

# EntwicklerCamp 2012

## Hands-On 1:

# JavaScript leicht gemacht

Gelsenkirchen, 26. März 2012

Innovative Software-Lösungen.


[www.assono.de](http://www.assono.de)

## Manuel Nientit

- Diplom-Wirtschaftsinformatik, Universität Paderborn
- seit 2003 entwickle ich mit und für Lotus Notes
- seit 2008 IT-Consultant bei assono
- seit 2009 auch Webentwicklung
- z.T. auch auf BlackBerry

 [mnientit@assono.de](mailto:mnientit@assono.de)

 <http://www.assono.de/blog>

 04307/900-406

## Was ist JavaScript?

- JavaScript hat nichts mit Java zu tun
- Ähnlichkeiten zur Java-Syntax gegeben
- JavaScript ist eine interpretierte Sprache
- Läuft in einer Sandbox
  - Zugriff auf lokale Dateien und/oder OS-Funktionen ist untersagt

**Lässt sich im Browser abstellen!**

## Entwicklung

- Kein spezieller Editor notwendig
- Aber es gibt Hilfen:
  - Komodo Edit
  - TopStyle
  - Aptana Studio
  - Notes kann auch ;)
  - ...
- Debugging schwierig -> z.B. Firebug
- **Achtung! Es gibt Unterschiede zwischen Browsern!**

# Hello World

- JS Anweisungen werden beim Laden ausgeführt
- Syntaxregeln:
  - Strings in Anführungszeichen - auch einfache
  - Jede Anweisung wird (optimalerweise) mit ";" beendet

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Hello World</title>
    <script type="text/javascript">
      alert("Hello World");
    </script>
  </head>
  <body>
    <h1>This should alert "Hello World" when loading.</h1>
  </body>
</html>
```

## Hello World – als Funktion

Definition als Funktion:

```
function hello(){  
    return 'Hello World';  
}
```

- syntaktisch kein Unterschied zwischen Prozeduren und Funktionen, da:
  - Kein Rückgabetyt definiert wird
  - Return optional ist
  - ohne return: Rückgabewert „null“
- Kommentare entweder einzeilig mit `// Kommentar`
- Oder mehrzeilig mit `/*Kommentar*/`

## Variablen

- Werden durch einfache Zuweisung mit "=" ohne Typ angelegt
  - Variable = wert;
- > loose typing
- Empfohlen wird die Nutzung von "var"
  - Entspricht einer Neudefinition
  - Variablen-Scope nur Funktionskontext

**Achtung: JavaScript ist case-sensitive!**

# Typen

„loose typing“ heißt nicht untypisiert

- Es gibt folgende 6 Typen:
  - string
  - number
  - boolean
  - object (auch z.B. arrays)
  - null – Eigenschaften ohne Wert bzw. Funktionen ohne Rückgabewert
  - undefined – nicht definierte oder gelöschte Variable



# typeof

typeof hilft nur bedingt

Typ	Ergebnis von typeof
object	'object'
function	'function'
array	'object'
number	'number'
string	'string'
boolean	'boolean'
null	'object'
undefined	'undefined'

## Boolean

- Folgende Werte sind "falsch"
  - false
  - „null“
  - „undefined“
  - "" (leere Zeichenkette)
  - 0
  - NaN
- Alle anderen Werte sind "true", auch
  - "0"
  - "false"

## Number

- alle Zahlen sind 64-bit Gleitkommazahlen („double“)
  - Rundungsdifferenzen durch Binär-Dezimal-Umwandlung
    - $4,98 + 0,2 = 5,1800000000000001$
  - NaN = Not a Number
    - „ansteckend“
    - ungleich allem anderen (inklusive NaN)
- > Test mit: `isNaN(Variablen-Name)`

# Array

- Nullbasiert
- Ist ein Objekt
- **Test:** `variable instanceof Array`

```
var arrayFix = new Array(100); //festes Array
var arrayFix = [100]; //Kurzform

var arrayDyn = new Array(); //dynamisches Array
var arrayDyn = []; //Kurzform

var arrayBekannt = new Array("One", "Two", "Three"); //wenn Inhalte bereits bekannt sind
var arrayBekannt = ["One", "Two", "Three"]; //Kurzform

alert(arrayBekannt[0]); //messagebox: "One"
arrayBekannt[0] = "Four";
alert(arrayBekannt.join(", ")); //-> messagebox: "Four, Two, Three"
```

## Bedingte Ausführung - if...then...else

```
if(Bedingung){  
    TuWas();  
    UndNochWas();  
  
