**HTML – Grundgerüst**

* HTML = Auszeichnungssprache keine Progr.ssprache
* 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.Zielpubliku**

**5. Linkverfolgung 6. Refresh 7.Wiederholung Indizieru**

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">

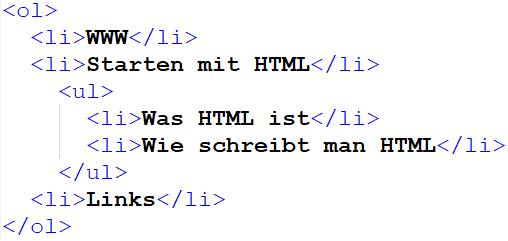
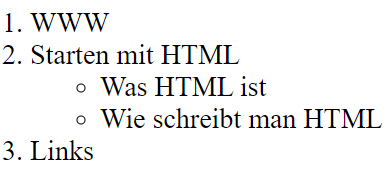
5. <meta name="robots" content="index,follow">

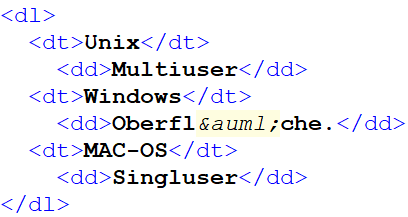
6.<meta http-equiv="refresh" content="10; URL=x.htm">

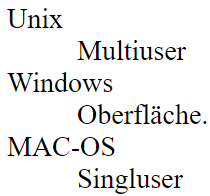
7. <meta name="revisit-after" content="14 days">

* **<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. Definitieons-/Glossar List**





Bei ol, *start=’’3’’* 🡪 Aufzählung beginnt bei 3. *value=’’6’’* fährt die Liste bei 6 weiter, *type=’’I’’* führt zu grosser römischer Nummerierung. Bei ul, type=’’square’’ 🡪 Quadrat als Aufzählungszeichen.

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

<img src="pfad/bildname" alt="Beschreibung" />

<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>

<a href="adresse" target="\_blank">Adresse</a>

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

<img src="Landkarte" usemap="#karte" />

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

</map>

* **Umlaute:**

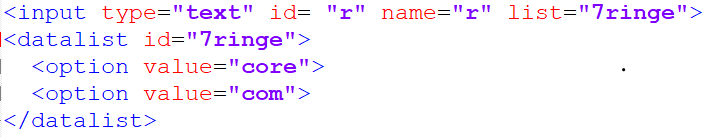
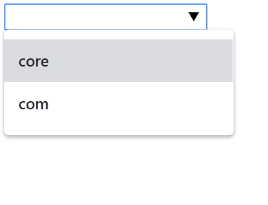
ü=&uuml; | ö=&ouml; | ä=&auml;

* **<iframe> Eingebettete Frames/Webpages**

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

</iframe>

**Formularelemente**

* **1.Datalist 2.Platzhalter Text**
* **1.Slider(range) 2.Number selector 3.Date**

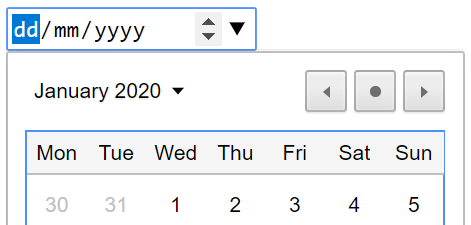
<input type="range" id="e" name="e" min="1" max="5">



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



<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;”-->

<img src="pfad/bildname" alt="Beschreibung" />

</body>

</html>  
**CSS – Grundgerüst**

body {

background-color: powderblue;

}

#myID {

color: blue;

}

.myClass {

color: red;

}

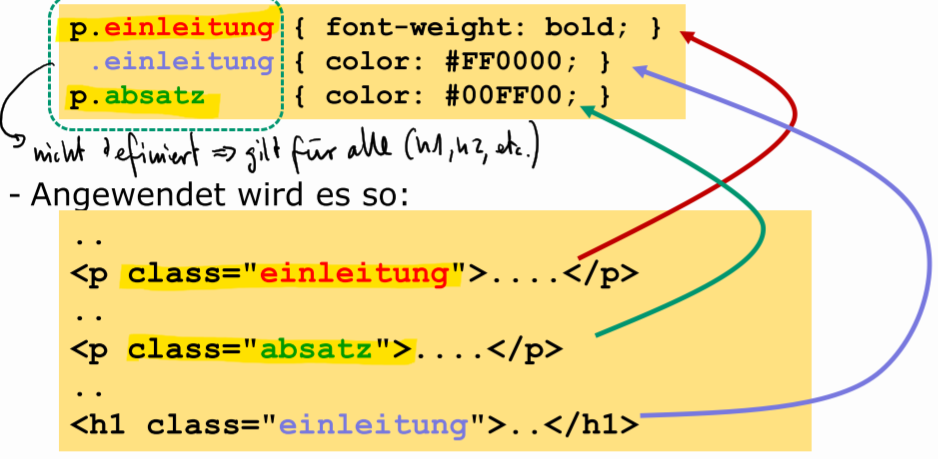
* **CSS-Reset**

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**

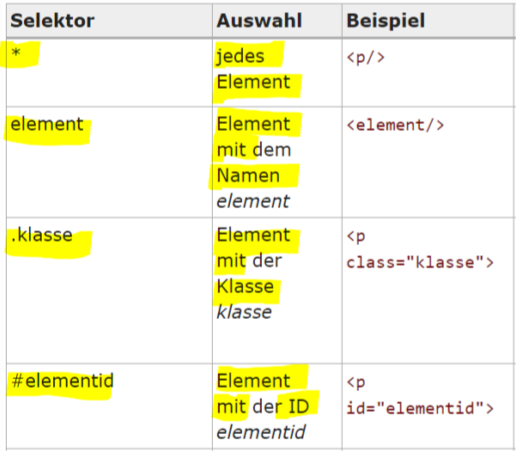


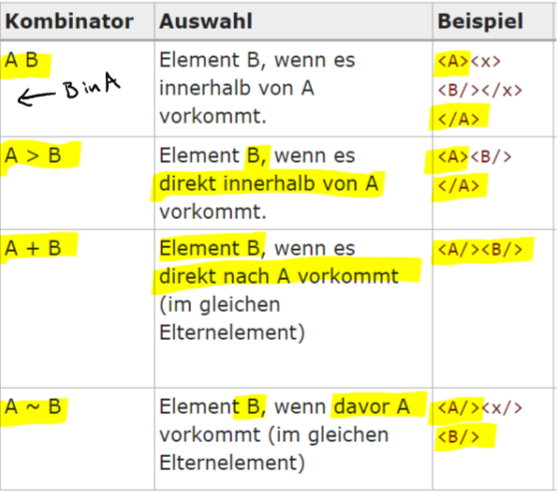


* **h3 innerhalb h2 innerhalb h1**

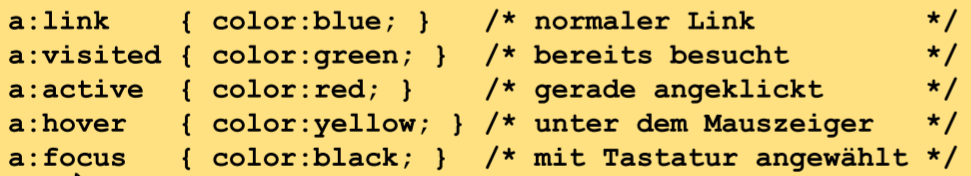


* **CSS-Selektoren: 1.einfache 2.kombinierte**

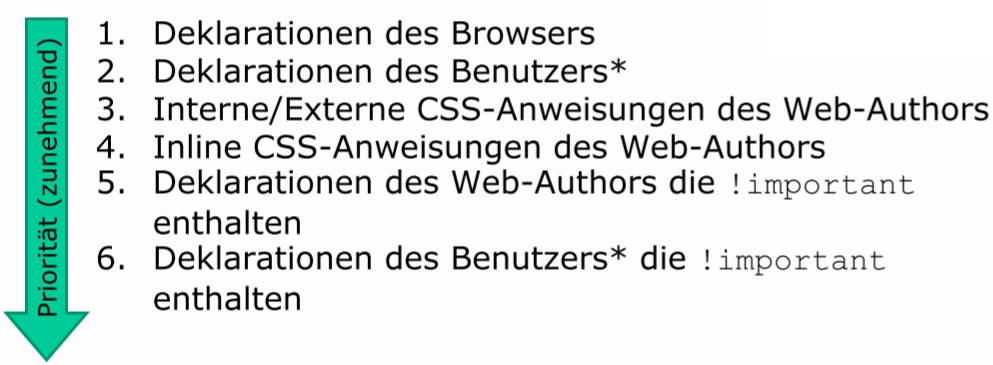




* **Hyperlinks**



* **Kaskadierung/Prioritätsreihenfolge von hoch>niedrig**





* **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 verschieden 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 Seitenspalte 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;

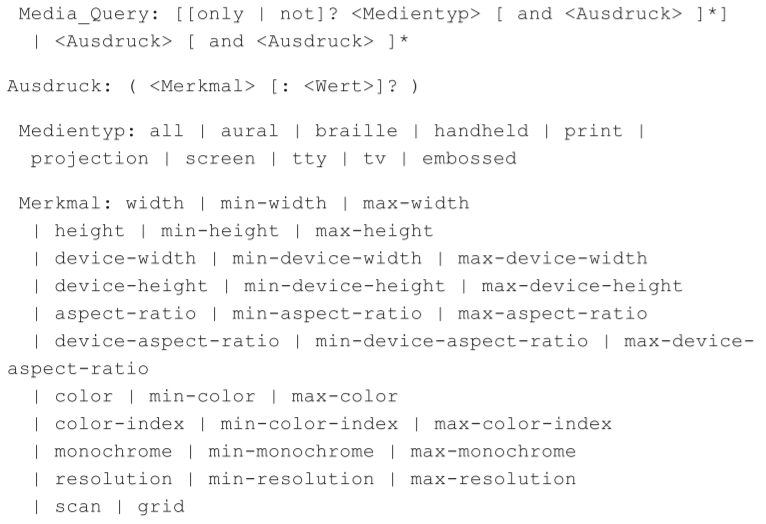
Zusätzlich mit Breite:

@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)" />

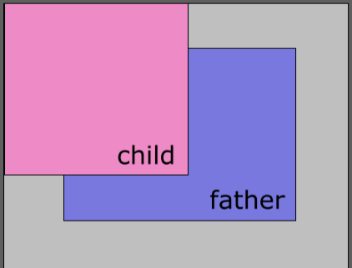
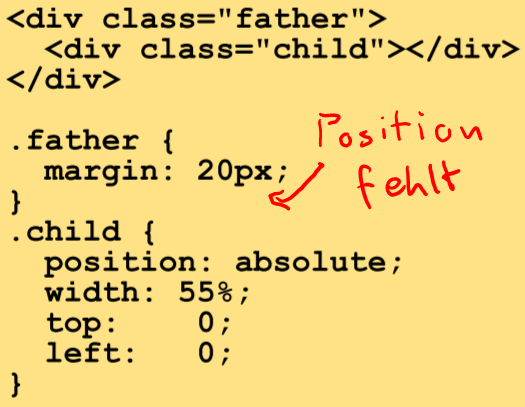
* **Media-Queris generell:**

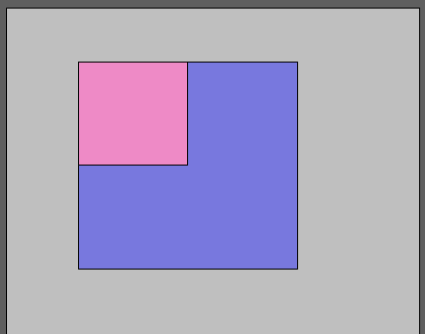
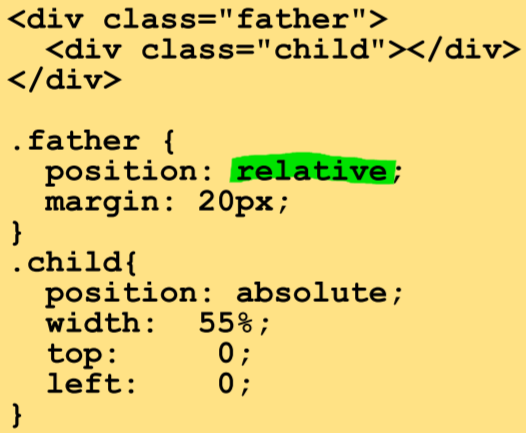


* **POSITIONIERUNG/FLEX**

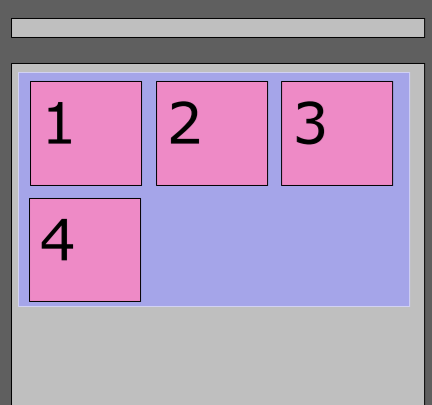
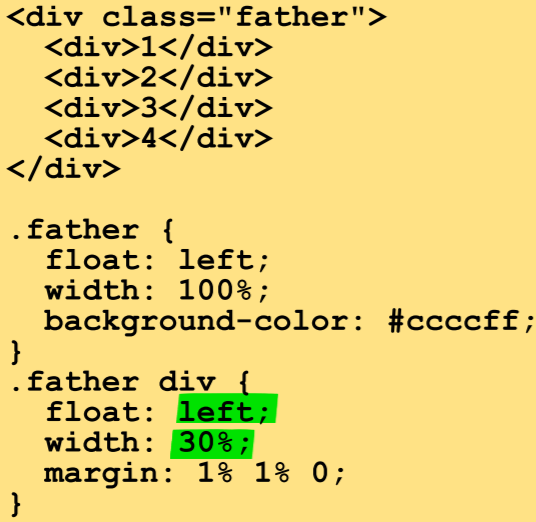
Es gibt fünf verschieden Positionswerte:

