



# Willkommen bei der Höheren Berufsbildung Uster

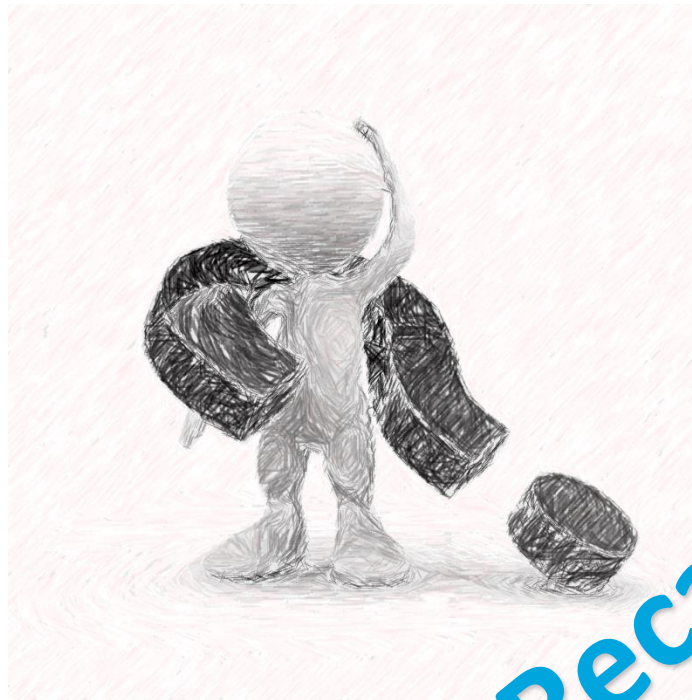
## JavaScript Raphael Ritter

Ein Angebot der Berufsfachschule Uster und der Höheren Fachschule Uster

# Agenda

- Demo Prüfung
- DOM
  - Lesen
  - Schreiben
- JavaScript Fehlerbehandlung

# Rückblick



Recap

# Unterlagen der HSR

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



# HSR

HOCHSCHULE FÜR TECHNIK  
RAPPERSWIL

FHO Fachhochschule Ostschweiz

[www.hsr.ch](http://www.hsr.ch)

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

## Demo Prüfung

- Hier ein Muster wie die Prüfungen jeweils aussehen
- Wir Lösen diese Prüfung gemeinsam