}else if (Bedingung2)  
    TuWasAnderes();  
  
else  
    DannHaltDas();
```

- Bedingung in Klammern
- Wenn zutreffend wird die nächste Anweisung ausgeführt
- Geschwungene Klammern fassen einen Block zu einer Anweisung zusammen

# Switch

```
switch(Ausdruck){  
    case wert1:  
        TuDas();  
        break;  
    case wert2:  
        TuDies();  
        break;  
    default:  
        OderDas();  
}
```

- Ohne die break-Anweisung wird der nächste case ausgeführt - sonst Beendigung der Switch-Anweisung
- Wenn nichts zutrifft, wird default ausgeführt.

## Vergleichsoperatoren

- == und != vergleichen Werte
- === und !== vergleichen Werte unter Berücksichtigung des Typs
  - 1 == "1" -> Wahr
  - 1 === "1" -> Falsch
- <, <=, >, >=, ! usw. wie in LotusScript

## Logische Operatoren

- "Ausdruck1 && Ausdruck2" gibt Ausdruck1 aus, wenn Ausdruck1 falsch ist, sonst Ausdruck2
- "Ausdruck1 || Ausdruck2" gibt Ausdruck1 aus, wenn wahr, sonst Ausdruck2



## Schleifen - for

```
for([Init];[Bedingung];[Increment]){
```

```
    Anweisungen
```

```
}
```

- [Init] setze Initialwert der Inkrementierungsvariable
- [Bedingung] Austrittsbedingung
- [Increment] Verändere Inkrementierungsvariable
- Mit "break" unterbricht man auch ohne erfüllte Bedingung die Schleife
- Mit "continue" springt man zum nächsten Inkrement

## Schleifen – while/do...while

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

- Anweisung wird so lange ausgeführt bis die Bedingung nicht mehr erfüllt ist

```
do{  
    Anweisung(en)  
}while(Bedingung)
```

- Wie oben, aber die Anweisung wird mindestens einmal ausgeführt

# Objekte

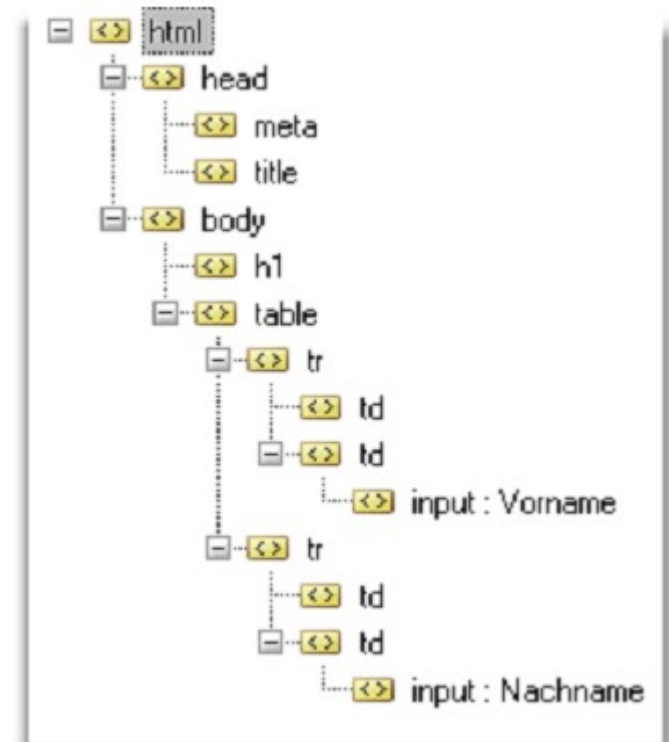
- Objektorientierte Programmierung ohne Basiskonstruktionen wie
  - Abstrakte Klassen
  - Zugriffsmodifikatoren

```
function objectName(initparam){
    this.objectmember = initparam; //die "this." Notation macht aus einer Variable eine Objekteigenschaft
    //Funktionsdeklaration durch Zuweisung zu einer Objekteigenschaft
    this.objectfunction = function(){
        return this.objectmember;
    };
    //Rückgabewert des Konstruktors
    return true;
}
//Objekt kann jetzt mit der "New"-Notation initiiert werden
var myobject = new objectName("myinit");
//Objektfunktionen werden über die Punktnotation aufgerufen
alert(myobject.objectfunction());