* + **static** (default, not affected by top, bottom ,left …)
  + **relative** (ist relativ zum normalen Ort plaziert)
  + **fixed** (Immer an derselben Position, auch beim scrollen, bsp. Navigation)
  + **absolute** (Relativ positioniert zu parent element(welches positioniert ist, also alles ausser static. Wenn keine postion: xyz; beim Parten gesetzt dann funktionierts nicht), wenn keines vorhanden, dann zu body. Also wird die box vom übergeordneten Element zum Koordinatensystem dieses Elementes.
  + **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**

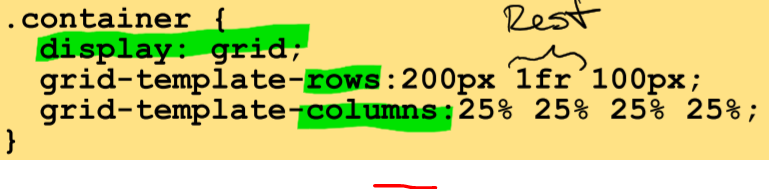
* **FLOAT (Wo stehe ich relativ zu anderen Elementen)**

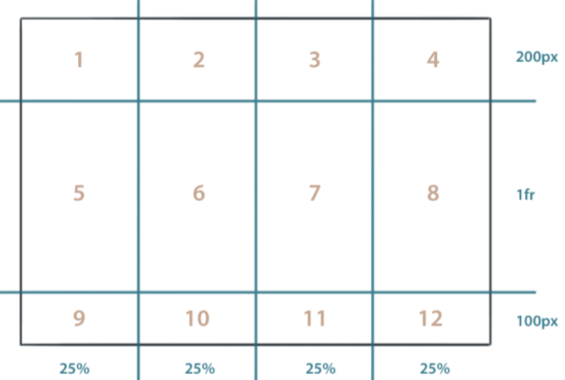
 

* **FLEX-Positionierung**

1. Zuerst sagt man einem Elemnt (father), dass dessen Kinder «flex» sind
2. Dann bestimmt man wie sich die Kinder verhalten
3. Properties:
   1. Flex-direction: (row-/column(-reverse))
   2. Flex-wrap: auto Zeilenumbruch: (no)wrap
   3. Justify-content: X-Achse: center, flex-start/end, space-between/around
   4. Align-content: Y-Achse: gleiche wie oben

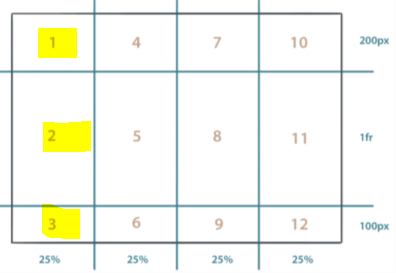
* **GRID (zusammen mit Flex)**
  + CSS Grid ermöglicht Gestaltungsraster auf Basis von CSS
  + Dem Elternelement wird mit Hilfe der Angabe **display:grid;** mitgeteilt, dass CSS Grids genutzt werden sollen.
  + Mit Hilfe der Eigenschaften **grid-template-columns** und **grid-template-rows** werden Rasterlinien gezeichnet.





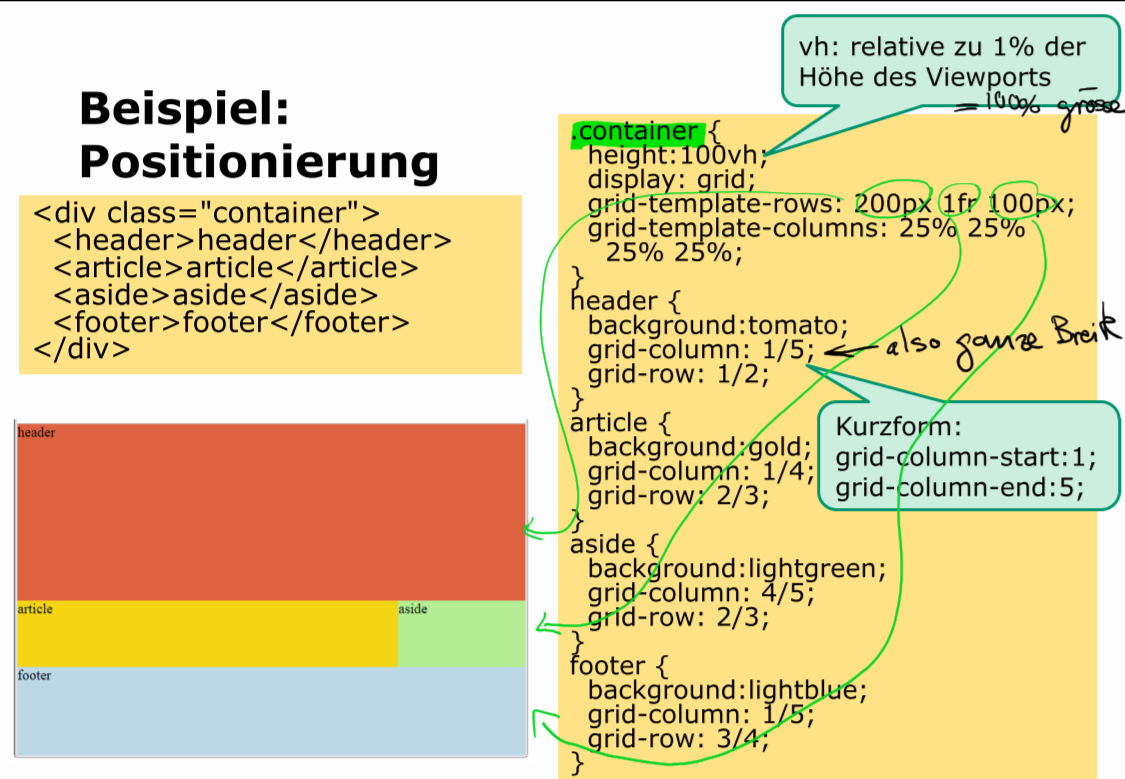
* + Mit der CSS-Eigenschaft **grid-auto-flow** kann beeinflusst werden, wie die Kind-Elemente im Grid einsortiert werden.



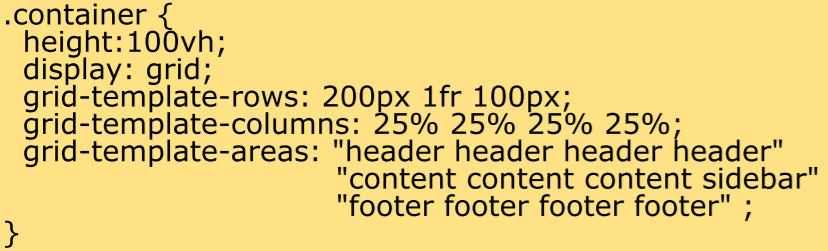


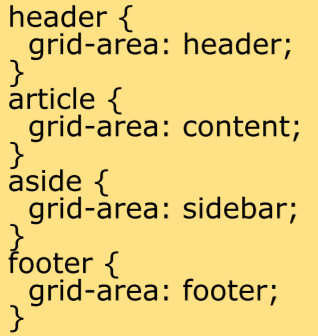
* **Elemente im Grid positionieren**

Achtung beginnt bei 1 (0%)



* **Gridzellen benennen**





Wird eine Gridzelle mit Punkt (.) bennant, dann wird diese Zelle durchsichtig sein. Sprich es könnte Beispielsweise der Hintergund 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**
  + Gelten nur für ganze Seiten
  + 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

- direkterer Zugriff auf die Darstellungsmöglichkeiten des Betrachters

**Nachteile:**

- Server kann die Ausführung der Scripte nicht

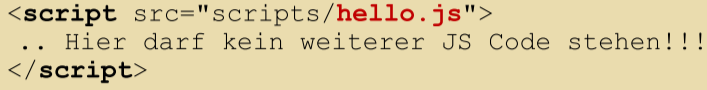
- Dateioperationen nicht möglich

- 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**





* **Möglichkeiten von JS**

- Validitätscheck von Formularwerten

- Dokumentenmanipulation 🡪 DOM

- Browsersteuerung

- Bearbeiten von Cookies

- Spielereien, Gimmicks

* **Variablen-Namen**
  + Gross/Kleinschreibung wird beachtet
  + Zeichen, Buchstaben, Zahlen und Unterstrich
  + Keine Leerzeichen
  + Keine Umlaute
  + Kein Bindestrich (=Subtraktionszeichen!)
  + Maximal 32 Zeichen lang

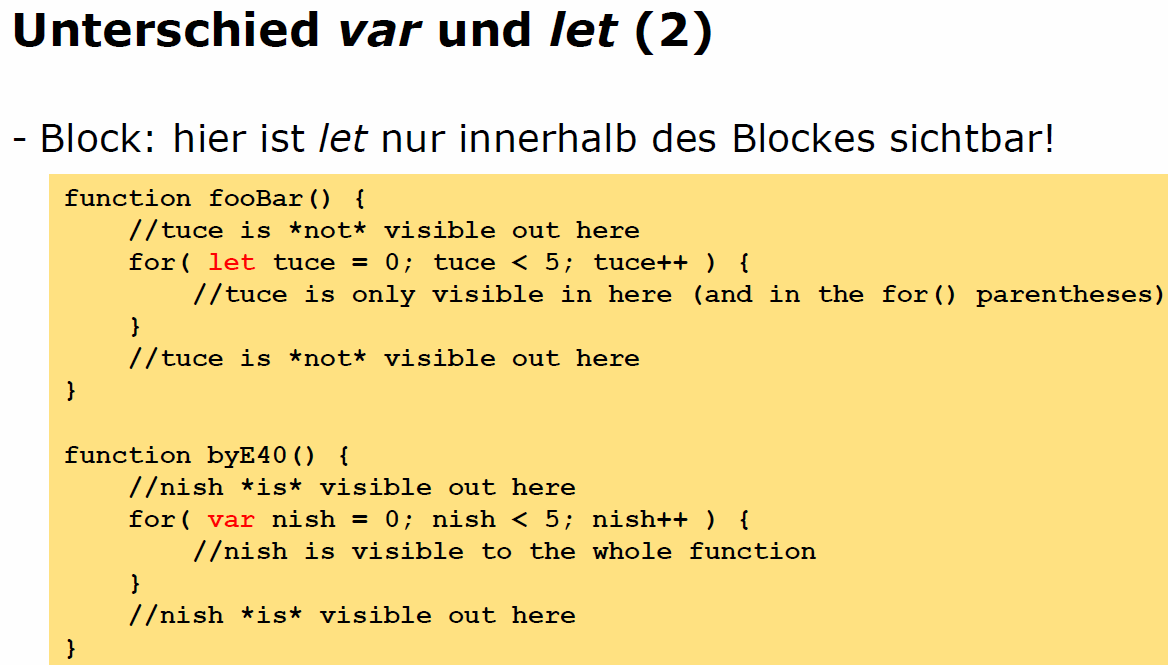
**Globale Variablen:**

Mit dem Vorstellen von «window.» kann eine Variable nach der ersten Verwendung nachher global benutzt werden. 🡪 **window.MeineVariable;**

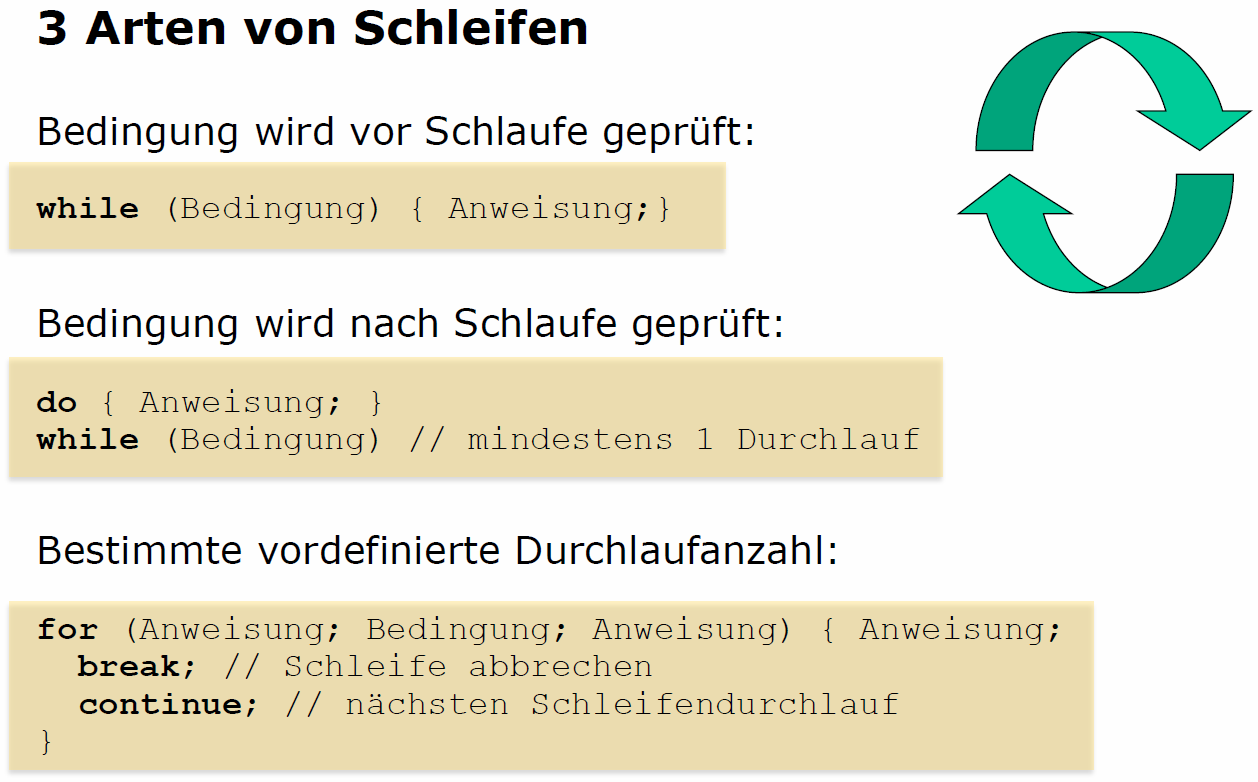
**Oder:** wenn Variable ausserhalb von Funktion gesetzt.