# Document Object Model

## JavaScript

# Document Object Model

HTML File  
**domDemo1.html**

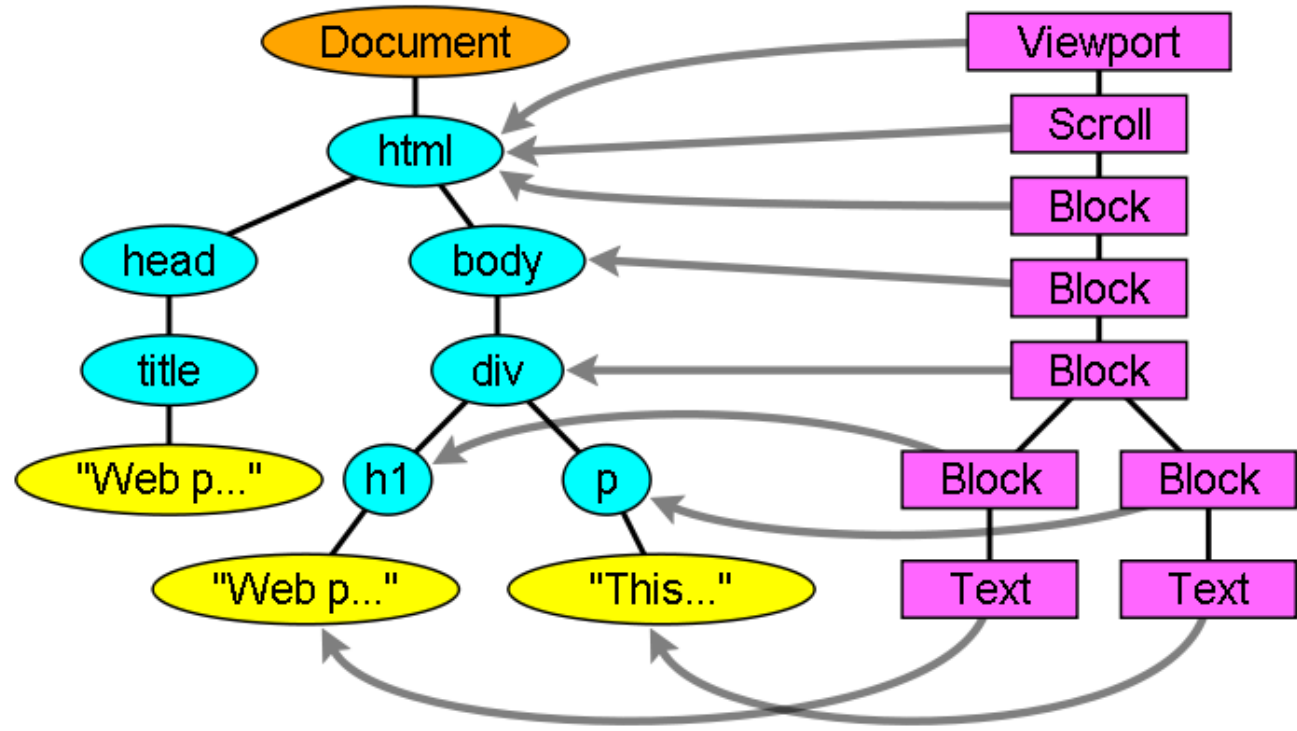
load, parse

DOM Tree

convert

Render Tree

```
<html>
<head>
  <title>Web p...
</title>
</head>
<body>
  <div>
    <h1>Web p...
  </h1>
    <p>This ...
  </p>
  </div>
</body>
</html>
```

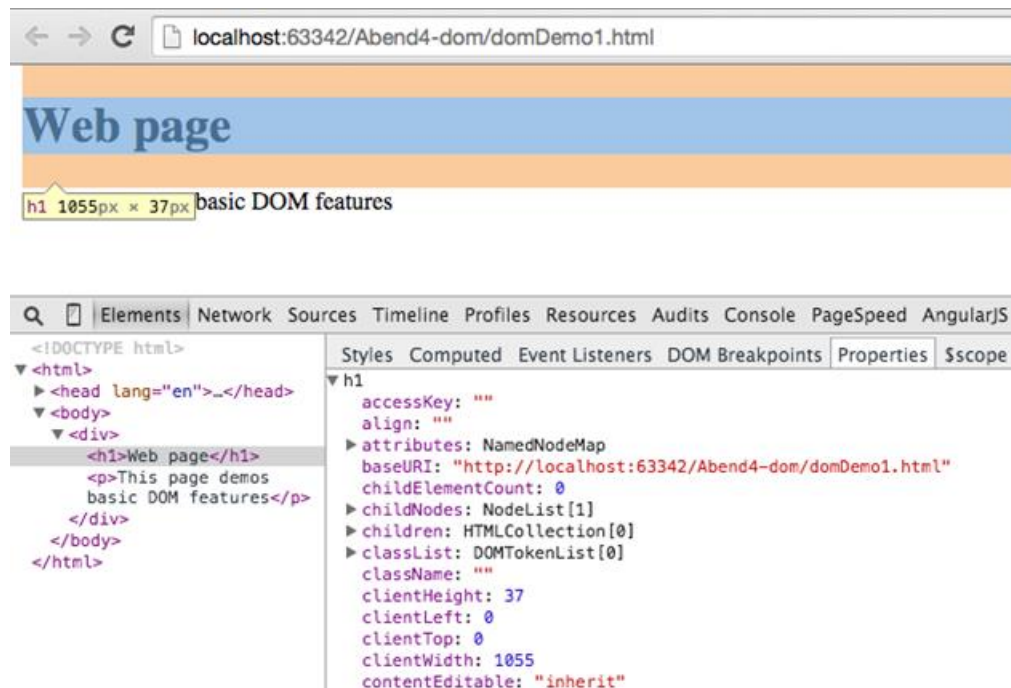


# Document Object Model

- HTML wird geparkt und die DOM Elemente können analysiert werden



```
<html>
<head>
  <title>Web p...
</title>
</head>
<body>
  <div>
    <h1>Web p...
  </h1>
  <p>This ...
</p>
</div>
</body>
</html>
```





