

## HTML – Grundgerüst

- HTML = Auszeichnungssprache keine Programmiersprache
- Tags/Markups sind html Anweisungen(`body`, `h1`, `script`) besser klein geschrieben für XML-kompatibilität
- Attribut ist Zusatzinfo für jeden Tag(key/value pairs)
  - `<body bgcolor="#AABBCC">` ↪ immer in Hochkommas
- Code einrücken für bessere Lesbarkeit.
- `<head>`: Metainfos, Titel, Stil, Scriptdefinition, Address- & Zielfensterbasis
- Metatags: 1. Sprache 2. Keywords 3. Inhalt 4. Zielpublikum 5. Linkverfolgung 6. Refresh 7. Wiederholung Indizierung
  1. `<meta http-equiv="language" content="deutsch, de">`
  2. `<meta name="keywords" content="a,b,c,d,e,f,g,h">`
  3. `<meta name="page-topic" content="Dienstleistung">`
  4. `<meta name="audience" content="alle">`
  5. `<meta name="robots" content="index,follow">`
  6. `<meta http-equiv="refresh" content="10; URL=x.htm">`
  7. `<meta name="revisit-after" content="14 days">`
- `<body>`: Anfang & Ende des sichtbaren Inhalt der Page
- `<div>/<span>`: Container zum Gruppieren & Attribute setzen. Div = block element; span=inline element
  - `<p>I love <span style="color:blue">blue</span> </p>`
- 1. Listen & 2. Definitions-/Glossar List

```
<ol>
  <li>www</li>
  <li>Starten mit HTML</li>
    <ul>
      <li>Was HTML ist</li>
      <li>Wie schreibt man HTML</li>
    </ul>
  <li>Links</li>
</ol>
```

1. WWW  
2. Starten mit HTML

- Was HTML ist
- Wie schreibt man HTML

3. Links

### Unix

Multiuser

### Windows

Oberfläche.

### MAC-OS

Singluser

Bei ol, `start="3"` → Aufzählung beginnt bei 3. `value="6"` fährt die Liste bei 6 weiter, `type=""` führt zu grosser römischer Nummerierung. Bei ul, `type="square"` → Quadrat als Aufzählungszeichen.

- 1. Grafiken 2. Link 3. Mail-Link

```

<a href="http://www.hslu.ch">HSLU T&A</a>
<a href="mailto:mars@nimm2.org">Kontakt</a>
```

- Verknüpfungen: 1. Link 2. Ziel 3. Neues Fenster

```
<a href="#Kapitel1">Kapitel 1</a>
<p id="Kapitel1">Kapitel 1</p>
```

- Klickbare Grafiken: 1. Bild 2. Map & klickbare Fläche

```

<map name="karte"> <area shape="circle" coords="50,50,45" href="Ziel.html" alt="Reiseziel" />
</map>
```

- Umlaute:

ü=&ampuuml | ö=&ampouml | ä=&auml;

- `<iframe>` Eingebettete Frames/Webpages

```
<iframe src="http://www.test.ch/" width="1000" height="400" name="innerframe" align="center">
<p>Ihr Browser unterst&ampuumltzt Frames nicht.</p>
</iframe>
```

### Formularelemente

- 1. `Datalist` 2. Platzhalter Text

```
<input type="text" id="r" name="r" list="7ringe">
<datalist id="7ringe">
  <option value="core">
  <option value="com">
</datalist>
```

```
<input type="text" id="i" name="i" placeholder="beautiful">
```

- 1. `Slider(range)` 2. `Number selector` 3. `Date`

```
<input type="range" id="e" name="e" min="1" max="5">
```

```
<input type="number" id="e" name="e" min="5" max="20">
```

3

```
<input type="date" id="datum" name="datum">
```

- **Regex für Eingabekontrolle**

```
<input id="zip" name="zip" pattern="[\d]{5}(-[\d]{4})">
```

- Beispiel

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Hello World</title>
```

```
<meta charset="utf-8">
```

```
<meta name="author" content="JD">
```

```
<style>h1 { color: white; }</style>
```

```
<link rel="stylesheet" href="styles.css">
```

```
</head>
```

```
<body>
```

```
<h1 style="color:red;">Hallo Welt</h1>
```

```
<!--tagname style="property:value;"-->
```

```

```

```
</body>
```

```
</html>
```

## CSS – Grundgerüst

```
body { background-color: powderblue; }
```

```
#myID { color: blue; }
```

```
.myClass { color: red; }
```

```
}
```

### CSS-Reset

CSS-Reset wird ein Stylesheet oder ein Teil eines Stylesheets genannt, der die Browservorgaben für die Anzeige der Elemente, oder auch nur Teile davon, entfernt. Der klassische CSS-Reset entfernt so jegliche margin/padding-Angaben und das für alle Elemente. Der Autor eines Stylesheets muss demnach bei der Definition von eigenen CSS-Eigenschaften Browserunterschiede nicht mehr beachten. Verschiedene Browser definieren normalerweise beispielsweise die Einrückung von Listen unterschiedlich.

- 1. Klassen/Selektoren 2. Mehrere gleichzeitig

```
p.einleitung { font-weight: bold; }
.p.einleitung { color: #FF0000; }
p.absatz { color: #00FF00; }
```

nicht spezifiziert ⇒ gilt für alle (h1, h2, etc.)

- Angewendet wird es so:

```
...
<p class="einleitung">...</p>
...
<p class="absatz">...</p>
...
<h1 class="einleitung">..</h1>
```

h1 0 h2 0 h3 { color: blue; }

- h3 innerhalb h2 innerhalb h1

h1 0 h2 0 h3 { color: blue; }

- CSS-Selektoren: 1.einfache 2.kombinierte

Selektor	Auswahl	Beispiel
*	jedes Element	<p>
element	Element mit dem Namen element	<element/>
.klasse	Element mit der Klasse klasse	<p class="klasse">
#elementid	Element mit der ID elementid	<p id="elementid">

Kombinator	Auswahl	Beispiel
A B ← B in A	Element B, wenn es innerhalb von A vorkommt.	<A><x> <B></x> </A>
A > B	Element B, wenn es direkt innerhalb von A vorkommt.	<A><B> </A>
A + B	Element B, wenn es direkt nach A vorkommt (im gleichen Elternelement)	<A><B>
A ~ B	Element B, wenn davor A vorkommt (im gleichen Elternelement)	<A><x> <B>

- Hyperlinks

```
a:link { color:blue; } /* normaler Link */
a:visited { color:green; } /* bereits besucht */
a:active { color:red; } /* gerade angeklickt */
a:hover { color:yellow; } /* unter dem Mauszeiger */
a:focus { color:black; } /* mit Tastatur angewählt */
```

- Kaskadierung/Prioritätsreihenfolge von hoch>niedrig

1. Deklarationen des Browsers
2. Deklarationen des Benutzers\*
3. Interne/Externe CSS-Anweisungen des Web-Authors
4. Inline CSS-Anweisungen des Web-Authors
5. Deklarationen des Web-Authors die !important enthalten
6. Deklarationen des Benutzers\* die !important enthalten

\* Benutzer => Website Besucher → also seine Browsetreinstellungen!

- Ausgabemedien: 1.Bildschirm 2.Drucker 3.Sprachausg.
- Das Attribut: media definiert, für welches Ausgabemedium der entsprechende Style definiert ist.

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

- MEDIA-QUERIES/RWD – Responsive web design

```
@media screen and (orientation: portrait) { body {
background-color: white; }}
```

```
@media screen and (orientation: landscape) { body {
background-color: black; }}
```

- Ausgabemedien: Verlinkung CSS in html-head

```
<link href="standard.css" rel="stylesheet" media="screen"
title="Standard-Layout">
```

Media kann verschiedene Werte wie: all, braille, handheld, print, tty oder tv haben.

Besser ist es jedoch @import zu verwenden. Man kann auch verschiedene css-files angeben je nach Ausgabemedium.

```
<head>
<style>
  @import url('komplett.css') all;
  @import url('print.css') print;
  @import url('screen.css') screen;
</style>
</head>
```

Der Header, die Seitenpalte und der Footer werden in der folgenden Vorgabe nur für den Fall ausgeblendet, falls die Webseite ausgedruckt wird:

```
@media print {
  div#header,
  div#sidebar,
  div#footer { display: none; }
}
```

- Mehrere Queries kombinieren (Screen und Projektor)

```
@import url("style.css") screen, projection;
```

Zusätzlich mit Breite:

```
@import url("style.css") screen and (min-width:
800px), projection and (min-width: 800px);
```

Not/only:

```
<link rel="stylesheet" type="text/css"
href="style.css" media="not screen and (color)" />
```

- Media-Queris generell:

```
Media_Query: [:only | not]? <Medientyp> [ and <Ausdruck> ]*
| <Ausdruck> [ and <Ausdruck> ]*
```

Ausdruck: ( <Merkmal> [: <Wert>] )

Medientyp: all | aural | braille | handheld | print |
projection | screen | tty | tv | embossed

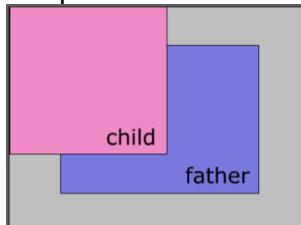
Merkmale: width | min-width | max-width
| height | min-height | max-height
| device-width | min-device-width | max-device-width
| device-height | min-device-height | max-device-height
| aspect-ratio | min-aspect-ratio | max-aspect-ratio
| device-aspect-ratio | min-device-aspect-ratio | max-device-
aspect-ratio
| color | min-color | max-color
| color-index | min-color-index | max-color-index
| monochrome | min-monochrome | max-monochrome
| resolution | min-resolution | max-resolution
| scan | grid

- POSITIONIERUNG/FLEX

Es gibt fünf verschiedenen Positionswerte:

- o static (default, not affected by top, bottom, left ...)
- o relative (ist relativ zum normalen Ort plaziert)
- o fixed (Immer an derselben Position, auch beim scrollen, bsp. Navigation)
- o absolute (Relativ positioniert zu parent element(welches positioniert ist, also alles außer static. Wenn keine position: xyz; beim Parent gesetzt dann funktioniert nicht), wenn keines vorhanden, dann zu body. Also wird die box vom übergeordneten Element zum Koordinatensystem dieses Elementes.)
- o sticky (basiert auf scroll-Position, zuerst relative und ab offset fix. (bsp. top: 0; scrollt bis Element am oberen Rand des Screens ist & ist dann fix & bleibt auch beim weiterscrollen oben am Screen.)

- Beispiele



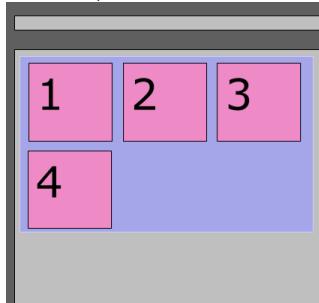
```
<div class="father">
  <div class="child"></div>
</div>