**Lokale Variablen:**

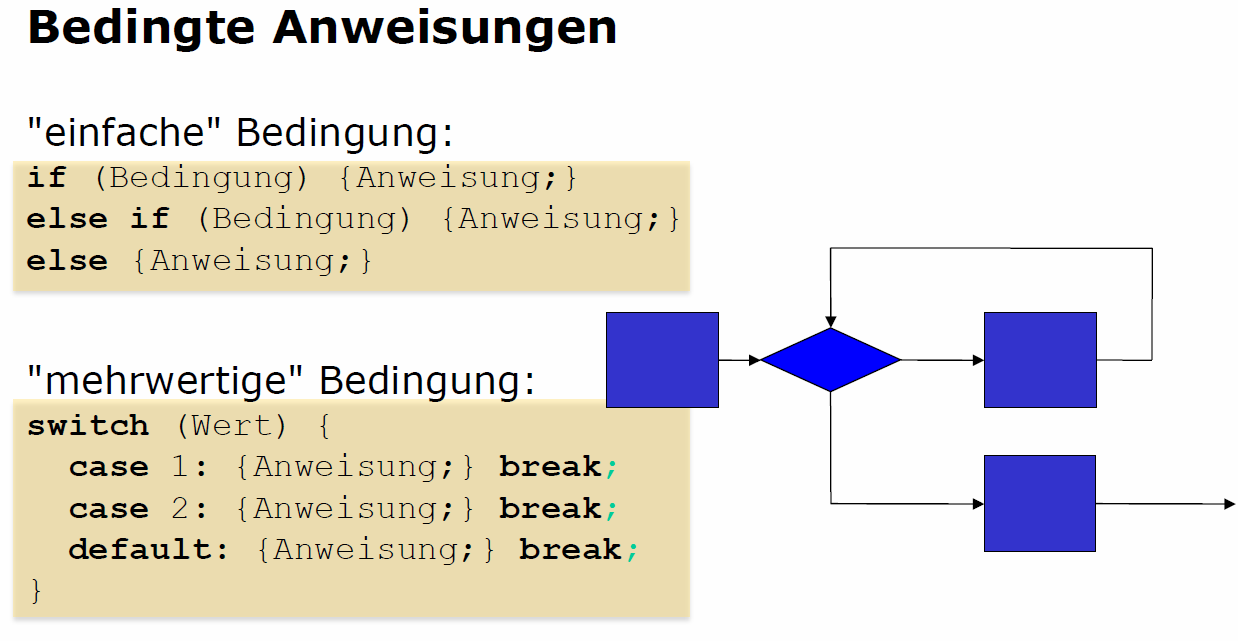
Innerhalb einer Funktion, also lokal mit var.



* **Schlaufen**



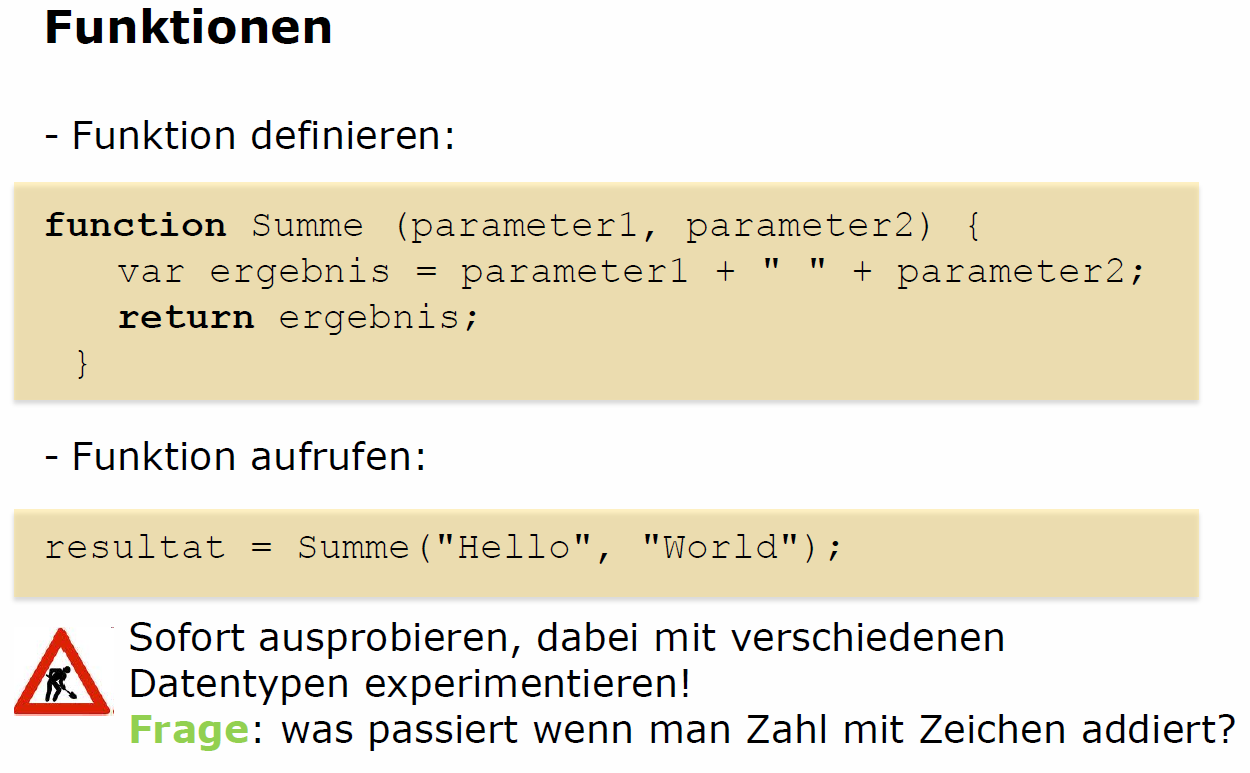
* **If-Schlaufen**



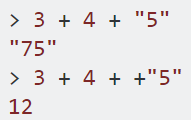
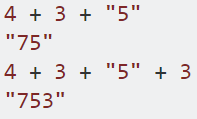
* **Boolische Ausdrücke (ergibt true oder false)**



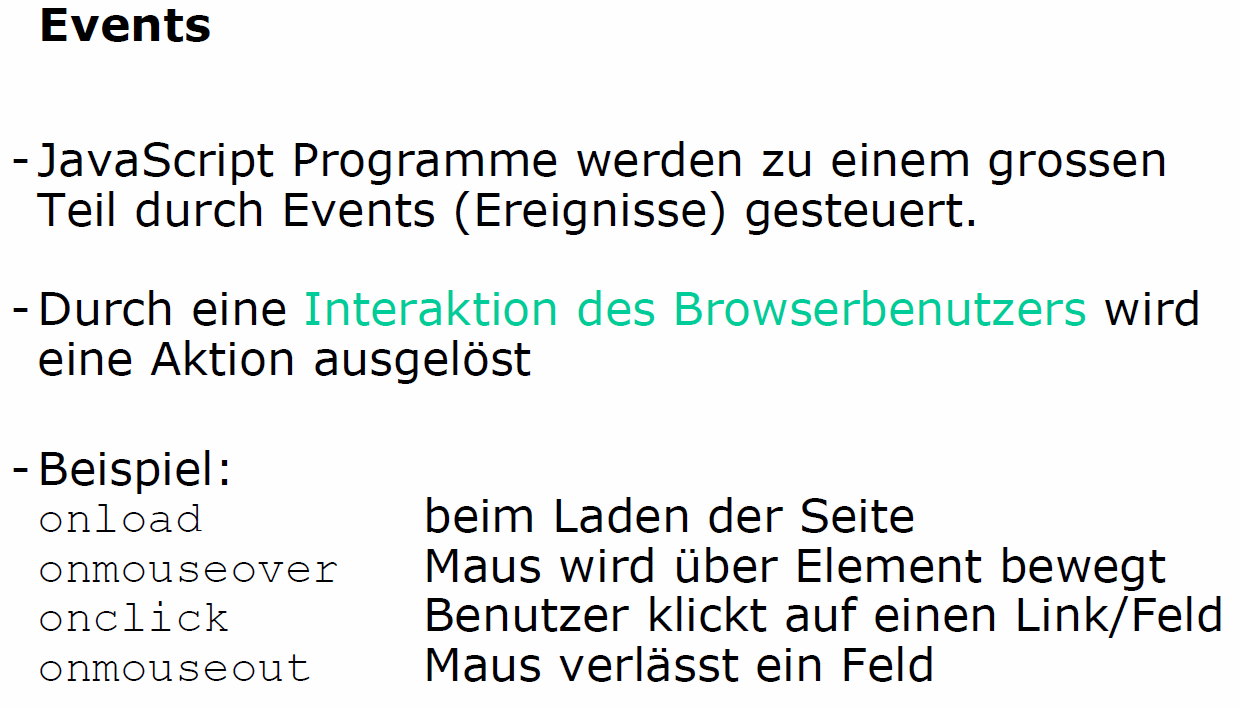
* && bedeutet AND
* | | bedeutet OR
* == bedeutet "ist gleich"
* **Funktionen**



When you add a number & a string in Javascript the result is always a string.

* **Events**

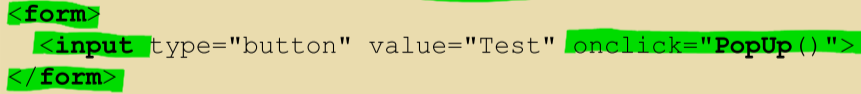


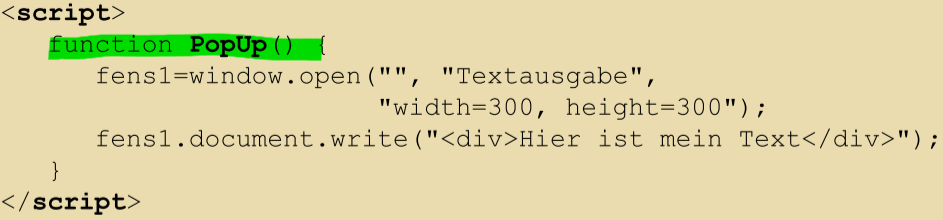
**Beispiel:**





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

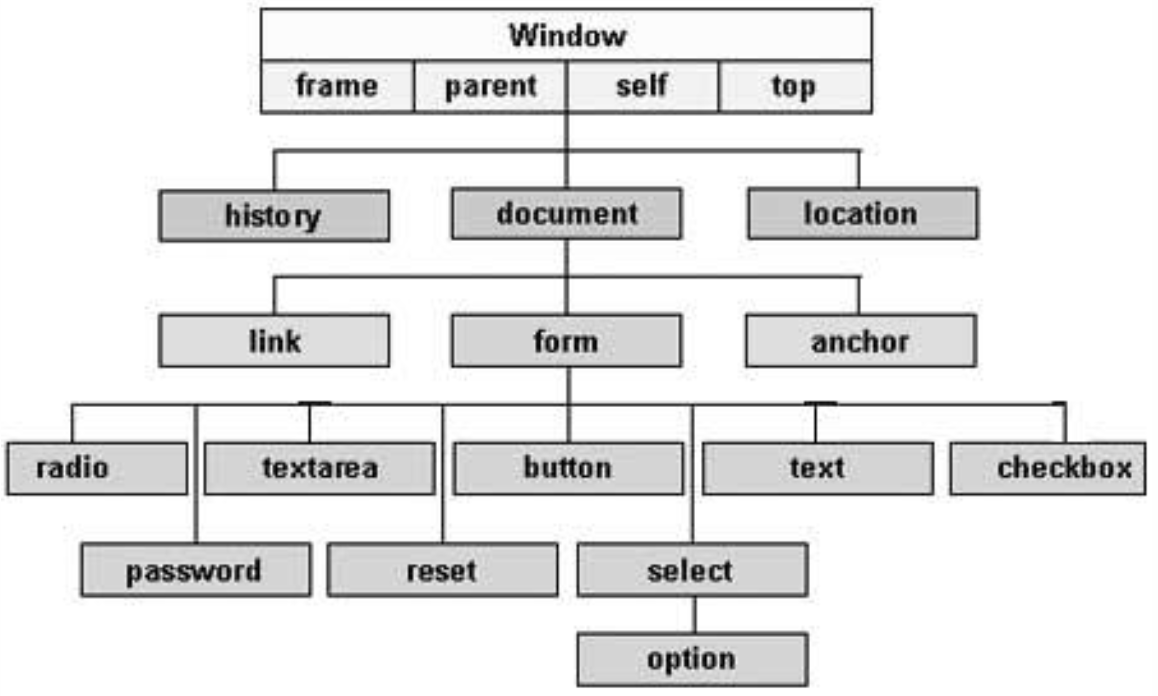




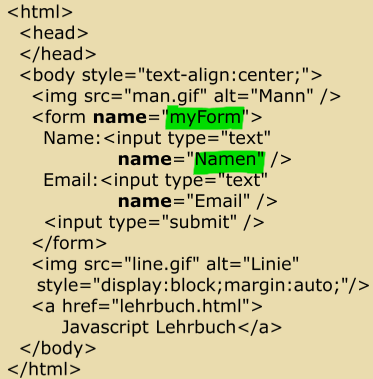
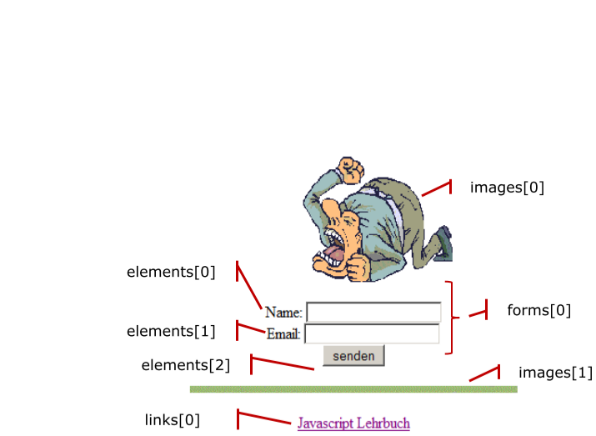
* **Vordefinierte Dialogboxen**
  + **Alert():** Meldung mit OK bestätigen
  + **Confirm():** Ja/Nein Abfrage
  + **Prompt:** erfordert Benutzereingabe



