

EINE EINFÜHRUNG IN PROGRAMMIERUNG VON  
WEBAPPLIKATIONEN MIT JAVASCRIPT.

# JAVASCRIPT KOMPAKT

# DIE ENTSTEHUNGSGESCHICHTE VON JAVASCRIPT



# BRENDAN EICH

- entwickelt 1995 die Sprache Mocha/LiveScript für den Netscape Navigator 2.0



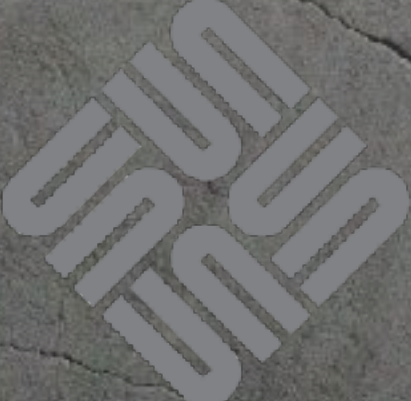


# JAVASCRIPT

- Sun Microsystems und Netscape kooperieren - Ziel: Javaapplets werden mit LiveScript gesteuert.
- LiveScript wird in Javascript 1.0 umbenannt, Netscape stellt LiveConnect zur Verfügung.



NETSCAPE®

 Sun

The Sun Microsystems logo, featuring a stylized sun icon made of horizontal lines, is positioned to the left of the word 'Sun' in a large, bold, sans-serif font.

# ECMA - EUROPEAN COMPUTER MANUFACTURER ASSOCIATION

- Die ECMA übernimmt die Standardisierung des allgemeinen Teils von Javascript:
- Die einfachen Variablentypen 'number', 'string' und 'boolean' werden genau beschrieben, deren Deklaration und Verhaltensweisen.
- Ebenso die komplexen Typen 'array', 'object' und 'function'.
- Dazu kommen einige Funktionsobjekte/-konstruktoren: Date, Math, RegExp, JSON
- Die Methode 'var' wird Pflicht, um Variablen zu deklarieren, Die 'use strict' Klausel trennt ECMA-konformes Scripting von 'freiem' Schreiben.

- Heute gelten die ECMA Standards 5.1 und 6.0 (ES2015+).
- ES 6.0 ergänzt zum Beispiel 'class', 'private', 'public', 'protected' und weitere OO-Muster, aber auch Ausdrücke aus anderen Sprachen

# DIE MOZILLA FOUNDATION: DOM - JAVASCRIPT

- Die Mozilla Foundation als Nachfolger der ehemaligen Netscape Entwicklergruppe entwickelt den DOM-Part von Javascript, der für das Verhalten im Browser verantwortlich ist und implementiert dies in seinen Browser 'Firefox'
- Darin befinden sich alle DOM-Objekte und Methoden, wie 'window', 'document', 'navigator' etc. Diese werden von allen Browserherstellern weitestgehend übernommen.
- Mozilla entwickelt das NICHT-DOM-Objekt console zur Ausgabe von Werten und Typen in der Browserconsole. (Firebug war auch die erste Konsole für das Debugging im Browser).



- Microsoft entwickelte hier teilweise eigene Lösungen, die besser zu Architektur der Microsoft-Software passen: zum Beispiel das ActiveXObject, das bei Mozilla XMLHttpRequest heisst.
- Script, VBScript waren Microsofts frühe Implementationen der Javascript Idee. Heute unterstützt Microsoft Standard Javascript, entwickelt aber mit Typescript eine verbesserte Notationsweise.



# DIE JAVASCRIPT ENGINES IN DEN BROWSERN

- Jeder Browser verwendet eine eigene Engine zur Interpretation und Kompilierung von Javascript.
- Chrome, Safari und Opera verwenden eine Version der Opensource Javascript Engine V8. Microsoft hat Chakra entwickelt und verwendet diese ab der Versionen 10 des Internet Explorer.

- Die Browser implementieren teilweise zusätzliche Objekte und Methoden, so gibt es zum Beispiel in Googles Chrome ein 'chrome' Objekt, das die Grundlagen für Chrome-Apps liefert.

# BEKANNTE ANWENDUNGEN VON JAVASCRIPT AUSSERHALB DES BROWSERS

- Adobe Flash
- Adobe PDF
- Nodejs

# JAVASCRIPT IN HTML EINBINDEN



# WO WIRD JAVASCRIPT EINGEBUNDEN?

- Im Allgemeinen wird Javascript heute am Ende des Body Elementes der HTML Datei.
- Das stellt einen reibungslosen Ladevorgang des DOM und CSS sicher, bevor mit dem Parsen und Kompilieren von Javascript begonnen wird.
- Skripte, die nicht auf das DOM zugreifen und nicht sehr lang sind, werden im Body positioniert, (z.B. das Google Analytics Skript).



# JAVASCRIPT INCLUDES

Das Einbinden externer Javascript-Dateien hat einige Vorteile:  
Es muss nur eine (einzige) Datei aktualisiert werden, alle  
Seiten nutzen automatisch das (geänderte) Skript.  
Die HTML Seiten sind übersichtlicher und bleiben sauber;

```
<html>
<head> ... </head>
<body>
    ...
    <script src="_scripts/javascript.js" ... ></script>
</body>
</html>
```

# ASYNC

```
<script async src="...">
```

Das Script wird asynchron zum Rest der Seite ausgeführt. Es startet, wenn es geladen ist.

# DEFER

```
<script defer src="...">
```

Das Script wird in der Ladereihenfolge ausgeführt.  
Deferred Scripts laufen vor allen anderen geladenen Skripten.

# CHARSET

```
<script ... charset=„UTF-8"></script>
```

Das charset Attribut spezifiziert das Character Encoding der inkludierten Datei.

