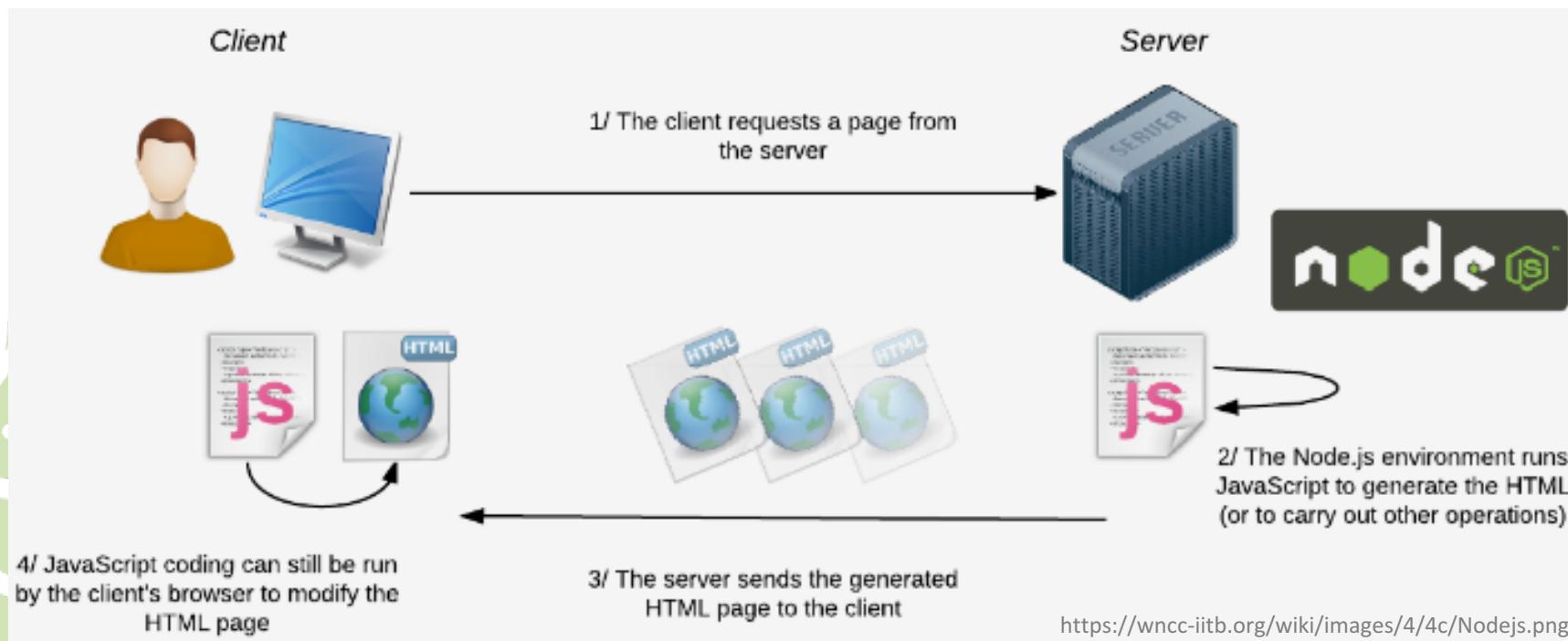
Serverseitige Programmierung

Teil 1: Node.js



Node.js

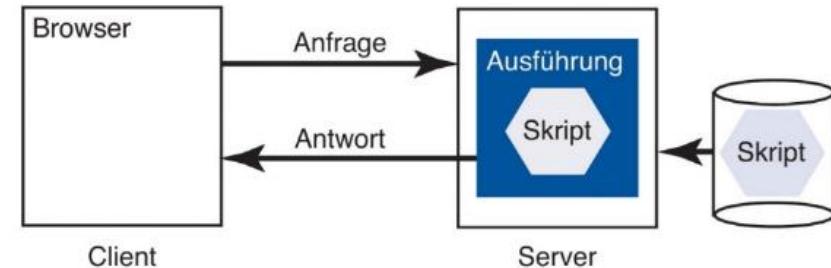
- Zur Realisierung performanter Webserver
 - JavaScript-Laufzeitumgebung
 - Basiert auf V8 JavaScript Engine von Chrome
- Download: <https://nodejs.org/>



Node.js



- Backend-Code wird in JS geschrieben
 - Läuft auf Server statt im Client
 - Wird dynamisch bei Anfrage ausgeführt und liefert Ergebnisseite zurück
 - Kann beliebige Formate liefern (HTML, JSON, Text...)
- Voller Zugriff auf Datenbanken und Dateien
 - Wichtiges Anwendungsfeld: serverseitige Verarbeitung von Formulardaten
 - Ermöglicht Persistenz
 - Datenbank muss jedoch separat installiert werden



Getting started...

- Startdatei erstellen
 - Z.B. *myapp.js* mit folgendem Inhalt

```
const http = require('http');
const server = http.createServer();
server.on('request',
  function(req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.write("Hello World!");
    res.end();
  }
);
server.listen(8080);
```

- Gehe in Konsole in Projektordner
 - Starte dort Server mit **node myapp.js**
- Aufruf im Browser: *http://localhost:8080/*



Header & CORS

- Header setzen

```
res.writeHead(200, { 'Content-Type': 'text/html' }) ;
```

- Content-Type

- Besteht aus Charset (z.B. UTF-8) und MIME-Type
- Bsp.: `text/html`, `text/plain`, `application/json`, `image/jpeg`, `image/png`, `audio/mpeg`, ...

- Mehrere Header setzen

```
res.writeHead(200, {  
  'Content-Type': 'application/json',  
  'Access-Control-Allow-Origin': '*'  
}) ;
```

- Access-Control-Allow-Origin

- Erlaube Cross-Origin Resource Sharing (CORS)



Module

- Eingebaute Module mit *require()* einbinden
 - Stellen so zusätzliche Funktionalitäten bereit
 - Beispiele

```
const http = require('http'); // HTTP-Server  
const url = require('url'); // URL-Objekt  
const fs = require('fs'); // File System
```

- NPM: Node.js Package Manager
 - Verwaltung von JavaScript-Bibliotheken (Module)
 - Einfacher Zugriff auf große Modul-Registry
 - Installieren: npm install [-g] *modName*



Anmerkungen für Entwicklung

- Konsolenausgabe (`console.log()`) erscheint hierbei auf Serverkonsole
- Hinweis: wenn man Inhalte ändert, muss Server neu gestartet werden!
 - Ggf. besetzten Port freigeben
- Auto-Restart des Servers mit *Nodemon*
 - Über NPM Modul nachinstallieren
 - `npm install -g nodemon`
 - Datei beim Entwickeln damit ausführen
 - `nodemon myapp.js`
 - Nach Änderung nur Reload der Webseite nötig



Formulare verarbeiten (GET)

- URL parsen, um Query-Parameter zu lesen
 - Query-Parameter als String: `q.search`
 - Parameter als JS-Objekt: `q.query`
- Beispiel

```
server.on('request', function(req, res) {  
  const q = url.parse(req.url, true);  
  q.query // Queryobjekt: {param1: val1, ...}  
  q.query.param1 // Zugriff auf Query-Parameter  
  ... // mache etwas mit Formulardaten  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end(JSON.stringify(q.query));  
});
```



Formulare verarbeiten (POST)

```
const queryString = require('querystring');

...
server.on('request', function(req, res) {
  if (req.method === 'POST') {
    let body = "";
    req.on('data', function(chunk) { body += chunk.toString(); });
    req.on('end', function() {
      console.log(queryString.parse(body));
      res.end('ok');
    });
  } else {
    res.end(`<!doctype html>
      <html><body><form action="/" method="post">
        <input type="text" name="uname"><br>
        <button>Senden</button>
      </form></body></html>`);
  }
});
```



