Bildungszentrum Uster
Höhere Berufsbildung Uster
HBU

JavaScript
Raphael Ritter

Ein Angebot der Berufsfachschule Uster und der Höheren Fachschule Uster

Agenda

- Unerwartete JavaScript Features
  - Truthy and falsy
  - Semicolon Insertion
  - NULL und UNDEFINED

- JavaScript Features
  - Scope & Context
  - Hoisting
  - Alles eine HashTable

- Objekte
- EventHandler

Bildungszentrum Uster
Höhere Berufsbildung
Uster
HBU

Rückblick

Unterlagen der HSR

Einige Inhalte sind aus den Folien / Übungen des CAS Frontend Engineering der Hochschule für Technik Rapperswil.

www.hsr.ch
http://hsr.ch/CAS-Front-End-Engineering.12432.0.html

Unerwartete JavaScript Features

Unerwartete Features

- Es gibt einige «Features» in JavaScript welche zu sehr unerwarteten Problemen führen können
  - Typenumwandlung (type conversion)
  - Numbers
  - Booleans
  - Arrays

- Details: http://people.mozilla.org/~jorendorff/es6-draft.html#sec-rules-of-automatic-semicolon-insertion

- *A semicolon is inserted before, when a Line terminator or "}" is encountered that is not allowed by the grammar.*
  - Ein Semikolon wird eingeführt falls die zusammengefügte Zeile zu einem Fehler führen würde.

```
var x = 10
var y = 20;
console.log(x + "," + y);
var a = 2 * 4
(a).toString();
```

```
var x = 10 var y = 20;   // Fehler
console.log(x + "," + y);

var a = 2 * 4(a).toString();
```

Semicolon insertion

- A semicolon is inserted at the end, when the end of the input stream of tokens is detected and the the parser is unable to parse the single input stream as a complete program.
  - Semicolon am Ende eines Programmes
  - A semicolon is inserted at the end, when a statement with restricted productions in the grammar is followed by a line terminator.
- Statement wird mit dem nächsten “line terminator” beendet.
  - PostfixExpressions (++ and --); continue; break; return;

Semicolon insertion

- Was macht diese Funktion?

```
function createUser(name)
{
  return
  {
     name : name
  }
}
```

Semicolon insertion

- Hier werden die Probleme dieses Features offensichtlich, da der Code so verändert wird, dass ein unerwartetes Verhalten eintritt

```
function createUser(name)
{
  return;
  {
     name : name
  }
}
```

```
function createUser(name)
{
  return   {
     name : name
  } ;
}
```

Semicolon insertion

- Falls Semikolons weggelassen werden, fügt der JavaScript-Interpreter diese ein.
- Automatisch gesetzte Semikolons können zu Bugs führen.
- Programmierer sollten sich **NIE** auf automatisch gesetzte Semikolons verlassen.
- Aber: Für bessere Übersicht sinnvoll:

```
var result = [4,2,1,5].map(function(x){ return x - 3}).filter(function(x){return x > 0}).sort()
console.log(result);

var result = [4,2,1,5]
   .map(function(x){ return x – 3 })
   .filter(function(x){ return x > 0 })
   .sort()
console.log(result);
```

Null und undefined

- undefined:
  - Variable ist nicht definiert. Z.B. var a wurde vergessen;
  - Variable ist nicht initialisiert. Z.B. var a = 1234
- Null:
  - Ist ein Wert von einer Variable.
- => *undefined ist ein Zustand und null ist ein Wert*

- **Achtung**: null == undefined resultiert in true

- *Hinweis: In alten Browsern ist undefined überschreibar.*

Null und undefined

- Aufgabe:
  Wie würden Sie überprüfen ob eine Variable undefined ist?
  Z.B. myVariable

- Wie würden Sie überprüfen ob ein Wert auf einem Objekt undefined ist?
- z.B. myVariable.a