Es wird benötigt, wenn die inkludierte Datei einen anderen Zeichensatz verwendet, als die HTML Datei.

# DIE KOMPONENTEN VON JAVASCRIPT





# JAVASCRIPT

- Javascript ist im Grunde keine Programmiersprache (das ist ECMA Script), sondern eine Objektbibliothek..
- -> Objektbasiert, nicht klassenbasiert.
- Objekt bedeutet hier entweder ein Objekt aus dem Document Object Model, sprich ein HTML-Knoten (Node), oder ein selbst erzeugtes.



THE JAVASCRIPT ARCHITECTURE

ECMA  
European Computer Manufacturer  
Association

Common Parts:  
Simple and Enhanced Variables,  
Object Constructors,  
strict coding rules

global unnamed object		
String	Number	Boolean
Array	Object	Function
Date	Math	PregExp
JSON	...	...

CLIENTSIDE SCRIPTING

SERVERSIDE SCRIPTING

Mozilla Foundation

Adobe

Microsoft

Ryan Dahl

Microsoft

Browser DOM

Flash, PDF

typescript

Node.js

Jurassic

window	document	navigator	console	...	...	...
--------	----------	-----------	---------	-----	-----	-----

???	socket	net	http	...	...	...
-----	--------	-----	------	-----	-----	-----

objects

frames[]	appName	...
images[]	geolocation	...
forms[]	...	
...		

methods

querySelector()	...	log()
getElementById()	...	dir()
...		

values

screenWidth	online
screenHeight	...



# ZWEI BESONDERHEITEN VON JAVASCRIPT

- ‚var‘ erzeugt Funktionssscopes.
- ‚let‘ (ES 6) erzeugt Controlsopes
- Objekte sind public.
- Werte haben Typen, es gibt aber keine implizite Typenüberprüfung.
- Typen können sich ändern.

# ECMA OBJEKT-PROTOTYPEN

- Das **namenlose globale Objekt**, das alle Variablen und Objekte enthält. (**function () { ... }()**)
- **Object** als allgemeiner Prototyp, von dem alle Objekte abgeleitet sind
- **Number** als Prototyp für Zahlen (64-Bit-Gleitkommazahlen gemäß IEEE 754)
- **String** als Prototyp für Zeichenketten (16-bit UCS-2 Zeichenketten)
- **Boolean** als Prototyp für boolesche Werte (true, false)
- **Function** als Prototyp für Funktionen
- **Array** als Prototyp für Arrays (numerisch indiziertes Objekt)

# TYPEN IN JAVASCRIPT

number	<code>a = 1;</code>	(kein int, float, double)
string	<code>a = 'Hallo'; a="H";</code>	(no char);
boolean	<code>a = 3&gt;4; a=false;</code>	
object	<code>a = {};</code>	
array	<code>a = [];</code>	
function	<code>a = function () {};</code>	



# UNIFIED NUMBER TYPE

```
a = 0.1; b = 0.2; c = 0.3;  
(a + b) + c === a + (b + c) // false
```

Die Teilung von zwei Integerzahlen  
kann einen Nicht-Integerwert hervorbringen.

```
10 / 3 = 3.3333333333333335
```

Unäre Operatoren können Strings in Zahlen konvertieren.

```
+"42" = 42
```

# + ODER CONCAT

Addition und Stringverkettung?

$$3 + 4 = 7$$

$$'$' + 3 + 4 = '$34'$$

$$3 + '4' = 34$$

$$3 + 4 + '5' = 75$$

# JSON - JAVASCRIPT OBJECT NOTATION

```
var obj = {  
  key_1 : 'Value 1',  
  key_2 : 42,  
  key_3 : false,  
  key_4 : [true, 2, 'drei'],  
  key_5 : { ... },  
  key_6 : function () { ... }  
}
```

# JSON NACH ALLGEMEINEN REGELN

```
{  
  "key_1" : "Value 1",  
  "key_2" : 42,  
  "key_3" : false,  
  "key_4" : [true, 2, "drei"],  
  "key_5" : { ... },  
}  
  
[  
  ["Michael", "Cologne", "michael@zenbox.de"],  
  ["Paula", "Stuttgart", "paula@zenbox.de"]  
]
```

# EINGEBAUTE OBJEKTE, DIE VON ECMASCRIPT DEFINIERT WERDEN.

- **Math** stellt Konstanten und Methoden für mathematische Operationen bereit. Math kann nicht als Konstruktor dienen.
- **Date** für Operationen mit Daten bzw. Zeitpunkten und Datumsformaten
- **RegExp** für reguläre Ausdrücke
- **JSON** zur Verarbeitung von Objekten



# DIE OBJEKT-BIBLIOTHEK DES BROWSERS

`window, document, navigator, screen, history, localStorage, console, XMLHttpRequest, ...`

Das `window`-Objekt selbst ist dabei de facto das globale Objekt, indem einer Variablen `window` das globale Objekt zugewiesen wurde.

```
(function () {  
    var window = this;  
})();
```

# DIE SPRACHELEMENTE VON JAVASCRIPT

# KOMMENTARZEICHEN

```
// slashslash für einzeiligen Kommentar  
/*  
    slashstar  
    für einen Block  
    Kommentar  
*/
```

# VERWENDUNG VON JAVADOC-KOMPATIBLEN KOMMENTAREN IST GUT FÜRS TEAMWORK

```
/* vim: set expandtab tabstop=4 shiftwidth=4 softtabstop=4: */  
/**  
 * Short description for file  
 *  
 * Long description for file (if any)...  
 *  
 * written in Javascript 1.7, jQuery 1.7.2  
 *  
 * @package      myApplication  
 * @author       Michael Reichart <reichart@michaelreichart.de>  
 * @author       Christian Marx  
 * @copyright    1999–2012 Michael Reichart  
 * @license      http://www.michaelreichart.de/license/1.txt  
 * @version      SVN: $Id$  
 * @link         http://host.net/package/myApplication  
 * @since        File available since Release 1.0.0  
 * @deprecated   File deprecated in Release 2.0.0  
 */
```