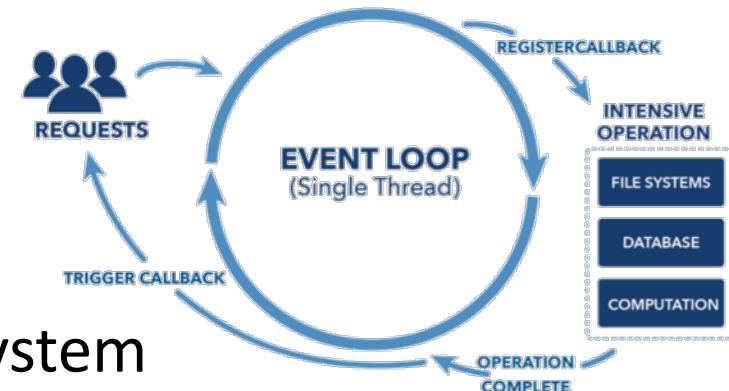
Node.js vs. herkömmliche Server

- Herkömmliche Server (z.B. Apache, IIS) sind Multi-Threaded Server
 - Für jeden HTTP-Request wird neuer Thread zum Verarbeiten genommen
 - Anfrage meist mit blockierender I/O verbunden (z.B. Dateizugriff, Datenbankabfrage)
 - Thread wartet bis dahin auf Ergebnis, braucht solange Speicherplatz
- Node.js ist Single-Threaded
 - Verwaltet Anfragen in Request-Queue
 - Falls Anfrage mit blockierender I/O verbunden, wird parallel Worker-Thread aufgerufen
 - Hauptthread kann währenddessen weiterlaufen und weitere Anfragen bearbeiten



Files lesen und schreiben

- In PHP oder ASP
 - Sende Aufgabe an Dateisystem
 - Warte auf Antwort
 - Gebe Inhalt zurück
 - Nächster Befehl
- In Node.js
 - Sende Aufgabe an Dateisystem
 - Bearbeite nächste Befehle
 - Wenn Dateisystem soweit ist, gebe Inhalt zurück
 - → Nicht-blockierende I/O



Files lesen

- Bestehende Datei asynchron lesen
 - `fs.readFile(file, callbackFunc);`

- Beispiel

```
let fs = require('fs');
server.on('request', function (req, res) {
  fs.readFile('document.html',
    function (err, data) {
      res.writeHead(200, {'Content-Type': 'text/html'});
      res.write(data);
      res.end();
    }
  );
}) ;
```

- Bestehende Datei synchron lesen (blockierend ☹)
 - `let txt = fs.readFileSync(file);`



Files schreiben

- An Datei anfügen (bzw. anlegen, falls nötig)

```
let fs = require('fs');
fs.appendFile(file, content, callback);
```

- Beispiel

```
server.on('request', function(req, res) {
  fs.appendFile('newFile.txt', 'Hello content!\n',
    function(err) {
      if (err) {throw err;};
      res.writeHead(200, {'Content-Type': 'text/plain'});
      res.end('Datei erfolgreich geschrieben.');
    });
});
```

- Inhalt ersetzen (bzw. neu anlegen, falls nötig)

```
fs.writeFile(file, content, callback);
```



URL Routing

- Basierend auf angefragter HTTP-Methode, URL, Parameter usw. erzeugt Server je andere Antwort
 - Dazu werden sog. Routen definiert
 - Enthalten Logik zum Anzeigen verschiedener Inhalte
 - Z.B. URL / für Startseite oder /contact für Kontaktinfos
 - Jeweils andere Dateien laden oder Datenbankinhalte anzeigen
- Einfaches Beispiel

```
const pfad = url.parse(req.url, true).pathname;
switch(pfad) {
  case '/': res.write('Hallo Startseite!'); break;
  case '/contact': res.write('Kontaktinformation'); break;
  default: res.write(pfad); break;
}
```
- Für komplexe Apps besser Webframeworks nutzen

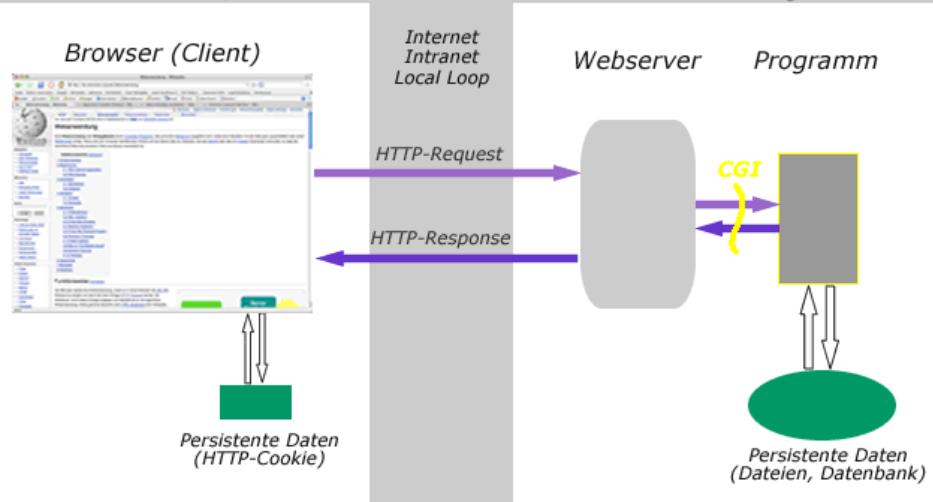


Webserver: HTML vs. API

- Webserver können HTML Seiten ausliefern
 - Normalweise benötigt jeder Request Refresh ☹
 - Anzeigelogik auf Server ☹
 - Anfangs ggf. etwas leichter aufzusetzen ☺
- Webserver können auch APIs ausliefern
 - Werden vom Frontend konsumiert

```
result.setHeader('Content-Type', 'application/json');
result.setHeader('Access-Control-Allow-Origin', '*');
result.end(data);
```
 - Asynchron, damit sehr schnelle Ladezeiten ☺
 - Oft sauberer, weil Komponenten klar getrennt sind ☺
- Unabhängig davon Elemente wie Navigation, Header und Footer inkludieren
 - Wartungsfreundlich ☺