.father {
  margin: 20px; Position fehlt
}
.child {
  position: absolute;
  width: 55%;
  top: 0;
  left: 0;
}

<div class="father">
  <div class="child"></div>
</div>

.father {
  position: relative;
  margin: 20px;
}
.child{
  position: absolute;
  width: 55%;
  top: 0;
  left: 0;
}
```

- FLOAT (Wo stehe ich relativ zu anderen Elementen)



```
<div class="father">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>

.father {
  float: left;
  width: 100%;
  background-color: #ccccff;
}

.father div {
  float: left;
  width: 30%;
  margin: 1% 1% 0;
```

- FLEX-Positionierung

1. Zuerst sagt man einem Elment (father), dass dessen Kinder «flex» sind
2. Dann bestimmt man wie sich die Kinder verhalten
3. Properties:

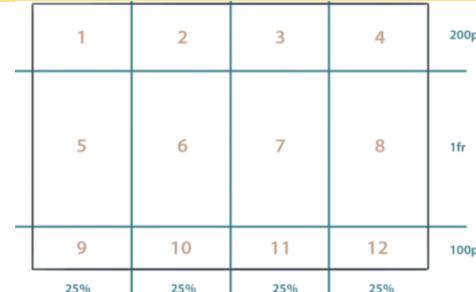
- a. Flex-direction: (row-/column(-reverse))
- b. Flex-wrap: auto Zeilenumbruch: (no)wrap
- c. Justify-content: X-Achse: center, flex-start/end, space-between/around
- d. Align-content: Y-Achse: gleiche wie oben

- GRID (zusammen mit Flex)

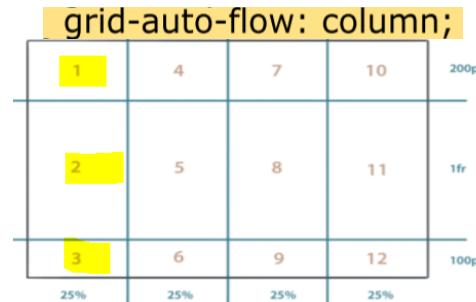
- o CSS Grid ermöglicht Gestaltungsraster auf Basis von CSS
- o Dem Elternelement wird mit Hilfe der Angabe **display:grid;** mitgeteilt, dass CSS Grids genutzt werden sollen.

- o Mit Hilfe der Eigenschaften **grid-template-columns** und **grid-template-rows** werden Rasterlinien gezeichnet.

```
.container {
  display: grid;
  grid-template-rows: 200px 1fr 100px;
  grid-template-columns: 25% 25% 25% 25%;
```



- o Mit der CSS-Eigenschaft **grid-auto-flow** kann beeinflusst werden, wie die Kind-Elemente im Grid eingesortiert werden.

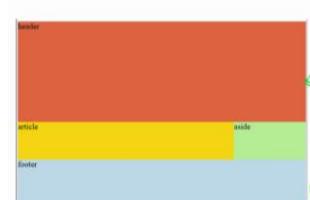


- Elemente im Grid positionieren

Achtung beginnt bei 1 (0%)

### Beispiel: Positionierung

```
<div class="container">
  <header>header</header>
  <article>article</article>
  <aside>aside</aside>
  <footer>footer</footer>
</div>
```



```
vh: relative zu 1% der Höhe des Viewports
100% größer

.container {
  height:100vh;
  display: grid;
  grid-template-rows: 200px 1fr 100px;
  grid-template-columns: 25% 25% 25% 25%;
```

header { background:tomato; grid-column: 1/5; grid-row: 1/2; *also ganz Brücke* }

article { background:gold; grid-column: 1/4; grid-row: 2/3; Kurzform: grid-column-start:1; grid-column-end:5; }

aside { background:lightgreen; grid-column: 4/5; grid-row: 2/3; }

footer { background:lightblue; grid-column: 1/5; grid-row: 3/4; }

- Gridzellen benennen

```
.container {
  height:100vh;
  display: grid;
  grid-template-rows: 200px 1fr 100px;
  grid-template-columns: 25% 25% 25% 25%;
```

grid-template-areas: "header header header header"  
"content content content sidebar"  
"footer footer footer footer";

Wird eine Gridzelle mit Punkt(.) benannt, dann wird diese Zelle durchsichtig sein. Sprich es könnte Beispielsweise der Hintergrund gesehen werden. Wie ein Guckloch Abstände mit: grid-row/column-gap

## Accessibility – WCAG

Es geht darum Webseiten so zu gestalten, dass sie von jedermann gelesen werden können (Barrierefrei).

- Regelungen von WAI, WCAG und W3C
- Probleme: 1. Clientseitige Scripts 2. Bilder ohne alt="" 3. Flash Animationen 4. Farbenwahl
- Sämtliche Dienstleistungen öffentlicher Gemeinwesen müssen barrierefrei sein.
- Prinzipien: 1. Wahrnehmbar 2. Bedienbar 3. Verständlich 4. Robust (Kompatibilität mit assist-Tec)
- Konformitätsstufen: A=min – AAA=max
  - o Gelten nur für ganze Seiten
  - o Vollständiger Prozess muss konform sein, also vom Kaufen bis Zahlen. → Alles barrierefrei

# JavaScript

Ein JavaScript-Programm ist entweder in das HTML Dokument eingebettet oder wird aus einer separaten Datei eingebunden.

- Code ist einsehbar!
- Der Browser interpretiert den Code
- JavaScript-Code im <head> oder <body> Bereich
- Sämtliche JS werden clientseitig ausgeführt

## Vorteile:

- kein Rechenaufwand für den Server
- der Betrachter entscheidet, was er zulässt
- direkter Zugriff auf die Darstellungsmöglichkeiten des Betrachters

## Nachteile:

- Server kann die Ausführung der Scripte nicht
- Dateioperationen nicht möglich
- Zugriffsmöglichkeiten auf den ClientPC könnten missbraucht werden
- Betrachter benötigt entsprechende JavaScript-Laufzeitumgebung, Server muss evtl. mehrere angepasste Scripte für verschiedene Browser / Versionen zur Verfügung stellen

- JS in body von externer Datei: 1. HTML 2. JS in hello.js

```
<script src="scripts/hello.js">
  .. Hier darf kein weiterer JS Code stehen!!!
</script>

document.write("<h2>Hello World mit Javascript</h2>");
```

## Möglichkeiten von JS

- Validitätscheck von Formularwerten
- Dokumentenmanipulation → DOM
- Browsersteuerung
- Bearbeiten von Cookies
- Spielereien, Gimmicks
- Variablen-Namen
  - o Gross/Kleinschreibung wird beachtet
  - o Zeichen, Buchstaben, Zahlen und Unterstrich
  - o Keine Leerzeichen
  - o Keine Umlaute
  - o Kein Bindestrich (=Subtraktionszeichen!)
  - o Maximal 32 Zeichen lang

## Globale Variablen:

Mit dem Vorstellen von «window.» kann eine Variable nach der ersten Verwendung nachher global benutzt werden. → window.MeineVariable;  
Oder: wenn Variable ausserhalb von Funktion gesetzt.  
Lokale Variablen:

Innerhalb einer Funktion, also lokal mit var.

## Unterschied var und let (2)

- Block: hier ist let nur innerhalb des Blockes sichtbar!

```
function fooBar() {
  //tuce is *not* visible out here
  for( let tuce = 0; tuce < 5; tuce++ ) {
    //tuce is only visible in here (and in the for() parentheses)
  }
  //tuce is *not* visible out here
}

function byE40() {
  //nish *is* visible out here
  for( var nish = 0; nish < 5; nish++ ) {
    //nish is visible to the whole function
  }
  //nish *is* visible out here
}
```

## Schlaufen

Bedingung wird vor Schleife geprüft:

```
while (Bedingung) { Anweisung; }
```



Bedingung wird nach Schleife geprüft:

```
do { Anweisung; }
while (Bedingung) // mindestens 1 Durchlauf
```

Bestimmte vordefinierte Durchlaufanzahl:

```
for (Anweisung; Bedingung; Anweisung) { Anweisung;
  break; // Schleife abbrechen
  continue; // nächsten Schleifendurchlauf
}
```

## If-Schlaufen

"einfache" Bedingung:

```
if (Bedingung) { Anweisung;}
else if (Bedingung) { Anweisung;}
else { Anweisung; }
```

"mehrwertige" Bedingung:

```
switch (Wert) {
  case 1: {Anweisung;} break;
  case 2: {Anweisung;} break;
  default: {Anweisung;} break;
}
```

## Boolische Ausdrücke (ergibt true oder false)

```
(a > b) && (x > y) || y == 10
```

- && bedeutet AND
- || bedeutet OR
- == bedeutet "ist gleich"

## Funktionen

- Funktion definieren:

```
function Summe (parameter1, parameter2) {
  var ergebnis = parameter1 + " " + parameter2;
  return ergebnis;
}
```

- Funktion aufrufen:

```
resultat = Summe("Hello", "World");
```

Sofort ausprobieren, dabei mit verschiedenen Datentypen experimentieren!

**Frage:** was passiert wenn man Zahl mit Zeichen addiert?

> 3 + 4 + "5"	4 + 3 + "5"	When you add a number & a string in
"75"	"75"	Javascript the result
> 3 + 4 + +"5"	4 + 3 + "5" + 3	is always a string.
12	"753"	

## Events

- JavaScript Programme werden zu einem grossen Teil durch Events (Ereignisse) gesteuert.

- Durch eine **Interaktion des Browserbenutzers** wird eine Aktion ausgelöst

## Beispiel:

beim Laden der Seite  
Maus wird über Element bewegt  
Benutzer klickt auf einen Link/Feld  
Maus verlässt ein Feld

## Beispiel:

```

```

```
<body onload= "alert('Willkommen');">
```

## - JS auf Knopfdruck: 1. HTML 2. JS-Script in HTML