# STANDARKOMMENTARE FÜR FUNKTIONEN

```
/**
 * Short description for Method
 *
 * Long description for method (if any)...
 *
 * @author      Michael Reichart
 * @author      Your Name
 * @version     1.0.0
 * @since       Method available since Release 1.0.0
 * @deprecated  File deprecated in Release 2.0.0
 *
 * @param  type $varName
 * @return type
 */
```

# VARIABLENNAMEN

```
var
  a, _b, $c,
  _, $,
  a1, b_3, c$, myCamelCaseVariablesName,
  x = null,
  __ia__is_ie7_askjeu = false;
```

Konstruktorfunktionen beginnen mit einem Grossbuchstaben.

```
function Auto () { ... }
var myAuto = new Auto();
```

# JAVASCRIPT OBJECT NOTATION - JSON

Ein Objekt ist eine dynamische Sammlung von Eigenschaften. Jede Eigenschaft hat eine String als Namen. Der String ist unique!  
Die Attribute sind public !

```
var tier = {  
    art      : "Hund",  
    beine    : 2,  
    fluegel  : 0,  
    bellen   : function(){  
        log('wau!');  
    }  
};
```

```
tier.beine = 4;  
tier.schwanz = 1 ;  
delete tier.fluegel;
```

# FUNKTIONEN UND FUNKTIONALE OBJEKTE

```
function log(msg) {  
    console.log('log: ' + msg);  
}
```

```
var log = function (msg) {  
    console.log('log: ' + msg);  
}
```



# NUMBERS UND NUMERISCHE LITERALE

```
var n1 = 9.81;    // dezimal  
var n2 = 0xa3;    // hexadezimal  
var n3 = 073;     // oktal
```

```
    .01024e4  
    1.024e+3  
    10.24E2  
    102.4E+1  
1024.e0  
1024.00  
1024  
10240e-1
```

# METHODEN FÜR NUMERISCHE WERTE

```
var a = 4
```

```
a.toExponential()
```

```
a.toFixed()
```

```
a.toLocaleString()
```

```
a.toPrecision()
```

```
a.toString()
```

```
a.valueOf()
```

# NAN - NOT A NUMBER

Ist das Ergebnis von nicht definierten oder fehlerbehafteten Rechenoperationen.  
NaN ist giftig: Jede arithmetische Operation, die mit einem NaN rechnet, wird NaN als Ergebnis liefern.  
NaN ist mit nichts anderem vergleichbar, auch mit sich selbst nicht.

```
NaN === NaN // false  
NaN !== NaN // true
```

# STRINGS UND STRING.LENGTH

- Eine Folge von 0 oder mehr 16 bit Unicode Buchstaben (UCS-2)
- Die "length" Eigenschaft gibt die Anzahl der 16-bit Zeichen in einem String zurück.
- Erweiterte Zeichen werden als 2 Zeichen gezählt.

# + VERBINDEN VON STRINGS

+ kann Strings verbinden oder addieren.

```
'$' + '1' + '2' === '$12';  
'$'.concat('1').concat('2');
```

```
'$' + 1 + 2 = ', $12';  
1 + 2 + „$“ = „3$“
```

# ZAHLEN IN EINEN STRING KONVERTIEREN

```
str = num.toString();
```

```
str = String(num);
```

# SPRINGS IN EINE ZAHL KONVERTIEREN

Mit der Number Funktion:  
`num = Number(str);`

Mit dem + Präfixoperator:  
`num = +str;`

Mit der parseInt/parseFloat Funktion:  
`num = parseFloat(str);`  
`num = parseInt(str);`

# DIE PARSEINT FUNKTION

Die Konvertierung mit `parseInt` endet beim ersten Nicht-Ziffern-Zeichen.

```
parseInt(str, 10);  
parseInt("12em", 10) === 12;
```

Das Anhängsel (10) stellt das Dezimalsystem ein und sollte immer verwendet werden.

```
parseInt("08")      === 0 (ES3)  
parseInt("08", 10) === 8
```



# STRING METHODEN

charAt()  
charCodeAt()  
compareLocale()  
concat()  
indexOf()  
lastIndexOf()  
localeCompare()  
match()  
replace()  
search()

slice()  
split()  
substring()  
toLocaleLowerCase()  
toLocaleUpperCase()  
toLowerCase()  
toString()  
toUpperCase()  
trim()  
valueOf()

# BOOLEAN UND FALSY VALUES

```
false === false;
```

```
0 == false;
```

```
'' == false;
```

```
null == false;
```

```
undefined == false;
```

```
Nan == false;
```

```
var variable;
```

```
variable == undefined == false;
```

Alle anderen Werte sind truthy!

# BOOLEAN UND FALSY VALUES

```
1 == '1'  
1 !== '1'  
1 === 1
```

```
0 == false;  
' ' == false;  
null == false;  
undefined == false;  
Nan == false;  
  
var variable;  
variable == undefined == false;
```

Alle anderen Werte sind truthy!

# ARRAYS

```
var array = [true, 'Zwei', 3];  
  
array[0]; array[1];  
  
array.push('new value')  
  
for (i = 0; i < a.length; i++) {  
    var val = a[i];  
}  
  
array[array.length] = 'new value';
```

# ARRAY METHODS

concat  
every  
filter  
forEach  
indexOf  
join  
lastIndexOf  
map  
pop  
push

reduce  
reduceRight  
reverse  
shift  
slice  
some  
splice  
toString  
unshift

# SORT

```
var n = [4, 8, 15, 16, 23, 42];  
n.sort();  
  
// n is [15, 16, 23, 4, 42, 8]
```

# ELEMENTE EINES ARRAY LÖSCHEN

```
myArray = ['a', 'b', 'c', 'd'];
```

```
delete myArray[1];  
// ['a', undefined, 'c', 'd']
```

```
myArray.splice(1, 1);  
// ['a', 'c', 'd']
```

# NULL UND UNDEFINED

- **null** ist der Standard-**leer**-wert für Variablen und Parameter.
- Es ist der Wert für fehlende Member in Objekten.
- **undefined** wird zurückgegeben, wenn ein Objekt nicht existiert.
- null und undefined sind falsy values.