- Zeit: 10 Minuten

Bildungszentrum Uster
Höhere Berufsbildung
Uster
HBU

JavaScript Features

Truthy and falsy

- In JavaScript werden folgende Inhalte einer Variable **IMMER** als false interpretiert
  - undefined, null
  - Boolean: false
  - Number: 0, NaN
  - String: ''
- Alle anderen Werte werden automatisch als true interpretiert
- Boolean(undefined) => false
- Boolean(0) => false
- Boolean(3) => true
- Boolean({}) // empty object => true
- Boolean([]) // empty array => true

Übungen

- WTF.js
  - Schaut euch das JavaScript File an und was die entsprechenden Outputs ergeben

Scope und Context

- Scope und Context ist nicht das gleiche
- Auch bei Programmierern, die schon länger mit JavaScript arbeiten nicht richtig bekannt.
- Jeder Funktionsaufruf hat einen Scope und einen Context
- Scope:
  - Sicherbarkeit der Definitionen (Funktionen / Variablen)
  - Wird bei jedem Aufruf neu erzeugt
- Context:
  - Owner vom aktuellen auszuführendem Code.
  - Context wird durch «this» representiert.

Scope und Context

- Hier sehen wir den Scope von verschiedenen Funktionen und Variablen (am ersten Vorlesungstag bereits angeschnitten)

```
var funcA = function(){
   var a = 1;
   var funcB = function(){
      var b = 2;
      console.log(a, b); // outputs: 1, 2
   }
   funcB();
   console.log(a, b); // Error! b is not defined
}
funcA();
funcB(); // Error! funcB is not defined
```

Scope und Context

- «this» ist der aktuelle Context. «this» referenziert je nach Aufrufart auf ein anders Objekt:
- Falls ein eine Funktion als Methode von einem Objekt aufgerufen wird. Ist this = objekt
  Beispiel: `object.foo();`
- Falls eine Funktion mit new() aufgerufen wird. Wird «this» mit einem neu erstellten Objekt abgefüllt.
  Beispiel. `new foo();`
- Falls eine «unbound» Funktion aufgerufen wird. Wird «this» mit dem globalen Objekt abgefüllt.

Scope und Context

- Jede Funktion kann mit apply() oder call() den Context gesetzt werden. In diesem fall werden die oben genannten Regeln ignoriert.

```
foo.call( { counter : 123} );
```

- Jeder Funktion kann mit bind ein Context vorgeben werden. In diesem fall werden die oben genannten Regeln ignoriert.
  Diese Regel wird ignoriert falls die «gebunde-funktion» mit new aufgerufen wird.

```
var boundFoo = foo.bind({counter : 11});
boundFoo();
```

Scope und Context

- Aus diesem Grund ist nicht garantiert was für ein Objekt sich hinter this verbirgt
  - Dies im Gegensatz zu fast allen anderen Programmiersprachen, wo this das instanziierte Objekt repräsentiert
- Um sicher zu sein, dass man auf das instanziierte Objekt zugreifen kann, empfiehlt es sich eine Variable für this zu erstellen

Scope und Context

- Hier ein Beispiel dazu mit jQuery
  - jQuery selbst behandeln wir später in diesem Kurs

```
var colours = ['red', 'green', 'blue'];
document.getElementById('element').addEventListener('click', function() {
    // this is a reference to the element clicked on

    var that = this;

    colours.forEach(function() {
        // this is undefined
        // that is a reference to the element clicked on
    });
});
```

```
$('#element').click(function(){
    // this is a reference to the element clicked on

    var that = this;

    $('.elements').each(function(){
        // this is a reference to the current element in the loop
        // that is still a reference to the element clicked on
    });
});
```

Scope und Context

- Wir werden diesbezüglich noch weitere Themengebiete in folgenden Vorlesungen ansehen
  - Execution Context
  - Scope Chain
  - Closures
  - Function Lookup

Übungen

- Scope\Scope.js
  - Überlegen Sie sich, was auf der Console ausgeben wird. Überprüfen Sie Ihre Überlegung.
  - Spielen Sie etwas mit den Variablen. Um ein Gefühl für den globalen und den funktionalen Scope zu bekommen.
- Scope\Scope1.js:
  - Ziel dieses Programmes ist es den Index der übergeben Buchstaben im Alphabet-Array zu finden und diese Indexes zurückzugeben.
  - Finden Sie die Ursache weshalb A funktioniert und B nicht.
  - Nutzen Sie den Debugger.
  - Ändern Sie den Code damit dieser funktioniert.