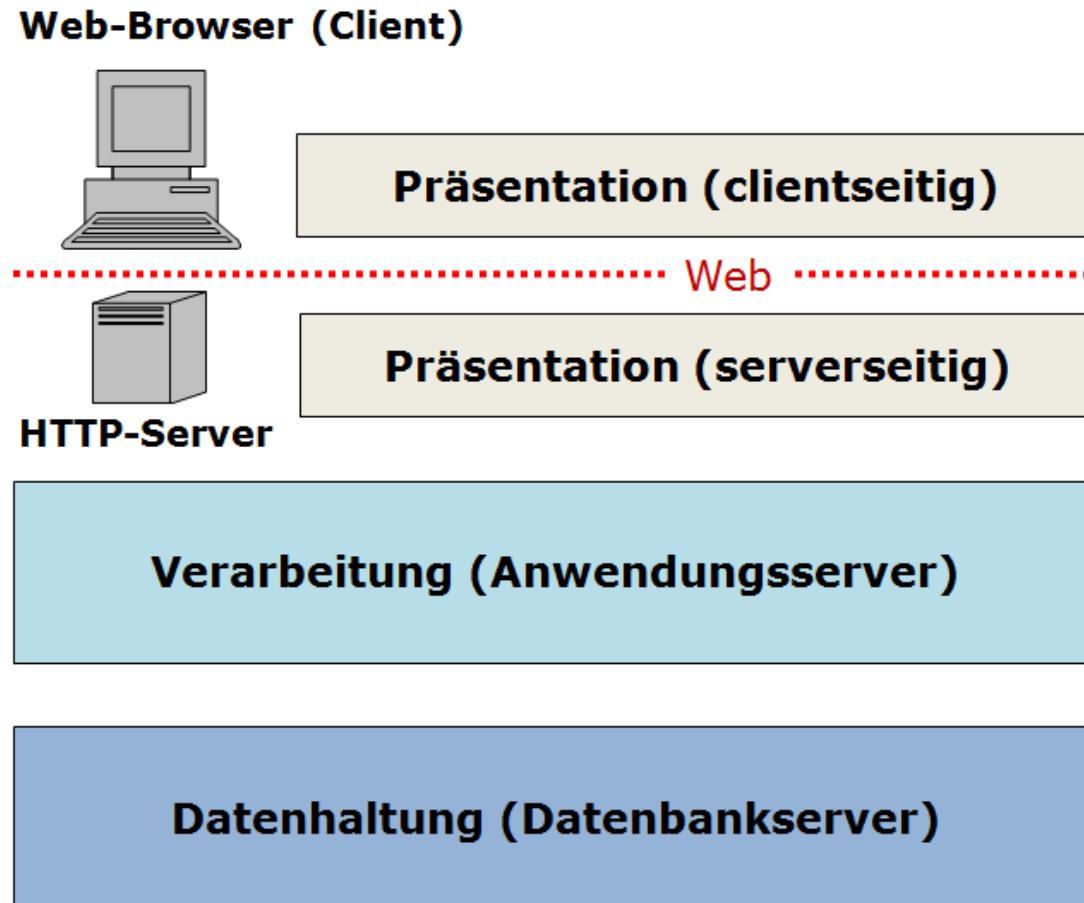


Webarchitekturen



Schichtenarchitektur

- Typische Architektur von Anwendungssystemen



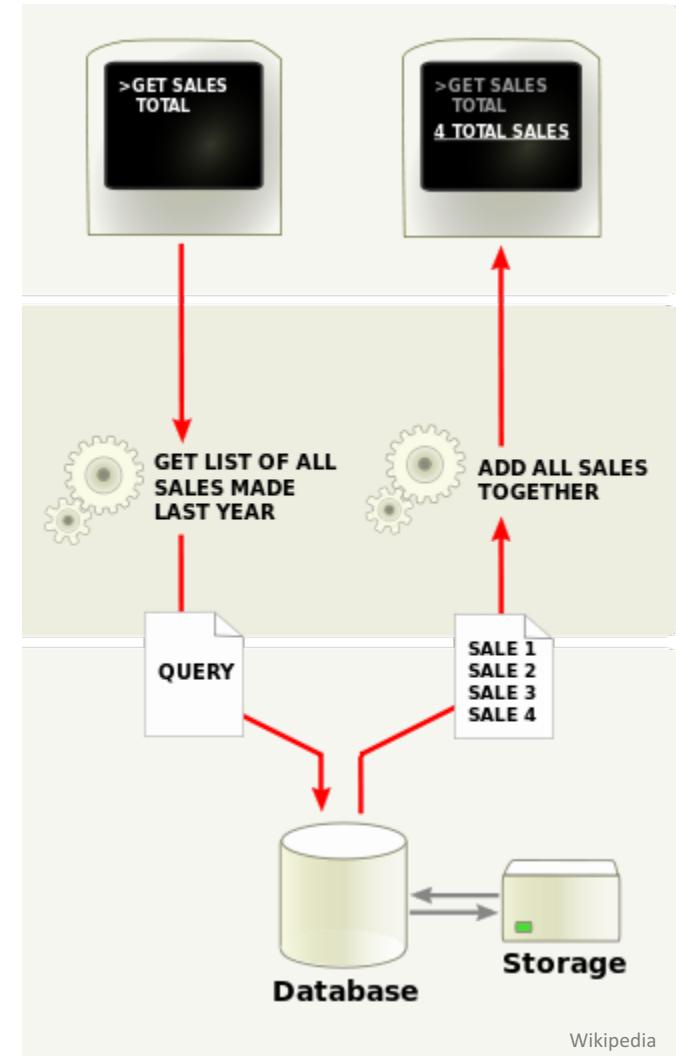
Schichtenarchitektur (*n*-tier)

- Unterteilung in aufeinander aufbauende Schichten
 - Abhängigkeiten eingeschränkt
 - Nur höhere Schichten dürfen auf tiefere zugreifen
 - Kommunikation zw. Ebenen über definierte Interfaces/ APIs
- 3-Schichten-Architektur
 - Präsentation
 - UI, Frontend, Client
 - HTML/CSS, JavaScript, Flash
 - Anwendungslogik
 - Geschäftslogik, Domänenschicht
 - Backend, Web-/Application-Server
 - Servlets, JSP, ASP, PHP, Node.js
 - Datenhaltung
 - Datenbank (MySQL, CouchDB etc.)



Client

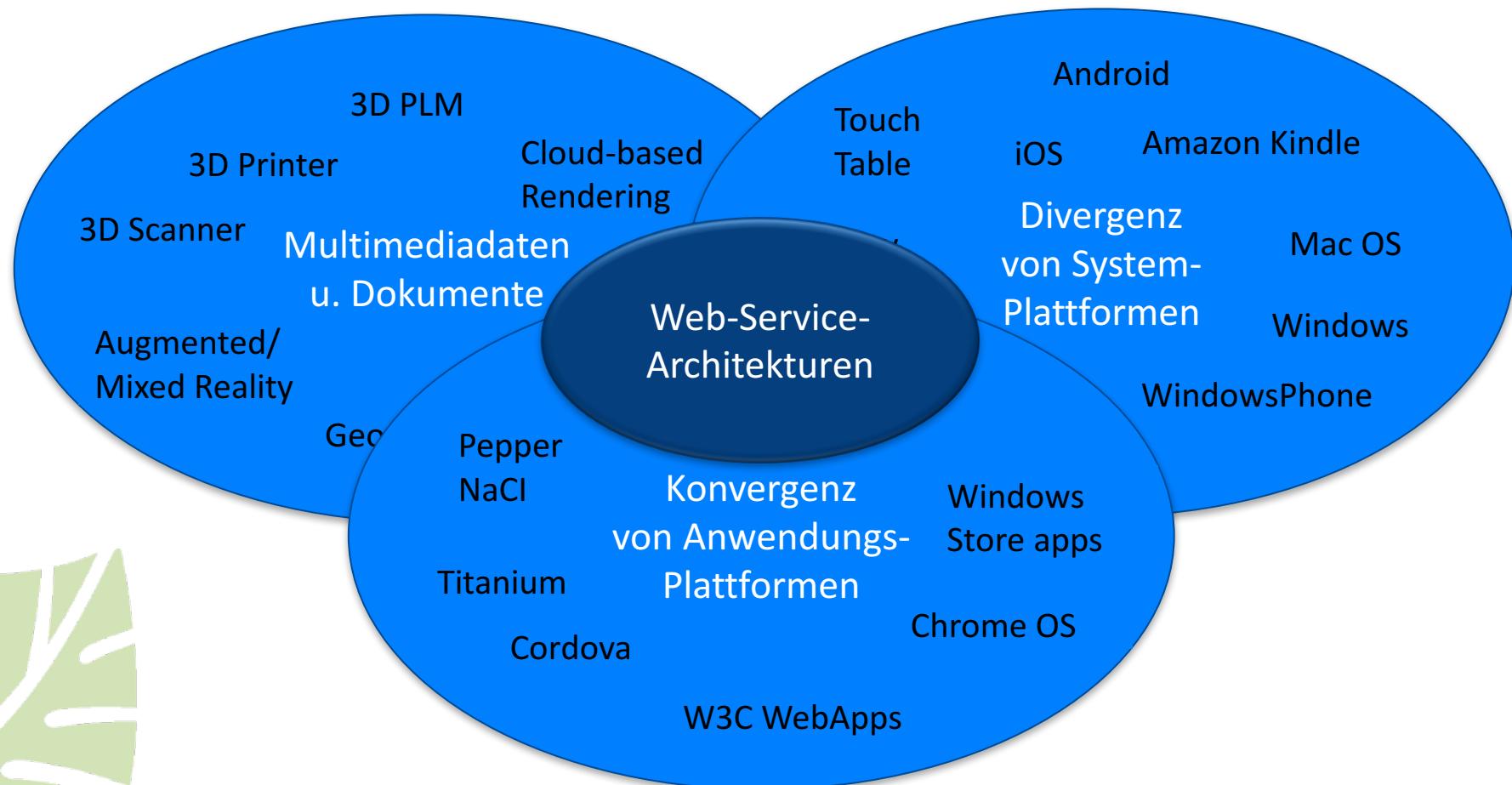
Server



Wikipedia



Technologie-Trends im Web



Service-Oriented Architecture: klar getrennte Komponenten, die über definierte Interfaces und verschiedene Protokolle kommunizieren