# CALL BY REFERENCES OR BY VALUE?

# CALL BY REFERENCE

- Objekte können als Argumente in Funktionen übergeben werden. Und können von Funktionen als Returnwert zurückgegeben werden.
- Objekte werden dabei als Referenz behandelt.

# CALL BY VALUE

- Die einfachen Variablentypen `string`, `number`, und `boolean` werden als Wert übergeben.

# WERTE ODER WERTE/TYPEN-VERGLEICH?

Der `===` Operator vergleicht Objektreferenzen, und nicht deren Werte. Nur wenn beide Operanden ein und dasselbe Objekt sind.

```
1 === true -> false  
1 ==  true -> true
```

# OPERATOREN

# OPERATOREN

- Arithmetrisch + - \* / %
- Vergleichend == != < > <= >= === !==
- Logisch && || !
- Bitweise & | ^ >> >>> <<
- Ternary ? :

# DEFAULT OPERATOR

```
function (arg) {  
    var value = arg || default;  
}
```



# BITWEISE

& | ^ >> >>> <<

Die bitweisen Operatoren konvertieren die Operanden zu einer 32-bit vorzeichenfähigen Integerzahl und liefern das Ergebnis wieder als 64-bit Fließkommazahl ab.

# STATEMENTS

# STATEMENTS

```
if  
switch  
while  
do  
for  
break  
continue  
return  
try ... catch/throw
```

# FOR STATEMENT

Iteriert durch alle Elemente eines Arrays:

```
for (var i = 0; i < array.length; i++) {  
    // within the loop,  
    // i is the index of the current member  
    // array[i] is the current element  
}
```

# FOR IN STATEMENT

Iteriert durch alle Elemente eines Objektes:

```
for (name in object) {  
    if (object.hasOwnProperty(name)) {  
        // within the loop,  
        // name is the key of current member  
        // object[name] is the current value  
    }  
}
```

# SWITCH STATEMENT

- Mehrwegverzweigung
- Der Switch-Wert muss keine Zahl, sondern kann auch ein String sein.
- Die einzelnen Cases können Ausdrücke und Programmzeilen beinhalten.
- Gefährlich: Cases fallen in den nächsten Case durch, solange der Case nicht durch ein break unterbrochen und beendet wird.

# SWITCH STATEMENT

```
switch (expression) {  
    case ';' :  
    case ',' :  
    case '.' :  
        punctuation();  
        break;  
    default:  
        noneOfTheAbove();  
}
```

# WHILE () {}

```
while (condition) {  
    doSomethingAwesome();  
}
```

```
var a = 0;  
while(a<4) {  
    doSomethingAwesome();  
    a++;  
}
```



# DO {} WHILE ()

```
do {  
    doSomethingAwesome();  
} while (condition)
```

```
var a = 0;  
do {  
    doSomethingAwesome();  
    a++;  
} while(a<4)
```

# THROW STATEMENT

```
throw new Error(reason);
```

```
throw {  
    name: exceptionName,  
    message: reason  
};
```

# TRY STATEMENT

```
try {  
    ...  
} catch (e) {  
    switch (e.name) {  
        case 'Error':  
            ...  
            break;  
        default:  
            throw e;  
    }  
}
```

# TRY STATEMENT

Die JavaScript Implementierung kann folgende  
Exception names ausgeben:

- 'Error'
- 'EvalError'
- 'RangeError'
- 'SyntaxError'
- 'TypeError'
- 'URIError'

# SCOPES - GÜLTIGKEITSBEREICHE VON VARIABLEN



# FUNKTIONEN BILDEN SCOPES

```
fn = function (arg) {  
    var member = arg || null;  
}
```

# ERZEUGEN GLOBALER VARIABLEN

```
<script>
  a = 2;           // global: window.a
  var c = 3;       // global: window.c
  function f () {
    b = 3;         // auch global!! window.b
    var d = 4;
  }
</script>
```

# LOKALE VARIABLEN - "VAR" INNERHALB VON FUNKTIONEN

```
function f () {  
    "use strict";  
    var a;    // Lokale Variable mit var  
    ... ;  
}
```



FUNKTIONEN, DIE SICH SELBST AUSFÜHREN.  
SIE WERDEN VOR ALLEN ANDEREN FUNKTIONEN INITIALISIERT UND KÖNNEN NEBENBEI  
DAZU VERWENDET WERDEN, PRIVATE KEYS UND METHODEN ZU VERWENDEN.

# IMMEDIATE FUNCTIONS

# IIFE - DIE IMMEDIATE INVOKED FUNCTION EXPRESSION

```
()();
```

Was, das soll eine Funktion sein?

Ja. Das erste Klammersymbol ist eine Javascriptausdruck für eine anonyme, sich nach dem Kompilieren sofort selbst ausführende Funktion.

Das zweite Klammersymbol ist eine Argumente-Klammer, mit der Argumente in die Funktion übergeben werden können.