## JavaScript

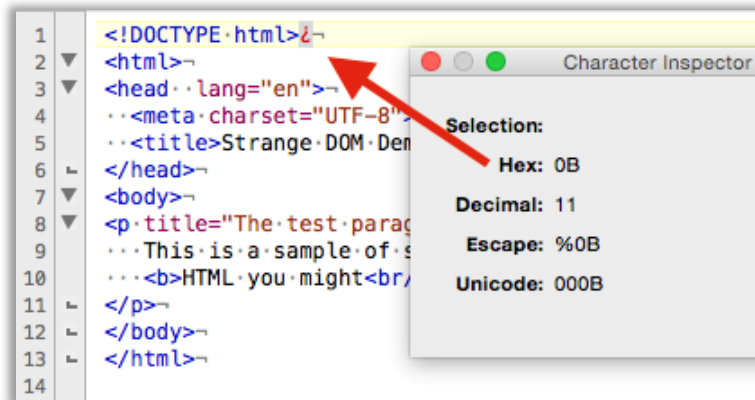
# Document Object Model

- HTML Parsing kann durch «special characters» fehlerhaft sein

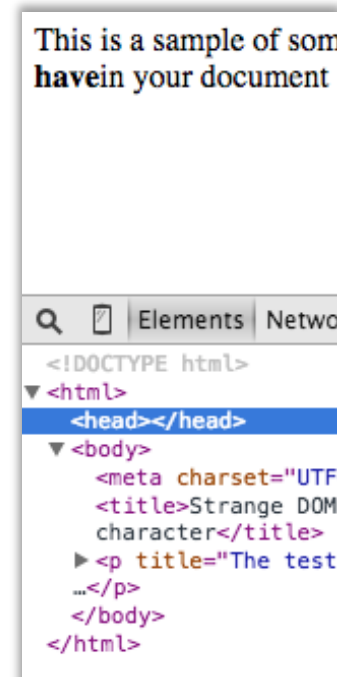
HTML  
File

load, parse

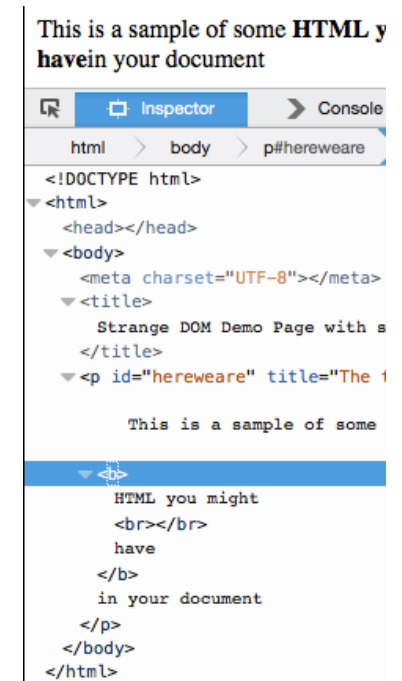
domDemo2strangeDOMTree-  
SpecialChar.html



