Vorlesungsreihe
EwA

Einführung in Javascript

Prof. Dr.-Ing. Thomas Wiedemann
email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Gliederung

- JavaScript
  - Einführung / Historie, Abgrenzung zu Java
  - Grundlegende Technologien & Syntax
  - Beispiele
- Ajax

Quelle(n) :

[1] Selfhtml http://wiki.selfhtml.org/wiki/JavaScript

[2] W3School_JS: http://www.w3schools.com/js/default.asp

und viele weitere Bücher und Webseiten

Übersicht zu Javascript

- JavaScript ist keine direkte Ableitung von Sun´s JAVA sondern eine Eigenentwicklung von Netscape aus dem Jahre 1995
- Hauptziel ist die Client-seitige Manipulation des Dokumentes im Browser
- wie bei HTML bestehen zum Teil erhebliche Unterschiede bei der Unterstützung der JavaScript-Versionen durch die Browser – kurze Versionshistorie:
  - JavaScript wurde eingeführt durch Brendan Eich zur Verwendung im Netscape Browser im Jahr 1995,
  - Sprache wurde standardisiert durch ECMA Standard association in 1997.

    Offizieller Standard-Name ECMA-262

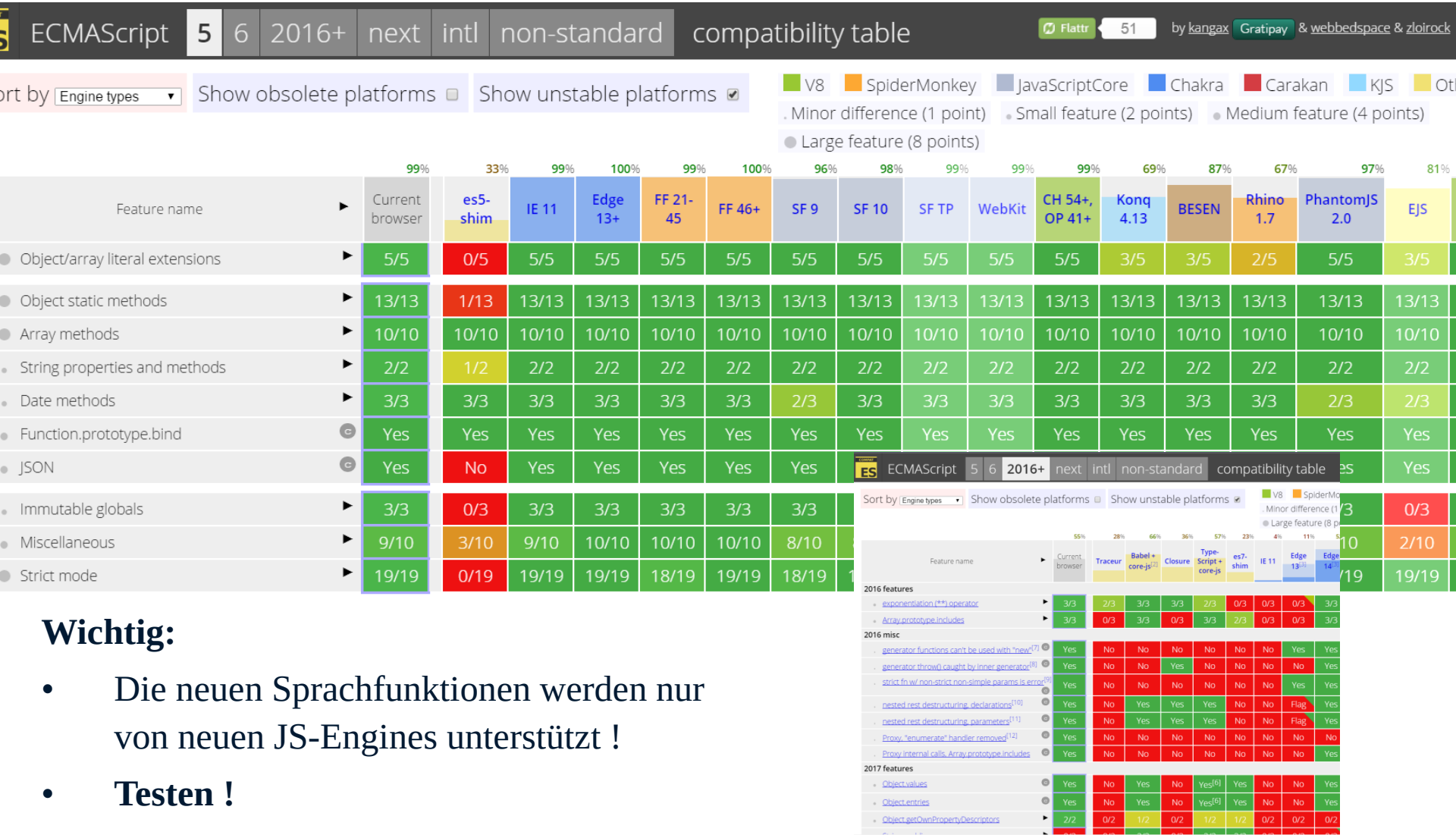
    Versionsüberblick aus: https://en.wikipedia.org/wiki/ECMAScript