Practice

Übungen

- Context\Context1.js
  - Überlegen Sie sich, was auf der Console ausgeben wird. Überprüfen Sie Ihre Überlegung.

Hoisting

- Dieses Feature nennt sich «Hoisting»
- JavaScript verschiebt alle Deklarationen von Methoden / Variablen an den Anfang des Scopes.
- Initialisierungen werden nicht ge-«hoisted»
- Kann zu Bugs führen.
- Funktionen-Definitionen werden auch verschoben.
- Achtung: Bei einer «Function-Expressions» wird die Zuweisung nicht verschoben.

Hoisting

- Beispiel bezüglich hoisting, nur dadurch können beide Funktionscodes ohne Fehler ausgeführt werden

Variante 1:

```
"use strict";
var a;
a  = 1;
console.log("0",a);
```

Variante 2:

```
"use strict";
a = 1;
console.log("0",a);
var a;
```

Objekte sind HashTables

- Eine HashTable ist eine Tabelle mit einer speziellen Indexstruktur
- Der Key um auf einen Inhalt zuzugreifen ist durch einen mathematischen Hash Wert abgelegt
- JavaScript nutzt diese Funktionalität für alles mögliche um so Code einfach aufrufbar und erweiterbar zu halten

https://de.wikipedia.org/wiki/Hashtabelle

Objekte sind HashTables

- Jedes Objekt ist eine HashTable
  - Ausnahme: null, undefined
- Zugriff auf Eigenschaften vom Objekt mit
  - obj[PropertyName] bzw. obj[PropertyName] = "A "
- Objekte
  - Wandeln den Array Operator immer in einen String.
  - Sample[1] == Sample["1"]

Objekte sind HashTables

- Array
  - Unterstützt ganzzahlige Nummern als Indexer
  - Nicht ganzzahlige Nummern werden zu Strings gewandelt.
  - [].length beachtet nur die echten Array Einträge.
  - => Ein Array sollte wie ein Array verwendet werden.
- Functions
  - Können wie Objekte mit Properties ergänzt werden

Bildungszentrum Uster
Höhere Berufsbildung
Uster
HBU

Objekte

Objektarten

- JavaScript kennt vier verschiedene Arten von Objekten

- JavaScript Objekte (in der Sprache enthalten)
- Browser Object Model (BOM) Objekte
- Document Object Model (DOM) Objekte
- Vom Entwickler geschriebene Objekte (Objektorientierte Programmierung)

- Alle Objekte verhalten sich gleich, aber die ersten drei Kategorien sind vordefiniert