**DOM – Document Object Model**

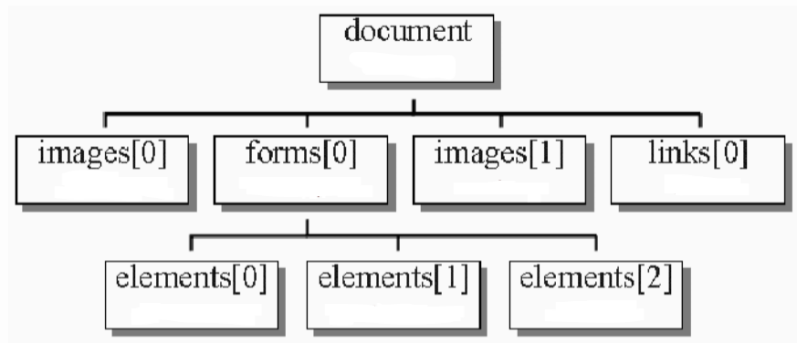


* **Beispiel 1:**



Zugriff:





first image

The whole form

second image

The link

1st text element

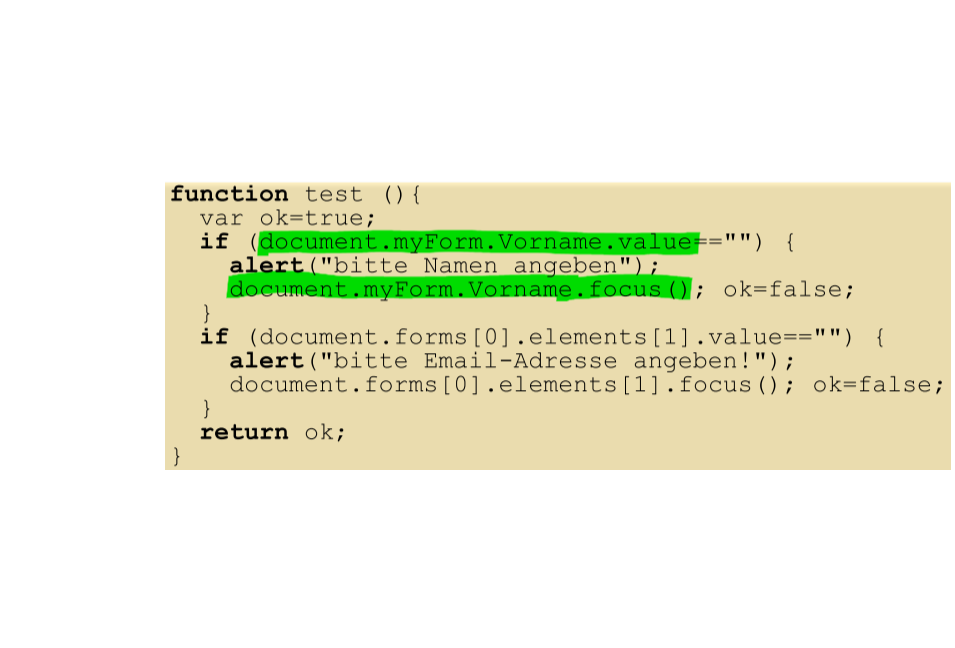
the button

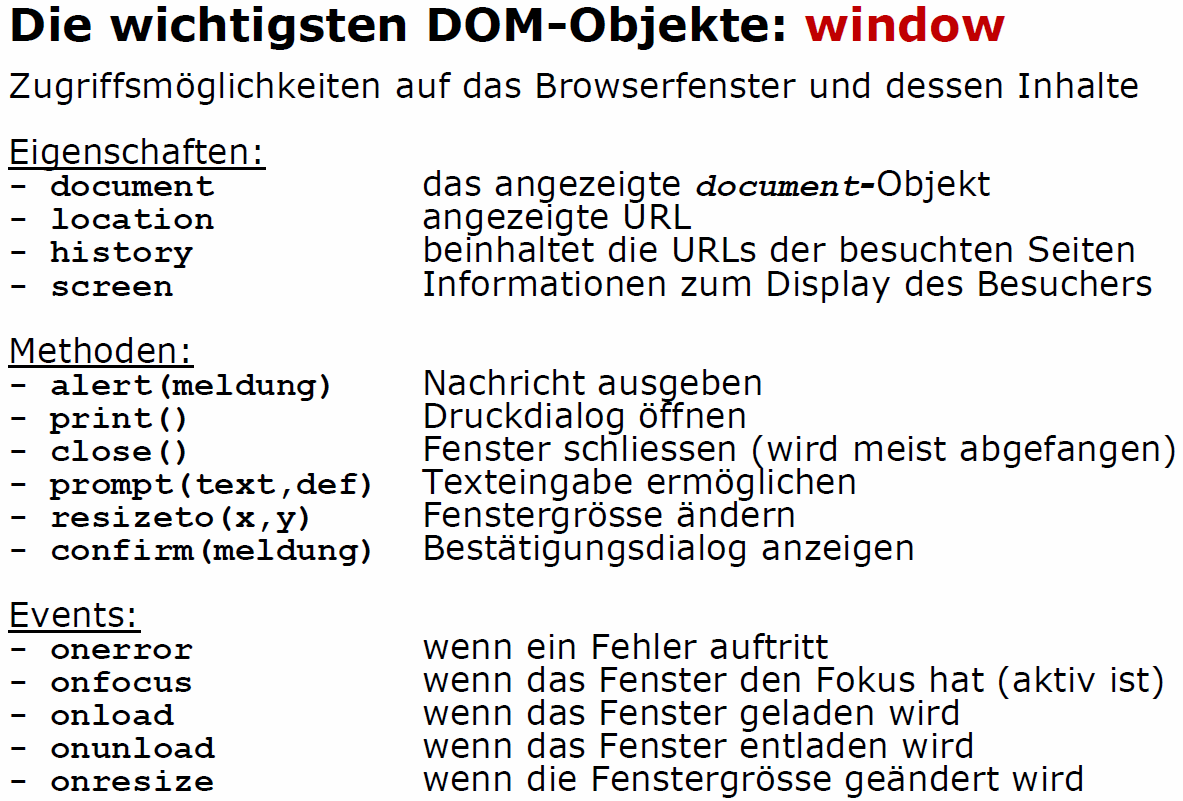
HTML-page

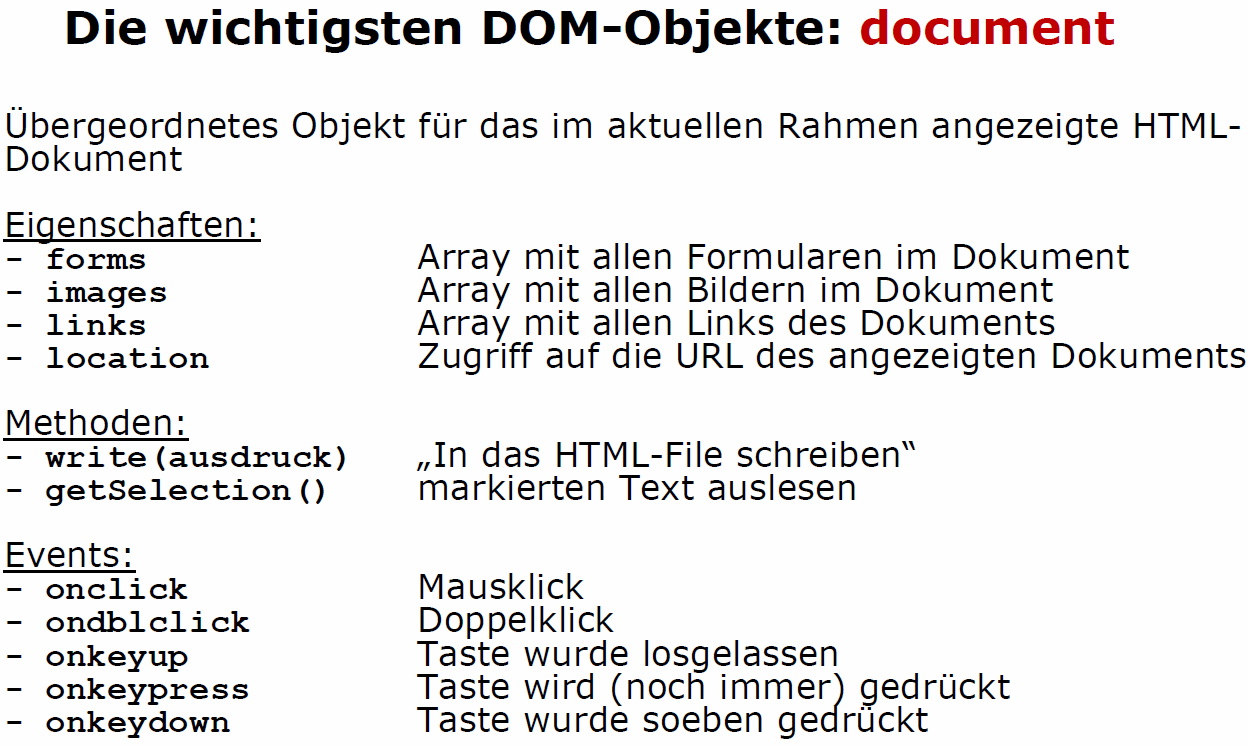
2nd text element

* **Beispiel 2:**

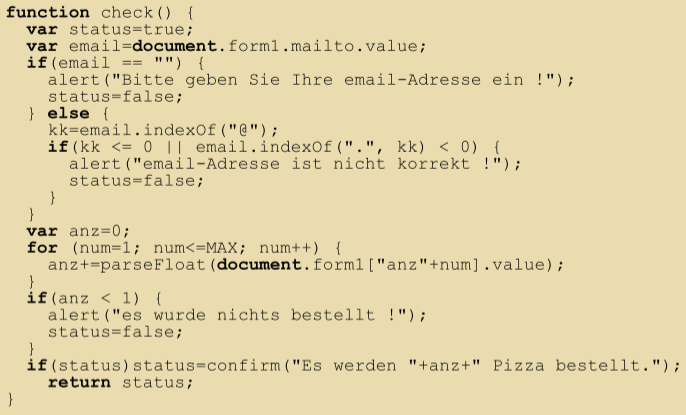








* **Intelligente Formulare (Mail & Bestellung prüfen)**



* **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 {

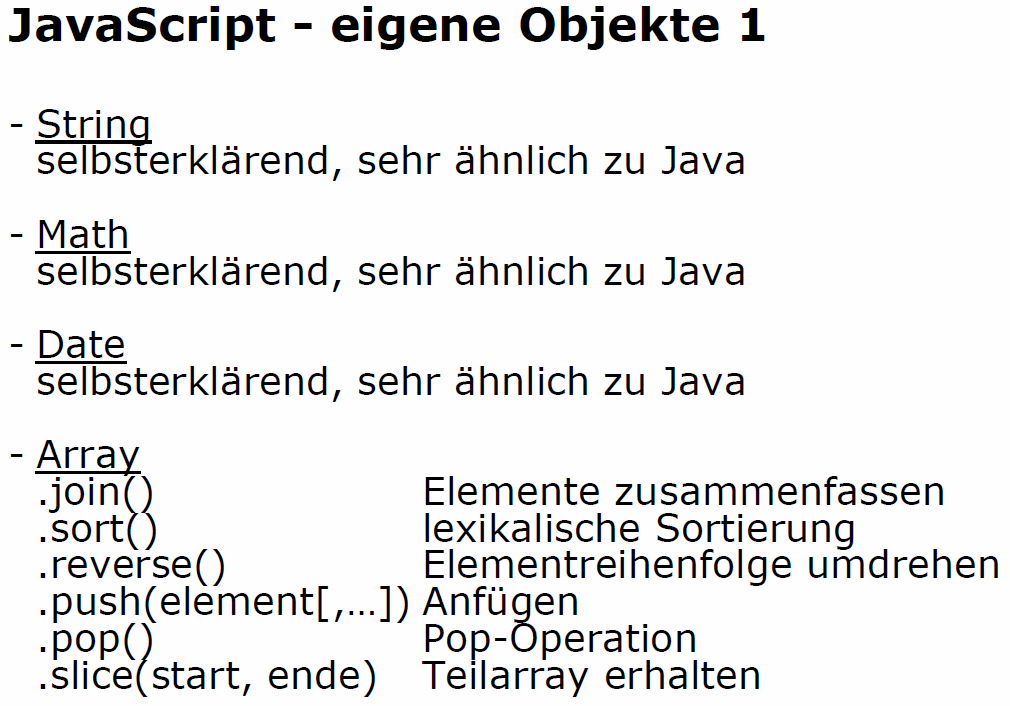
jetzt = new Date();

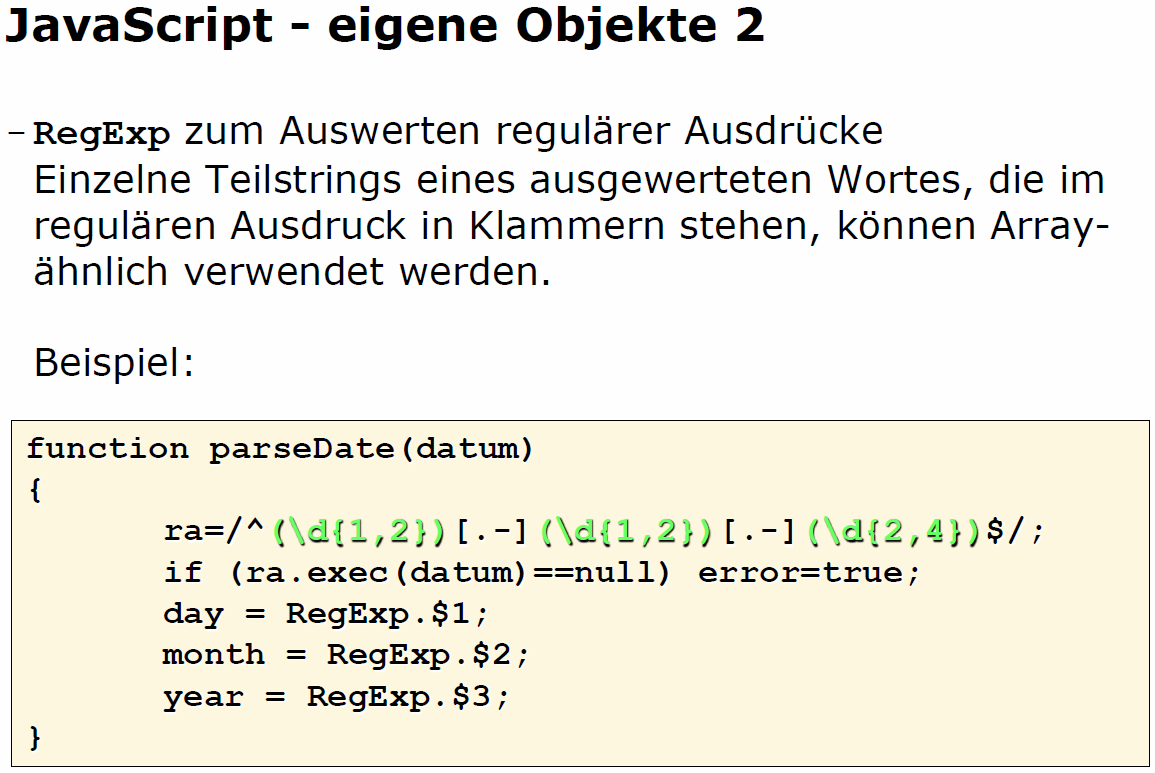
timeout= new Date(jetzt.getTime()+1000\*60);