    ECMAScript 3 (1999) with regular epressions, more string functions, try/catch,
    - **ECMAScript 5 (2009) new „strict mode“ with improved error checking, JSON-support**
    - **ECMAScript 2015 (before called Version 6 - “Harmony”) new syntax for complex apps , new options like for/of – loops**
    - **ECMAScript 2016 (=V7) June 2016 (danach meist im Juni jeweils eine neue Version)**
    - **ECMAScript 2018 (=V9) June 2018**
    - **Aktuell ECMAScript 2019 (=V10) June 2019**

      http://www.ecma-international.org/ecma-262/10.0/index.html

Unterstützung durch die JavaScript-engines

- Die verschiedenen Versionen werden nur sehr unterschiedlich unterstützt !
- ➔ http://kangax.github.io/compat-table/es2016plus/



Wichtig:

- Die neuen Sprachfunktionen werden nur von neuen JS-Engines unterstützt !
- **Testen !**

Einbindung von Javascript in HTML-Seiten

- Javascript-Code wird entweder zwischen <script..>-Tags (meist im Header oder zu Beginn) eingebunden oder aus einer externen Datei geladen (Erw. *.js) oder in HTML-Tags zum Aufruf komplexerer Funktionen verwendet
- Das Attribut „text/javascript“ ist in HTML5 nicht mehr notwendig !

```
<html><head><title>Test</title>
<script type="text/javascript">
        alert("Hallo Welt!");
</script> </head> <body> </body>

<script src="quadrat.js“></script>
....
<input type="button" value="Quadrat
errechnen" onClick="Quadrat()" >
 </html>
```

Datei: quadrat.js

```
function Quadrat()
{ var Ergebnis = document.Formular.Eingabe.value *
document.Formular.Eingabe.value; alert("Das Quadrat von " +
document.Formular.Eingabe.value + " = " + Ergebnis); }
```

Allgemeine Syntax-Regeln für JavaScript

- generelle Schreibweise an Java angelehnt, wenn auch nicht 100% ig
- Anweisungen werden mit ; abgeschlossen (nicht zwingend notwendig, aber zwecks Kompatibilität zu Java sehr sinnvoll) Quadrat = Zahl * Zahl;
- Achtung: automatic semicolon insertion (ASI) behebt ; - Fehler teilweise !
- Anweisungsblöcke zur Zusammenfassung größerer Anweisungsblöcke mit { } insbesondere bei Schleifen und Bedingungen

  if (Zahl > 1000) { Zahl = 0; Neustart(); }
- Alle Schleifen und Bedingungen werden analog zu JAVA bzw. C unterstützt

  if () { } else { } / while () { }

- **Eigene Bezeichner** analog zu JAVA: keine Leerzeichen / nur Buchstaben und Ziffern (Keine Umlaute!) und _
  - erstes Zeichen muss ein Buchstabe sein
  - Groß- und Kleinschreibung werden unterschieden!
- Kommentare mit /* */ oder //

JavaScript Daten Typen

Definition

- numerische und nicht-numerische Typen
- Der Identifier var kann (muss aber nicht) verwendet werden var i = 0 ;
  (siehe auch Scope-Anmerkungen)

Javascript erlaubt die meisten bekannten Datentypen

```
var length = 16;       // Integer Number  - aber intern als 64bit float
var length = 16.5;     // Float Number
```

➔ Generell werden alle Zahlen intern als 64bit-Float repräsentiert !!!

```
var points = x * 10;      // Number assigned by an expression literal
var lastName = "Johnson";    // String assigned by a string literal
var x = true; var y = false;  // boolean
```

JS-Variable sind dynamisch definiert – der Typ kann zur Laufzeit modifiziert werden :

```
var x;                  //  x is undefined – ergibt bei Ausgabe „undefined“ !
var x = 5;              // Now x is a Number
var x = "John";    // Now x is a String
```

Der Scope von JavaScript Variablen / Konstanten

Definitionen AUSSERHALB von Funktionen sind GLOBAL !

```
var   j ,  k = 1;  // global
```

Definitionen innerhalb MIT VAR in Funktionen sind LOKAL :

```
function myFunction(p1, p2) {
        var  i = 2;  //   this i is visible only inside this function
        return p1 * p2´*  i ;     /
};
```

Achtung: Def. OHNE var sind GLOBAL !!!! (kann ggf. kritisch sein !)

```
function myFunction(p1, p2) {
        i = 2;  //   this i is visible GLOBAL,also outside the function
        return p1 * p2´*  i ;     /
};
```