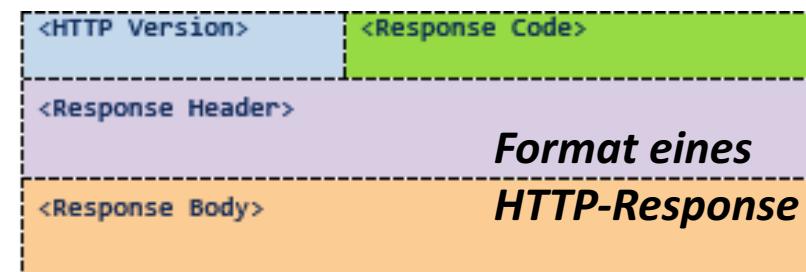
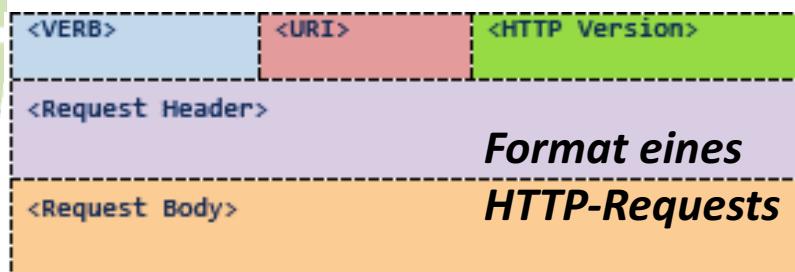
Webdienste

- Webservice
 - Dienst, den Web-Apps nutzen können
 - Webserver stellt Informationen in strukturierten Format zur Verfügung
 - Nicht primär zur direkten Anzeige gedacht ist, sondern für Maschine-Maschine-Kommunikation
- Microservice
 - Kapselt Teile der Anwendungslogik
 - Modularer Softwareaufbau
 - Dienste weitgehend entkoppelt
 - Kommunizieren über sprachunabhängige Schnittstellen
 - Geben oft JSON (oder XML) zurück



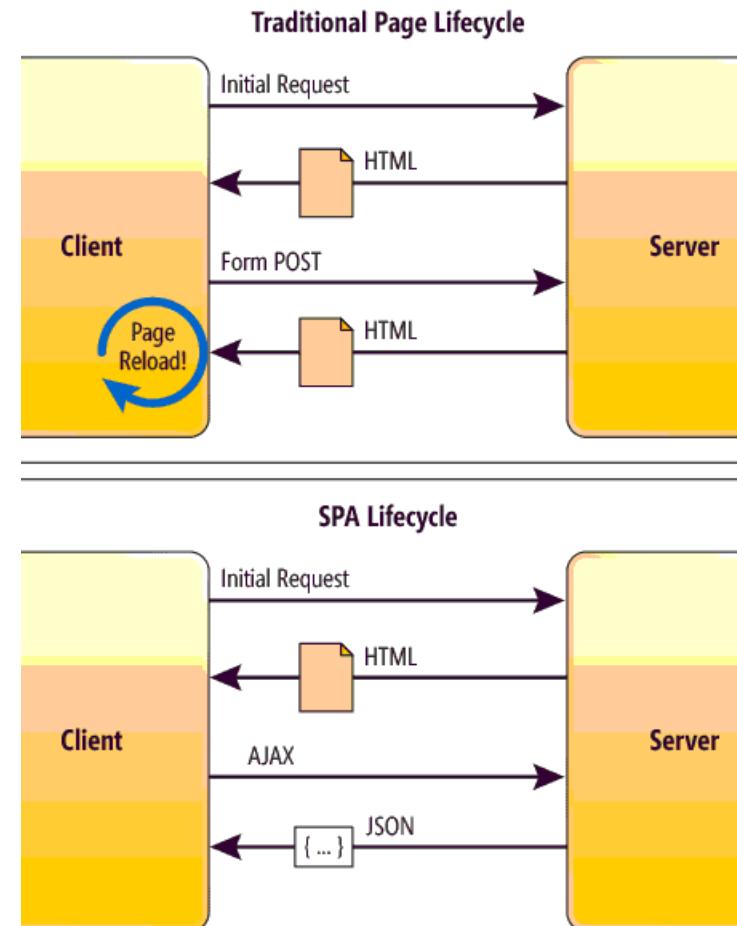
REST (Representational State Transfer)

- Moderner Webservice, bei dem Aktionen durch passende HTTP-Methoden ausgelöst werden
 - GET (zum Lesen), PUT (Ressourcen erstellen), POST (erstellen oder bearbeiten), DELETE (löschen)
 - CRUD-Operationen (d.h. create, read, update, delete) entsprechen HTTP-Methoden
 - Zustandslos: Jede Anfrage beinhaltet alle nötigen Infos
 - Ergebnisse auf Server cache-bar, um Anfragen zu beschleunigen
 - REST-konforme Dienste haben eindeutige Adresse
 - Alle Ressourcen identifizierbar über eindeutige URI
 - Verzeichnisartige URI-Struktur mit festen Regeln
 - Bsp.: <http://www.service.org/discussion/{year}/{month}/{topic}>



Single Page Applications (SPA)

- Usually uses REST APIs combined with JSON
- Only modified areas/ data asynchronously loaded (e.g., content area but not footer)
 - Requires JS to be enabled
 - Less bandwidth and very fast loading time
- Component-based software design
 - Each area as component
 - With scoped CSS rules
- Example frameworks
 - React, Riot, Vue



Model-View-Controller Architektur

- Zur Verhaltensmodellierung direkt-manipulativer Interfaces über kooperierende Interaktionsobjekte
- Strukturierung von Mensch-Computer-Interaktionen durch strikte Trennung von Zuständigkeiten
 - Datenhaltung, GUI, Programmlogik
- Ziel ist lose Kopplung der Objekte
 - Jede Komponente kann getrennt entwickelt, getestet und ggfs. im Gesamtsystem ausgetauscht werden
- Meist findet Observer-Pattern Anwendung
 - Observer registriert sich beim zu beobachtenden Objekt
 - Beobachtetes Objekt informiert Beobachter bei Änderungen

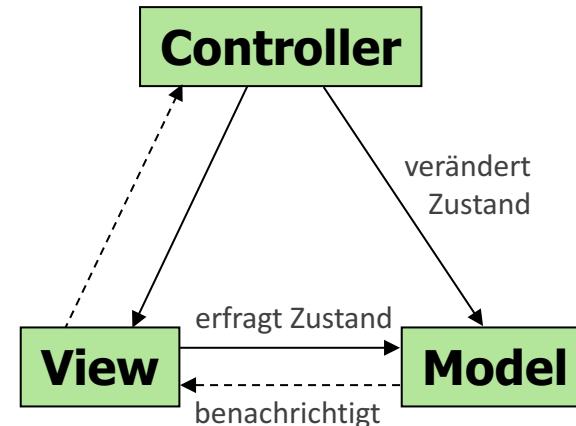


Model-View-Controller Architektur

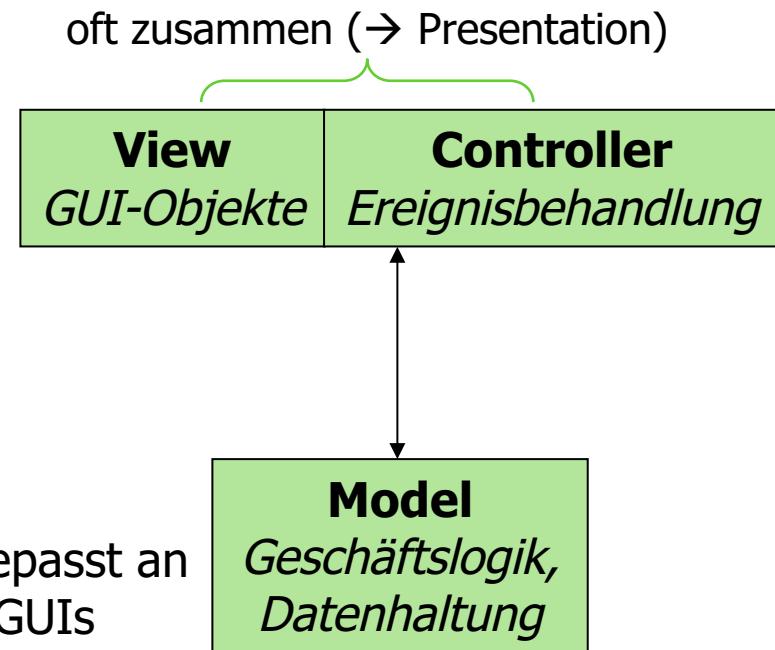
- Model: Was wird angezeigt?
 - Anwendungsfunktionen und Datenstrukturen (funktionaler Teil, kennt GUI nicht)
 - Implementiert durch "normale" Klassen-Hierarchie
- View: Wie wird es angezeigt?
 - Präsentation der Daten mit GUI-Komponenten
 - Model löst bei Änderungen Benachrichtigungen aus
→ Ansichten werden aktualisiert
 - View empfängt Dialogereignis, ruft entsprechende Methode von Control zur Umsetzung des Ereignisses auf
- Controller: Steuerung
 - Übersetzt Eingaben im View in Änderungen im Model
 - Manipuliert Model mit angebotenen, passenden Methoden
 - Controller i.d.R. als Listener/Eventhandler angebunden
 - Reagiert damit auf Modelländerungen