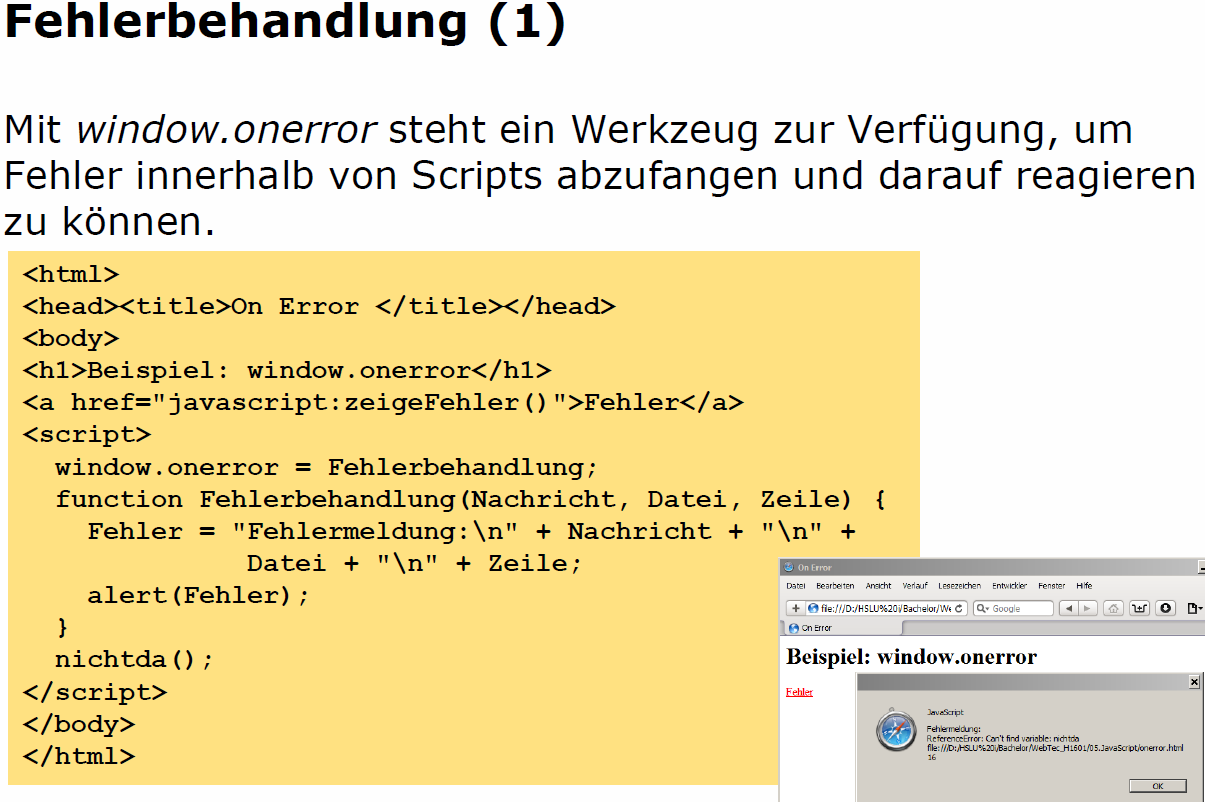
document.cookie = "Hallo Besucher"

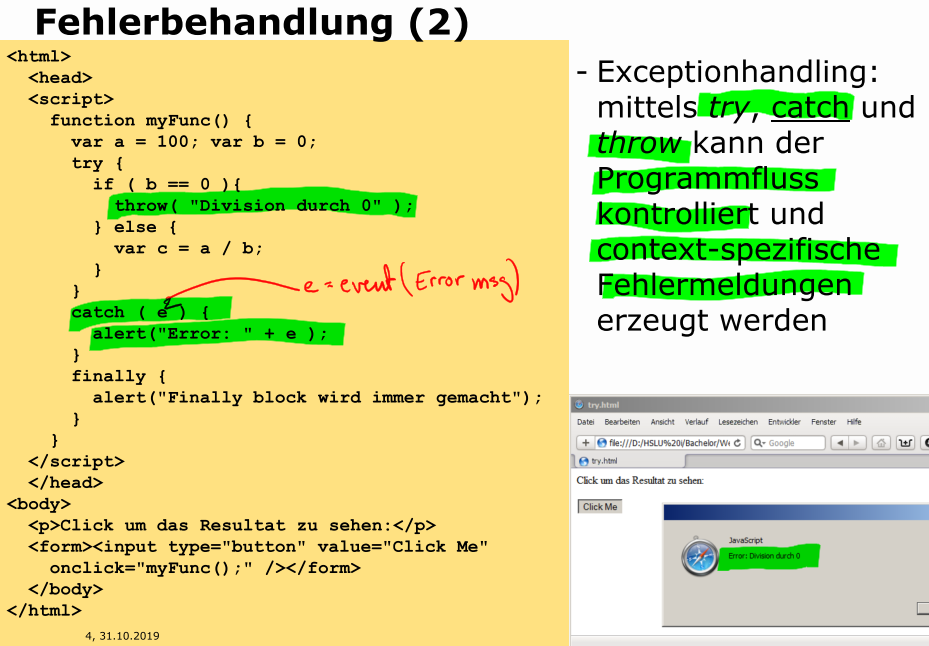
+"expires"+timeout.toGMTString()+";";

}

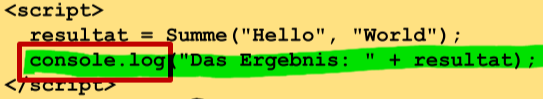








* **Ausgabe in Konsole**

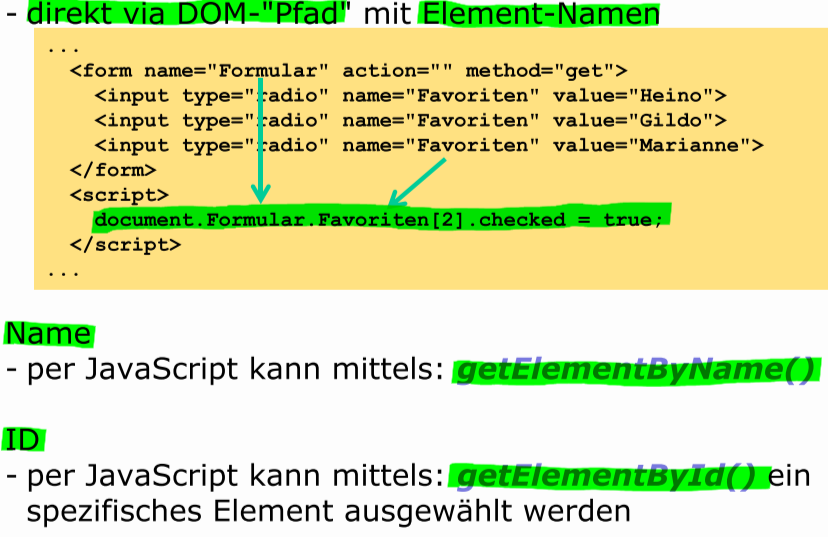


**Name oder ID**

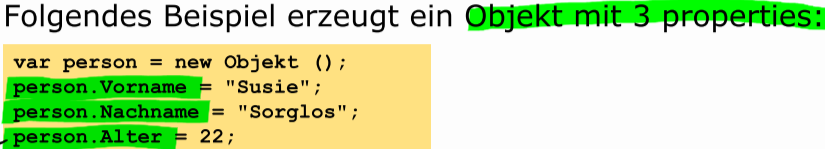
Name wird eigentlich nur in **Formularen** verwendet und zwar bei den <input> Elementen um diese zu gruppieren.

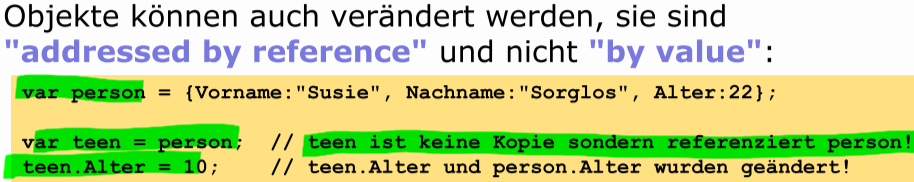
Nur Formularelemente mit Namen werden an Server gesendet

**Zugriff: direkt, Name oder ID**



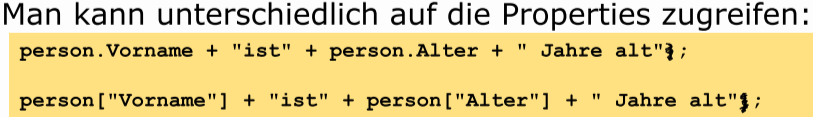
* **Objekt**



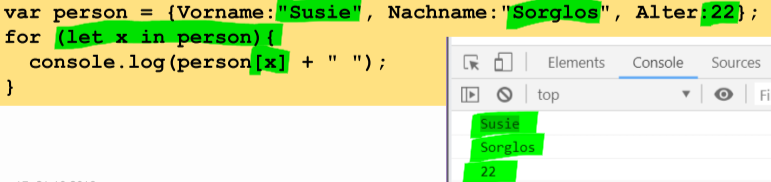


* **Properties**

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



Oder:

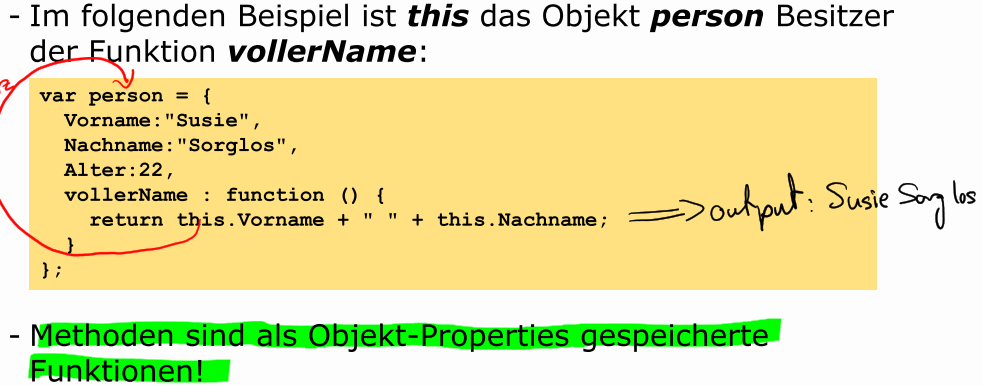


* **Neue properites hinzufügen/entfernen**

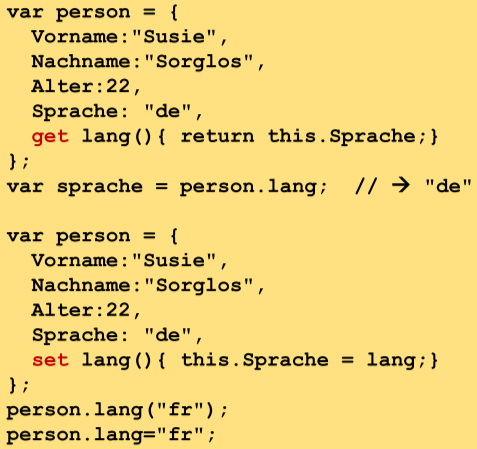




* **Objekt Methoden**

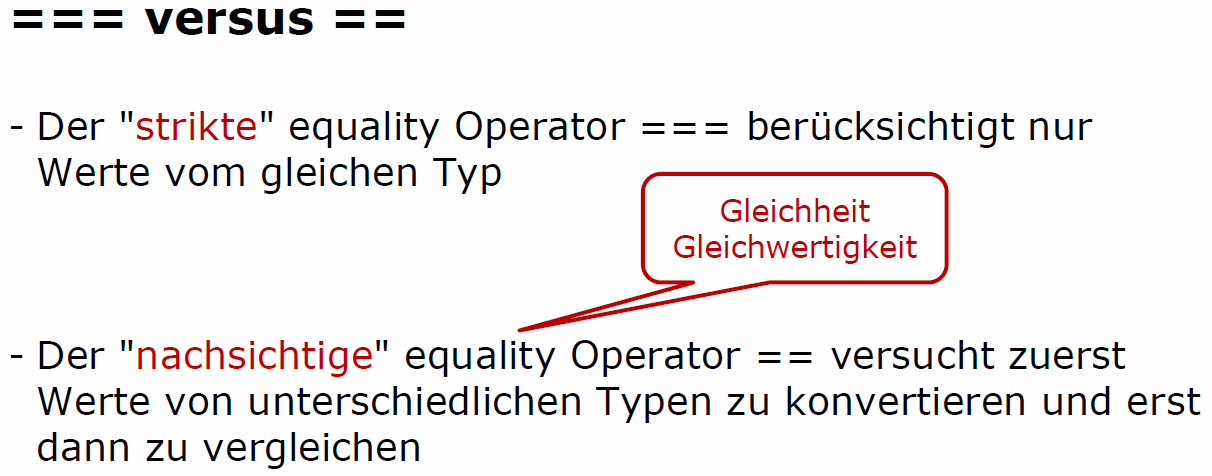


* **Getter und Setter Methoden (get/set)**



* **Zugriff auf Objekt Methoden**





**DEBUGGING:**

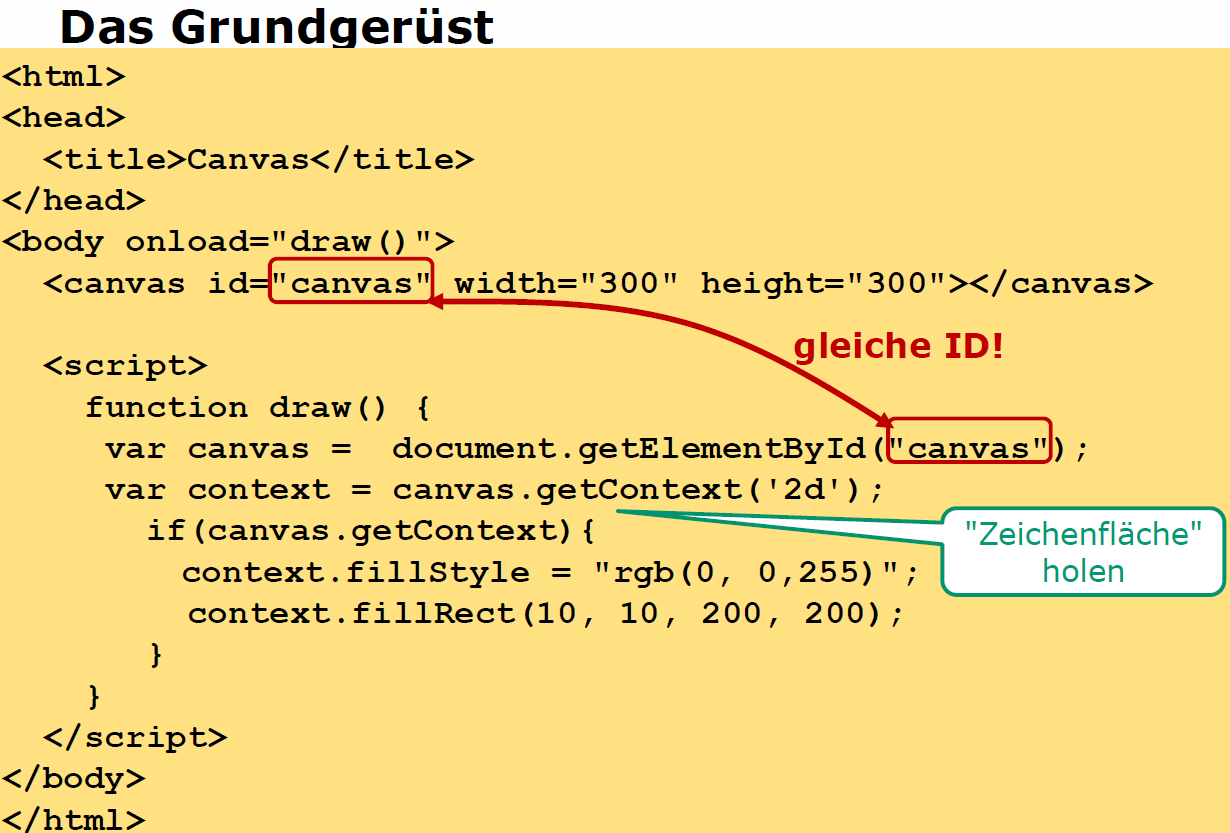
Meldung in Console-Log vom Browser ausgeben:

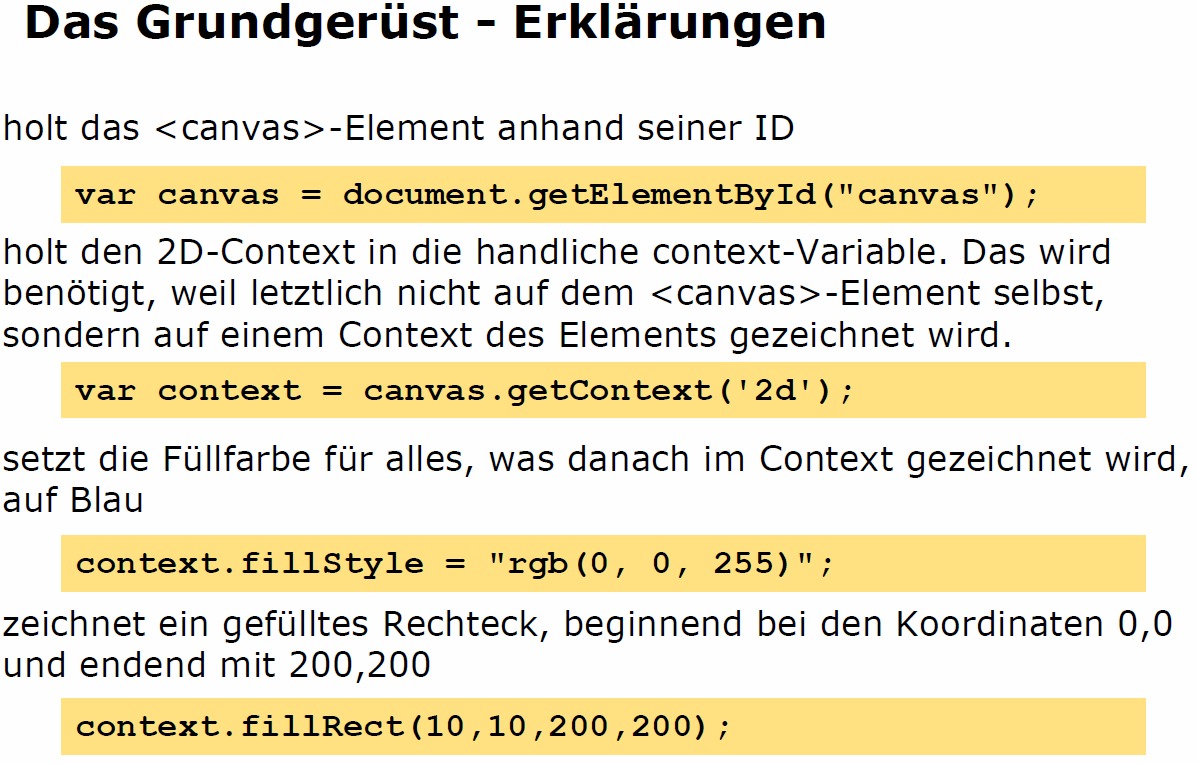
console.log("Das Ergebnis: " + resultat);

**CANVAS**

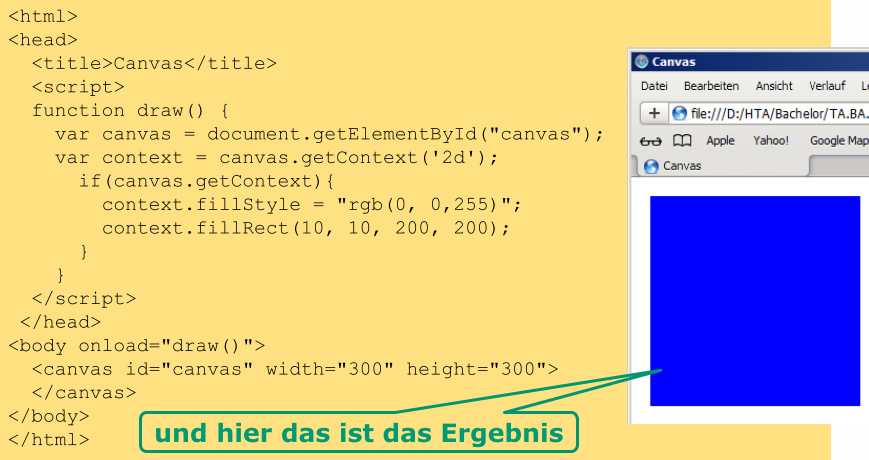
- Mit <canvas> wurde ein HTML-Element eingeführt, mit dem dynamisch Bitmap-Grafiken gerendert und in Dokumente integriert werden können.

- Zum Erstellen/Zeichnen kann eine Scriptsprache zum Einsatz kommen. Wir verwenden aus naheliegenden Gründen JavaScript.