Definitionen von Konstanten mit const

```
const   plz   = '01069';
```

Der „strict“ - mode

Automatische Definition bei erste Verwendung ➔ global

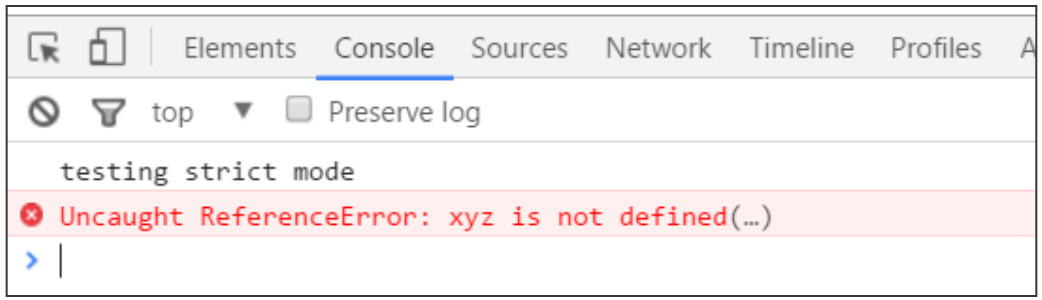
```
xyz   = 1;  // automatic definition without  var
```

Unbeabsichtigte Schreibfehler definieren damit eine neue Variable (sehr schlecht zu finden und zu debuggen ! ➔ Folgefehler !

```
xzy =  3;   // new variable  (which was not intended …)
```

Neuer “strict mode” ab ECMAScript 5 generiert error-message

```
"use strict";     // new strict directive
var   xyz = 1;   //  ok !
      xzy = 3;  //  error message  in console window
```



JavaScript-Arrays

Definition und Initialisierung

3 Optionen der Anlage

- var arr = Array(1,2,3); // Anlage von 3 Elementen und Initialisierung
- var arr = [1,2,3];   // analoge Kurzform (meist bevorzugt)
- var arr = new Array(1,2,3); // als neues Objekt
- Achtung bei Einzelwert: Array(10); // interpretiert 10 als Elementanzahl !!!
- Mischbelegungen möglich: Array(1, 2, „“Wert3“ , true , 5)
- **Hinweis: intern wird ein Array als ein spezielles Objekt (mit entspr. Methoden) und einer numerischen Indexierung (vgl. auch JS-Objekte)**

Zugriff und Interation über Arrays

```
arr[0] =  11; //  erstes Element wird mit  0 referenziert
for (var i = 0; i < arr.length; i++)      // .length – liefert Array-Länge als N+1 !
        {   console.log( arr[i]); }
arr.forEach(function(color) {    // 2. Option mit foreach und Funktion
            console.log(color); }); // ignoriert leere Elemente
arr.length = n ;  // löscht Array bei   n= 0 oder setzt neue Länge !
```

JavaScript-Array-Funktionen

Array-Management (immer auf Basis von arr = [1,2,3])

- **Zusammenfügen**: arr = arr.concat("4", "5"); // arr = [1,2,3, "4","5"]
- **Nach Text wandeln**: var text = arr.join ("+"); // text = "1+2+3"
- **Letztes Element holen:** var last = arr.pop(); // arr = [1,2], last = 3
- **Element am Ende hinzufügen:** arr.push(4); // arr = [1,2,3,**4**]
- **Erstes Element holen:** var first = arr.shift(); // arr = [2,3] first = 1
- **Element am Ende hinzufügen:** arr.unshift(0); // arr = [**0**,1,2,3]
- Vollständige Liste der Meth.: http://www.w3schools.com/jsref/jsref_obj_array.asp

Sortieren

- **Reversieren:** arr. reverse(); // arr [3,2,1]
- **Sortieren**: arr.sort(); // Achtung – sortiert nur alphabetisch (nach String)
- var arr2 = [3,30 ,1,10,]; arr2.sort(); // -> [1,10,3,30]

bei numerischen Elementen muss entspr. Vergleichsfunktion definiert werden

- arr2.sort(function(a, b) { return a - b}); // => [1,3, 10, 30]

JavaScript Funktionen

Funktionen werden mit dem Keyword „function“ definiert :

```
function Name(parameter1, parameter2, parameter3)
{   code of the function ….
    return   ret_value;  }
```

- Die Parameter werden typenlos definiert !
- Optional kann mit return … ein Rückgabewert zurück gegeben werden.