//literal object notation
var literalobject = {
    eigenschaftenName: Eigenschaftwert, //Eigenschaften werden mit Kommata getrennt
    objektfunktion: function(){
        return this.eigenschaftenName;
    }
}
```

## DOM – Document Object Model

- Jedes HTML-Dokument lässt sich als Baum darstellen
- Die Wurzel heißt "window"
- Man kann auch mit "document" starten
- Jeder "Kindknoten" heißt "element"
- Elemente haben:
  - Attribute (z.B. width)
  - Namen (nicht eindeutig)
  - Id's (eindeutig)
  - Klassen (für CSS)
  - Events (z.B. onclick)
  - Werte (value)



## DOM – Elemente und Werte

- `document.getElementById`: Element jeglichen Typs mit Hilfe der ID
- `document.getElementsByName`: Element(e) mit Namen
- `document.getElementsByTagName`: Element(e) eines bestimmten Typs
- `document.getElementsByClassName`: Elemente mit einer bestimmten CSS-Klasse
- `element.value` gibt den Wert zurück bzw. setzt ihn
- Elemente können auch direkt mit der ID angesprochen werden, wenn man den "Pfad" kennt
  - `document.Formname.ElementID`
  - Ähnlich wie: „Notesdocument.FeldName“

## DOM - Manipulation

- `document.createElement(name)` erzeugt ein Element
- `element.appendChild(element)` hängt an das aktuelle Element ein Weiteres an
- Mit `element.innerHTML` kann man neue Elemente (als String) einfügen
- `element.setAttribute(attributname, attributwert)` ändert das angegebene Attribut des Elementes
  - `element.getAttribute(attributName)` umgekehrt

## DOM – Manipulation weitere

- AppendChild() → Kindelement hinzufügen
- HasChildNodes() → auf Kindelement prüfen
- InsertBefore() → Element einfügen
- RemoveChild() → Element löschen
- ReplaceChild() → Kindelement ersetzen

## DOM – Style Attribut

- Das Style-Attribut ist eine Besonderheit, da es mehrere Style-Informationen beinhalten kann
  - > `Style="width: 50px; font-size: bold;"`
  - `element.setAttribute("style", "width:60px;")` wird alle anderen Styles überschreiben
  - Mit `element.style.StyleName` erhält man direkten Zugriff auf den jeweiligen Style
  - > `elem.style.backgroundColor = "red";` (oder `#FF0000`)
- oder
- > `elem.style[AttributName] = Wert;`



## Events

HTML-Elemente feuern events. Die Wichtigsten:

- Das Form-Element feuert `onsubmit()` ~ `onQuerySave()` ideal für client-seitige Validierung