# AUFBAU EINER IMMEDIATE FUNCTION

```
var extArgs;
```

```
( function (intArgs) { ... } )(extArgs);
```

```
/*  
innerhalb der Immediate Function wird in der Regel eine  
anonyme Funktion platziert, die durch die Immediate Function  
ausgeführt wird. Ihr werden die Argumente übergeben.  
*/
```

# IMMEDIATE FUNCTION MIT RÜCKGABEWERT

```
var extArgs,  
    ExtObj;
```

```
extObj = (function (intArgs) {  
    var intObj = {};  
    ...  
    return intObj;  
} )(extArgs);
```

```
/*
```

Da die Funktion sofort ausgeführt ist, gibt sie anstelle eines Funktionszeigers ein Ergebnis zurück.

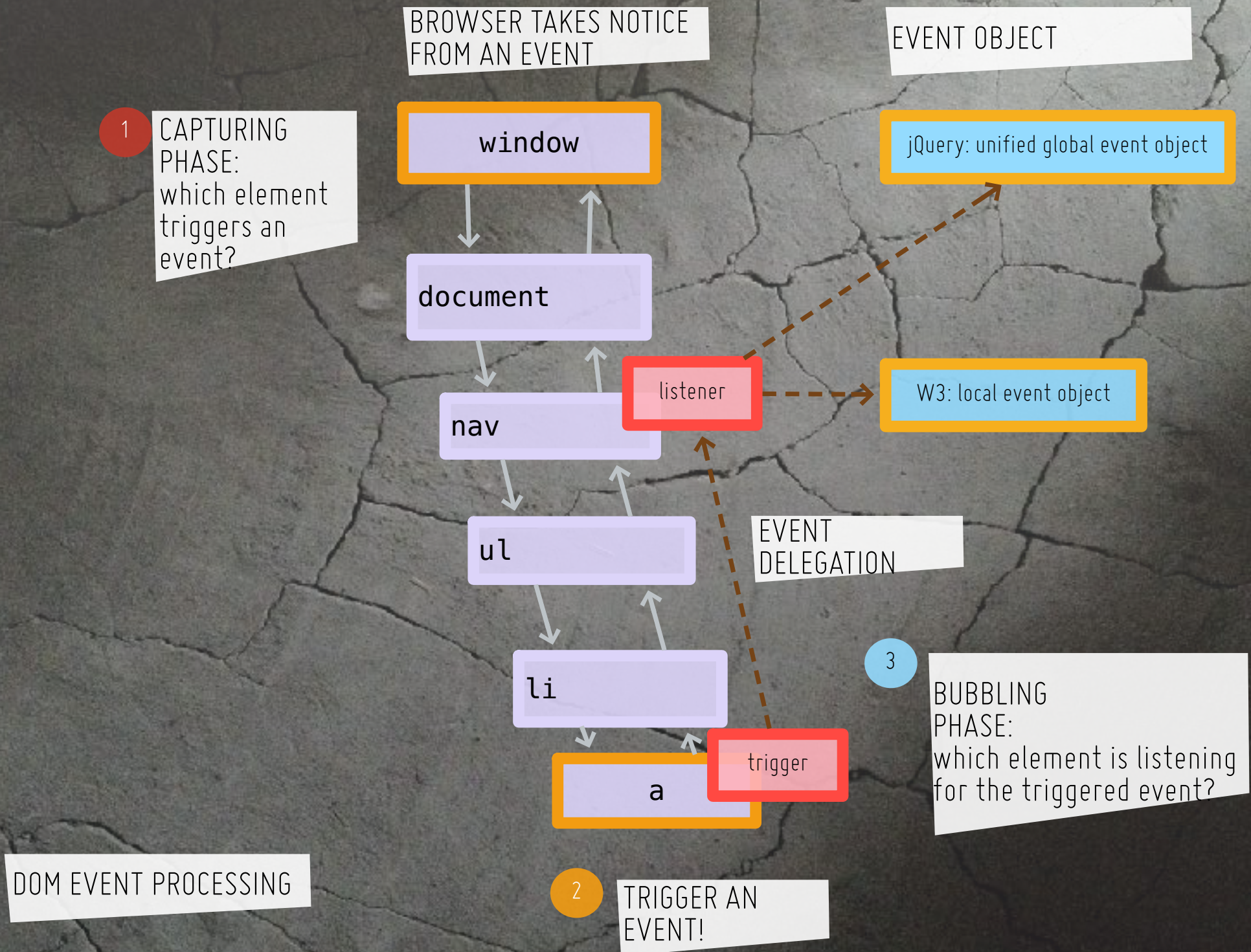
```
*/
```

# EIN OBJEKT MIT PUBLIC UND PRIVATE KEYS.

```
var animal = (function (type, legs, wings, sound) {  
  
    var type    = type    || 'Hund',  
        legs    = legs    || '4',  
        wings   = wings   || '0',  
        sound   = sound   || 'wau'  
  
    ;  
    function getType () { return type; }  
    function setType (value) { type = value; }  
    return ({  
        getType : getType,  
        setType : setType,  
        sound   : sound  
    });  
})();  
  
// Getter!  
console.log('Getterzugriff: ' + animal.getType());
```

# JAVASCRIPT EVENTS







# EVENTS

- Alle Elementobjekte und weitere Objekte besitzen die Methode `addEventListener`.
- Ein Eventlistener enthält einen Ereignistyp, ein Elementobjekt und eine Handler-Funktion.



```
ELEMENT.ADDEVENTLISTENER( "EVENT",  
                           HANDLERFUNKTION,  
                           CAPTURING);
```

- "event" ist ein String und enthält den Ereignistyp: "click", "mouseover", "load", "submit" ...
- Das Handler-Funktionsobjekt, das ausgeführt werden soll.
- Der dritte Parameter bestimmt, für welche Event-Phase der Handler registriert werden soll. Die Werte sind true oder false.
- false steht für die Bubbling-Phase  
(sollte als Standard verwendet werden).  
true für die Capturing-Phase.