Bsp. einer Funktionsdefinition ::

```
function myFunction(p1, p2)
{  return p1 * p2;  } // the function returns the product of p1 and p2
```

Aufruf der Funktion

- Direkt aus dem Code `var myresult = myFunction(10, 20) ;`
- Durch Einbau in Event-Handler (siehe Events) -> “…“ beachten !!

```
<input type="button"  onclick=" myFunction(10, 20) ">
```

Achtung: Es erfolgt keine automatische Parameterlistenprüfung – ggf. selbst erledigen :

```
if (b  === undefined) {   b = 1  ;  }
```

- über die Methode .arguments kann ein Abruf ALLER AKT. Parameter erfolgen !

JavaScript Funktions – Expressions

Funktionen können auch einer Variable zugeordnet und aufgerufen werden:

```
var  myfunc = function myfunction  (p1, p2)   {   return p1 * p2;    }
```

Aufruf der Funktion

```
var x =  myfunc ( 3,4 ) ;
```

- da der Aufruf über die Variablenzuweisung läuft, kann der Funktionsname auch weggelassen werden -> sog. **anonyme Funktionen**

```
var  myfunc = function          (p1, p2)   {   return p1 * p2;    }
```

Spezielle JavaScript – Eigenschaften

1. Hoisting : Deklarationen (auch var …) werden AN DEN ANFANG gesetzt :

```
function myfunc(a,b) {  return a * b; }
myfunc(4, 5);
function myfunc(a,b) {  return a * b; }
```

Automatischer „Selbst-Aufruf“ der Funktion (self-invoking)

```
( function myfunction  (p1, p2)
    {   return p1 * p2;    }  )  ( )
```

JavaScript-Objects

Allgemeines

- **In JavaScript sind fast alle Sprachelemente OBJEKTE** (mit Ausnahme der primitiven Datentypen, aber auch diese können als Objekte gewrappt werden)
- **JavaScript stellt aber kein herkömmlicher OO-Klassensystem zur Verfügung, sondern orientiert auf die direkte Definition und Verwendung von Objekten !**
- **(vgl. Konzept: „**Object composition over class inheritance")
- **JS-Objekte sind ähnlich zu assoziativen PHP-Listen oder C-Hashtables**

Definition und Initialisierung

3 Optionen der Anlage

- `var x1 = { firstName:"Tom", age : 25 };` // Object definition with Literal
- `var x2 = new Object();` // Instanzierung eines leeren Objektes mit new und

`x2.firstname = "Tom"; x2.age = 25;` // Deklaration und Belegung der Wertepaare - Definition einer Konstruktorfunktion und mehrmaliger Aufruf

```
function Person ( name , age )
        {  this.firstname = name; this.age = age }
var p1 = Person ("tom", 25); var p2 = Person ("Eva" , 21);
```

JavaScript-Objects – Zugriff und Referenzierung

Zugriffsoptionen

- **Dot-Syntax**: x1.firstName
- **Array-Syntax:** x1[“age“] ; oder auch mit name=„age“; x2[name];
- **Zugriff auf alle Eigenschaften eines Objektes**

```
for ( xvalue  in  x1 ) {
        text +=   x1 [xvalue];
    }
```

Manipulation

- JS-Objekte können auch nach der Deklaration geändert werden
- **Neue Eigenschaft (Attribut):** x1.newvalue = “123“;
- **Eigenschaft löschen:** delete x1.newvalue;

JavaScript-Objects – Methoden

Definition

- **Als Objekteigenschaft über die verschiedenen Objektdeklarationen**
- **Syntax:** methodenname : function (parameter) { code }
- **Bsp.:** changeName = function (name) { this.firstname = name; }
- **oder in Objektkonstrukturfunktion:**
  … , this.changeName = function (name) { this.firstname = name; }

Aufruf:

- objekt.funktionsname (parameter) ;
- Bsp: p1.changename(“Lisa“);
- **Achtung: Immer Methodenaufruf mit ()**
  – sonst wird Funktionsdefinition ausgegeben !!

JavaScript-Objects – Prototypen

Allgemein: **Objektprototypen sind ein gewisser Ersatz für die fehlende Vererbungshierarchie in JavaScript**

Jede Objektinstanz hat einen Prototypen :

- **Entweder die vorab definierte Konstruktorfunktion**
- **oder das Object – Objekt von JS selbst**

- **Über die Eigenschaft prototype können auch nachträglich neue Eigenschaften oder Methoden zu einem prototype hinzugefügt werden:**

```
Person.prototype.adress   =  "Dresden";
Person.prototype.changeAdress = function (  newAdress )
                        {     adresse = NewAdress  }
```

JSON (JavaScript Object Notation)

- definiert ein Format zum Datenaustausch
- entspricht der JS-Objektsyntax , wobei die Objekteigenschaften aber **definitiv in Hochkommata** geschrieben werden müssen !

```
{   “firstName “  :   “Tom", “age“ : “25“  };
```

- JSON-Array

```
var text =   '{ "employees":[   {"firstName":"John", "lastName": "Muller" },
                                {"firstName":"Anna", "lastName":"Smith"} ]  }';
```