Model-View-Controller Architektur



übliche Darstellung

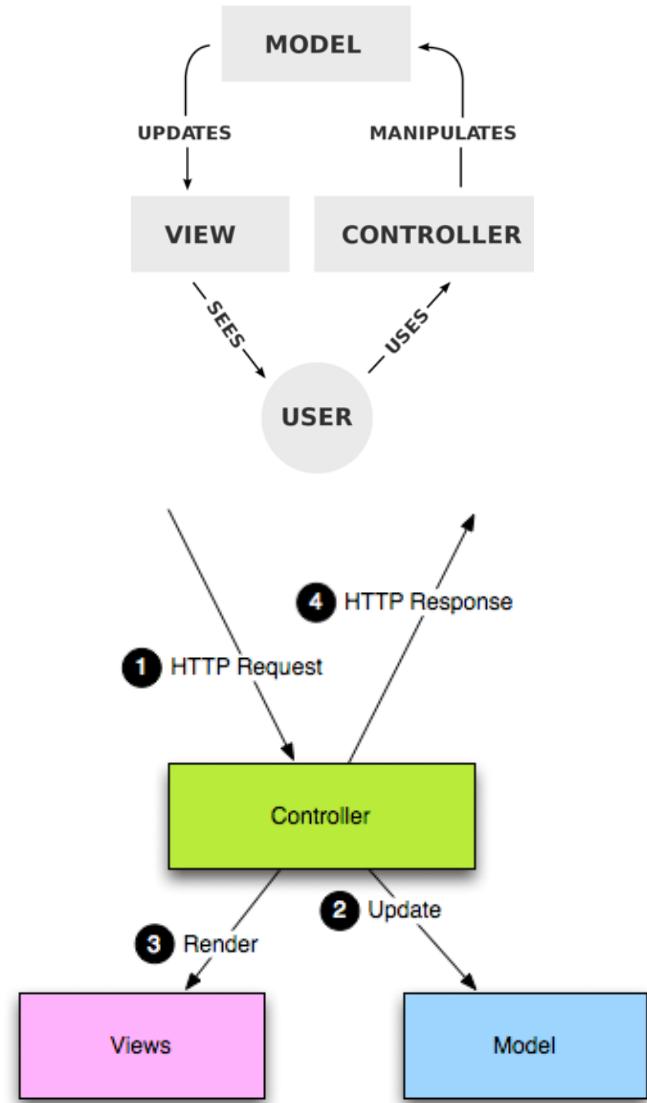


angepasst an
OO-GUIs

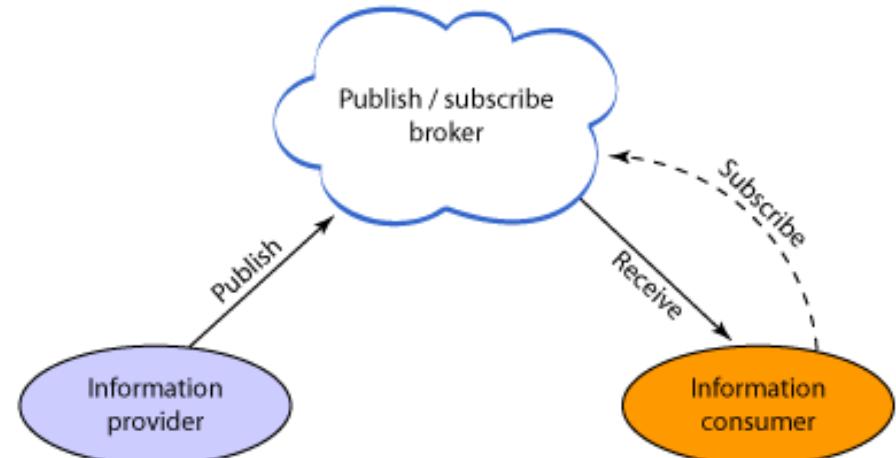
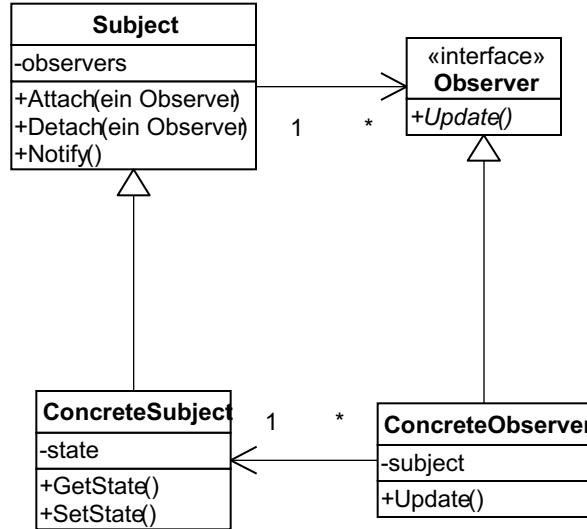
- View-Controller-Kombination
 - Implementiert durch Fensterklassen (mit Eventhandler)
 - Verschiedene Sichten auf Datenmodell je interaktiv
 - Ermöglicht synchronisierte Datenänderung
 - Anwendungsfunktionalität u. graphische Oberfläche sind getrennt

Model-View-Controller (MVC)

- Trennt Anwendung in Teile
 - Präsentation bzw. GUI
 - Views (werden aktualisiert, nachdem Modell aufgrund von Änderungen Nachricht auslöst)
 - Controller (dient dazu, Änderungen **nur** im Modell auszulösen)
 - → Frontend (Client: HTML, JavaScript)
 - Anwendungslogik & Datenhaltung
 - Modell (weiß nicht, wie viele und welche Views dafür existieren)
 - → Backend (Server: Python, PHP, SQL)
- Oft Basis von Webframeworks
 - Controller i.d.R. aktiver Vermittler zwischen Model u. View während Request-Response-Zyklus
 - View erzeugt hier lediglich (HTML-) Code, den Webbrowswer anzeigt



Model-View als Observer-Pattern



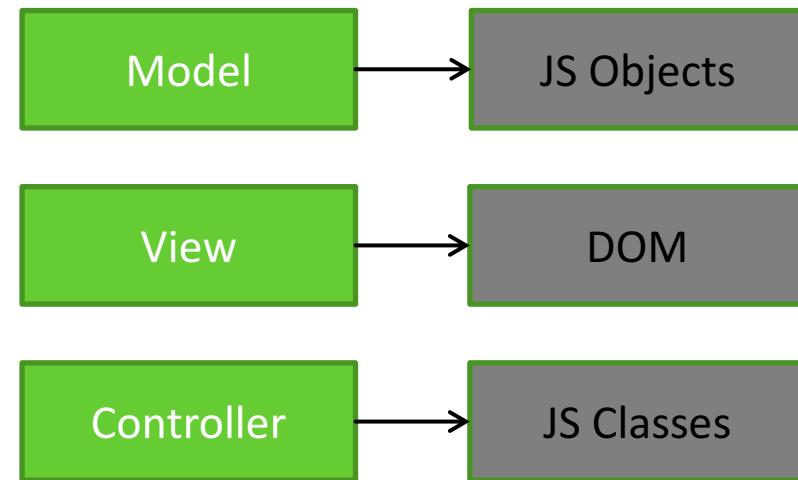
Das bekannte
Observer-Pattern

Model-View-Architektur

- Publisher (Subject) hat Liste von bei ihm registrierten Subscribers (Observer), welche er bei Änderungen informiert
 - Daher auch Publisher-Subscriber-Pattern genannt
 - View (Buttons...) und Control (als Listener) in Komponenten festgelegt

Beispiel: AngularJS

- MVC JavaScript framework for rich Web application development
 - Allows for modern single page apps
 - Browser and server can have their own MVC systems
- Locale MVC app
 - Model not necessarily the data from database
 - Observer interface allows decoupling Model from rest