```
<form>
  <input type="button" value="Test" onclick="PopUp()">
</form>
<script>
  function PopUp() {
    fens1=window.open("", "Textausgabe",
      "width=300, height=300");
    fens1.document.write("<div>Hier ist mein Text</div>");
  }
</script>
```

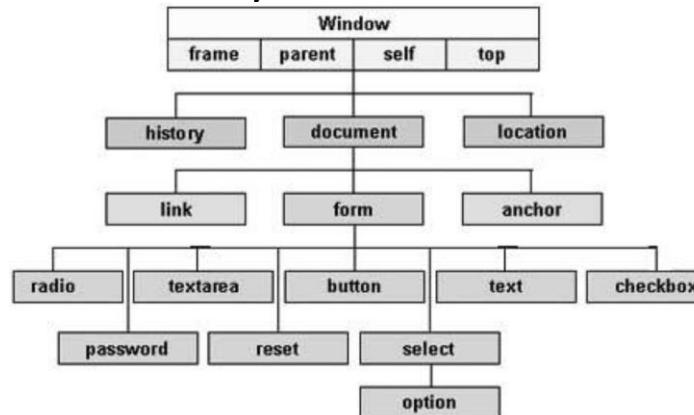
## - Vordefinierte Dialogboxen

- o Alert(): Meldung mit OK bestätigen
- o Confirm(): Ja/Nein Abfrage

- Prompt: erfordert Benutzereingabe

```
if (confirm("Are you 18years old?"))
{ alert ("Welcome"); }
```

## DOM – Document Object Model



### Beispiel 1:

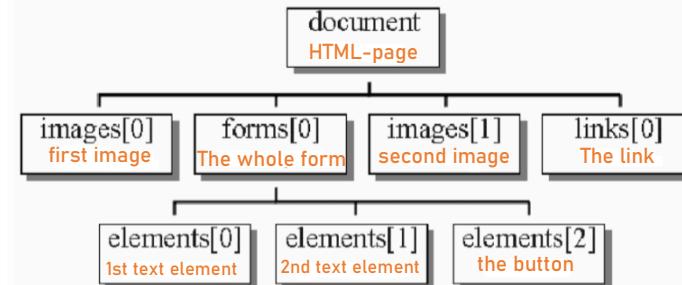
```
<html>
<head>
</head>
<body style="text-align:center;">

<form name="myForm">
  Name:<input type="text" name="Name" />
  Email:<input type="text" name="Email" />
  <input type="submit" />
</form>

<a href="lehrbuch.html"> Javascript Lehrbuch </a>
</body>
</html>
```

### Zugriff:

```
name = document.forms[0].elements[0].value;
name = document.myForm.Namen.value;
```



### Beispiel 2:

```
<form id="myForm" action="form.html" method="post"
  onSubmit="return test();">
  Name: <input type="text" id="Vorname"/>
  Email:<input type="text" id="Mail"/>
  <input type="submit" value="senden" />
</form>

function test () {
  var ok=true;
  if (<input type="text" id="Vorname".value=="") {
    alert("bitte Namen angeben");
    document.myForm.Vorname.focus(); ok=false;
  }
  if (document.forms[0].elements[1].value=="") {
    alert("bitte Email-Adresse angeben!");
    document.forms[0].elements[1].focus(); ok=false;
  }
  return ok;
}
```

## Die wichtigsten DOM-Objekte: window

Zugriffsmöglichkeiten auf das Browserfenster und dessen Inhalte

### Eigenschaften:

- document das angezeigte document-Objekt
- location angezeigte URL
- history beinhaltet die URLs der besuchten Seiten
- screen Informationen zum Display des Besuchers

### Methoden:

- alert(meldung) Nachricht ausgeben
- print() Druckdialog öffnen
- close() Fenster schließen (wird meist abgefangen)
- prompt(text,def) Texteingabe ermöglichen
- resizeto(x,y) Fenstergröße ändern
- confirm(meldung) Bestätigungsdialog anzeigen

### Events:

- onerror wenn ein Fehler auftritt
- onfocus wenn das Fenster den Fokus hat (aktiv ist)
- onload wenn das Fenster geladen wird
- onunload wenn das Fenster entladen wird
- onresize wenn die Fenstergröße geändert wird

## Die wichtigsten DOM-Objekte: document

Übergeordnetes Objekt für das im aktuellen Rahmen angezeigte HTML-Dokument

### Eigenschaften:

- forms Array mit allen Formularen im Dokument
- images Array mit allen Bildern im Dokument
- links Array mit allen Links des Dokuments
- location Zugriff auf die URL des angezeigten Dokuments

### Methoden:

- write(ausdruck) „In das HTML-File schreiben“
- getSelection() markierten Text auslesen

### Events:

- onclick Mausklick
- ondblclick Doppelklick
- onkeyup Taste wurde losgelassen
- onkeypress Taste wird (noch immer) gedrückt
- onkeydown Taste wurde soeben gedrückt

## - Intelligente Formulare (Mail & Bestellung prüfen)

```
function check() {
  var status=true;
  var email=<input type="text" id="Mail".value;
  if(email == "") {
    alert("Bitte geben Sie Ihre email-Adresse ein !");
    status=false;
  } else {
    kk=email.indexOf("@");
    if(kk <= 0 || email.indexOf(".", kk) < 0) {
      alert("email-Adresse ist nicht korrekt !");
      status=false;
    }
  }
  var anz=0;
  for (num=1; num<=MAX; num++) {
    anz+=parseFloat(<input type="text" id="anz"+num".value);
  }
  if(anz < 1) {
    alert("es wurde nichts bestellt !");
    status=false;
  }
  if(status)status=confirm("Es werden "+anz+" Pizza bestellt.");
  return status;
}
```

## - JS & Cookies

Cookies können nicht nur per Server sondern auch per JS gesetzt werden. Zugriff mit: `document.cookie`

```
if (document.cookie) { // existiert ein Cookie?
```

```
  alert (document.cookie);}
```

```
else {
```

```
  jetzt = new Date();
  timeout= new Date(jetzt.getTime()+1000*60);
  document.cookie = "Hallo Besucher"
  + "expires"+timeout.toGMTString() +";"
```

## JavaScript - eigene Objekte 1

### - String

selbsterklärend, sehr ähnlich zu Java

### - Math

selbsterklärend, sehr ähnlich zu Java

### - Date

selbsterklärend, sehr ähnlich zu Java

### - Array

- .join() Elemente zusammenfassen
- .sort() lexikalische Sortierung
- .reverse() Elementreihenfolge umdrehen
- .push(element,...) Anfügen
- .pop() Pop-Operation
- .slice(start, ende) Teilarray erhalten

## JavaScript - eigene Objekte 2

- RegExp zum Auswerten regulärer Ausdrücke  
Einzelne Teilstrings eines ausgewerteten Wortes, die im regulären Ausdruck in Klammern stehen, können Arrayähnlich verwendet werden.

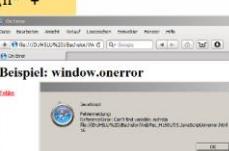
Beispiel:

```
function parseDate(datum)
{
    ra=/^(\d{1,2})[-](\d{1,2})[-](\d{2,4})$/;
    if (ra.exec(datum)==null) error=true;
    day = RegExp.$1;
    month = RegExp.$2;
    year = RegExp.$3;
}
```

### Fehlerbehandlung (1)

Mit `window.onerror` steht ein Werkzeug zur Verfügung, um Fehler innerhalb von Scripts abzufangen und darauf reagieren zu können.

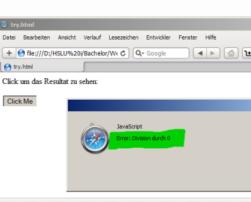
```
<html>
<head><title>On Error </title></head>
<body>
<h1>Beispiel: window.onerror</h1>
<a href="javascript:zeigeFehler()">Fehler</a>
<script>
window.onerror = Fehlerbehandlung;
function Fehlerbehandlung(Nachricht, Datei, Zeile) {
    Fehler = "Fehlermeldung:\n" + Nachricht + "\n" +
        Datei + "\n" + Zeile;
    alert(Fehler);
}
nichtda();
</script>
</body>
</html>
```



### Fehlerbehandlung (2)

```
<html>
<head>
<script>
function myFunc() {
    var a = 100; var b = 0;
    try {
        if (b == 0)
            throw "Division durch 0";
    } else {
        var c = a / b;
    }
} catch (e) {
    e = event(Error msg);
    alert("Error: " + e);
}
finally {
    alert("Finally block wird immer gemacht");
}
</script>
</head>
<body>
<p>Click um das Resultat zu sehen:</p>
<form><input type="button" value="Click Me"
    onclick="myFunc();"/></form>
</body>
</html>
```

- Exceptionhandling:  
mittels `try`, `catch` und `throw` kann der Programmfluss kontrolliert und context-spezifische Fehlermeldungen erzeugt werden



### Ausgabe in Konsole

```
<script>
    resultat = Summe("Hello", "World");
    console.log("Das Ergebnis: " + resultat);
</script>
```

Name oder ID

Name wird eigentlich nur in Formularen verwendet und zwar bei den `<input>` Elementen um diese zu gruppieren. Nur Formularelemente mit Namen werden an Server gesendet

Zugriff: direkt, Name oder ID

- direkt via DOM-`"Pfad"` mit Element-Namen

```
...
<form name="Formular" action="" method="get">
    <input type="radio" name="Favoriten" value="Heino">
    <input type="radio" name="Favoriten" value="Gildo">
    <input type="radio" name="Favoriten" value="Marianne">
</form>
<script>
    document.Formular.Favoriten[2].checked = true;
</script>
...
```

Name

- per JavaScript kann mittels: `getElementByName()`

ID

- per JavaScript kann mittels: `getElementById()` ein spezifisches Element ausgewählt werden

### Objekt

Folgendes Beispiel erzeugt ein Objekt mit 3 properties:

```
var person = new Objekt ();
person.Vorname = "Susie";
person.Nachname = "Sorglos";
person.Alter = 22;
```

Objekte können auch verändert werden, sie sind "addressed by reference" und nicht "by value":

```
var person = {Vorname:"Susie", Nachname:"Sorglos", Alter:22};

var teen = person; // teen ist keine Kopie sondern referenziert person!
teen.Alter = 10; // teen.Alter und person.Alter wurden geändert!
```

### Properties

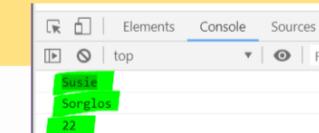
Properties sind Werte welche mit einem Objekt assoziiert sind. Sie können geändert, gelöscht, hinzugefügt werden aber einige sind auch "read only"

Man kann unterschiedlich auf die Properties zugreifen:

```
person.Vorname + " ist " + person.Alter + " Jahre alt";
person["Vorname"] + " ist " + person["Alter"] + " Jahre alt";
```

Oder:

```
var person = {Vorname:"Susie", Nachname:"Sorglos", Alter:22};
for (let x in person){
    console.log(person[x] + " ");
}
```



### Neue properties hinzufügen/entfernen