- JSON-Notationen können sehr leicht in JS-Objekte gewandelt werden

```
var obj = JSON.parse(text);
```

- und Umwandlung eines Objektes in JSON

```
var text2 = JSON.stringify(obj);
```

JavaScript-Sets und Maps

Definition und Initialisierung

3 Optionen der Anlage

- var arr = Array(1,2,3); // Anlage von 3 Elementen und Initialisierung
- var arr = [1,2,3];   // analoge Kurzform (meist bevorzugt)
- var arr = new Array(1,2,3); // als neues Objekt
- Achtung bei Einzelwert: Array(10); // interpretiert 10 als Elementanzahl !!!
- Mischbelegungen möglich: Array(1, 2, „“Wert3“ , true , 5)
- Hinweis: intern werden Array als Maps (siehe entspr. Folie gespeichert)
- var lastName = "Johnson";   // String assigned by a string literal
  var cars = ["Saab", "Volvo", "BMW"];   // Array - Definition
  var x = {firstName:"John", lastName:"Doe"};   // Object definition

JS-Variable sind dynamisch definiert – der Typ kann zur Laufzeit modifiziert werden :

```
var x;                 //  x is undefined
var x = 5;            // Now x is a Number
var x = "John";    // Now x is a String
```

Hilfreiche JavaScript-Befehle

- Normale Messageboxen werden angezeigt mit

```
<script type="text/javascript"> alert("Hallo Welt!"  + zahl ); </script>
```

- Zur Intepretation von Rechenanweisungen zur Laufzeit kann mit eval ein mathematischer Ausdruck ausgewertet werden (aber möglichst nicht auf Eingabewerten des Nutzers (Sicherheitsprobleme) :

```
function Rechne (Operation)
      { var Ergebnis = eval(Operation);
        alert("Ergebnis: " + Ergebnis); }
```

- Fehler können mit onerror abgefangen werden :

```
<script type="text/javascript">
    window.onerror = Fehlerbehandlung;
function Fehlerbehandlung (Nachricht, Datei, Zeile) {
  Fehler = "Fehlermeldung:\n" + Nachricht + "\n" + Datei + "\n" + Zeile;
  alert(Fehler);   return true; }
nichtda();  //  Aufruf einer nicht existenten  Funktion
</script>
```

Anbindung von Javascript an Browser-Events

- Das W3C-Konsortium hat in HTML entsprechende Event-Handler definiert, welche bei Auslösen der entsprechenden Aktion das Script aufrufen :

  onAbort (bei Abbruch)
  onBlur (beim Verlassen)
  onChange (bei erfolgter Änderung)
  onClick (beim Anklicken), onDblClick (bei doppeltem Anklicken)
  onError (im Fehlerfall)
  onFocus (beim Aktivieren)
  onKeydown (bei gedrückter Taste), onKeypress (bei gedrückt gehaltener Taste)
  onKeyup (bei losgelassener Taste)
  onLoad (beim Laden einer Datei)
  onMousedown (bei gedrückter Maustaste), onMousemove (bei bewegter Maus)
  onMouseout (beim Verlassen des Elements mit der Maus)
  onMouseover (beim Überfahren des Elements mit der Maus)
  onMouseUp (bei losgelassener Maustaste)
  onReset (beim Zurücksetzen des Formulars)
  onSelect (beim Selektieren von Text)
  onSubmit (beim Absenden des Formulars)
  onUnload (beim Verlassen der Datei)
  javascript: (bei Verweisen)

Javascript - Objektreferenz

- Die Eventhandler stehen in Bezug zu allen sichtbaren und unsichtbaren Bestandteilen des Browsers. Diese Bestandteile werden durch die JavaScript-Objekthierarchie definiert:

window (Anzeigefenster), frames (Frame-Fenster)
document (Dokument im Anzeigefenster)
HTML-Elementobjekte (Alle HTML-Elemente des Dokuments)
node (Alle Knoten des Elementenbaums)
all (Alle HTML-Elemente des Dokuments - Microsoft)
style (CSS-Attribute von HTML-Elementen)
anchors (Verweisanker im Dokument)
applets (Java-Applets im Dokument)
forms (Formulare im Dokument)
elements (Formularelemente eines Formulars)
options (Optionen einer Auswahlliste eines Formulars)
images (Grafikreferenzen im Dokument)
layers (Layer im Dokument - Netscape)
links (Verweise im Dokument)
event (Anwenderereignisse), history (besuchte Seiten)
location (URIs), Array (Ketten von Variablen)
Boolean (Ja/Nein-Variablen), Date (Datum und Uhrzeit)
Function (JavaScript-Funktionen)