# EIN BEISPIEL

```
window.addEventListener("load", start, false);

function start () {
    var pElement = document.getElementById("interaktiv");
    pElement.addEventListener("click", klickverarbeitung,
false);
}
function klickverarbeitung () {
    document.getElementById("interaktiv").innerHTML +=
        " Das ist dynamisch generierter Text.";
}
```

# EVENT-HANDLER ENTFERNEN: REMOVEEVENTLISTENER

Um die mit `addEventListener` registrierten Handler wieder zu entfernen, gibt es die Methode `removeEventListener`. Die Methode erwartet dieselben Parameter, die `addEventListener` beim Registrieren bekommen hat.

```
function beenden () {  
    pElement.removeEventListener("click", klickverarbeitung,  
false);  
}
```

# EIGENE EVENTS

```
// Einen neuen Event Typ anmelden  
var e = new Event('send');  
  
// Einen neuen CustomEvent Typ anmelden  
var e = new CustomEvent('send', {detail: 'some data'});  
  
// Den Event abschicken  
domObj.dispatchEvent(e);  
  
// Der Eventlistener, wie gewohnt  
domObj.addEventListener('send', function (event) { ... }));
```

# EVENTLISTENER IN IE < 9

```
window.attachEvent("onload", start);

function start () {
    var pElement = document.getElementById("interaktiv");
    pElement.attachEvent("onclick", klickverarbeitung);
}

function klickverarbeitung () {
    document.getElementById("interaktiv").innerHTML +=
        " Das ist dynamisch generierter Text.";
}
```

# EVENTS IN IE < 9 ENTFERNEN

```
function beenden () {  
    pElement.detachEvent("onclick", klickverarbeitung);  
}
```

# EINE BROWSERÜBERGREIFENDE EVENTFUNKTION

```
function addEvent (obj, type, fn) {  
    if (obj.addEventListener) {  
        obj.addEventListener(type, fn, false);  
    } else if (obj.attachEvent) {  
        obj.attachEvent('on' + type, function () {  
            return fn.call(obj, window.event);  
        });  
    }  
}
```

Eine bessere unter:

[http://therealcrisp.xs4all.nl/upload/addEvent\\_dean.html](http://therealcrisp.xs4all.nl/upload/addEvent_dean.html)

# DAS EVENTOBJEKT MIT PREVENTDEFAULT

```
function zeigeVollbild (eventObjekt) {  
    // Browserübergreifender Zugriff auf das Event-Objekt  
    if (!eventObjekt) eventObjekt = window.event;  
  
    // Existiert die Methode preventDefault? Dann rufe sie auf.  
    if (eventObjekt.preventDefault) {  
        // W3C-DOM-Standard  
        eventObjekt.preventDefault();  
    } else {  
        // Andernfalls setze returnValue  
        // Microsoft-Alternative für Internet Explorer < 9  
        eventObjekt.returnValue = false;  
    }  
};
```



## THE EVENT OBJECT

```
event
target
delegateTarget

which
type
meta-/shift-/alt-/ctrlKey

pageX
pageY

preventDefault()
stopPropagation()

...
```

## AN ONCLICK EVENT HANDLER

```
fn = {
  onclick : function (event) {
    event.preventDefault();
    event.stopPropagation();
    url = $(event.target).attr('href');
    fn.loadContentOf(url)
  },
  loadContentOf : function (url) {...}
}
```

USE `event.target`, NOT `this`!



# ECMA OBJEKTE

MATH



# DIE METHODEN DES MATH OBJECTS

```
var a = Math.abs(3.14152);  
var u = 4 * Math.PI * r;
```

abs()  
acos()  
asin()  
atan()  
atan2()  
ceil()  
cos()  
exp()  
floor()

log()  
max()  
min()  
pow()  
random()  
round()  
sin()  
sqrt()  
tan()  
PI

# DATE



# DER DATE KONSTRUKTOR

```
var now = new Date();
```

```
now.getDate(); // Tage des Monats (1–31)
```

Erzeugt eine Date-Instanz, die einen Zeitmoment enthält.

Datumsobjekte basieren auf einem Wert in Millisekunden seit dem 1. Januar 1970, UTC.

# DER DATE KONSTRUKTOR

```
var today = new Date();  
var birthday = new Date("December 17, 1995 03:24:00");  
var birthday = new Date("1995-12-17T03:24:00");  
var birthday = new Date(1995,11,17);  
var birthday = new Date(1995,11,17,3,24,0);
```

# METHODEN DES DATE OBJEKTES

Date.now()	Returns the numeric value corresponding to the current time - the number of milliseconds elapsed since 1 January 1970 00:00:00 UTC.
Date.parse()	Parses a string representation of a date and returns the number of milliseconds since 1 January, 1970, 00:00:00, local time.
Date.UTC()	Accepts the same parameters as the longest form of the constructor (i.e. 2 to 7) and returns the number of milliseconds since 1 January, 1970, 00:00:00 UTC.



# GETTER DER DATE INSTANZEN

<code>.getDate()</code>	Returns the day of the month (1-31) for the specified date according to local time.
<code>.getDay()</code>	Returns the day of the week (0-6) for the specified date according to local time.
<code>.getFullYear()</code>	Returns the year (4 digits for 4-digit years) of the specified date according to local time.
<code>.getHours()</code>	Returns the hour (0-23) in the specified date according to local time.
<code>.getMilliseconds()</code>	Returns the milliseconds (0-999) in the specified date according to local time.
<code>.getMinutes()</code>	Returns the minutes (0-59) in the specified date according to local time.
<code>.getMonth()</code>	Returns the month (0-11) in the specified date according to local time.
<code>.getSeconds()</code>	Returns the seconds (0-59) in the specified date according to local time.