```
var person = {Vorname:"Susie", Nachname:"Sorglos", Alter:22};
person.Nationalitaet = "Schweiz";
var person = {Vorname:"Susie", Nachname:"Sorglos", Alter:22};
delete person.Alter;
```

### Objekt Methoden

- Im folgenden Beispiel ist `this` das Objekt `person` Besitzer der Funktion `vollerName`:

```
var person = {
    Vorname:"Susie",
    Nachname:"Sorglos",
    Alter:22,
    vollerName : function () {
        return this.Vorname + " " + this.Nachname; ==> Output: Susie Sorglos
    }
};
```

- Methoden sind als Objekt-Properties gespeicherte Funktionen!

### Getter und Setter Methoden (get/set)

```
var person = {
    Vorname:"Susie",
    Nachname:"Sorglos",
    Alter:22,
    Sprache: "de",
    get lang(){ return this.Sprache; }
};
var sprache = person.lang; // → "de"

var person = {
    Vorname:"Susie",
    Nachname:"Sorglos",
    Alter:22,
    Sprache: "de",
    set lang(){ this.Sprache = lang; }
};
person.lang("fr");
person.lang="fr";
```

### Zugriff auf Objekt Methoden

```
var name = person.vollerName(); // → "Susie Sorglos"
```

### window Objekt

Repräsentiert Fenster eines Browsers. Beinhaltet `document`, `location`, `history`.

Beispiel: 1.Seite neu laden 2.Seite zurück

```
<html>
<head><title>new Location</title>
<script>
    function newDoc() {
        window.location.assign("http://www.w3schools.com")
    }
</script>
</head>
<body>
    <input type="button" value="Load new document" onclick="newDoc()">
</body>
</html>

<script>
function goBack() {
    window.history.back()
}
</script>
```

## - document objekt

```
<script>
// example: traversing a document
var a = document.getElementsByTagName('a'),
a_len = a.length,
i, title;
for (i=0; i < a_len; i++) {
  title = a[i].getAttribute('title');
  if (title) {
    console.log(title);
  }
}
</script>
```

alle links in variable

## - Mathematische Operatoren

- String: "123" wird zu 123  
"abc" wird zu NaN

- Boolean: false wird zu 0  
true wird zu 1

- Null: 0

- Undefined: NaN

- Object: `toString()` zu String zu Number

(NaN: not a number)

"abc" + "def" // ergibt "abcdef"  
"5" + 5 // ergibt "55" ↪ nicht wahr da +(Ausnahme)  
"5" - 5 // ergibt 0 (mathematische Operation) ↪ wahr da -  
null + 5 // "null5" ↪ nicht wahr, da String + Null=String

== versus ==

- Der "strikte" equality Operator === berücksichtigt nur Werte vom gleichen Typ

Gleichheit  
Gleichwertigkeit

- Der "nachsichtige" equality Operator == versucht zuerst Werte von unterschiedlichen Typen zu konvertieren und erst dann zu vergleichen

## CANVAS

- Mit `<canvas>` wurde ein HTML-Element eingeführt, mit dem dynamisch Bitmap-Grafiken gerendert und in Dokumente integriert werden können.

```
<body onload="draw()>
<canvas id="canvas" width="300" height="300"></canvas>

<script>
function draw() {
  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');
  if(canvas.getContext) {
    context.fillStyle = "rgb(0, 0, 255)";
    context.fillRect(10, 10, 200, 200);
  }
}
</script>
</body>
</html>
```

gleiche ID!

"Zeichenfläche" holen

## Das Grundgerüst - Erklärungen

holt das `<canvas>`-Element anhand seiner ID

```
var canvas = document.getElementById("canvas");
```

holt den 2D-Context in die handliche context-Variable. Das wird benötigt, weil letztlich nicht auf dem `<canvas>`-Element selbst, sondern auf einem Context des Elements gezeichnet wird.

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

setzt die Füllfarbe für alles, was danach im Context gezeichnet wird, auf Blau

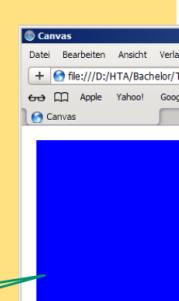
```
context.fillStyle = "rgb(0, 0, 255);
```

zeichnet ein gefülltes Rechteck, beginnend bei den Koordinaten 0,0 und endend mit 200,200

```
context.fillRect(10, 10, 200, 200);
```

### Beispiel

```
<html>
<head>
<title>Canvas</title>
<script>
function draw() {
  var canvas = document.getElementById("canvas");
  var context = canvas.getContext('2d');
  if(canvas.getContext) {
    context.fillStyle = "rgb(0, 0, 255)";
    context.fillRect(10, 10, 200, 200);
  }
}
</script>
</head>
<body onload="draw()>
<canvas id="canvas" width="300" height="300">
</canvas>
</body>
</html>
```



und hier das ist das Ergebnis

## Fehlermeldung bei Browserinkompatibilität

```
<body onload="draw()>
<canvas id="testcanvas" width="800px" height="200px">
<p>Dieses Beispiel benötigt einen Webbrowser mit aktivierter Canvas-Unterstützung.</p>
</canvas>

<noscript>
<p>Dieses Beispiel benötigt einen Webbrowser mit aktivierter JavaScript-Unterstützung.</p>
</noscript>
</body>
```

### - Linien zeichnen

`lineTo (x-Koordinate, y-Koordinate)`

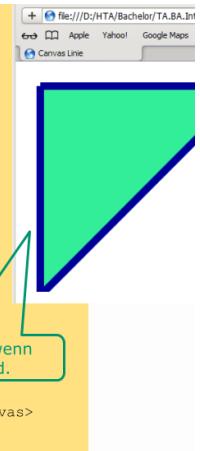
Cursor bewegen:

`moveTo (x-Koordinate, y-Koordinate)`

## Beispiel

```
<html>
<head>
<title>Canvas Linie</title>
<script>
function draw () {
  var canvas = document.getElementById("canvas");
  var ctx = canvas.getContext("2d");
  ctx.beginPath();
  ctx.lineWidth=10;
  ctx.moveTo(25,25);
  ctx.lineTo(305,25);
  ctx.lineTo(25,305);
  ctx.lineTo(25,25);
  ctx.fillStyle="#33ee99";
  ctx.fill();
  ctx.strokeStyle="#000099";
  ctx.stroke();
  ctx.closePath();
}
</script>
</head>
<body onload="draw()>
<canvas id="canvas" width="300px" height="300px"></canvas>
</body>
</html>
```

Und so sieht es aus, wenn beides kombiniert wird.



## Kreise – Kurven – Segmente (2)

- Mit Hilfe der Methode `arc` können Kreise und Kreisbögen gezeichnet werden. Gezeichnet wird der Kreis(-bogen) ausgehend vom Mittelpunkt (x,y). An diesen wird der Radius angebracht.

- Die Länge des Kreisbogens hängt vom Start- und Endwinkel ab, der von der positiven x-Achse aus gemessen wird.

`arc(x, y, Radius, startWinkel, endWinkel, Richtung);`

x: x-Koordinate des Kreismittelpunkts

y: y-Koordinate des Kreismittelpunkts

startWinkel: Startwinkel in rad, gemessen von der positiven x-Achse

endWinkel: Endwinkel in rad, gemessen von der positiven x-Achse

Richtung: true=gegen den Uhrzeigersinn, false=mit dem Uhrzeigersinn

## Segmente 1

`arc(x, y, Radius, startWinkel, endWinkel, Richtung);`



```
context.beginPath();
context.arc(100, 100, 50, 0, 1.5 * Math.PI, false);
context.lineWidth = 10;
context.stroke();
```



```
context.beginPath();
context.arc(100, 100, 50, 0, Math.PI, false);
context.lineWidth = 10;
context.stroke();
```



```
context.beginPath();
context.arc(100, 100, 50, Math.PI, 2 * Math.PI, false);
context.lineWidth = 10;
context.stroke();
```

## Textdarstellung

```
function draw() {
    var canvas = document.getElementById("canvas");
    var context = canvas.getContext('2d');
    var x=20;
    var y=80;

    if(canvas.getContext) {
        context.font = "36pt Helvetica";
        context.fillStyle = "red";
        context.lineWidth=4;
        context.strokeStyle="#4488aa";
        context.fillText("Reflection", x, y,400);
        context.font = "58pt Verdana";
        context.strokeText("Reflection", x,2* y);
    }
}
```



## Beispiel allgemein

```
<script>
function draw() {
    var canvas = document.getElementById("canvas");
    var context = canvas.getContext('2d');

    context.beginPath();
    context.moveTo(200,38);
    // Ober Ecke
    context.lineTo(350,300);
    // Rechte Ecke unten
    context.lineTo(50,300);
    // Linke Ecke unten
    context.lineTo(200,38);
    context.strokeStyle="#ff0000";
    context.lineWidth="16";
    context.lineJoin="round";
    context.fillStyle="#ffff00";
    context.fill();
    context.stroke();

    context.fillStyle="#787878";
    context.fillRect(185,308,30,300);

    // Ausrufezeichen einfügen
    context.beginPath();
    context.font="120pt Verdana";
    context.fillStyle="#343434";
    context.fillText("!",170,250);
    context.closePath();
}
</script>
```

```
...
<body onload="draw()">
<canvas id="canvas"
width="800px" height="800px"></canvas>
</body>
```



## Bestimmte Seite laden:

```
window.location.assign("http://www.xx.ch")
```

## Eine Seite zurück gehen:

```
window.history.back()
```

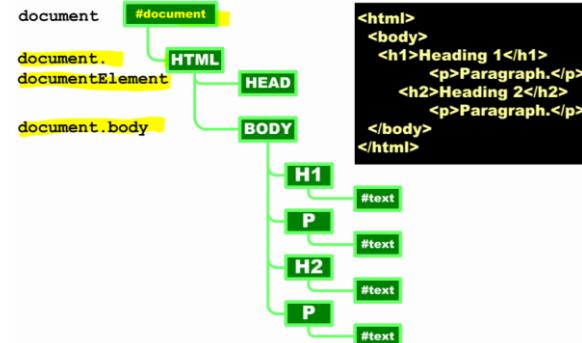
## HTTP Strict Transport Security (HSTS)

- HTTP Strict Transport Security (HSTS) ist eine Security-Erweiterung, welche Zugriffe eines Clients (Browser) auf einen Server nur mittels HTTPS erlaubt.
- Dadurch wird sichergestellt, dass die Verbindung nicht über eine unsichere HTTP-Verbindung aufgebaut werden kann, die potentiell anfällig für Angriffe sein könnte.
- HSTS ist im Response-Header als Strict-Transport-Security definiert und sobald der unterstützte Browser diesen Header erhält, weiß er, dass er alle Informationen über HTTPS liefern kann.
- HSTS ist definiert in RFC-6797 (<https://tools.ietf.org/html/rfc6797>)

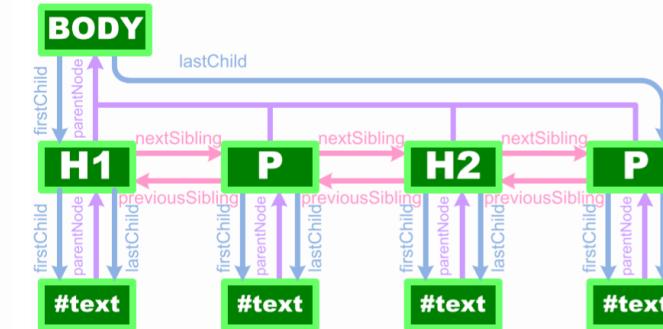
## Frameworks

Grundlagen aller Frameworks & Libraries ist DOM.  
DOM ist bereits sobald alles von der Seite geladen ist.

### - DOM- Document Tree Structure



### - Navigation innerhalb des DOM (child, sibling, parent)



## jQuery

jQuery ist eine kompakte, OpenSource JavaScript Bibliothek welche die Interaktion zwischen HTML/CSS und JavaScript erheblich vereinfacht.

### - Vorarbeiten

jQuery muss zuerst als Bibliothek geladen werden.