Beispiel: AngularJS

```
<!DOCTYPE html>
<html ng-app>
<head>
    <title></title>
</head>
<body>
    <div class="container">
        Name: <input type="text" ng-model="name"> {{ name }}
    </div>

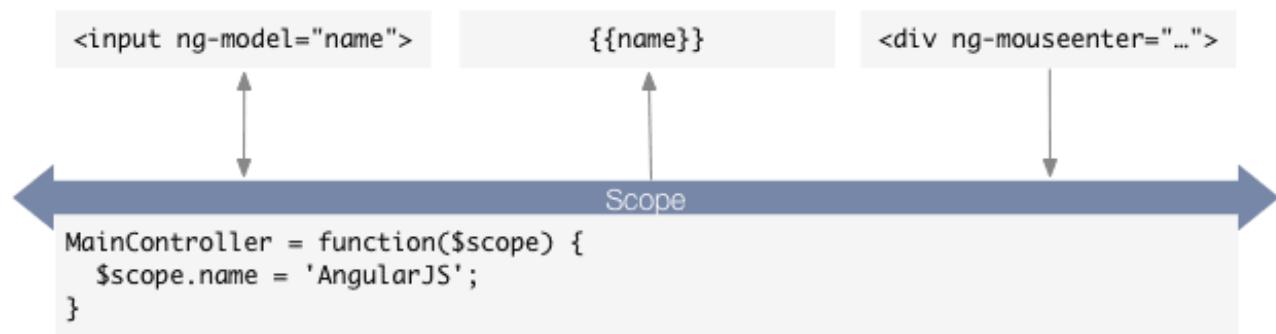
    <script src="scripts/angular.js"></script>
</body>
</html>
```

Direktive

Direktive

Expression

Zwei-Wege-Datenbindung



Framework vs. Bibliothek

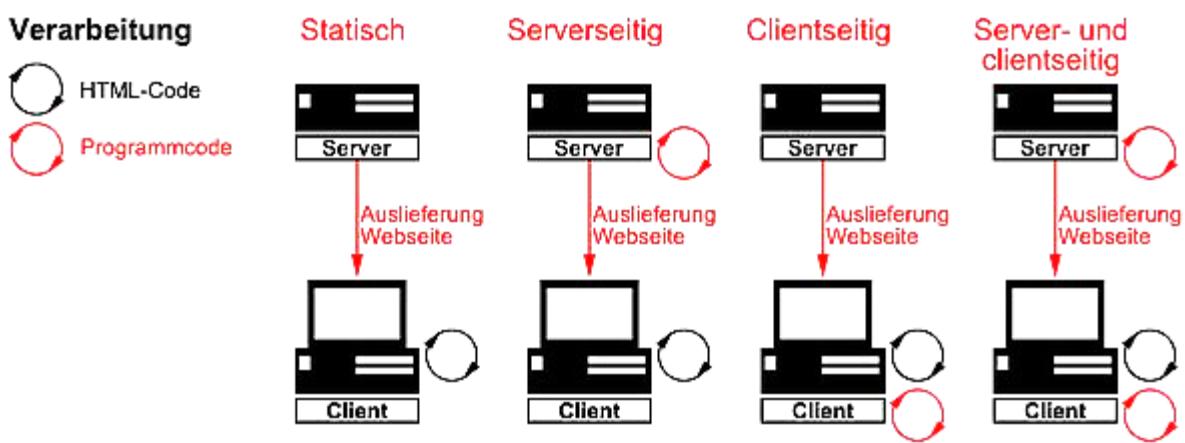
- Anpassbares Applikationsskelett
 - Wiederverwendbarer Entwurf (→ Reuse)
 - Beschrieben durch Menge abstrakter „Klassen“ und Zusammenspiel der Instanzen
 - Wiederverwendbares Verhalten:
Eigener Code wird durch Framework aufgerufen (AngularJS)
 - Unterschied zu Bibliothek
 - Bietet wiederverwendbare Funktionalität:
Methoden werden von eigenem Code aufgerufen (jQuery)
- Wahl des Frameworks als technische Basis
 - Bedingt i.d.R. Wahl der Programmiersprache
 - Reduktion von Entwicklungszeiten und -kosten
 - Webframeworks nutzen meist MVC-Architektur
 - Beinhalten oft Templatesystem (→ View) u. Datenbankzugriff
 - Beispiele: Play Framework, Flask (mit Template-Engine Jinja2)



Entwicklerwerkzeuge

- Versionsverwaltung (z.B. Git, SVN)
- Code-Analyse-Tools (z.B. JSLint, ESLint)
- Minifier (z.B. Google Closure Compiler)
 - Zum Komprimieren des Quelltextes
 - Entfernt Whitespace, verkürzt Variablennamen etc.
- Bündeln und Modularisieren (z.B. Webpack)
 - Baut und bündelt Module
 - Löst Abhängigkeiten auf
 - Bietet viele nützliche Zusatzmodule
 - CSS-Prefixer für bessere Browserkompatibilität
 - Babel, um ES6 zu ES5 zu wandeln (inkl. Polyfill)





Serverseitige Programmierung

Teil 2: PHP



Content-Management-Systeme

- Verwalten Inhalte eines Webauftretts
 - Beispiele: WordPress, Drupal, Joomla, TYPO3

The image displays two side-by-side screenshots of Content Management System (CMS) administration panels.

Left Screenshot (WordPress Admin):

- Dashboard:** Shows a list of posts with titles like "RWTH Aachen University uses X3DOM for Collaborative 3D Viewer", "X3DOM 1.7 Released", "Open Source 3D Component Editor using X3DOM - Draft", "Vicomtech's VolumeRC showcases X3DOM", "Cultural Heritage Project 3D-ICONS uses X3DOM - Draft", "CREALIS 3D uses X3DOM", and "X3DOM used for professional BIM visualization".
- Menu:** Includes links for Posts, Media, Links, Pages, Comments, X3DOM Examples, Appearance, Plugins, Users, Tools, Settings, and Collapse menu.

Right Screenshot (Another CMS Admin):

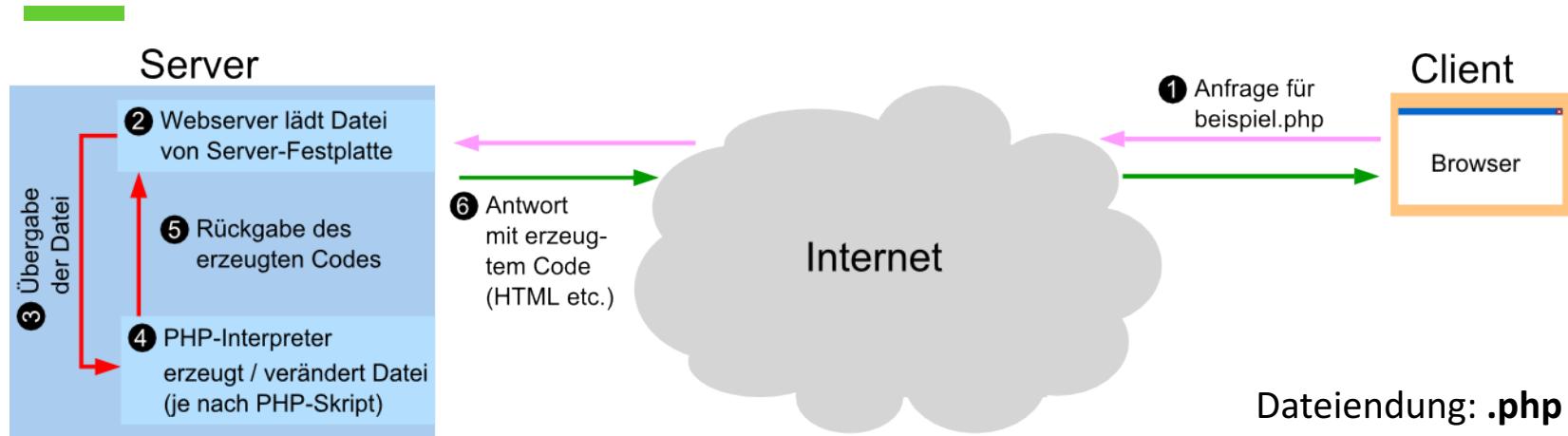
- Dashboard:** Shows a list of themes with "x3dom new design" selected.
- Appearance:** Shows the Main Index Template (index.php) code:

```
<?php get_header(); ?>

<div class="container">
    <div class="row">
        <div class="col-lg-8">

            <?php if (have_posts()) : while (have_posts()) : the_post(); ?>
                <div>
                    <h2><a href=<?php the_permalink(); ?>><?php the_title(); ?></a></h2>
                    <p><?php the_date('d.m.Y'); ?>
                        <?php
                            $categories = get_the_category();
                            $separator = ' ';
                            $output = '';
                            if($categories)
                            {
                                foreach($categories as $category)
                                {
                                    $output .= '<span class="label label-primary">' . $category->cat_name . '</span> ' ;
                                }
                                echo trim($output, $separator);
                            }
                        ?>
                    </p>
                    <div style="font-size: 15px;">
                        <?php post_body(); ?>
                    </div>
                </div>
            </?php endif(); ?>
        </div>
    </div>
</div>
```
- Templates:** Lists various template files: 404 Template (404.php), archive-xex_example.php, Footer (footer.php), front-page.php, Theme Functions (functions.php), Header (header.php), home.php, Main Index Template (index.php), Page Template (page.php), Search Results (search.php), Sidebar (sidebar.php), single-xex_example.php, Single Post (single.php), Styles, and Stylesheet.