JavaScript Objekte

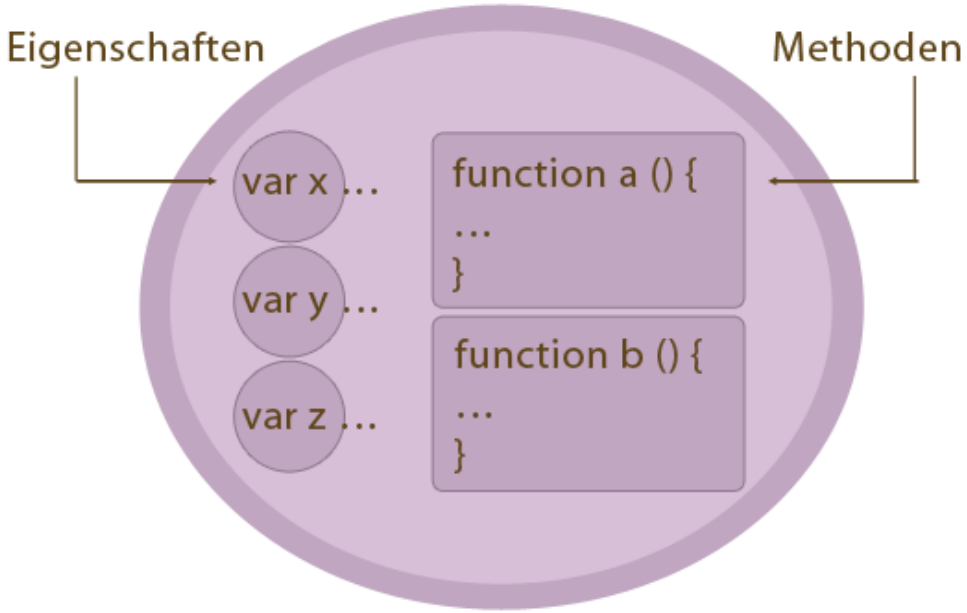
- sind ein Behälter für logisch zusammengehörenden Variablen und Funktionen
- diese werden als Eigenschaften und Methoden des Objekts bezeichnet
- Objekte kapseln Daten und Logik nach aussen ab und bieten eine einfache Schnittstelle

Bildungszentrum Uster
Höhere Berufsbildung
Uster
HBU

JavaScript Objekte

- Beispiel

```
var myObj = new MyObejct();
var x = myObj.x;          // Zugriff auf Eigenschaft
myObj.a();                // Funktionsaufruf
```



JavaScript Objekte

- Die wichtigsten JavaScript Objekte sind
  - Date
  - String
  - Number
  - Math
  - Array

Date Objekt

- Das Date Object repräsentiert ein Datum

- Erzeugung

```
var d = new Date();
var d = new Date(milliseconds);
var d = new Date(dateString);
// Date("March 20, 2010 12:20:25");
var d = new Date(year, month, day, hours, minutes, seconds,
milliseconds);
```

String Objekt

- Stellt eine Zeichenkette dar

```
var txt = new String("string");
```
ist das gleiche wie
```
var txt = "string";
```

Eigene Objekte

- In JavaScript werden eigene Objekte nicht wie in anderen Sprachen als Klassen definiert
- Es ist mehr eine Erweiterung der Funktionen
- Echte Objektfunktionalität ist für JavaScript 2.0 definiert, aber derzeit ist nicht absehbar ob sich dieser Standard durchsetzen wird

Beispielobjekte

```
function Farbe (R, G, B)
{
  this.R = R;
  this.G = G;
  this.B = B;
  this.hex = "#";
}

function HintergrundWechseln () {
  var Hintergrund = new Farbe("E0", "FF", "E0");
  document.bgColor = Hintergrund.hex + Hintergrund.R +
Hintergrund.G + Hintergrund.B;
}
```