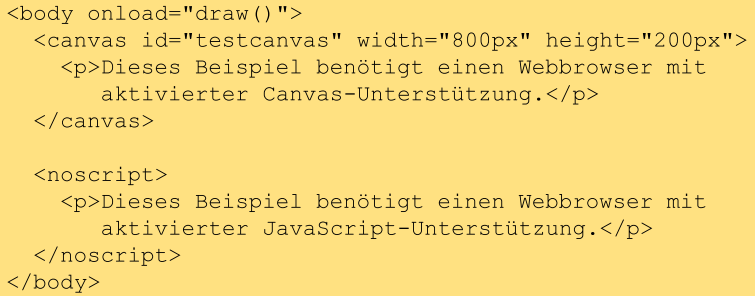


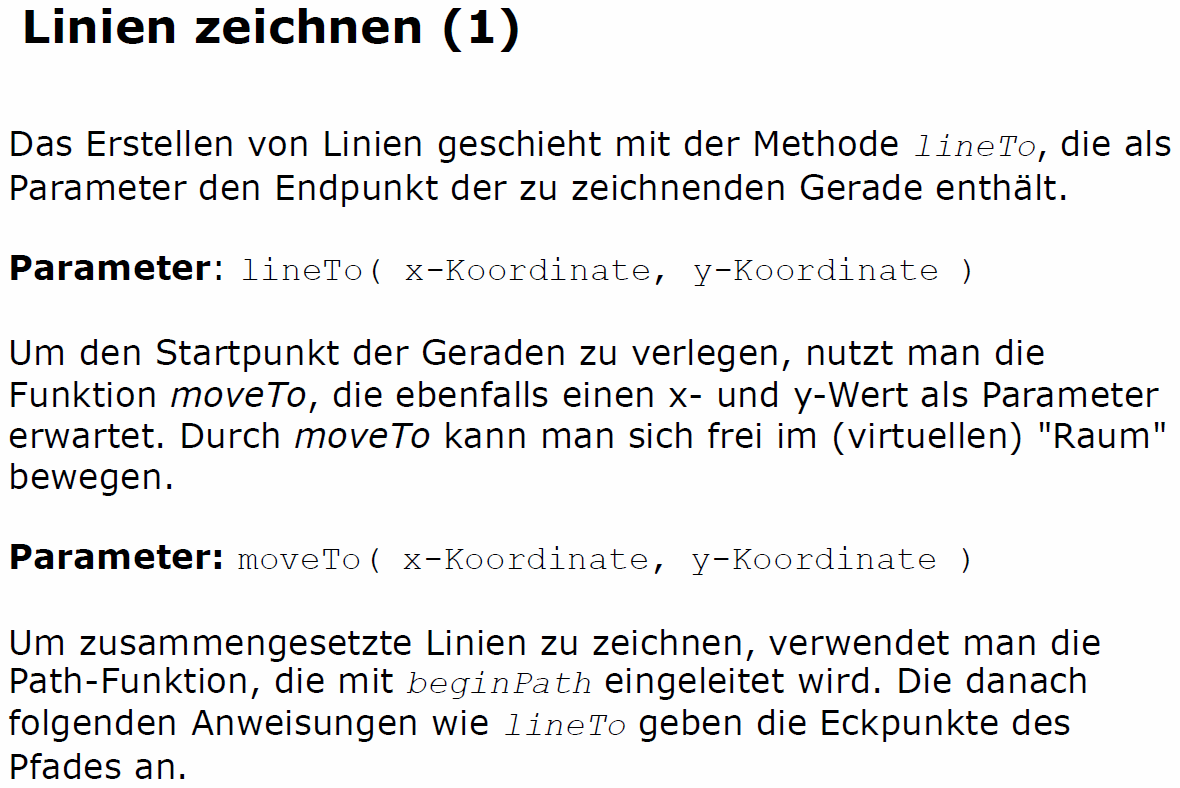


**Beispiel**

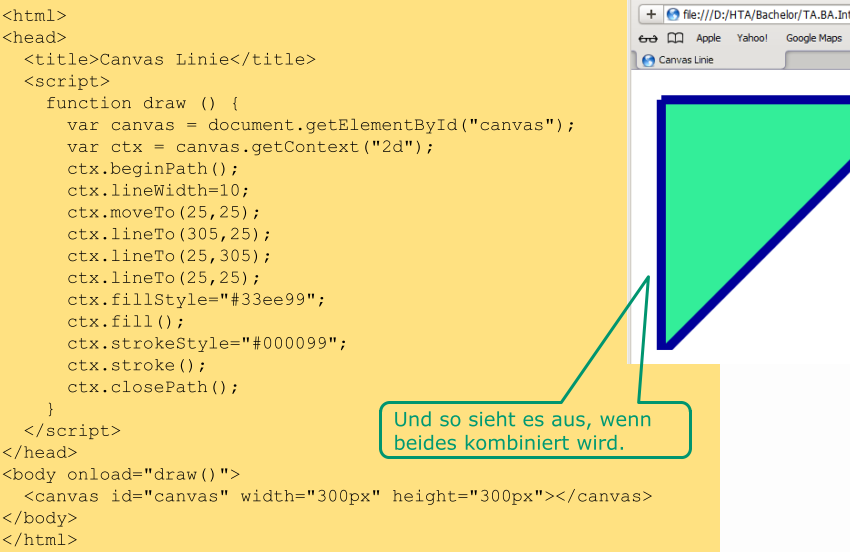


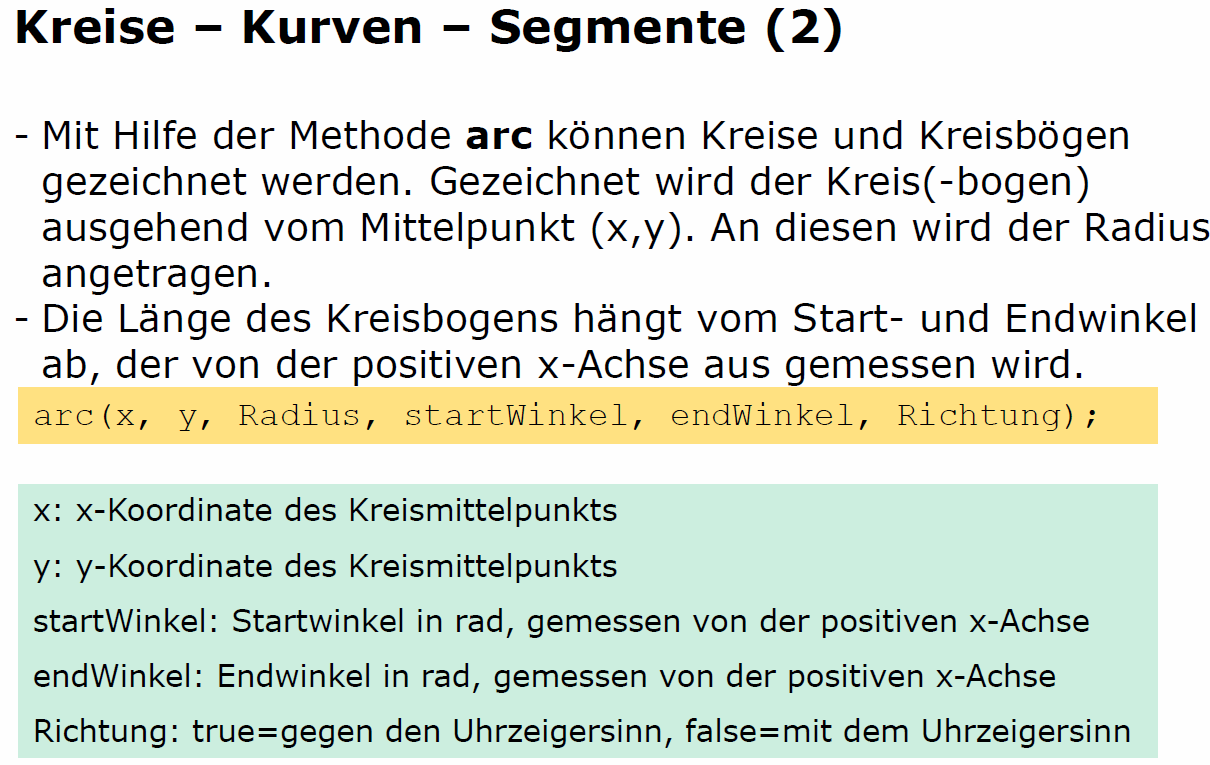
**Fehlermeldung bei Browserinkompatibilität**

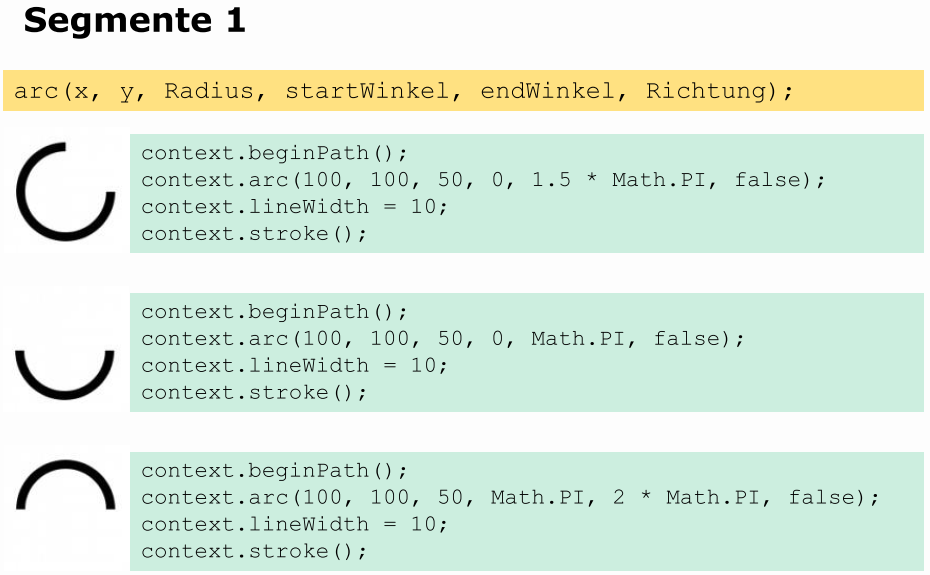




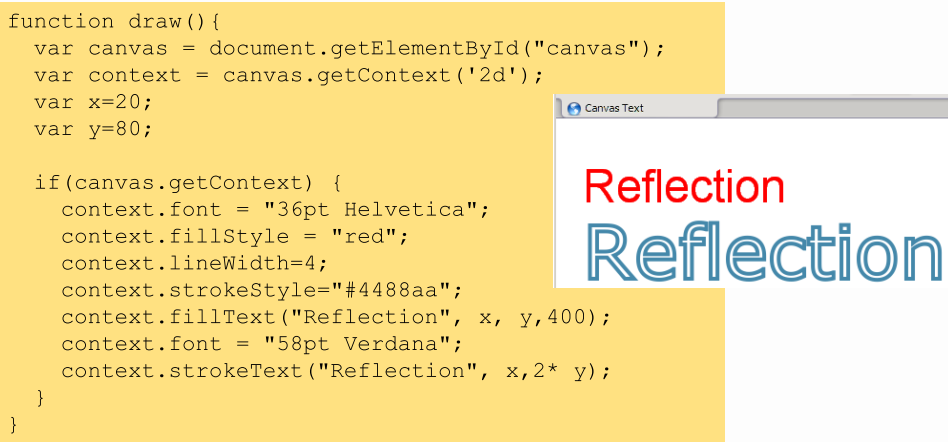
**Beispiel**







**Textdarstellung**



**Beispiel allgemein**

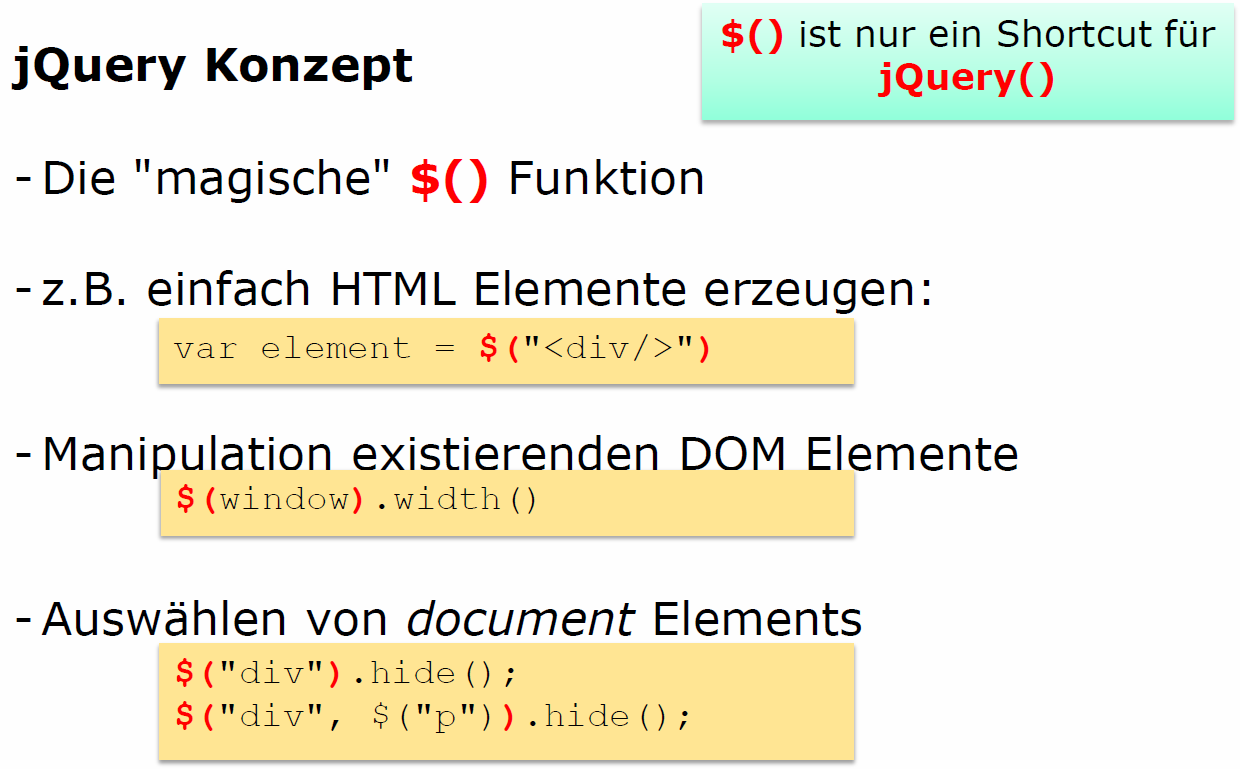
Bestimmte Seite laden:

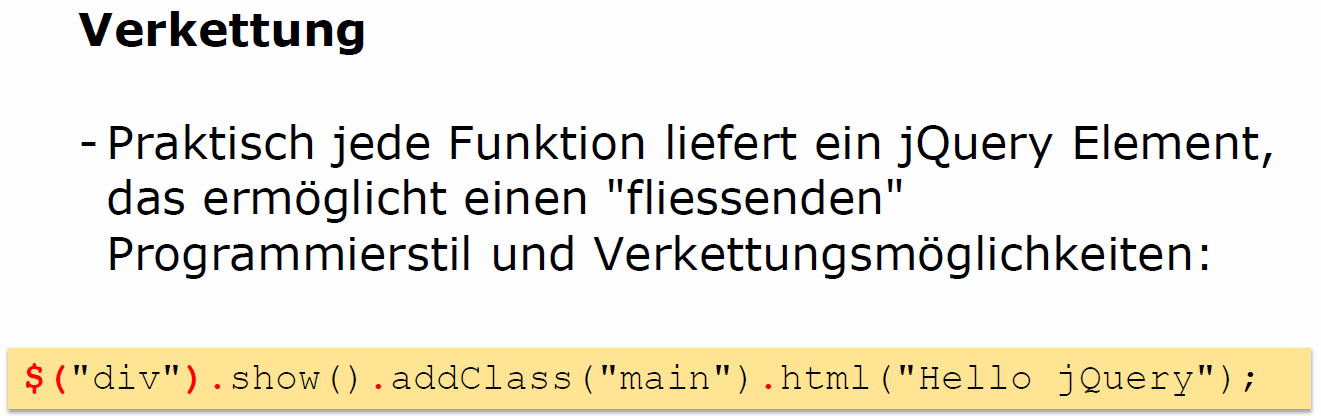
window.location.assign("http://www.xx.ch")

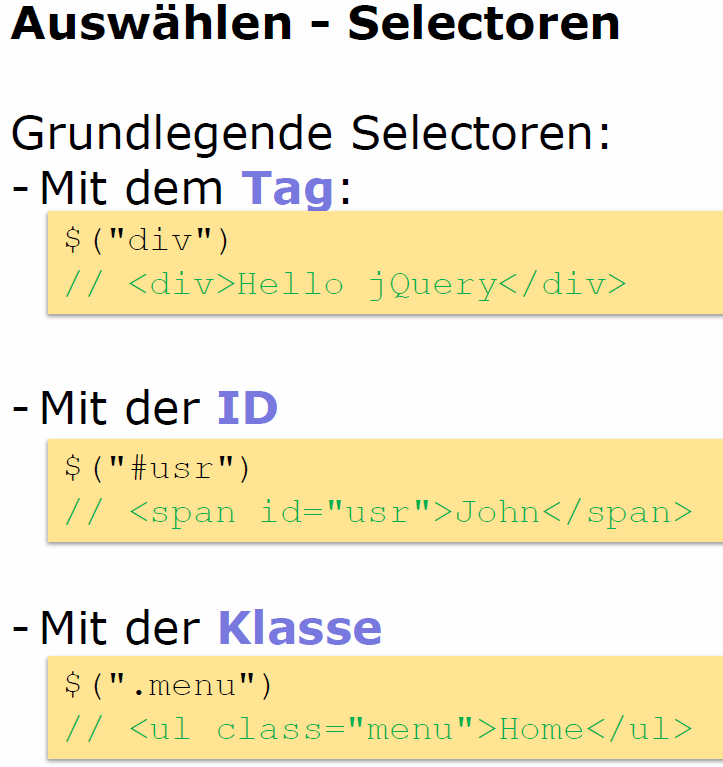
Eine Seite zurück gehen:

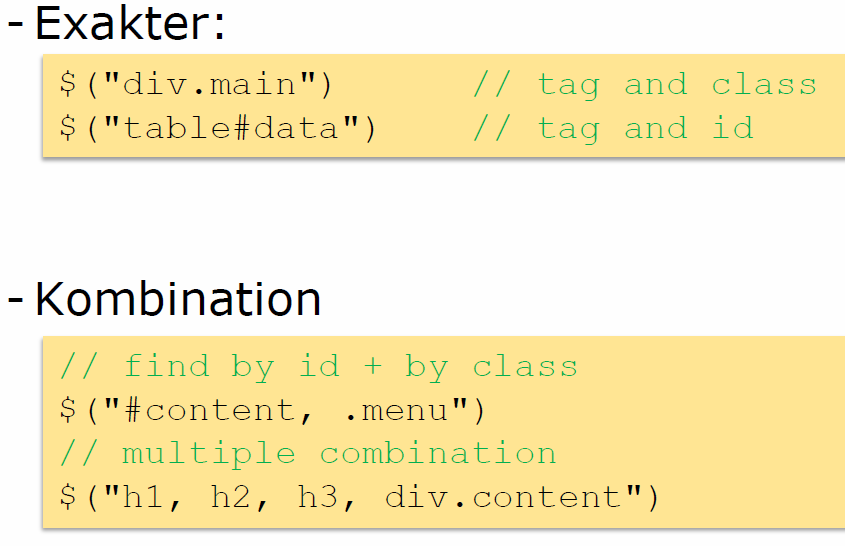
window.history.back()