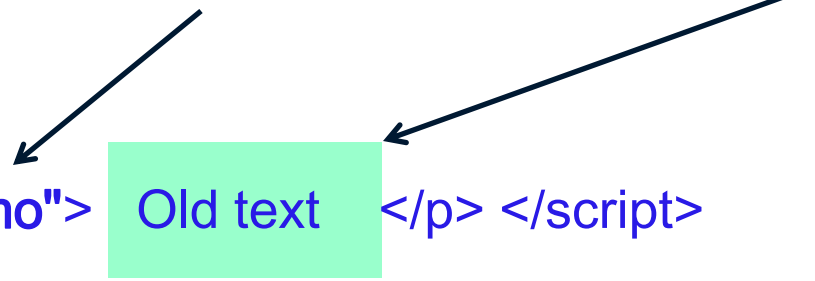
Hierarchie

Referenzierung von HTML-Elementen mit JavaScript

- Die Methode **getElementById** sucht die erste Referenz mit einer gegebenen ID and führt die danach angegebene Aktion aus
- Häufig verwendet zum Auslesen oder Setzen von HTML-Elementen innerhalb des DOM (Document Object model (DOM) – siehe Folgeseiten)
- Beispiel :

```
<button onclick="changetext()">Change text</button>
<script>
function changetext() {
        alert(  document.getElementById("demo").innerHTML ) ;
        document.getElementById("demo").innerHTML = "NewText";  }
</script>

<p id="demo"> Old text </p> </script>
```



- innerHTML() referenziert auf den Haupt-HTML-Inhalt

Referenzierung von HTML-Elementen mit JavaScript II

Andere Methoden zum Referenzieren

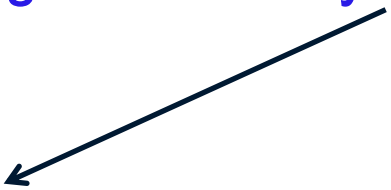
- Methode **getElementsByTagName** sucht alle Elemente mit Tag-Namen
- Sinnvoll für Massenaktionen (meist kein eindeutige Referenz)

- Analog mit **getElementsByClassName** suchen ach Class-Names

- Ebenfalls häufig verwendet zum Auslesen oder Setzen von HTML-Elementen innerhalb des DOM

- Beispiel :

```
function changetext() {
        document.getElementsByClassName("d2").innerHTML = "NewText";
      }
</script>
  <p class="d2">  Old text    </p> </script>
```



Beispiele zur Arbeit mit Ereignissen und Objekten

- **Typische Aufgabe – Prüfung von Formulareingaben**

```
<script type="text/javascript">  function checkeFormular()
{ if (document.Formular.User.value == "")
{ alert("Bitte Namen eingeben!"); document.Formular.User.focus(); return false; }
if (document.Formular.Mail.value.indexOf("@") == -1)
{ alert("Keine E-Mail-Adresse!"); return false; }  var chkZ = 1;
for (i = 0; i < document.Formular.Alter.value.length; ++i)
 if (document.Formular.Alter.value.charAt(i) < "0" ||
       document.Formular.Alter.value.charAt(i) > "9") chkZ = -1;
  if (chkZ == -1) { alert("Altersangabe keine Zahl!");
       document.Formular.Alter.focus(); return false; } } </script> </head>

<body> <form name="Formular“ action=http://.../f.php
       method="post" onsubmit="return checkeFormular()">
Name: <input type="text" size="40" name="User">
E-Mail: <input type="text" size="40" name="Mail">
 Alter: <input type="text" size="40" name="Alter">
 Formular: <input type="submit" value="Absenden">
<input type="reset" value="Abbrechen"> </pre> </form>
```

Testen und Debuggen von JavaScript

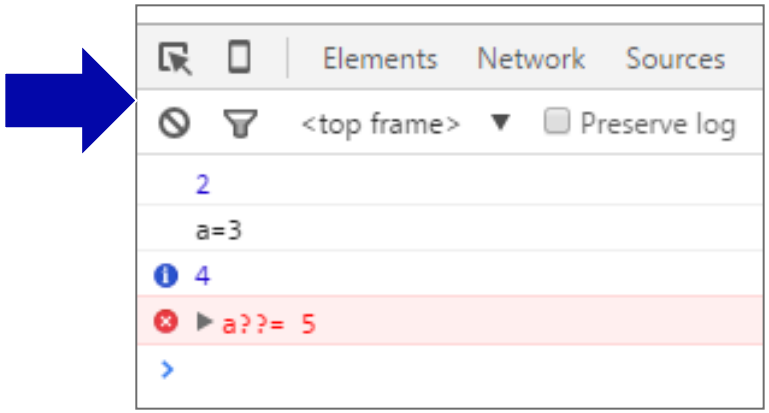
Details unter http://wiki.selfhtml.org/wiki/JavaScript/Tutorials/

- **Testausgaben mit** alert(“Info …“);
- **Abfangen von Fehlern mit** try .. catch

```
try {     adddlert("Welcome guest!");     // fehlerhafter Befehl !
                                    } catch(err) {
                                document.getElementById("demo").innerHTML = "Error: " + err.message;     }
```