Beispielobjekte

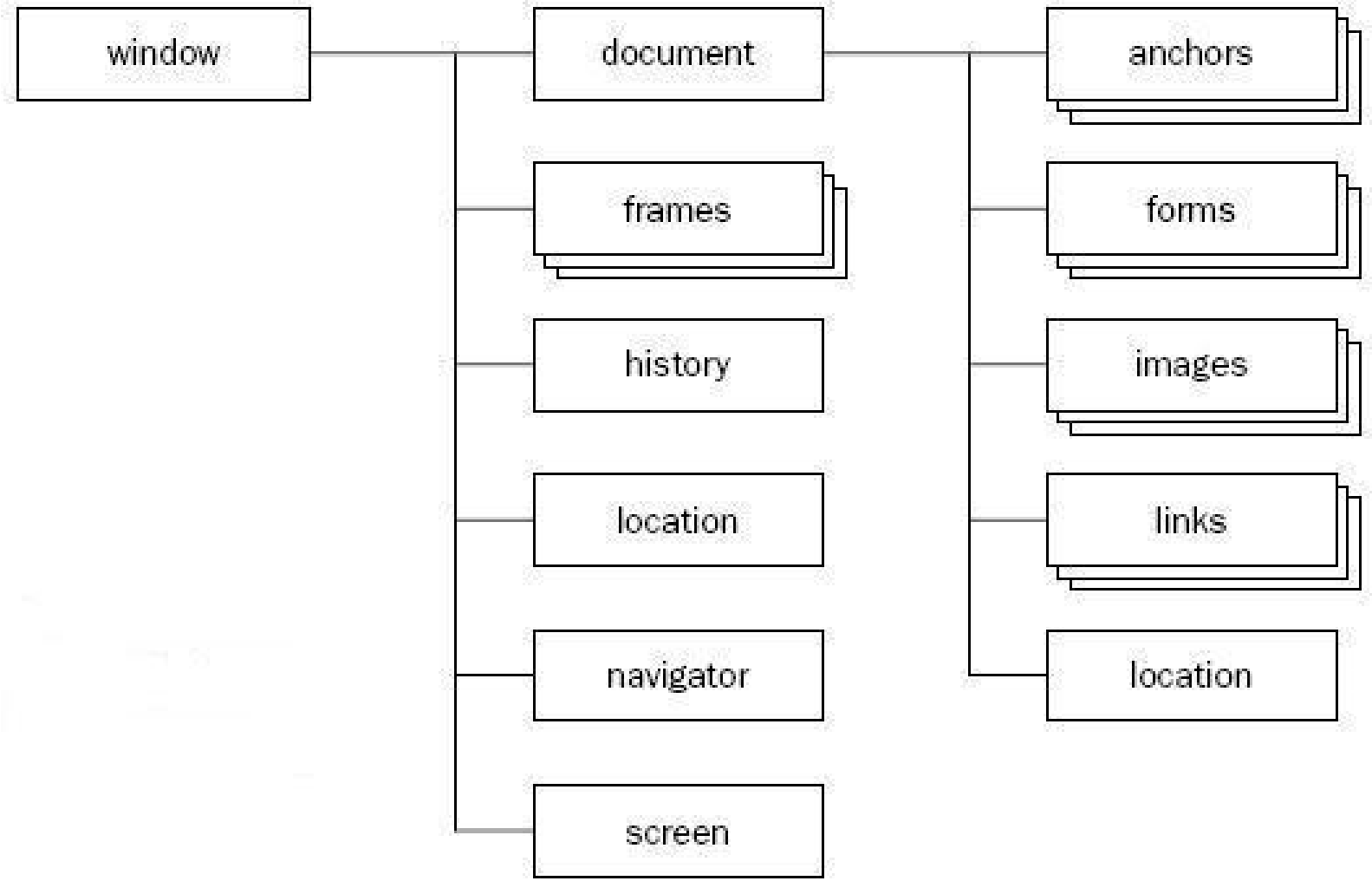
```
function team() {
        this.name;
        this.city;
        this.members = new Array();
        this.add_member = add_member;
}

function member() {
        this.gname;
        this.fname;
        this.birthday;
}

function add_member(member) {
        this.members.push(member);
}
```

```
var fire_lions = new team();
fire_lions.name = "FireLions";
fire_lions.city = "Feuerstadt";
fire_lions.founded = 1960;

var hs = new member;
hs.gname = "Hans";
hs.fname = "Sander";
hs.birthday = "21.03.1980";
fire_lions.add_member(hs);

document.write("Team-Mitglieder: "
+ fire_lions.members.length +
"<br>");
document.write("Vorname: "
+ fire_lions.members[0].gname +
"<br>");
document.write("Zuname: "
+ fire_lions.members[0].fname +
"<br>");
```