# GETTER DER DATE INSTANZEN

<code>.getTime()</code>	Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC (negative for prior times).
<code>.getTimezoneOffset()</code>	Returns the time-zone offset in minutes for the current locale.
<code>.getUTCDate()</code>	Returns the day (date) of the month (1-31) in the specified date according to universal time.
<code>.getUTCDay()</code>	Returns the day of the week (0-6) in the specified date according to universal time.
<code>.getUTCFullYear()</code>	Returns the year (4 digits for 4-digit years) in the specified date according to universal time.
<code>.getUTCHours()</code>	Returns the hours (0-23) in the specified date according to universal time.
<code>.getUTCMilliseconds()</code>	Returns the milliseconds (0-999) in the specified date according to universal time.
<code>.getUTCMinutes()</code>	Returns the minutes (0-59) in the specified date according to universal time.

# GETTER DER DATE INSTANZEN

<code>.getUTCMonth()</code>	Returns the month (0-11) in the specified date according to u
<code>.getUTCSeconds()</code>	Returns the seconds (0-59) in the specified date according to
<code>.getFullYear()</code>	Returns the year (usually 2-3 digits) in the specified date acco Use <code>getFullYear()</code> instead.

# SETTER

- `setDate()`
- `setFullYear()`
- `setHours()`
- `setMilliseconds()`
- `setMinutes()`
- `setMonth()`
- `setSeconds()`
- `setTime()`

- `setUTCDate()`
- `setUTCFullYear()`
- `setUTCMonth()`
- `setUTCSeconds()`
- `setUTCHours()`
- `setUTCMilliseconds()`
- `setUTCMinutes()`
- `setYear()`

# CONVERSION GETTER

- `toDateString()`
- `toISOString()`
- `toJSON()`
- `toGMTString()`
- `toLocaleDateString()`
- `toLocaleFormat()`
- `toLocaleString()`

- `toLocaleTimeString()`
- `toSource()`
- `toString()`
- `getTimeString()`
- `toUTCString()`
- `valueOf()`

# REGEXP



# REGEXP

- Ein regulärer Ausdruck spezifiziert eine Syntax eines Suchausdrucks für Texte.
- Reguläre Ausdrücke folgen einer Syntax.
- RegExp erlaubt keine Leerzeichen oder Kommentare  
→ Sie sind schwer zu lesen.

# BEISPIELE FÜR REGULÄRE AUSDRÜCKE

Ein einfacher regulärer Ausdruck in Javascript:

```
var reg = /ab+c/;  
var reg = new RegExp("ab+c");  
  
var myArray = reg.exec("cdbbdsbz");
```

Ein komplexer Ausdruck für Email-Adressen:

```
var reg = /^[a-zA-Z0-9][\w\.-]*@(?:[a-zA-Z0-9][a-zA-Z0-9_-]+\.)  
+[A-Z,a-z]{2,5}$/;
```

```
var reg = /(\w+)\s(\w+)/;  
var str = "Jonas Schmidt";  
var newstr = str.replace(reg, "$2, $1");  
console.log(newstr);
```



# BEISPIELE FÜR REGULÄRE AUSDRÜCKE

```
var reg = /\(?\d{3}\)?([-\./])\d{3}\1\d{4}/;

function testInfo(phoneInput){
    var OK = reg.exec(phoneInput.value);

    if (!OK)
        window.alert(RegExp.input + " ist keine Telefonnummer mit Vorwahl!");
    else
        window.alert("Danke! Ihre Telefonnummer ist " + OK[0]);
}
```

# METHODEN DIE REGULÄRE AUSDRÜCKE VERWENDEN

EXEC	EINE METHODE VON REGEXP, DIE SUCHE NACH EINER ÜBEREINSTIMMUNG IN EINER ZEICHENKETTE DURCHFÜHRT UND GIBT EIN
TEST	EINE METHODE VON REGEXP, DIE EINE ZEICHENKETTE AUF EINE ÜBEREINSTIMMUNG ÜBERPRÜFT UND GIBT DEN INDEX DER ÜBEREINSTIMMUNG GIBT
MATCH	EINE METHODE VON STRING, DIE EINE SUCHE NACH EINER ÜBEREINSTIMMUNG IN EINER ZEICHENKETTE DURCHFÜHRT UND GIBT EIN
SEARCH	EINE METHODE VON STRING, DIE EINE ZEICHENKETTE AUF EINE ÜBEREINSTIMMUNG ÜBERPRÜFT UND GIBT DEN INDEX DER ÜBEREINSTIMMUNG GIBT
REPLACE	EINE METHODE VON STRING, DIE EINE SUCHE NACH EINER ÜBEREINSTIMMUNG IN EINER ZEICHENKETTE DURCHFÜHRT UND DIE ÜBEREINSTIMMUNG DURCH EINE ANDERE ERSETZT
SPLIT	EINE METHODE VON STRING, DIE ANHAND EINES REGULÄREN AUSDRUCKS ODER EINER REGEXP EINE ZEICHENKETTE IN EINER LISTE VON ZEICHENKETTEN AUFLÖST

# BESONDERE ZEICHEN UND AUSDRÜCKE

//	Start – Ende eines RegExp
\/	escaped Slash (\*, ...)
^	zu Beginn
\$	am Ende
.	Wildcard, beliebiges Zeichen
	Alternative

# ZEICHENBEREICHE

[ ] Set/Bereich von Zeichen [0-9], [A-Za-z0-9]  
findet jedes dieser Zeichen

[xyz] Zeichenbereich, der alle notieren Zeichen erfasst  
[^xyz] Erfasst keines der notierten Zeichen

# BESONDERE ZEICHEN UND AUSDRÜCKE

<code>\b</code>	Wortgrenze (Anfang/Ende)
<code>\s</code>	Jedes Leerzeichen, Tabulator oder Umbruch
<code>\d</code>	Jede Ziffer
<code>\w</code>	Jedes Vorschubzeichen
<code>\n</code>	Zeilenvorschub
<code>\w</code>	Jeder Buchstabe, inkl. <code>_</code>

# MENGENANGABEN

+	einen oder mehrere der vorangehenden Zeichengruppe
?	Keinen oder einen ...
*	keinen oder mehr ...
{x}	x mal ...
{x,}	x mal oder mehr ...
{x,y}	x bis y mal ...