- **Konsolenausgaben** (innerhalb Browserdebugger – browserabhängig)
  - Aufruf meist mit Ctrl + Shift + I)
  - Mit console.log(a); // auch mit texten und % -Platzhaltern wie in C !
  - Mit consol.info(…) und console.error(…) mit spezieller Hervorhebung

```
a = 2; console.log(a);
a = 3; console.log("a=%d",  a );
a = 4; console.info(a);
a = 5; console.error("a??= %f" , a);
```



- **je nach Browser auch mit**
  - **Setzen von Breakpoints**
  - **Function-Call- Stackanzeige**

Beispiele und Details

- Alle weiteren Details und Beispiele zu JavaScript
  - Grundlagen :

    http://wiki.selfhtml.org/wiki/JavaScript
  - Größere Anwendungsbeispiele :

    http://wiki.selfhtml.org/wiki/JavaScript/Anwendung_und_Praxis

Weitere Unterlagen unter

http://www.w3schools.com/js/

Ajax (Asynchronous JavaScripting and XML)

- kombiniert JavaScript, HTML, DHTML, DOM und XML
- erzeugt stark interaktive Seiten und ermöglicht damit flüssigeres Arbeiten
- keine manuelle Interaktion des Users mit dem Server, sondern eigenständige Kommunikation von Javascript mit dem Server
- Ajax-Kommunikation durch das XMLHttpRequest-Objekt mit einem meist asynchronen XML-Datenaustausch
- **XMLHttpRequest-Objekt** ist verfügbar ab Microsoft Internet Explorer 5.0, Mozilla Firefox 1.0, Netscape 7.1, Apple Safari 1.2, Opera Mobile Browser 8.0
- Achtung : leider noch unterschiedliche Objektreferenzen auf das XMLHttpRequest-Objekt !!

Methoden von XMLHttpRequest

Die wichtigsten Methoden von XMLHttpRequest **:**

- Öffnen einer Verbindung zum Server

  open(httpReqMeth, url, async)
  open(httpReqMeth, url, async, usr, pwd)

  httpReqMeth - definiert http-Methode ('GET', 'POST', 'PUT')
  url = URL des Dienstes

  async = true für asynchrone Kommunikation (Client wartet nicht auf Antwort, sondern es wird Callback-Funktion später aktiviert)

- Absenden eines Request

  send(postReq) ; - postReq = null für 'GET'-Anfragen oder Key-Value-Paare für 'POST'-Anfragen (z.B. “Key1=Value1&Key2=2")

- **abort()** - Bricht eine aktuell laufende Anfrage ab
- **setRequestHeader(key, value)** - fügt dem HTTP-Header Werte zu

Die wichtigsten Attribute von XMLHttpRequest

- **onreadystatechange** - verweist auf die Event-Handler-Callback-Methode, welche bei Zustandsänderungen des XMLHttpRequest-Objekts aufgerufen wird (siehe ,
- **readyState** – aktueller Status des Request mit folgenden Werten :

  0 = uninitialized 1 = loading 2 = loaded

  3 = interactive **4 = complete**
- **status** - das Ergebnis des http-Request (= http-Status)
  - z.B.: **200 = OK** 404 = Not Found
- **statusText** - der HTTP-Status als Textmitteilung
- **responseText** - die Serverantwort als einfacher Text
- **responseXML** - die Serverantwort im XML-Format

Die Grundstruktur eines Ajax-Request

```
<script language="JavaScript" type="text/javascript">
var url = "http://localhost/checkiptxt.jsp"; var req;
function starteAjax()
{ try { if( window.XMLHttpRequest )
        { req = new XMLHttpRequest(); }   // Version für Firefox & Co.
        else if( window.ActiveXObject )
                { req = new ActiveXObject( "Microsoft.XMLHTTP" ); }   // IE
        else { alert( "Ihr Webbrowser unterstuetzt leider kein Ajax!" ); }
         req.open( "GET", url, true );
         req.onreadystatechange = meineCallbackFkt;
         req.send( null );
        }   catch( e ) { alert( "Fehler: " + e ); } }
    function meineCallbackFkt()
        { if( 4 == req.readyState ) { if( 200 != req.status )
                { alert( "Fehler " + req.status + ": " + req.statusText ); }
        else { alert(  req.responseText ); } } }
</script>
```

Ajax-Request mit XML-Antwort

- Request analog zu vorheriger Seite und Anforderung XML-Dok.

```
if( req.overrideMimeType ) { req.overrideMimeType( 'text/xml' );
```