Browser Object Model

- Das BOM ist eine in JavaScript enthaltene Sammlung von Objekten
- Es ist nicht standardisiert, aber seit IE9 haben sich die Hersteller fast auf eine gemeinsames API geeinigt
- Wird manchmal mit dem Document Object Model (DOM) verwechselt oder sogar als DOM Level 0 bezeichnet

Browser Object Model



Window Object

- Das window Objekt repräsentiert ein Fenster im Browser
- Oberstes Objekt in der BOM Hierarchie

- Methoden:
  - confirm() Dialogfenster zum Bestätigen
  - prompt() Dialogfenster zur Werteingabe
  - alert() Dialogfenster zur Anzeige
  - open() Neues Fenster öffnen

Window Object

- Methoden zum verzögerten oder repetitiven Ausführen von Anweisungen
  - setInterval()
    Führt eine Anweisung repetitiv immer wieder aus
  - clearInterval()
    Bricht einen Endlosvorgang ab, der mit setInterval() begonnen wurde
  - setTimeout()
    Führt eine Anweisung nach einer bestimmten Verzögerungszeit aus
  - clearTimeout()
    Bricht einen Timeout ab der mit setTimeout() gestartet wurde

Window Object

- setInterval(code, interval) erwartet zwei Parameter
  - code
    JavaScript Anweisung die wiederholt werden soll, meistens ein Funktionsaufruf
  - interval
    Zeit in Millisekunden bis zum nächsten Ausführen

- Analoge Methode: setTimeout(code, timeout)

Window Object

- Beispiele mit setInterval() und clearInterval()
- var interval = window.setInterval("show()", 1000);
  die Funktion show() wird alle 1000 ms aufgerufen (Endlosschleife)

- die Variable interval ist die Referenz auf das gestartete Interval

- window.clearInterval(interval);
  stoppt das Interval

- setTimeout() und clearTimeout() werden analog aufgerufen (einmalige verzögerte Ausführung)

History Object

- Das history Object repräsentiert die History des Browsers
- Wichtige Methoden
  - back()
    zurückspringen (gleiche Wirkung wie der Back Button)
  - forward()
    vorwärtsspringen (gleiche Wirkung wie der Forward Button)

Location Object

- Das location Object repräsentiert die aktuelle URL des Browser
- Attribute
  - href (URL)
- Methoden:
  - reload() neu laden (gleiche Wirkung wie Reload Button)

- Beispiel

```
window.location.href = "neueSeite.html";
```

Navigator Object

- Informationen über den Browser selbst (z.B. Version)

- Attribute
  - appName (offizieller Name des Browsers)
  - appVersion (Browser-Version)
  - cookieEnabled (Cookies erlaubt)
  - language (Browser-Sprache)
  - platform (Plattform, auf der der Browser läuft)
  - userAgent (HTTP-Identifikation des Browsers)

Übungen

- Übung 1.1 selbständig lösen
- Übung 1.2 selbständig lösen

Bildungszentrum Uster
Höhere Berufsbildung Uster
HBU

EventHandler

EventHandler

- Eventhandler sind eine wichtige Schnittstelle zwischen HTML und JavaScript.
- Eventhandler erlauben es, auf Aktionen des Anwenders zu reagieren
- Eventhandler werden als Attribute von HTML-Elementen notiert und beginnen immer mit on
- Als Wert für diese Attribute wird in der Regel eine JavaScript Funktion aufgerufen.
- Es ist auch möglich, einzelne JavaScript-Anweisungen als Werte zu notieren, dies führt aber recht schnell zu einem unübersichtlichen Quelltext des HTML-Dokuments.

Bildungszentrum Uster
Höhere Berufsbildung
Uster
HBU

EventHandler

- Texteingabefeld

```
<input type="text" name="in" onblur="check(this.value)">
```

- Überschrift

```
<h1 onclick="alert('click')">Überschrift</h1>
```

Universale EventHandler

- Die folgende Tabelle listet nur solche Event-Handler auf, die im HTML-Standard als weitgehend universell und für die meisten HTML-Elemente einsetzbar festgehalten sind.
- Es gibt noch weitere Event-Handler, die jedoch nur für bestimmte Elemente gültig sind.