# GRUPPIERUNGEN

(...) Gruppe; höchstens 9 Gruppen  
?: (...) Ausschlußgruppe

Parameter, auch Kombinationen sind möglich

/.../g globale Suche  
/.../i unabhängig von Groß- und Kleinschreibung  
/.../m Multiline

# JSON





# DAS JSON - OBJEKT

```
var obj = JSON.parse(text [, reviver])
```

Schreibt ein als JSON-String gegebenes Objekt in ein Objekt in den Speicher. So kann ein per Ajax gelieferter JSON-String weiterverarbeitet werden.

```
var = string = JSON.stringify(obj [, replacer] [, space])
```

Schreibt ein im Speicher liegendes Objekt in einen String um. Der String kann nun zum Server geschickt oder gespeichert werden.

# DIE BROWSEROBJEKTE IN JAVASCRIPT

# BROWSER OBJEKTE UND UNTEROBJEKTE

## **Window**

document

anchors

DOM

forms

elements

options

images

links

history

location

Seite

- Referenz auf das globale anonyme Objekt
- enthält das DOM (Document Object Modell)
- Ein Array mit allen anchor-Elementen aus dem DOM
- Ein Array mit allen form-Elementen aus dem DOM
- Ein Array mit allen Elementen einer Form
- 
- Ein Array mit allen img-Elementen aus dem DOM
- Ein Array mit allen Links aus dem DOM
- Ein Array mit allen besuchten URIs
- Ein Objekt mit Informationen zur aktuellen Seite

## **Navigator**

mimeTypes

plugins

- Ein Objekt mit Browserinformationen
  - Ein Array mit allen mimeTypes
- Ein Array mit allen Plugin Informationen

## **Screen**

- Ein Objekt mit Eigenschaften zum Bildschirm

# DAS WINDOW OBJEKT

Das windows Objekt liefert Informationen über das aktuell angesprochene Fenster (das das Dokument enthält).  
Es kennt u.a. folgende Eigenschaften:

```
window.frames  
window.location  
window.name  
window.opener  
window.parent  
window.status
```

und einige mehr.

# DAS WINDOW OBJEKT HAT METHODEN

Methoden sind Fähigkeiten (was kann ...) eines Objektes.

```
windows.alert()  
windows.blur()  
windows.clearTimeout()  
windows.close()  
windows.confirm()  
windows.focus()  
windows.prompt()  
windows.scroll()  
windows.setTimeout()
```

# AUßERDEM REAGIERT ES AUF EVENTS

Das Window Objekt kennt u.a. folgende Eventhandler:

<code>onBlur = ...</code>	<code>onFocus = ...</code>
<code>onLoad = ...</code>	<code>onUnload = ...</code>

Zum Beispiel stellt der folgende Event sicher, dass das, was innerhalb der anonymen Funktion steht, erst ausgeführt wird, wenn das Dokument vollständig geladen ist.

```
window.onload = function () {  
    ...  
}
```

# DAS NAVIGATOR OBJEKT

Das Navigator Objekt liefert Informationen über den verwendeten Browser.

```
navigator.appName  
navigator.appVersion  
navigator.appCodeName  
navigator.mimeTypes  
navigator.plugins  
navigator.userAgent  
navigator.plugins
```

<http://de.selfhtml.org/javascript/objekte/index.htm>

– JAVASCRIPT OBJEKTREFERENZ



# DOM MANIPULATION

# OBJEKTE DES DOM IM JAVASCRIPT

<code>document</code>	–	Zugriff auf HTML-Elemente
<code>window</code>	–	Zugriff auf den Browser
<code>console</code>	–	Ausgaben zum Debugging

# METHODEN UND ATTRIBUTE VON WINDOW

`window`            ist ein globales Objekt  
`document`        ist ein Attribut von `window`

```
window.document.title = 'Hello World';  
===  
document.title = 'Hello World';
```

# METHODEN UND ATTRIBUTE VON WINDOW

`innerwidth/innerheight`

→ Größe des HTML Bereiches

`setTimeout()`

→ Timer, wird für Animationen verwendet

# DOKUMENTENKNOTEN (DOM NODES) SIND OBJEKTE

```
<html>  
  <head>...</head>  
  <body>  
    <div id="box"> ...</div>  
  </body>  
</html>
```

→

```
document.getElementById( 'box' );  
document.querySelector( '#box' );
```

# METHODEN, UM DEN DOM ZU MANIPULIEREN:

```
var node = document.createElement('h1');  
node.setAttribute('class', 'h1-main');  
var child = document.createTextNode('text');  
node.appendChild(child);  
document.body.appendChild(node);
```

# IDENTIFIZIEREN VON KNOTEN

```
var node = document.getElementById( 'content' );
```

```
getElementsByClassName( "green" )  
getElementsByTagName( "p" )  
querySelectorAll( "p.green" )  
querySelector( )
```

# INTERAKTION VERWENDET EVENTS, EVENTS KANN MAN HÖREN

```
document.getElementById('catch01')  
    .addEventListener("keydown",  
        function (event) {  
            log('catch01 '+event.keyCode);  
        }  
    );
```



# ERST ERZEUGEN, DANN EINFÜGEN ...

## - AUCH TEXTKNOTEN SIND KNOTEN

```
var head      = document.createElement('h1');  
var headText = document.createTextNode('Schule ist doof');  
  
head.appendChild(headText);  
  
document.body.appendChild(head);
```