PHP: Hypertext Preprocessor



- Serverseitig interpretierte Skriptsprache
 - Syntax angelehnt an C/C++ und Perl
 - Zwei Modi: Kopieren (HTML) und Interpretieren (PHP)

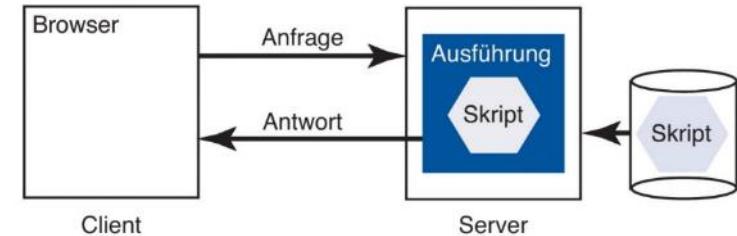
- Beispiel

```
<html>
  <body>
    <?php echo "Hallo Web-Apps!<br>"; ?>
  </body>
</html>
```



PHP

- Erleichtert Arbeit mit Datenbanken (MySQL)
 - Ermöglicht so Persistenz
- PHP-Dateien benötigen Server, können nicht von Browser interpretiert werden
 - PHP-Code wird dynamisch bei Anfrage ausgeführt und Ergebnisseite zurückgeliefert
 - Wichtiges Anwendungsfeld ist serverseitige Verarbeitung von Formulardaten
 - Daten stehen in sog. superglobalen Variablen zur Verfügung (→ `$_GET` bzw. `$_POST`)



Sprachkonstrukte

- Dateien inkludieren
 - Bsp.: `include_once "main.php";`
- Variablen beginnen mit `$`
 - Werden durch Verwendung direkt deklariert
 - Typ ergibt sich aus zugewiesenen Wert
- Operatoren i.d.R. analog zu JavaScript
 - Unterschied: String-Konkatenation über `.`
Bsp.: `"foo"."bar"`
- Kontrollstrukturen ebenfalls ähnlich
 - Auswahl (`if/else`), Schleifen (`for, while`)
- Ausgabe von Text und HTML mit `print`
 - Bsp.: `print "Sie wiegen $weight kg.
"`



BMI revisited: Formulare mit PHP

```
<form action="auswerten.php" method="post">
    <span>Gewicht</span>
    <input type="text" name="gewicht"><br>
    <span>Gr&ouml;&szlig;e</span>
    <input type="text" name="groesse"><br>
    <input type="submit" value="Absenden">
</form>
```

Eingabeformular in
HTML-Dokument

```
<?php
    $weight = $_POST["gewicht"];
    $height = $_POST["groesse"];
    $bmi = $weight / ($height * $height);

    echo "Mit ".$weight." kg und ".$height." m ".
        "haben Sie einen Body Mass Index von ".$bmi."<br>";
    if ($bmi < 20)
        echo "Damit sind Sie zu d&uuml;nn!<br>";
    else if ($bmi > 24)
        echo "Damit sind Sie zu dick!<br>";
    else
        echo "Damit haben Sie Normalgewicht.<br>";
?>
```

Aufgerufenes Skript
(*auswerten.php*)

Analog für GET



Funktionen

- Eingabeparameter wie in JS durch Komma getrennt
 - Rückgabewerte analog über *return*-Anweisung

```
function calcBMI($w, $h) {  
    return $w / ($h * $h);  
}
```

- Parameterübergabe mittels „*call by value*“
- Auch möglich (vgl. C/C++): *call by reference*

```
function swap(&$val1, &$val2) {  
    $tmp = $val1;  
    $val1 = $val2;  
    $val2 = $tmp;  
}
```

- Beim Aufruf müssen aktuelle Parameter Variablen sein
- Ohne zusätzliche Kennzeichnung mit „global“ kein Zugriff auf außerhalb deklarierte Variablen



Vordefinierte Funktionen

- Zeit (*time()*) und Datum (*date(...)*)
- Mathematische Funktionen (z.B. *pi()*)
- Stringverarbeitung
 - Ähnlich zu C mit String als Argument
 - Z.B. `strlen($str)`, `strcmp($str1, $str2)` usw.
- Reguläre Ausdrücke (RegEx)
 - Aus Sicherheitsgründen (→ XSS) wichtig für serverseitige Validierung von Formularen
 - Bsp. (Suche nach Wort-Leerzeichen-Wort):
`preg_match("/^([a-zA-ZäöüÄÖÜ]+)\s+([a-zA-ZäöüÄÖÜ]+)$/", $input_string, $output_array);`



Felder

- Werden mit Zahlen 0, 1, 2, ... indiziert
 - Assoziatives Feld als Sonderform (Hash Table)

- Name-Wert-Paare, durch „=>“ getrennt

```
$dict = array("Peter" => "0163/12345678");
```

- Array erzeugen

```
$tiere = array("Maus", "Ratte", "Frettchen");
```

- An Feld anhängen

```
$tiere[] = "Schlange";
```

- Array durchlaufen

```
$n = count($tiere);  
for ($i=0; $i<$n; $i++)  
    echo "$tiere[$i]<br>";
```

- String in Array splitten

```
$csv = "Peter;Petersen;Rosenweg 3;64283 Darmstadt";  
$splitted = explode(";", $csv);
```



Felder

- Vordefinierte assoziative Felder
 - Bsp. 1: `$_GET[FormName]` (z.B. „cb“ wie unten)
 - Bsp. 2: `$_SERVER["HTTP_USER_AGENT"]`
 - Hält alle benötigen Angaben zum verwendeten Browser
 - Beispielhafter Wert: *Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0*

- *m-aus-n-Auswahl mit Checkboxen*

```
<input type="checkbox" name="cb[]" value="red">
<input type="checkbox" name="cb[]" value="green">
<input type="checkbox" name="cb[]" value="blue">
```

- Zugriff mit PHP (Alternative Schleife mit ‚foreach‘)

```
foreach ($_GET["cb"] as $cb)
    echo "$cb<br>";
```



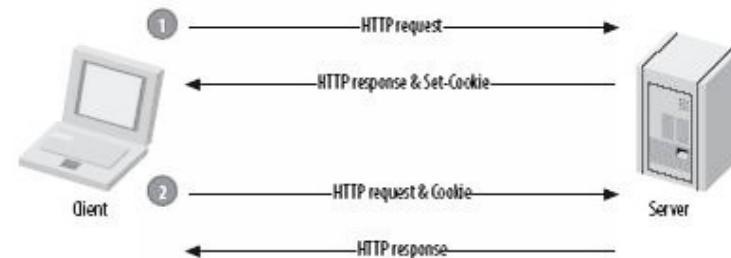
Superglobale Variablen

- Vordefinierte und ggfs. automatisch gefüllte Arrays
 - Mit Hilfe von *isset()* testen, ob Wert gesetzt wurde
- **\$_SERVER**: Server-Informationen
- **\$_ENV**: Umgebungsvariablen
- **\$_SESSION**: Sessionspezifische Variablen
- **\$_GET**: HTTP Get Variablen
 - Zugriff auf per Get gesendete Formularelemente
 - Name des Formularelements ist Zugriffsschlüssel
 - Name-Wert-Paare mittels Query-String übertragen
- **\$_POST**: HTTP Post Variablen
 - Zugriff auf per Post gesendete Formularelemente
 - Name des Formularelements ist Zugriffsschlüssel
- **\$_COOKIE**: Cookie-Informationen