Universale EventHandler

Event	Bedeutung
onclick	Für den Fall, dass der Anwender ein Element anklickt
ondblclick	Für den Fall, dass der Anwender ein Element doppelt anklickt
onmousedown	Tritt ein, wenn der Anwender die Maustaste gedrückt hält
onmouseup	Tritt ein, wenn der Anwender die Maustaste gedrückt hat und sie nun wieder loslässt
onmouseover	Tritt ein, wenn der Anwender mit der Maus über ein Element fährt
onmousemove	Tritt ein, wenn der Anwender die Maus bewegt, unabhängig davon, ob die Maustaste gedrückt ist oder nicht
onmouseout	Tritt ein, wenn der Anwender mit der Maus über ein Element fährt und dieses dabei verlässt
onkeypress	Tritt ein, wenn der Anwender eine Taste drückt und diese gedrückt hält
onkeydown	Tritt ein, wenn der Anwender, während er ein Element aktiviert hat, eine Taste drückt
onkeyup	Tritt ein, wenn der Anwender, eine Taste gedrückt hat und diese wieder loslässt

Rückgabewerte

- Eventhandler liefern einen Rückgabewert an den Browser.
- Diese Werte steuern, wie der Browser dieses Ereignis weiter verarbeitet.
- Eine Funktion die in einem Eventhandler angegeben wurde, kann einen booleschen Wert zurückliefern.
- Wird diese Funktion im Eventhandler mit return aufgerufen, reagiert dieser auf den Rückgabewert:
  - bei true wird die normale Ereignisbehandlung des Browsers ausgeführt
  - bei false wird diese abgebrochen

Rückgabewerte

- Das Absenden eines Formulars beim Click auf einen Submit-Button. Bei false wird dieses unterdrückt.

```
<form action="ziel.html" method="get"
      onsubmit="return confirm('Sind Sie sicher?');">
  <input type="submit" value="Daten absenden">
 </form>
```

Registrierung

- Eventhandler als Attribute eines HTML-Elements zu notieren

```
<body onload=“welcome()“>
```

- Eventhandler als Methode eines DOM Objektes registrieren

```
function welcome() {
  alert("Willkommen");
}
window.onload = welcome;
```

- Mittels addEventListener

```
element.addEventListener("event", handlerfunktion, capturing);
```

addEventListener

- element.addEventListener("event", handlerfunktion, capturing);
- Die Methode addEventListener() erwartet drei Parameter
  - Der erste Parameter gibt an, welcher Ereignistyp überwacht werden soll (bspw. click, mouseover, mousedown)
    - Der Name des gewünschten Ereignistyps muss bei der Parameterangabe in Anführungszeichen stehen
- Der zweite Parameter gibt die Funktion an, welche beim Auftreten des Events ausgeführt werden soll.
- Der dritte Parameter von addEventListener() bestimmt in welcher Phase ein Event-Handler aktiv wird
  - Mit true wird die Handler-Funktion in der Capture-Phase aufgerufen, mit false in der Target- oder Bubbling-Phase
  - In der Regel wird false gewählt

removeEventListener

- Zu addEventListener / attachEvent gibt es ein Gegenstück removeEventListener bzw. detachEvent, welches den Event Handler wieder vom Event trennt

```
element.removeEventListener("event", handlerfunktion, capturing);
element.detachEvent("event", handlerfunktion);
```

Kompatibilität

- element.addEventListener("event", handlerfunktion, capturing); funktioniert in IE Versionen 8 und älter nicht.
  - Stattdessen kann element.attachEvent("event", handlerfunktion) verwendet werden

Übungen

- Übung 1.3 selbständig lösen

Rückblick