```
<script src="jquery.js"/>
```

Im JS File "referenzieren":

```
var script = document.createElement('script');
script.src = '//code.jquery.com/jquery-1.11.0.min.js';
document.getElementsByTagName('head')[0].appendChild(script);
```

### - Beispiel: 1.vorher 2.nachher

```
<style>
.green { color:#009933; }
</style>
<script>
window.onload = function() {
    var elements = document.getElementById("box").getElementsByTagName("p");
    for (var i = 0; i < elements.length; i++) {
        if (elements[i].firstChild.data == "Zweiter Absatz") {
            elements[i].className = "green";
        }
    }
}</script></head>
<body>
<div id="box">
<p>Erster Absatz</p>
<p>Zweiter Absatz</p>
<p>Dritter Absatz</p>
</div>
</body>
</html>
```

```
<html><head>
<title>Nun mit jQuery</title>
<style>
.green { color:#009933; }
</style>
<script src="js/jquery-1.6.2.js"></script>
<script>
$(document).ready(function() {
    $('#box p:contains("Zweiter Absatz")').addClass("green");
});
</script></head>
<body>
<div id="box">
<p>Erster Absatz</p>
<p>Zweiter Absatz</p>
<p>Dritter Absatz</p>
</div>
</body>
</html>
```

## jQuery Konzept

**\$()** ist nur ein Shortcut für **jQuery()**

- Die "magische" **\$()** Funktion

- z.B. einfach HTML Elemente erzeugen:

```
var element = $("<div/>")
```

- Manipulation existierenden DOM Elemente

```
$(window).width()
```

- Auswählen von **document** Elements

```
$("div").hide();
$("div", $("p")).hide();
```

- **jQuery Philosophie: GET → ACT**

```
$(“div”).hide()
$(“<span/>”).appendTo(“body”)
$(“:button”).click()
```

Get

Act

## Verkettung

- Praktisch jede Funktion liefert ein jQuery Element, das ermöglicht einen "fliessenden" Programmierstil und Verkettungsmöglichkeiten:

```
$("div").show().addClass("main").html("Hello jQuery");
```

### - Auswählen – Selectoren (Mehr am Schluss)

#### - Mit dem Tag:

```
$("#div")  
// <div>Hello jQuery</div>
```

#### - Mit der ID

```
$("#usr")  
// <span id="usr">John</span>
```

#### - Mit der Klasse

```
$(".menu")  
// <ul class="menu">Home</ul>
```

#### - Exakter:

```
$("div.main") // tag and class  
$("table#data") // tag and id
```

#### - Kombination

```
// find by id + by class  
$("#content .menu")  
// multiple combination  
$("h1, h2, h3, div.content")
```

#### - Hierarchisch

```
$("table #id") // descendants  
$("tr > td") // children  
$("label + input") // next  
$("#content ~ div") // siblings
```

#### - per Index

```
$("tr:first") // first element  
$("tr:last") // last element  
$("tr:lt(2)") // index less than  
$("tr:gt(2)") // index gr. than  
$("tr:eq(2)") // index equals
```

#### - Sichtbarkeit

```
$("div:visible") // if visible  
$("div:hidden") // if not
```

#### - Attribute

```
$("div[id]") // has attribute  
$("div[dir='rtl']") // equals to  
$("div[id^='main']") // starts with  
$("div[id$='name']") // ends with  
$("a[href*='msdn']") // contains
```

## - Formulare

```
$("input:checkbox") // checkboxes  
$("input:radio") // radio buttons  
$("button") // buttons  
$("text") // text inputs
```

## - Filter

```
$("input:checked") // checked  
$("input:selected") // selected  
$("input:enabled") // enabled  
$("input:disabled") // disabled
```

## - Beispiel Selektoren: 1. Selektiertes Element 2. jQuery

```
<select name="cities">  
<option value="1">London</option>  
<option value="2" selected="selected">Rom</option>  
<option value="3">Paris</option>  
</select>
```

```
$("select[name='cities'] option:selected").val()
```

### - Document Traversal

Ein Selector gibt ein "Pseudo Array" von jQuery-Objekten zurück!

folgendes ergibt die Anzahl der ausgewählten Elemente

```
$( "div" ).length
```

und das retourniert das 2te DOM Element der Auswahl

```
$( "div" ).get(2) oder $( "div" )[1]
```

dieses hingegen retourniert des 2te jQuery Element

```
$( "div" ).eq(2)
```

**each(fn)** traversiert jedes selektierte Element und ruft fn() auf

```
var sum = 0;  
$("div.number").each(  
    function(){  
        sum += (+this.innerHTML);  
    });
```

**this** – ist das aktuelle DOM Element

**each(fn)** arbeitet auch als Indexer ("Durchnummrierter")

```
$("table tr").each(  
    function(i){  
        if (i % 2)  
            $(this).addClass("odd");  
});
```

**\$(this)** – konvertiert DOM nach jQuery

i - Index des aktuellen Elements

## - Suchen in der Auswahl

```
// select paragraph and then find  
// elements with class 'header' inside  
$("p").find(".header").show();
```

// equivalent to:

```
$(".header", $("p")).show();
```

## - Verkettung

```
$("<li><span></span></li>") // li  
.find("span") // span  
.html("About Us") // span  
.andSelf() // span, li  
.addClass("menu") // span, li  
.end() // span  
.appendTo("ul.main-menu"); // li
```

## - HTML-Manipulationen

### - Setzen oder auslesen des "Inner Content"

```
$("p").html("<div>Hello $!</div>");  
// escape the content of div.b  
$("div.a").text($(".div.b").html());
```

### - Ersetzen von Elementen

```
// select > replace  
$("h1").replaceWith("<div>Hello</div>");  
// create > replace selection  
$(<div>Hello</div>).replaceAll("h1");
```

### - Setzen oder auslesen von Werten

```
// get the value of the checked checkbox  
$("input:checkbox:checked").val();  
// set the value of the textbox  
$(":text[name='txt']").val("Hello");  
// select or check lists or checkboxes  
$("#lst").val(["NY", "IL", "NS"]);
```

## - CSS-Manipulationen

```
// get style  
$("div").css("background-color");
```

```
// set style  
$("div").css("float", "left");
```

```
// set multiple style properties  
$("div").css({color: "blue",  
padding: "1em",  
margin-right: "0px",  
margin-left: "10px"});
```

## CSS Klassen Manipulationen

```
// add and remove class
$("p").removeClass("blue").addClass("red");
```

```
// add if absent, remove otherwise
$("div").toggleClass("main");
```

```
// test for class existence
if ($("#div").hasClass("main")) { /*...*/ }
```

### - Effekte

#### Anzeigen und ausblenden

```
// just show
$("#div").show();
// reveal slowly, slow=600ms
$("#div").show("slow");
// hide fast, fast=200ms
$("#div").hide("fast");
// hide or show in 100ms $("#div").toggle(100);
```

#### Sliding und Fading

```
$("#div").slideUp();
$("#div").slideDown("fast");
$("#div").slideToggle(1000);
$("#div").fadeIn("fast");
$("#div").fadeOut("normal");
// fade to a custom opacity
$("#div").fadeTo("fast", 0.5);
```

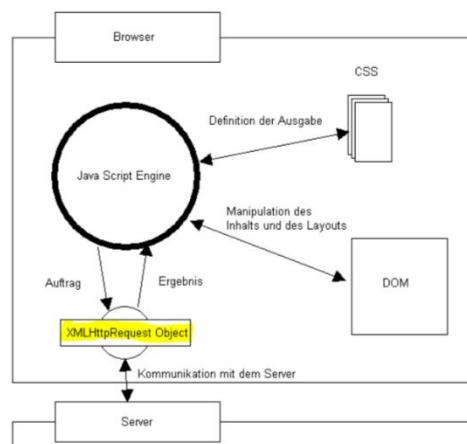
#### - Animation

```
$("#div").animate({width: "90%"},100)
    .animate({opacity: 0.5},200)
    .animate({borderWidth: "5px"});
```

## AJAX (Asynchronous JS and XML)

Es bezeichnet ein Konzept der asynchronen Datenübertragung zwischen einem Server und dem Browser, das es ermöglicht, innerhalb einer HTML-Seite eine HTTPAnfrage durchzuführen, ohne die Seite komplett neu laden zu müssen.

Das eigentliche Novum besteht in der Tatsache, dass nur gewisse Teile einer HTML-Seite oder auch reine Nutzdaten sukzessiv bei Bedarf nachgeladen werden, womit Ajax eine Schlüsseltechnik zur Realisierung des Web 2.0 darstellt.



### - 1. AJAX-Request 2.Fehlerbehebung 3.Bilder nachladen

```
<body>
<p id="output"></p>
<script>
xhr = new XMLHttpRequest();
xhr.onload = function() {
let output = document.getElementById('output');
output.innerText = xhr.responseText;
}
xhr.open('GET', 'http://192.168.56.101/hello.txt', true);
xhr.send();
</script>

```

Instanz eines XMLHttpRequest-Objektes erzeugen  
Weiterverarbeitung des Ergebnisses bei Erfolg  
Ggf. URL/IP und Pfad anpassen  
asynchron

```
xhr = new XMLHttpRequest();

// Fehler aufgetreten
xhr.onerror = function() {
let output = document.getElementById("output");
output.innerText = "Fehler bei Abruf von hallo.txt";
}

// Zeitüberschreitung (Server antwortet nicht)
xhr.ontimeout = function() {
let output = document.getElementById("output");
output.innerText = "Zeitüberschreitung bei Abruf von hallo.txt";
}
```

Simulieren z.B. mittels Aufruf einer nicht existierenden URL  
Simulieren mittels timeout von einer Millisekunde: xhr.timeout = 1;

```
</head><body>
<button id = "more" onClick="load();>Load more...</button>
<script>
function load() {
xhr = new XMLHttpRequest();
xhr.onload = function() {
// füge neue Reihe vor dem "load more" Button ein
let more = document.getElementById("more");
more.insertAdjacentHTML("beforebegin", xhr.responseText + "<br>");

more.scrollIntoView(); // button soll sichtbar bleiben
if (row == 10) { more.remove(); } // entferne Button bei der letzten Reihe
}
xhr.open("GET", "row" + row + ".html", true);
xhr.send();
++row;
}
let row = 1;
load(); // erste Bildreihe
</script>
</body></html>
```

xhr.responseText: Enthält bereits gültiges HTML  
Inhalt von row-<n>.html:  
  
  
  
Ein File pro Zeile. <n> ersetzt durch Zeilennummer. Inhalt in diesem Beispiel statisch, typischerweise aber dynamisch

### - mit jQuery bild nachladen

per Mausklick einen Text vom Server nachladen und "animiert" in der Website einblenden.