- Buttons u.a. feuern `onclick()` -> zum Aufruf von JS-Funktionen
- Input-Felder feuern `onchange()` beim Verlassen des Feldes, wenn geändert -> gut für Input-Translation
- Fast alle Elemente feuern `onmouse` (-click, -over etc.) Events z.B. für Hovereffekte und Kontextabhängige Hilfe

## Übung - Anmeldeformular

- Erstellt ein Formular mit 3 Feldern:
  - Vorname
  - Nachname
  - Alter
- Input-Translation für Alter (Ganze Zahl)
  - Nutze `parseInt(String)`, um die Zahl zu erzeugen
  - `Wert.toFixed(Stellen)`, um die Zahl der Dezimalstellen festzulegen
- Der Vor- und Nachname muss eingetragen werden (`onSubmit()`)

# Debugging

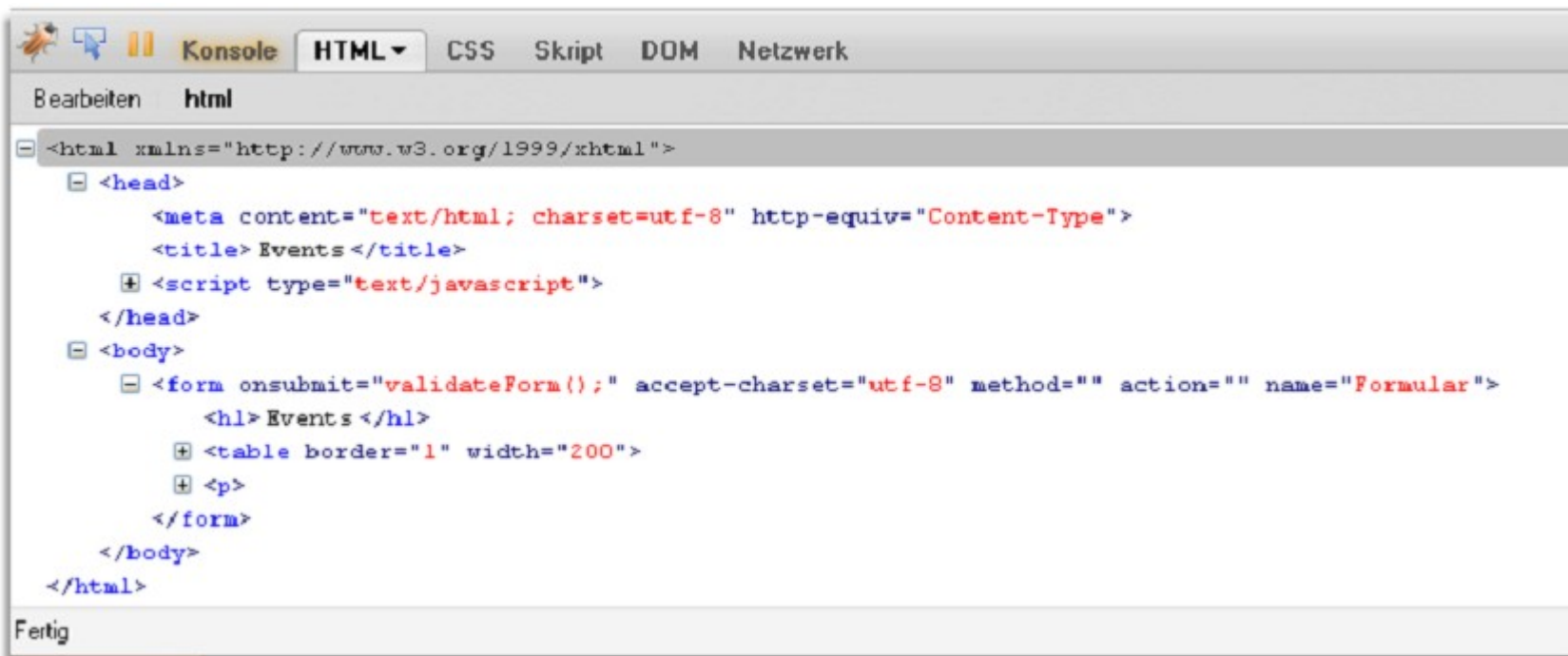
„The world needs more alerts“ 😄

- alert(String) hilfreich, um Status von Variablen auszugeben, muss aber einzeln bestätigt werden
- window.status verändert die Statusbar, wenn erlaubt
- FireBug o.ä.



# Demonstration Firebug

- Herunterzuladen als Firefox-Addon



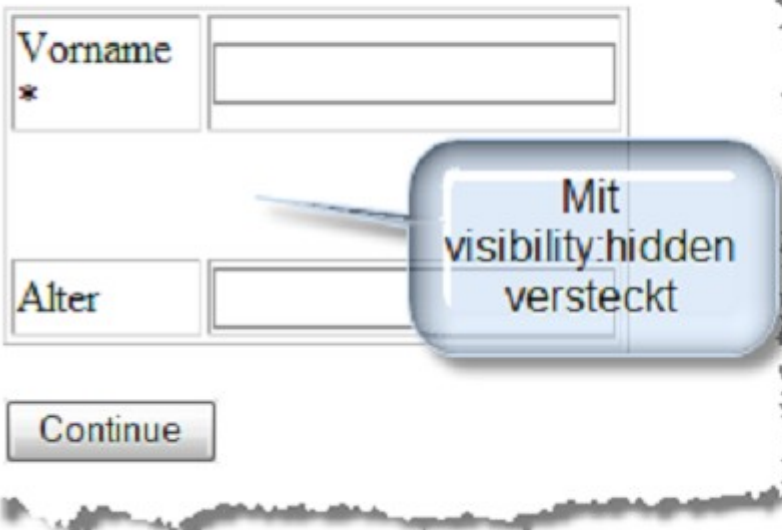
```
Firefox-Toolbar: Konsole HTML CSS Skript DOM Netzwerk
Bearbeiten html
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <title> Events </title>
    <script type="text/javascript">
  </head>
  <body>
    <form onsubmit="validateForm();" accept-charset="utf-8" method="" action="" name="Formular">
      <h1> Events </h1>
      <table border="1" width="200">
        <p>
      </form>
    </body>
  </html>
Fertig
```

## Hide-When – Visibility vs. Display

- Style visibility – verändert nur die Sichtbarkeit
  - hidden
  - visible
  - collapse wie hidden für Reihen und Spalten – nicht in IE
- display:none – verändert die "Flow"-Eigenschaft
  - Zum Anzeigen entweder Attribut löschen oder Anzeigeeigenschaft "inline", "block" o.ä. definieren

## Visibility vs. Display

### Events



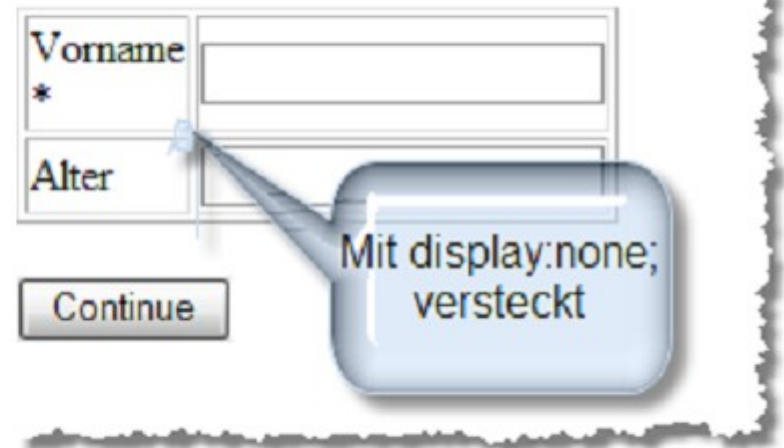
Vorname \*

  
  
Alter

Mit `visibility:hidden`  
versteckt

Continue

### Events



Vorname \*

  
Alter

Mit `display:none;`  
versteckt

Continue

## Übung mit Hide When

- Wir erweitern die Maske von eben um ein Feld Zweiter Vorname, das versteckt ist, bis der Vorname eingegeben wurde
  - Woher wissen wir das?
  - Visibility oder Display?

## Script Bibliotheken

- Wiederverwertbar
- Austauschbar