- Neue Auswertung in der Callback-Funktion :

```
function meineCallbackFkt()
{  if( 4 == req.readyState ) { if( 200 != req.status )
      { alert( "Fehler " + req.status + ": " + req.statusText ); }
  else { ergebnis = req.responseXML.documentElement;
 // Hole  Werte aus dem XML-Response
 var zahlAusgabe = ergebnis.getElementsByTagName('zahl')[0].firstChild.data;
 var textAusgabe = ergebnis.getElementsByTagName('text')[0].firstChild.data;
 var ipAusgabe = ergebnis.getElementsByTagName('ip')[0].firstChild.data;
   // Schreibe Werte  in die HTML-Seite
      document.getElementById("zahlAusgabe").value = zahlAusgabe;
     document.getElementById("textAusgabe").value = textAusgabe;
     document.getElementById("ipAusgabe").innerHTML = ipAusgabe;
} } }
```

Demo Ajax-Request für Auto-Vervollständigen

1. **HTMLSeite mit Formular :**

```
<form name="formular" action="...">
    <input type="text" id="eingabefeld" onKeyUp="meinAjaxAufruf( this.value )"/> <br>
    <div id="auswahlbox"></div> </form>
```

2. **Ajax - Request mit aktuellem Inhalt des eingabe-Feldes**

```
document.formular.eingabefeld.focus();
var url = "autovervollstaendigung.jsp?eingabe=" + escape( eingabetext ); req.open(
"GET", url, true );
```

3. **Ajax – Response in Formular einbauen**

```
var text = req.responseText;
if( text != "" )
{ auswahlarray = text.split( ";" );
    for( var idx in auswahlarray )
        { auswahlinhalt += "<a href='javascript:meinMausklick(" + idx + ")' id='" + idx;
        auswahlinhalt += "' class='aw' onmouseover=sel("+idx+")'>";
         auswahlinhalt += auswahlarray[idx] + "</a>"; }
document.getElementById( "auswahlbox" ).innerHTML = auswahlinhalt; }
```

Generelle Ajax- Kritik

Vorteile

- im Gegensatz zu Flash oder ähnlichen Technologien wird kein Browser-Plugin benötigt, auch unabhängig von Betriebssystem
- Schnelleres, flüssigeres Arbeiten (kein unnötiges Laden von statischen Daten bei erneuten Request -> Einsparung von Bandbreite)

Nachteile :

- JavaScript-Unterstützung muss aktiviert sein !
- Noch Probleme mit unterschiedlichen Browserversionen (sollte sich durch W3C-Standardisierung legen)

Weitere Ajax- Kritik (ggf. technisch lösbar)

Generell umfangreiche Tests erforderlich

- ggf. Erleichterung durch entsprechende Frameworks (Fehlerhandlg.)
- siehe Dojo-Toolkit - http://dojotoolkit.org oder auch JQuery (siehe VL)

Probleme mit Browserversionen

- Durch Fallunterscheidungen auf Clientseite lösbar (ineffizient)
- -> Server-seitige Browsererkennung und spezifische Javascript-Codes

Verwendung der „Zurück“-Schaltfläche

- Funktionalität der „Zurück“- **Schaltfläche kann nicht** gewährleistet werden, da diese nicht über Ajax-Aktionen informiert wird
- Lösung durch Füllung von zusätzlichen Inline-Frame-Elementen oder speziellen Rückruf-Funktionen (bei Frameworks)

Bandbreitenprobleme bei ständigem Polling

- Da nur Kommunikation von Client-> Server->Server muss Client bei Verdacht auf Serveraktual. ggf.- pollen -> Netzlast !!!!
- Lösungen: Serverresponse künstlich verzögern bis zum Eintreten des Ereignisses oder eines Timeout´s

Weitere Ajax- Kritik (ggf. technisch lösbar)

Analog zu den vorherigen Problemen :

- **Lesezeichenspeicherung ?**
- **Barrierefreiheit ?**
- **Suchmaschinen erkennen die dyn. Ajax-Inhalte nicht 100%ig (Google analysiert seit Mai 2016 zumindest offiziell AJAX/JS-Requests)**

  Lösungen :

  - zusätzliche Metatags oder Überschriften-Elemente für die Indizierung
  - Zusätzliche (unsichtbare) Links werden auf der Webseite für die Suchroboter
  - einer Suchmaschine gedacht sind.
  - zweite Webseite mit statischen Links wird entworfen. Diese ist für eine Suchmaschine voll zugänglich (Achtung: als Cloaking einstufbar)
  - Neu: Ajax-Interpreter in den Suchmaschinen selbst (???)
- Gesamtfazit:
  - Ajax ist eine in Kombination mit aufsetzenden Frameworks eine sehr nützlich Technolgie zum Aufbau desktopähnlicher Web-Apps

Ajax- Demos

- Im Netz verfügbare Demos und Details

http://www.ajax-net.de/index.php?option=com_wrapper&Itemid=62

http://de.wikipedia.org/wiki/Ajax_%28Programmierung%29