```
<head> <!-- jQuery einbinden -->
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
// JavaScript-Code hier...
$(document).ready(function() {
$("a").click(function() {
$.get("test.txt", function(text) {
$("p:first").fadeOut('fast',function() {
$("p:first").html(text).fadeIn('normal');
});
});
});
});
</script>
</head>
<body>
<p>Bitte auf den Link klicken um den Text zu laden!</p>
<p><a href="#">Datei test.txt laden...</a></p>
</body>
```

### - JSON Syntax (Unterschied JS-Schlüsselname mit '')

```
{
  "name" : "Muster",
  "vorname" : "Hans",
  "adresse" : "Musterstrasse 23",
  "stadt" : "Musterstadt",
  "plz" : 1234, Numerische Werte sind unterstützt (keine Anführungszeichen)
  "geburtsdatum" : "1.1.1980", Date nicht unterstützt. Als String speichern und in Date-Objekt umwandeln.
  "anstellung" : { "firma" : "Mustermann", "seit" : "1.1.2000" },
  "kinder" : [ "Anna", "Tim", "Nils" ]
}
```

Verschachtelte Objekte wie in JavaScript  
Arrays wie in JavaScript

### Umwandlung JS-Objekt nach JSON (stringify)

```
let person = { name: "Muster",
  vorname: "Hans",
  adresse: "Musterstr. 23" };
```

```
let personJSON = JSON.stringify(person); JSON-Objekt ist Teil des JavaScript-Sprachumfangs
console.log(personJSON);
```

### Ausgabe:

```
{"name":"Muster", "vorname":"Hans", "adresse":"Musterstr. 23"}
```

### Vorteile JSON

Kompakt, effizient, sehr »programmiersprachenähnlich«

Nachteile: nicht erweiterbar

Nachteil XML: sehr ausführlich (Platzverschw.)

Überführung in programmiersprachige Datael. Aufwändig.

## Umwandlung JSON nach JavaScript

```
let personJSON = {"name": "Muster", "vorname": "Hans", "adresse": "Musterstr. 23" };

let person = JSON.parse(person);
Achtung: parse wirft Exception bei fehlerhaften Eingabedaten

console.log("Name: " + person.name + " / Vorname: " + person.vorname);
```

**Ausgabe:**  
Name: Muster / Vorname: Hans

### - JSON & AJAX

jQuery kann mittels Funktion each auf die Schlüsselwertpaare über auf JSON-Elemente iterieren:

```
$.getJSON("http://192.168.56.101/muster.json", function(data) {
    $.each(data, function(key, val) {
        console.log("Key: " + key + " / value: " + val);
    });
});
```

muster.json:

```
{"name": "Muster",
 "vorname": "Hans",
 "adresse": "Musterstr. 23"}
```

**Ausgabe**  
Key: name / value: Muster  
Key: vorname / value: Hans  
Key: adresse / value: Musterstr. 23

## Geolocation/Web Storage

### - Geolocation

Abfrage aktueller Standort (Sport, Route, Wetter, Buchen)

#### 1.Verfügbarkeit prüfen 2.Positionsabfrage

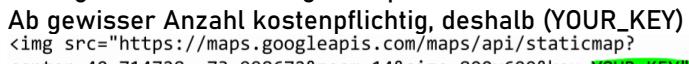
```
function getLocation() {
    if (navigator.geolocation) {
        // Geolocation-Funktionalität verfügbar
    } else {
        // Geolocation-Funktionalität nicht verfügbar
    }
}
```

```
<body>
<p id="output"></p>
<script>
    function getLocation() {
        navigator.geolocation.getCurrentPosition(zeigePosition);
    }

    function zeigePosition(position) {
        document.getElementById("output").innerHTML =
            "Breite: " + position.coords.latitude + " / " +
            "Länge: " + position.coords.longitude;
    }

    // Ausgabe
    getLocation();
</script>
</body>
```

### Anzeige auf Karte in Google-Maps

Ab gewisser Anzahl kostenpflichtig, deshalb (YOUR\_KEY)  
  
 center=40.714728,-73.998672&zoom=14&size=800x600&key=YOUR\_KEY" key sichtbar, Absicherung auf best. IP/URL ok. Rest block

## Kontinuierliches Tracking

```
<body>
    <p id="output"></p>
    <button onclick="start()">Start</button>
    <button onclick="stop()">Stop</button>

    <script>
        function start() {
            watchId = navigator.geolocation.watchPosition(showPosition);
        }

        function stop() {
            navigator.geolocation.clearWatch(watchId);
        }

        function showPosition(position) {
            document.getElementById("output").innerHTML =
                "Breite: " + position.coords.latitude +
                " / Länge: " + position.coords.longitude;
        }
    </script>
</body>
```

### - WebStorage

Daten können lokal im Browser als Key/value-pairs gespeichert werden. Unterschied zu Cookies, müssen nicht immer mitgeschickt werden. Vorteile/Idee:  
 Offline Verfügbar, Schneller Zugriff(lokal), speichern von Daten lokal, später sync, wenn online (Mails, Dokumente)  
**Zugriffskontrolle (per Domain &Protokoll)**  
 Nur Seiten von der gleichen Domain und dem gleichen Protokoll können auf die Daten zugreifen ([http](http://) != [https](https://)). Subdomains (sub.example.com) werden dabei als eigenständige Domain gesehen.

WebStorage Typen (Ablaufdatum: 1. Ohne, 2. Sessionende)  
 1. localStorage: Behalte Daten ohne Ablaufdatum  
 2. sessionStorage: Behalte Daten bis Browser (Tab) geschlossen(Auch Private Browsing(localStorage))

#### 1. Verfügbarkeit 2.Verwendung

```
if (typeof(Storage) == "undefined") {
    // keine Unterstützung von Web Storage
}
```

**Speichern** eines Wertes <VALUE> für Schlüssel <KEY>:  
 localStorage.setItem(<Schlüssel>, <Werte>);  
 Beispiel: localStorage.setItem("name", "Anna");

**Abfragen** des Schlüssels <KEY>:  
 <WERT> = localStorage.getItem(<KEY>);  
 Beispiel: localStorage.getItem("name"); // returns "Anna"

**Entfernen** des Schlüssels <KEY> (inklusive Wert):  
 localStorage.removeItem(<KEY>);  
 Beispiel: localStorage.removeItem("name");

### - WebWorkers (Parallele Ausführung)

Ziel: Schnell mit Nutzung mehrer Ausführungseinheiten. Wichtig: Eine Webseite arbeitet mittels Events.

(Reihenfolge wichtig). Gleichzeitige Ausführung mehrere Events und somit die Nutzung mehrere Ausführungseinheit ist nicht möglich.

Webworkers sind separate Ausführungsstränge, welche neben der Event-Verarbeitung laufen.

Auslagerung von rechenintens. Aufgaben an WebWorkers

1. Verfügbarkeit 2. Verwendung Hauptthread/3.WebWorker

```
if (typeof(Worker) == "undefined") {
    // keine Unterstützung von Web Workers
}
```

Schritt 1: Worker-Object erstellen: Parameter ist eine URL!  
 worker = new Worker("WebWorkerChild.js");

Schritt 2: Auf Resultat warten mittels Message-Eventhandler:  
 worker.onmessage = function(event) {
 let result = event.data;
 // Verarbeite Resultat:
 console.log("Resultat: " + result);
 }

Schritt 3: Arbeit übergeben (mehrfach wiederholbar):  
 worker.postMessage(work);

Schritt 4: Worker beenden:  
 worker.terminate();

```
onmessage = function(event) {
    let work = event.data;
    // Berechne das Resultat
    let res = calculate(work);
    // Melde das Resultat
    postMessage(res);
}
```

Oben: teil in body/script, sendet Daten zur Berechnung an WebWorker. Welcher den code in separatem JS file ausführt. Bild links

### - WebSockets

Konzept: Server kann Daten nach Verbindungsaufbau mittels Messages direkt an Client schicken. Ohne Anfrage/Antwort-Modell.

WO/Wann?: Chat, kooperative Textverarbeitung, Resultatmeldung, Handelsplattformen, multiplayer games, etc.

1. Client baut Verbind. zu Server auf(Prot: ws/wss)
2. Senden/Empfangen von Messages

Client

Socket

senden →  
empfangen ←

Server

Socket  
empfangen  
senden

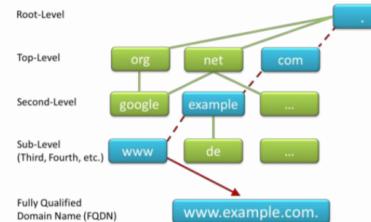
# DNS

DNS ist ein verteiltes, hierarchisches System zur Konvertierung von Rechnernamen (URLs) in IP -Adressen. DNS stellt ein hierarchischer Namensraum dar:

- An der Spitze steht Root (als Punkt dargestellt).
- auf der nächsten Ebene sind die **Top-Level-Domains** wie com, ch und andere angeordnet.

Jede Top-Level-Domain wird in **Second Level Domains** unterteilt, welche wiederum im mehrere Subdomains geteilt werden können

Am Schluss kommt noch der Hostnamen hinzu so dass der **FQDN** (fully qualified domain name) gebildet wird wie z.B. [www.example.com](http://www.example.com)



Zonen definieren die logische oder räumliche Zusammengehörigkeit von Rechnergruppen.

## HTTP-Requests

Requests-Header weisen folgende Struktur auf:

METHOD URL HTTP/version

General Header

Request Headers

Entity Header (optional)

Leerzeile

Request Entity (falls vorhanden)

Eine Request, der eine HTML-Seite anfordert, sieht beispielsweise so aus:

GET

<http://www.bla.de/verzeichnis1/seite2.html>

HTTP/1.1

Date Thursday, 14-Oct-99 17:55 GMT

User-agent: Mozilla/4.6

Accept: text/html, text/plain

## HTTP-Response

Der Aufbau einer HTTP-Response ist ähnlich zur Request:

HTTP/version Status-Code Reason-Zeile

General Header

Response Header

Entity Header (optional)

Leerzeile

Resource Entity (falls vorhanden)

Eine komplette Response, die eine HTML-Datei vom Server übermittelt, sieht beispielsweise so aus:

```

HTTP/1.1 200 OK
Via: HTTP/1.1 proxy_server_name
Server: Apache/1.3
Content-type: text/html, text, plain
Content-length: 78
<html>
<head>
<title>HTTP</TITLE>
</head>
<body>
<p> HTTP/1.1-Demo</p>
</body>
</html>
  
```

## HTTP Response Codes

**1xx:** Informelle Meldungen: Request erhalten, Bearbeitung wird durchgeführt.

**2xx:** Erfolg: Request wurde erfolgreich erhalten, verstanden und angenommen.

**3xx:** Weiterleiten: Weitere Aktionen müssen eingeleitet werden, damit eine Request vollständig bearbeitet werden kann.

**4xx:** Clientfehler: Die Request enthält ungültigen Syntax oder kann nicht bearbeitet werden.

**5xx:** Serverfehler: Der Server kann eine gültige Request nicht bearbeiten.

### GET-Methode

Die mit Abstand wichtigste Methode ist GET. Sie dient zur Anforderung eines Dokuments oder einer anderen Quelle. Eine Quelle wird dabei durch den Request-URL identifiziert. Man unterscheidet zwei Typen: conditional GET und partial GET. Beim Conditional -GET-Typ ist die Anforderung von Daten an Bedingungen geknüpft. Die genauen Bedingungen sind dabei im Header-Feld "Conditional" hinterlegt

### POST-Methode

Den umgekehrten Weg nimmt die POST-Methode: Sie übermittelt in erster Linie Formulareingaben an einen Webserver. Aber auch die Kommentierung bestehender Quellen, Übermittlung von Nachrichten an Foren und Erweiterung von Online-Datenbanken sind mit POST möglich. Die an den Server übermittelten Daten sind in der Entity-Sektion enthalten. Auch die POST-Methode übermittelt einen URL. In diesem Fall dient dieser

lediglich als Referenz, welche Routine auf dem Server die Bearbeitung der Daten übernimmt.

### OPTIONS-Methode

Über diese Methode kann der Client Informationen über verfügbare Kommunikationsoptionen abrufen. So lassen sich insbesondere Beschränkungen von Quellen auf einem HTTP-Server oder auch einem Proxyserver ermitteln, ohne das eine bestimmte Aktion eingeleitet oder gar ein Datentransfer stattfindet.

### HEAD-Methode

Diese Methode ist GET in seiner Funktionsweise sehr ähnlich. Einziger Unterschied: HEAD fordert lediglich den Header eines Dokuments oder Quelle an. Im Gegensatz zu GET übermittelt der Server aber nicht die eigentlichen Daten. HEAD eignet sich insbesondere dazu, die Größe von Quellen, Typ oder Objektattribute ausfindig zu machen. Der Server übermittelt auf eine HEAD-Anfrage die Metainformationen, die identisch mit den Informationen der GET-Request sind.

### PUT-Methode

Dieser Typ erlaubt die Modifikation bestehender Quellen beziehungsweise Erzeugung neuer Daten auf dem Server.

### DELETE-Methode

Mit Hilfe dieses Typs werden Daten auf dem HTTP-Server gelöscht, die durch den URL identifiziert sind.

### TRACE-Methode

Über diese Methode kann der Client Requests verfolgen, die über mehrere Knotenpunkte laufen. Dies ist insbesondere bei der Übermittlung der Request über einen oder mehrere Proxyserver interessant. Das letzte Glied der Kette generiert die Antwort.

### CONNECT-Methode

Die CONNECT-Methode ist in der HTTP/1.1-Spezifikation für Verbindungen reserviert, bei denen Proxyserver dynamisch als Tunnel agieren. In der Praxis kann es sich beispielsweise um SSLTunnel handeln.

# Webserver

Wichtige Logs bei Ubuntu/Apache:  
 /var/log/apache2/error.log  
 /var/log/apache2/access.log

## Serverseitiges Scripting CGI

(Common Gateway Interface) ist eine vordefinierte Schnittstelle des Webserver um Daten an einen verarbeitenden Prozess zu schicken und anschliessend die Resultate der Verarbeitung wieder entgegenzunehmen. Für die CGI-Schnittstelle muss der Webserver-Software 3 Dinge zur Verfügung stellen:

- Umgebungsvariablen (z.B. SERVER\_NAME)
- Weiterleitung von Ausgaben,
- Einholen von Formulareingaben oder Aufrufparametern

## SSI

Server Side Includes ist eine "Spezialität" vom Apache, ist aber auch beim IIS möglich. Wenige, einfache Befehle welche direkt in die HTML Seite eingebettet werden. Diese werden durch den Webserver ausgeführt. Website-Besucher sehen SSI Befehle nie sondern immer nur deren Ergebnis nach der Verarbeitung!

```
<!--#echo var="DOCUMENT_NAME"-->
```

Statt DOCUMENT\_NAME können die folgenden Variablen verwendet werden:

AUTH_TYPE	Autorisation des Clients, wenn vorhanden
CONTENT_LENGTH	Umfang einer Benutzereingabe
CONTENT_TYPE	MIME Typ
DATE_GMT	Greenwich Mean Time (GMT)
DATE_LOCAL	Datum u. Uhrzeit, lokale Server-Zeit
DOCUMENT_NAME	Name der Datei, in der der Befehl steht
DOCUMENT_URI	Pfad zu der Datei, in der der Befehl steht
LAST_MODIFIED	Datum der letzten Änderung einer Datei
PAGE_COUNT	Zugriffe auf eine Datei seitdem sie online ist
HTTP_REFERER	URL von der ein Besucher kam

REMOTE_ADDR	IP Adresse des Benutzers
REMOTE_HOST	Domain Name des Besuchers
REMOTE_USER	User ID (sofern vorhanden)
REQUEST_METHOD	HTTP Methode (GET oder POST)
SERVER_NAME	Hostname des Servers
SERVER_PORT	Benutzer Port (für gewöhnlich 80)
SERVER_PROTOCOL	HTTPD Version
SERVER_SOFTWARE	Verwendete Server Software und Version
TOTAL_HITS	Summe der Ausgelieferten Seiten seitdem der Server online ist
QUERY_STRING	übergebene Argumente

## PHP

Wichtige Konfiguration in php.ini:

register\_globals=off

Verhindert direktes Überscheinen von Variablen durch Get/Post/Cookies übermittelte Werte.

## Formatierte Stringausgabe (sprintf):

```
sprintf("%04d-%02d-%02d", $year, $month, $day);
```

## Einbinden von externen Dateien:

```
include ("blackboard.php");
```

Aber besser:

```
require ("bulletinboard.php");
```

Da dies bei einem Fehler sofort beendet wird, sonst könnte Code sichtbar werden.

## Formulare:

Mittels `$_REQUEST['vorname']` kann auf den eingegebenen Wert vom Input-Feld mit dem Namen 'vorname' zugegriffen werden.

## Bei Übermittlung mit der POST-Methode:

`$_POST[]` oder `$HTTP_POST_VARS[]`

## Bei Übermittlung mit der GET-Methode:

`$_GET[]` oder `$HTTP_GET_VARS[]`

## Cookie erzeugen

```
bool setcookie ( string name [, string value [, int expire[, string path [, string domain [, int secure]]]]] );
```

## Beispiel:

```
setcookie ("TestCookie", $value, time() + 3600);
/* verfällt in 1 Stunde = 3600 sek.*
```

## Zugriff auf Cookies

Sind die Cookies einmal gesetzt, kann man beim nächsten Seitenaufruf anhand des `$_COOKIE[]` Arrays auf diese zugreifen.

Existenz von Cookies prüfen:

```
print_r($_COOKIE);
```

## SESSION starten, ID auslesen und beenden

```
session_start();
```

```
session_id();
```

```
session_destroy();
```

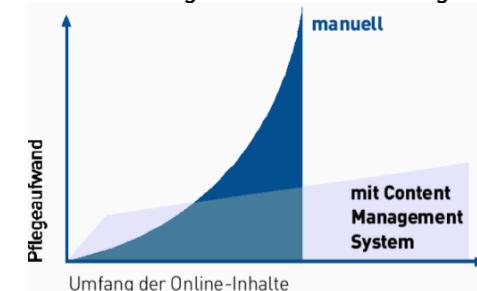
Jede Variable die in einer Session verwendet werden soll muss im globalen `$_SESSION[]` Array registriert werden!

## CMS

"Ein CMS ist ein Werkzeug, das vielen verschiedenen (zentralen) technischen und dezentralen nichttechnischen Mitarbeitern ermöglicht, eine Vielzahl von Inhalten (z.B. Text, Grafik, Video usw.) zu erstellen, zu bearbeiten, zu verwalten und schliesslich zu veröffentlichen, und zwar unter zentralen Randbedingungen bzgl. Regeln, Prozessen und Workflow, die ein konsistentes und gültiges Aussehen im Web sicherstellen"

Vorteile, vermeiden von:

1. starke Fehleranfälligkeit
2. grosser Pflegeaufwand
3. Hoher Schulungs-/Einarbeitungsaufwand
4. z.T. redundante Datenhaltung
5. Schwierige Administration
6. Probleme bei Einhaltung CI
7. Keine Trennung Inhalt, Struktur, Daten





## jQuery Selectors

Selector	Example	Selects
<code>*</code>	<code>\$("*")</code>	All elements
<code>#id</code>	<code>\$("#lastname")</code>	The element with id="lastname"
<code>.class</code>	<code>\$(".intro")</code>	All elements with class="intro"
<code>.class,.class</code>	<code>\$(".intro,.demo")</code>	All elements with the class "intro" or "demo"
<code>element</code>	<code> \$("p")</code>	All <p> elements
<code>e1,e2,e3</code>	<code> \$("h1,div,p")</code>	All <h1>, <div> and <p> elements
<code>:first</code>	<code> \$("p:first")</code>	The first <p> element
<code>:last</code>	<code> \$("p:last")</code>	The last <p> element
<code>:even</code>	<code> \$("tr:even")</code>	All even <tr> elements
<code>:odd</code>	<code> \$("tr:odd")</code>	All odd <tr> elements
<code>:first-child</code>	<code> \$("p:first-child")</code>	All <p> elements that are the first child of their parent
<code>:first-of-type</code>	<code> \$("p:first-of-type")</code>	All <p> elements that are the first <p> element of their parent
<code>:last-child</code>	<code> \$("p:last-child")</code>	All <p> elements that are the last child of their parent
<code>:last-of-type</code>	<code> \$("p:last-of-type")</code>	All <p> elements that are the last <p> element of their parent
<code>:nth-child(n)</code>	<code> \$("p:nth-child(2)")</code>	All <p> elements that are the 2nd child of their parent
<code>:nth-last-child(n)</code>	<code> \$("p:nth-last-child(2)")</code>	All <p> elements that are the 2nd child of their parent, counting from the last child
<code>:nth-of-type(n)</code>	<code> \$("p:nth-of-type(2)")</code>	All <p> elements that are the 2nd <p> element of their parent
<code>:nth-last-of-type(n)</code>	<code> \$("p:nth-last-of-type(2)")</code>	All <p> elements that are the 2nd <p> element of their parent, counting from the last child
<code>:only-child</code>	<code> \$("p:only-child")</code>	All <p> elements that are the only child of their parent
<code>:only-of-type</code>	<code> \$("p:only-of-type")</code>	All <p> elements that are the only child, of its type, of their parent
<code>parent &gt; child</code>	<code> \$("div &gt; p")</code>	All <p> elements that are a direct child of a <div> element
<code>parent descendant</code>	<code> \$("div p")</code>	All <p> elements that are descendants of a <div> element
<code>element + next</code>	<code> \$("div + p")</code>	The <p> element that are next to each <div> elements
<code>element ~ siblings</code>	<code> \$("div ~ p")</code>	All <p> elements that are siblings of a <div> element
<code>:eq(index)</code>	<code> \$("ul li:eq(3)")</code>	The fourth element in a list (index starts at 0)
<code>:gt(no)</code>	<code> \$("ul li:gt(3)")</code>	List elements with an index greater than 3
<code>:lt(no)</code>	<code> \$("ul li:lt(3)")</code>	List elements with an index less than 3
<code>:not(selector)</code>	<code> \$("input:not(:empty)")</code>	All input elements that are not empty
<code>:header</code>	<code> \$(":header")</code>	All header elements <h1>, <h2> ...
<code>:animated</code>	<code> \$(":animated")</code>	All animated elements
<code>:focus</code>	<code> \$(":focus")</code>	The element that currently has focus
<code>:contains(text)</code>	<code> \$(":contains('Hello')")</code>	All elements which contains the text "Hello"
<code>:has(selector)</code>	<code> \$("div:has(p)")</code>	All <div> elements that have a <p> element

<code>:empty</code>	<code> \$("":empty")</code>	All elements that are empty
<code>:parent</code>	<code> \$(":parent")</code>	All elements that are a parent of another element
<code>:hidden</code>	<code> \$("p:hidden")</code>	All hidden <p> elements
<code>:visible</code>	<code> \$("table:visible")</code>	All visible tables
<code>:root</code>	<code> \$(":root")</code>	The document's root element
<code>:lang(language)</code>	<code> \$("p:lang(de)")</code>	All <p> elements with a lang attribute value starting with "de"
<code>[:attribute]</code>	<code> \$("[href])</code>	All elements with a href attribute
<code>[:attribute=value]</code>	<code> \$("[href='default.htm'])")</code>	All elements with a href attribute value equal to "default.htm"
<code>[:attribute!=value]</code>	<code> \$("[href!=default.htm])")</code>	All elements with a href attribute value not equal to "default.htm"
<code>[:attribute\$=value]</code>	<code> \$("[href\$=.jpg])")</code>	All elements with a href attribute value ending with ".jpg"
<code>[:attribute*=value]</code>	<code> \$("[title]=Tomorrow")")</code>	All elements with a title attribute value equal to 'Tomorrow', or starting with 'Tomorrow' followed by a hyphen
<code>[:attribute^=value]</code>	<code> \$("[title^='Tom'])")</code>	All elements with a title attribute value starting with "Tom"
<code>[:attribute~value]</code>	<code> \$("[title~='hello'])")</code>	All elements with a title attribute value containing the specific word "hello"
<code>[:attribute*=value]</code>	<code> \$("[title*='hello'])")</code>	All elements with a title attribute value containing the word "hello"
<code>:input</code>	<code> \$(":input")</code>	All input elements
<code>:text</code>	<code> \$(":text")</code>	All input elements with type="text"
<code>:password</code>	<code> \$(":password")</code>	All input elements with type="password"
<code>:radio</code>	<code> \$(":radio")</code>	All input elements with type="radio"
<code>:checkbox</code>	<code> \$(":checkbox")</code>	All input elements with type="checkbox"
<code>:submit</code>	<code> \$(":submit")</code>	All input elements with type="submit"
<code>:reset</code>	<code> \$(":reset")</code>	All input elements with type="reset"
<code>:button</code>	<code> \$(":button")</code>	All input elements with type="button"
<code>:image</code>	<code> \$(":image")</code>	All input elements with type="image"
<code>:file</code>	<code> \$(":file")</code>	All input elements with type="file"
<code>:enabled</code>	<code> \$(":enabled")</code>	All enabled input elements
<code>:disabled</code>	<code> \$(":disabled")</code>	All disabled input elements
<code>:selected</code>	<code> \$(":selected")</code>	All selected input elements
<code>:checked</code>	<code> \$(":checked")</code>	All checked input elements

## CMS: Statische Systemarchitektur

### Vorteile:

- hohe Ausfallsicherheit des Webangebots
- Geringere Anforderungen an die Hardware
- physikalische Trennung von Web- und Datenbank-Server möglich
- Weiterverwendung bestehender Serverarchitektur inkl. Firewalls
- Leichte Erstellung von Offlineversionen

### Nachteile:

- Geringerer Aktualitätsgrad des Angebots
- Dynamische und personalisierte Abfragen benötigen weitere Datenbankanwendung

## CMS: Dynamische Systemarchitektur

### Vorteile:

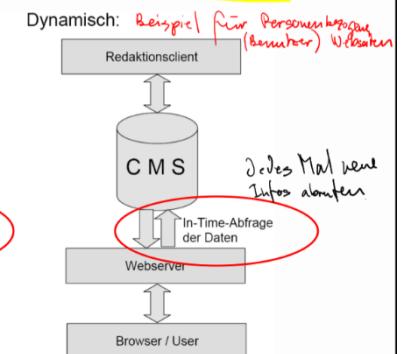
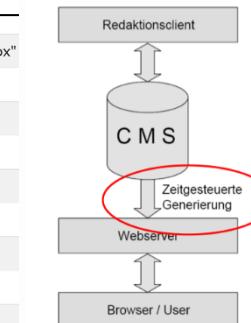
- Leichtere Unterstützung des Zusammenwachsens der drei Cs (Content, Commerce, Community)
- sehr hoher Aktualitätsgrad des Angebots
- leichtere Einbindung dynamischer Abfragen

### Nachteile:

- keine Offlineversionen möglich
- Totalausfall des Angebots bei Fehlern der Datenbank
- schwache Performance bei starken Zugriffen
- sehr hohe Hardwareanforderungen

## CMS-Systemarchitektur (2): Statisch vs. dynamisch

### Statisch:



Dies würde ein <a>-Element auswählen, das:

Verfügt über die folgenden Klassen: `class1, class2, and class3`

Hat die folgende ID: `someID`

Hat das folgende Attribut: `attr1`

Hat die folgenden Attribute und Werte: `attr2` mit Wert `something`, `attr3` mit Wert `something`

Hat die folgenden Zustände: `first-child` und `first-of-type`

Sie können auch verschiedene Selektoren mit einem Komma trennen:

Dies würde auswählen:

Alle <a> Elemente

Alle Elemente, die die Klasse `class1`

Ein Element mit der ID `#someID`

Core	Selectors	Effects	Events	Ajax
<b>jQuery Function</b>				
\$(selector, context)	jQuery		.load(fn)	.ajaxComplete(cb)
\$(<html>, owner)	jQuery		.ready(fn)	.ajaxError(cb)
\$callback)	jQuery		.unload(fn)	.ajaxSend(cb)
<b>No Conflict</b>	jQuery.noConflict(bool)	jQuery		.ajaxStart(cb)
<b>Attributes</b>				.ajaxStop(cb)
<b>General Attributes</b>				.ajaxSuccess(cb)
.attr(attribute)	obj			
.attr(attribute, val)	jQuery			
.attr(attribute, fn)	jQuery			
.removeAttr(attribute)	jQuery			
.html()	string			
.html(string)	jQuery			
.text()	string			
.text(string)	jQuery			
.val()	string, array			
.val(value)	jQuery			
<b>CSS</b>				
<b>Style Properties</b>				
.css(property)	string			
.css(property, val fn)*	jQuery			
<b>Class Attribute</b>				
.addClass(name fn)*	jQuery			
.removeClass(name fn)*	jQuery			
.hasClass(name)	bool			
.toggleClass(name fn, switch)*	jQuery			
<b>Dimensions</b>				
.height()	int			
.height(val)	jQuery			
.width()	int			
.width(val)	jQuery			
.innerHeight()	int			
.innerWidth()	int			
.outerHeight(margin)	int			
.outerWidth(margin)	int			
<b>Offset</b>				
.offset()	obj{top,left}			
.offset(coordinates)*	obj			
.position()	obj{top,left}			
.scrollLeft()	int			
.scrollLeft(val)	jQuery			
.scrollTop()	int			
.scrollLeft(val)	jQuery			
<b>Data</b>				
jQuery.data(element, key, value)	jQuery			
jQuery.data(element, key)	obj			
.queue(queueName)	array			
.queue(queueName, newQueue cb)	jQuery			
.clearQueue(queueName)*	jQuery			
.dequeue(queueName)	jQuery			
<b>Miscellaneous</b>				
<b>Collection Manipulation</b>				
.each(fn)	jQuery			
<b>Collection Manipulation</b>				
.get(index)	element, array			
.index(selector element)*	num			
.size()	num			
.toArray()*	array			
<b>Properties of the Global jQuery Object</b>				
jQuery.support	obj			
jQuery.browser	map			
jQuery.browser.version	string			