```
<script type="text/javascript" src="Pfad/name.js"></script>
```

- Wenn die Datei "lokal" liegt, dann ist relative Referenzierung zu empfehlen
- Es können auch Webreferenzen angegeben werden
- In Notes/Domino befindet sich die Bibliothek im "Hauptverzeichnis"



## Fehlerhandling: try..catch..finally

```
try{  
    Anweisungen;  
    throw new UserException();  
}catch(e[if e == "ExceptionType"]){  
    handleFehler();  
}finally{  
    WirdAufJedenFallGetan();  
}
```

- Fehler im try – block werden mit catch gefangen und können da behandelt werden (Log-Eintrag)
- Der Name ist frei-definierbar
- Im finally-block können Bedingungen für die weitere Bearbeitung gesetzt werden

## JS in Notes-Forms

- In Forms (Pages, Views etc.) gibt es JS-events entsprechend den events einer HTML-Seite
- Im JS Header werden JS-Funktionen oder Variablen definiert
- Im HTML Head können JS-Bibliotheken eingebunden werden
- In WebQuerySave kann man einen Agenten aufrufen, der für die serverseitige Validierung zuständig ist

**Achtung: JavaScript kann Browser-seitig abgestellt werden!**

## Exkurs: WebQuerySave

Serverseitige Validierung wird dringend empfohlen:

- WebQuerySave wird vor dem Speichern des Dokumentes gestartet
- Der Agent kann die Speicherung mit SaveOptions="0" verhindern
- Alle print-statements im Agenten werden an den Browser zurück gegeben -> HTML wird interpretiert
- Das Feld "\$\$return" auf der Form wird ausgewertet, wenn WebQuerySave keinen String-Output hat

## Einsatz von Frameworks

- Frameworks bieten häufig Convenience-Funktionen
- Bügeln häufig Unterschiede zwischen Browsern aus (z.B. bei Ajax/XMLHttpRequest)
- Bieten zusätzliche Funktionen für UI-Manipulation wie z.B. D'n'D
- Immer darauf achten, dass das Framework aktuell ist und gepflegt wird
  - Dojo – wird auch in Notes (XPages) verwendet und liegt im Server-Pfad
  - script.aculo.us
  - Prototype
  - JQuery
  - u.a.

## Weitere Session zu JS


- Mi. 11:00 Track 1: Xpages (8) Dojo Toolkits zur Optimierung – Mathias Bierl

## Ressourcen

- <http://de.selfhtml.org/javascript/index.htm> – nicht mehr ganz aktuell, aber gut strukturiert (als Wiki neuer und besser gepflegt)
- <http://www.w3schools.com/default.asp>
- Douglas Crockford:
  - <http://oreilly.com/catalog/9780596517748/>
  - <http://video.yahoo.com/watch/111593/1710507>
- Dave Flanagan: <http://oreilly.com/catalog/9780596805531/>
- Dustin Diaz, Ross Harmes: Pro JavaScript Design Patterns
- <http://caniuse.com/> - Zum Prüfen der Browser - Kompatibilität mit HTML 5, CSS3, SVG u.ä.

## Fragen?

jetzt stellen – oder später:

-  [mnientit@assono.de](mailto:mnientit@assono.de)
-  <http://www.assono.de/blog>
-  04307/900-406



Folien unter

<http://www.assono.de/blog>