Session-Tracking

- Verbindung zw. Client und Server zustandslos
 - Problem bei Warenkorbanwendungen u.ä.
 - Datenweitergabe an nachfolgende Anfragen nötig
- Lösungsmöglichkeiten
 - Persistente Cookies
 - Server sendet Cookie,
wird mit jedem Request zurückgesendet
 - In PHP mittels *setcookie()* → muss vor HTML-Code stehen
 - Versteckte Formularfelder
 - Werden spezifisch von Server für Client erzeugt
 - Damit Datenweitergabe von Anfrage zu Anfrage



```
<input type="hidden" name="unsichtbar" value="sessID">
```



Session-Tracking

- PHP-Sessions
 - Session ist Zeit, in der Browser mit geg. Server interagiert
 - Session-Informationen sind temporär, werden nach Verlassen der Website gelöscht
 - Session-Daten zugreifbar machen

```
session_start()
```

 - Erzeugt Session oder setzt sie fort; muss als erstes im PHP-Skript stehen, da Übertragung als Cookie bzw. Query String
 - Dann `$_SESSION` mit Name-Wert-Paaren setzen bzw. daraus lesen, falls in Session bereits gesetzt

```
if (!isset($_SESSION['counter'])) {  
    // Page Counter; u.a. Wert aus File lesen bzw. init.  
    $_SESSION['counter'] = $counter; // eigener Schlüssel  
}
```
 - `$_SESSION` ist assoziatives Feld und global verfügbar, Name-Wert-Paare können beliebig selbst gewählt werden



Dateien

- Haben Dateizeiger/Handle (wie in C)
- Öffnen: `$fp = fopen($fileName, mode)`
 - Zugriffsmodi
 - "r"/"r+": zum Lesen (und bei r+ noch Schreiben) öffnen, Beginn am Dateianfang, Datei muss existieren
 - "w"/"w+": zum Schreiben öffnen (w+ wie oben), Beginn am Dateianfang, erzeugt ggfs. Datei
 - "a"/"a+": zum Schreiben öffnen, Einfügen am Dateiende
 - Fehlerbehandlung: `fopen(...)` or die („ErrorMsg“)
- Auf Existenz prüfen: `file_exists($fileName)`
- Datei schließen: `fclose($fp);`

Dateien Lesen bzw. Schreiben

- Lesen
 - `$content = fread($fp, filesize($fileName));`
 - Zweiter Param.: #Bytes (hier: ganze Datei)
 - Einzelne Zeilen einlesen bis Dateiende
 - `while (!feof($fp)) { $line = fgets($fp); }`
- Schreiben
 - `fwrite($fp, $content)`
 - Evtl. vorher an Dateianfang gehen: `rewind($fp);`
 - Gleichzeitiges Lesen und Schreiben bei parallelen Anfragen problematisch
 - Lsg.: Datei mit `flock($fp, LOCK_EX)` sperren
 - Wieder entsperren mit `flock($fp, LOCK_UN)`



Datenbanken im Web

- Relationale DBs (Abfragesprache SQL)
 - Daten werden in Tabellen verwaltet
 - Bekanntester Vertreter: MySQL/MariaDB
 - Serverseitiger Zugriff mittels spezieller PHP-API
 - Bsp.: \$r = mysql_query("SELECT * FROM messwerte;");
- Dokumentenorientierte DBs (sog. NoSQL DBs)
 - Daten werden in JSON-Dokumenten verwaltet
 - Tabellenzeile entspricht Dokument
 - Tabellenspalte entspricht Datenfeld
 - Z.B. MongoDB und CouchDB
 - Clientseitiger Zugriff mit JavaScript & Ajax möglich
 - Nutzen zur Kommunikation oft RESTful Webservices



Exkurs: MySQL (Admin-Sicht)

The screenshot shows the phpMyAdmin interface for a MySQL database named 'test'. The left sidebar displays the database schema with the following structure:

- cdcol
- information_schema
- mysql
- performance_schema
- phpmyadmin
- test** (selected)
 - Prozeduren
 - Tabellen
 - Neu
 - chat
 - messwerte** (selected)
 - Indizes
 - Spalten
 - users
 - webauth**

The main panel shows the 'messwerte' table structure and data. At the top, there is an SQL query editor with the following update statement:

```
UPDATE `test`.`messwerte` SET `Name` = 'Frökk' WHERE `messwerte`.`Key` = 5;
```

Below the query, a message indicates:

Zeige Datensätze 0 - 4 (5 insgesamt, Die Abfrage dauerte 0.0002 Sekunden)

Underneath, there is a select query:

```
SELECT *  
FROM `messwerte`  
LIMIT 0 , 30
```

At the bottom, a table displays the data from the 'messwerte' table:

	Key	Name	Gewicht	Groesse	BMI	Datum
<input type="checkbox"/>	1	Hrollur	350	1.38	183.785	2014-06-13 17:34:08
<input type="checkbox"/>	2	Heidi	54	1.66	19.5965	2014-06-13 17:34:36
<input type="checkbox"/>	3	Heidi	54.6	1.667	19.6481	2014-06-13 17:34:58
<input type="checkbox"/>	5	Frökk	400	1.4	204.082	2016-04-06 13:46:21
<input type="checkbox"/>	6	Karl	20	0.5	80	2014-06-24 14:52:45

Below the table, there are buttons for selecting all rows, marking selected rows, editing, deleting, and exporting.

Exkurs: MySQL (Codebeispiel)

```
$db_server = mysql_connect("localhost", "root", null);
if (!$db_server)
    die("Unable to connect to MySQL: ".mysql_error());
mysql_select_db("test")
    or die("Unable to select database: ".mysql_error());

$query = "SELECT * FROM messwerte;";
$result = mysql_query($query);
if (!$result)
    die ("Database access failed: ".mysql_error());

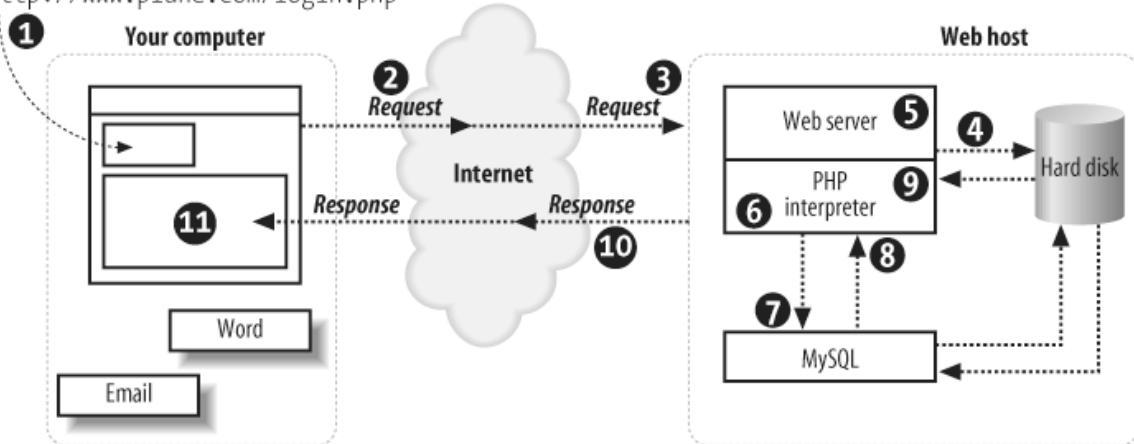
echo "<h2>Messwerte</h2>";
$rows = mysql_num_rows($result);

echo "<table border='1' cellpadding='3'>";
echo "<tr><td>#</td><td>Name</td><td>Gewicht</td><td>Größe</td><td>BMI</td></tr>";

for ($j = 0; $j < $rows; ++$j) {
    echo '<tr><td>'.mysql_result($result, $j, 'Key').'</td>';
    echo '<td>'.mysql_result($result, $j, 'Name').'</td>';
    echo '<td>'.mysql_result($result, $j, 'Gewicht').'</td>';
    echo '<td>'.mysql_result($result, $j, 'Groesse').'</td>';
    echo '<td>'.mysql_result($result, $j, 'BMI').'</td></tr>';
}
echo "</table>";

mysql_close($db_server);
```





Noch Fragen?