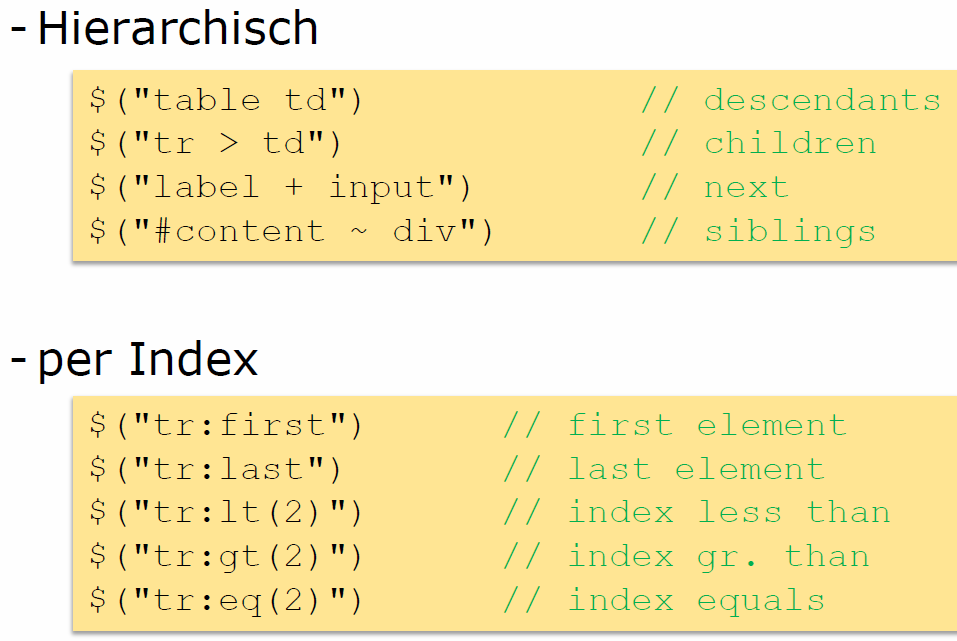


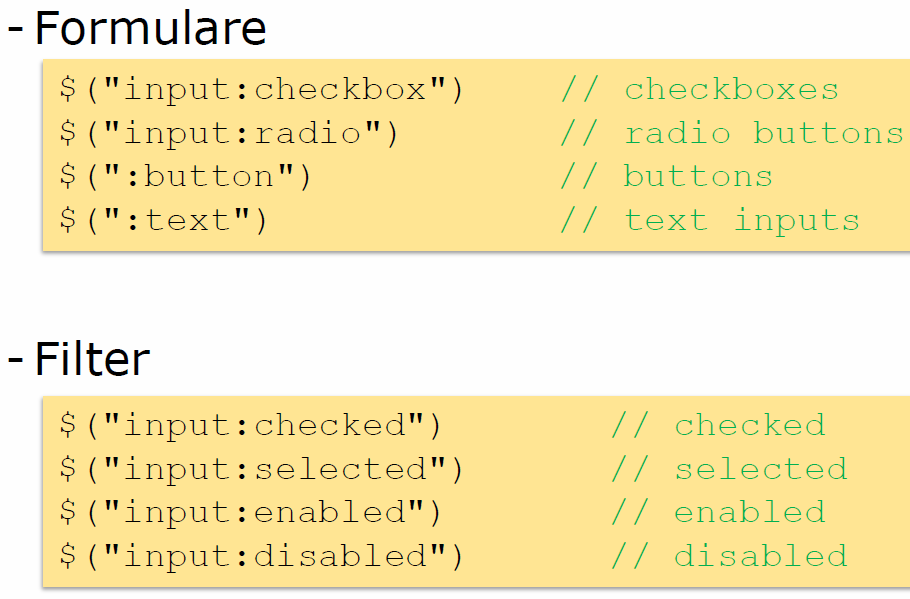


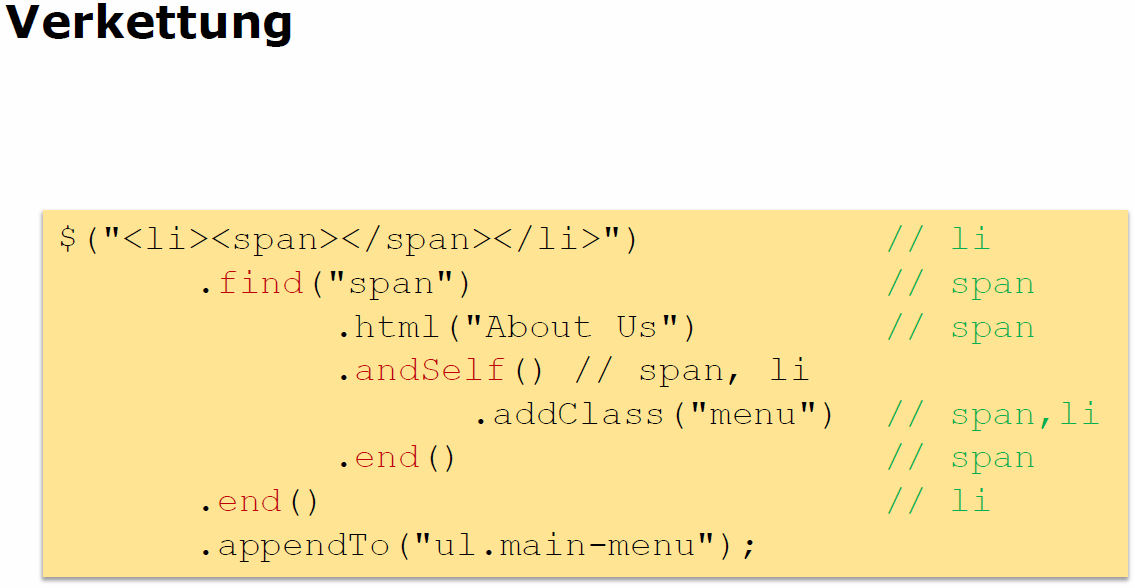


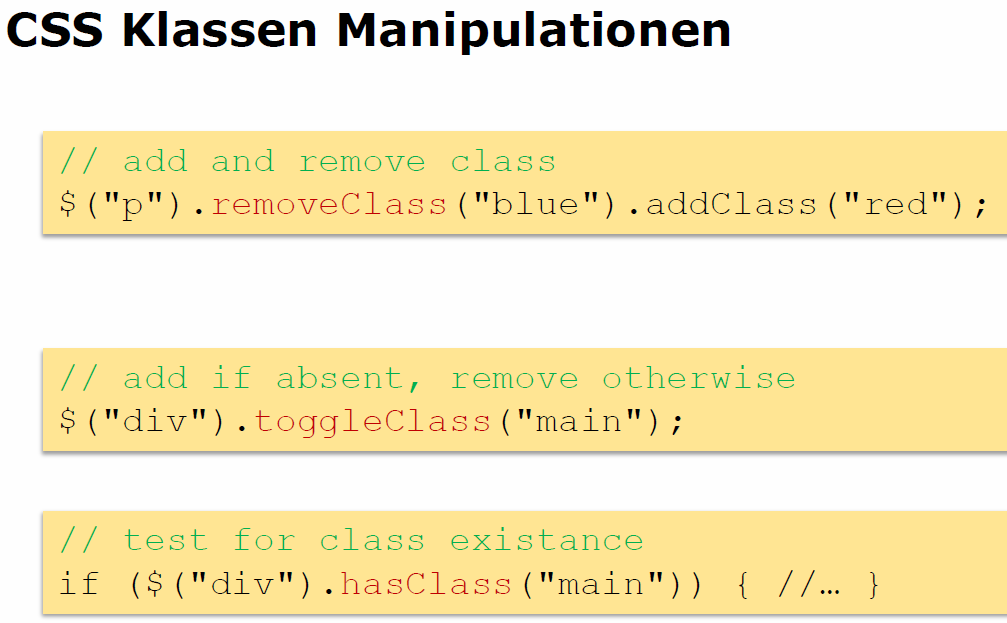




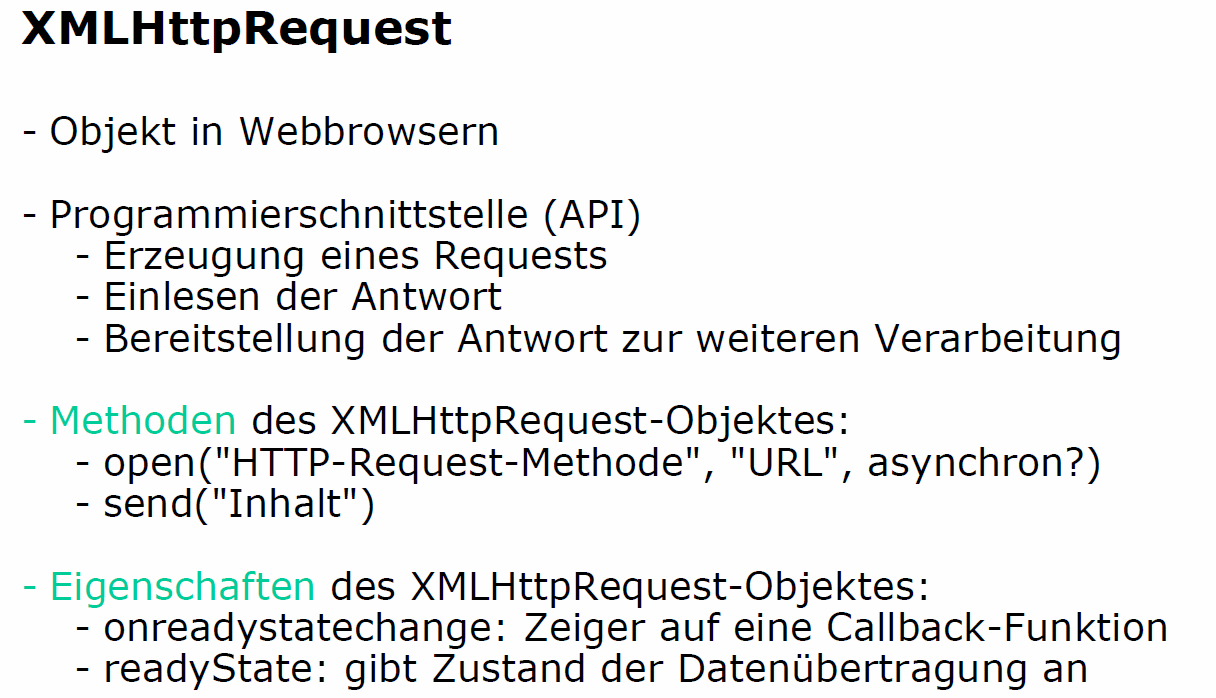








**AJAX**



**Internetworking**

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 anschliessenden 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 Überscheiben von Variabeln 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);

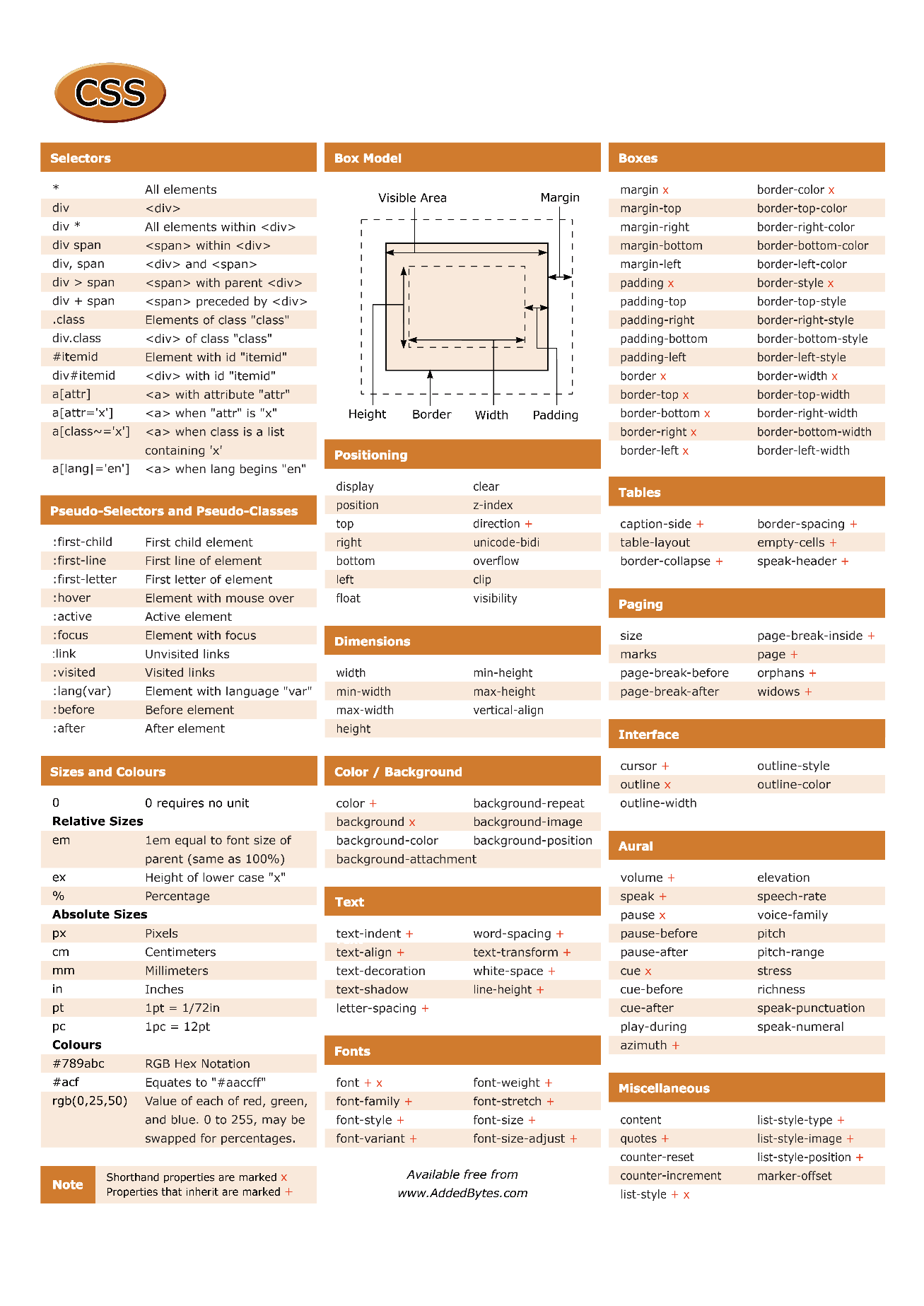
**SESSION starten, ID auslesen und beenden**

session\_start();

session\_id();

session\_destroy();

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





Überschrift

/line

/daten