DOM Tree Chrome

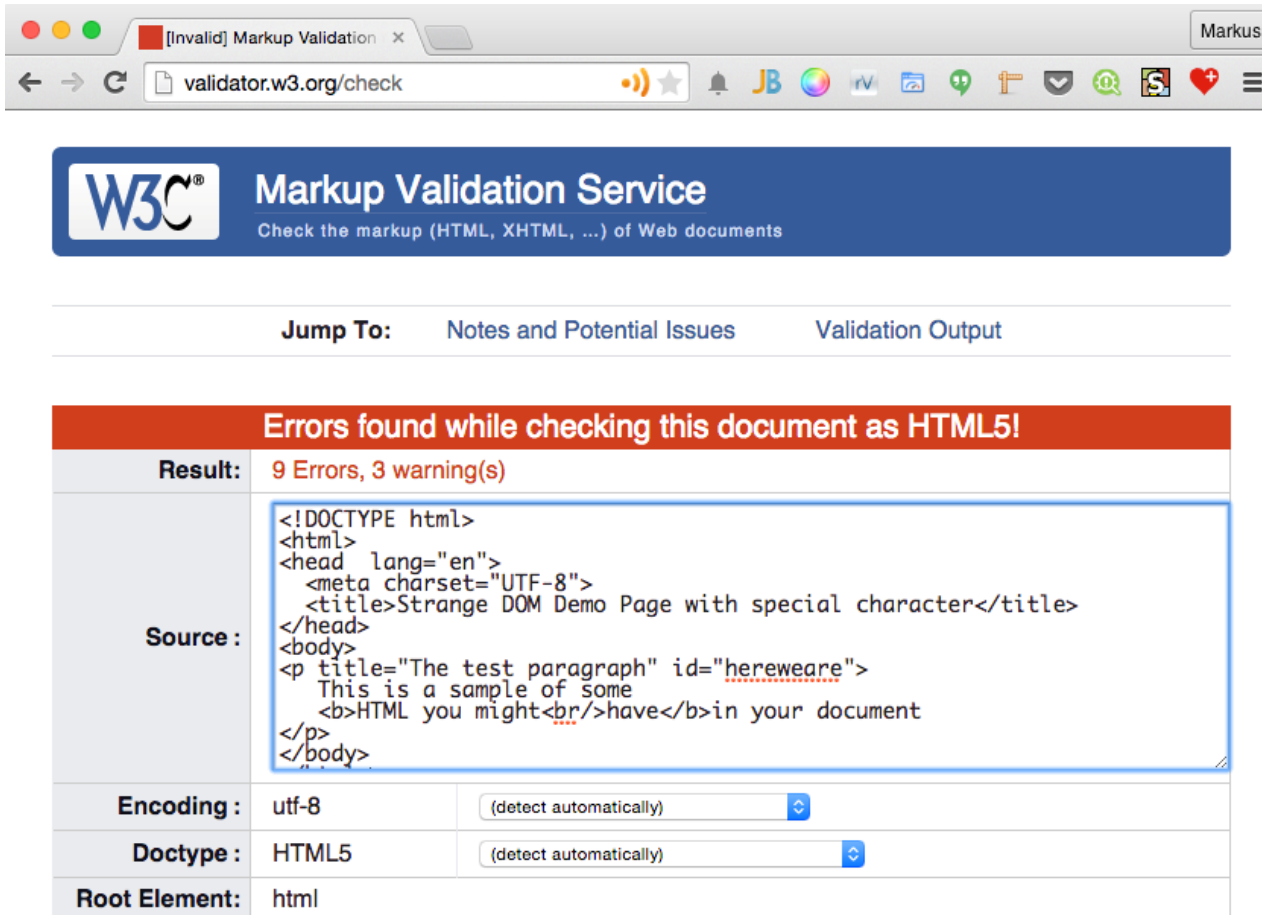


DOM Tree Firefox



# Document Object Model

- HTML Files sollten daher immer validiert werden!



The screenshot shows a web browser window with the address bar displaying `validator.w3.org/check`. The page title is "[Invalid] Markup Validation". The main heading is "Markup Validation Service" with the subtitle "Check the markup (HTML, XHTML, ...) of Web documents". Below this, there are tabs for "Jump To: Notes and Potential Issues" and "Validation Output". The "Validation Output" tab is active, showing a red header "Errors found while checking this document as HTML5!". The "Result:" section indicates "9 Errors, 3 warning(s)". The "Source:" section displays the HTML code being validated, with red squiggly lines under the `id="hereweare"` attribute and the `<br/>` tag. The "Encoding:" is set to "utf-8", "Doctype:" is "HTML5", and "Root Element:" is "html".

Errors found while checking this document as HTML5!	
<b>Result:</b>	9 Errors, 3 warning(s)
<b>Source:</b>	<pre>&lt;!DOCTYPE html&gt; &lt;html&gt; &lt;head lang="en"&gt;   &lt;meta charset="UTF-8"&gt;   &lt;title&gt;Strange DOM Demo Page with special character&lt;/title&gt; &lt;/head&gt; &lt;body&gt;   &lt;p title="The test paragraph" id="hereweare"&gt;     This is a sample of some     &lt;b&gt;HTML you might&lt;br/&gt;have&lt;/b&gt;in your document   &lt;/p&gt; &lt;/body&gt;</pre>
<b>Encoding:</b>	utf-8 (detect automatically)
<b>Doctype:</b>	HTML5 (detect automatically)
<b>Root Element:</b>	html

# Document Object Model

- Jeder Text im html File wird ins DOM übersetzt SCHRITT für SCHRITT

- Das DOM kann abgefragt werden (API)

`document.querySelector('#button5')`

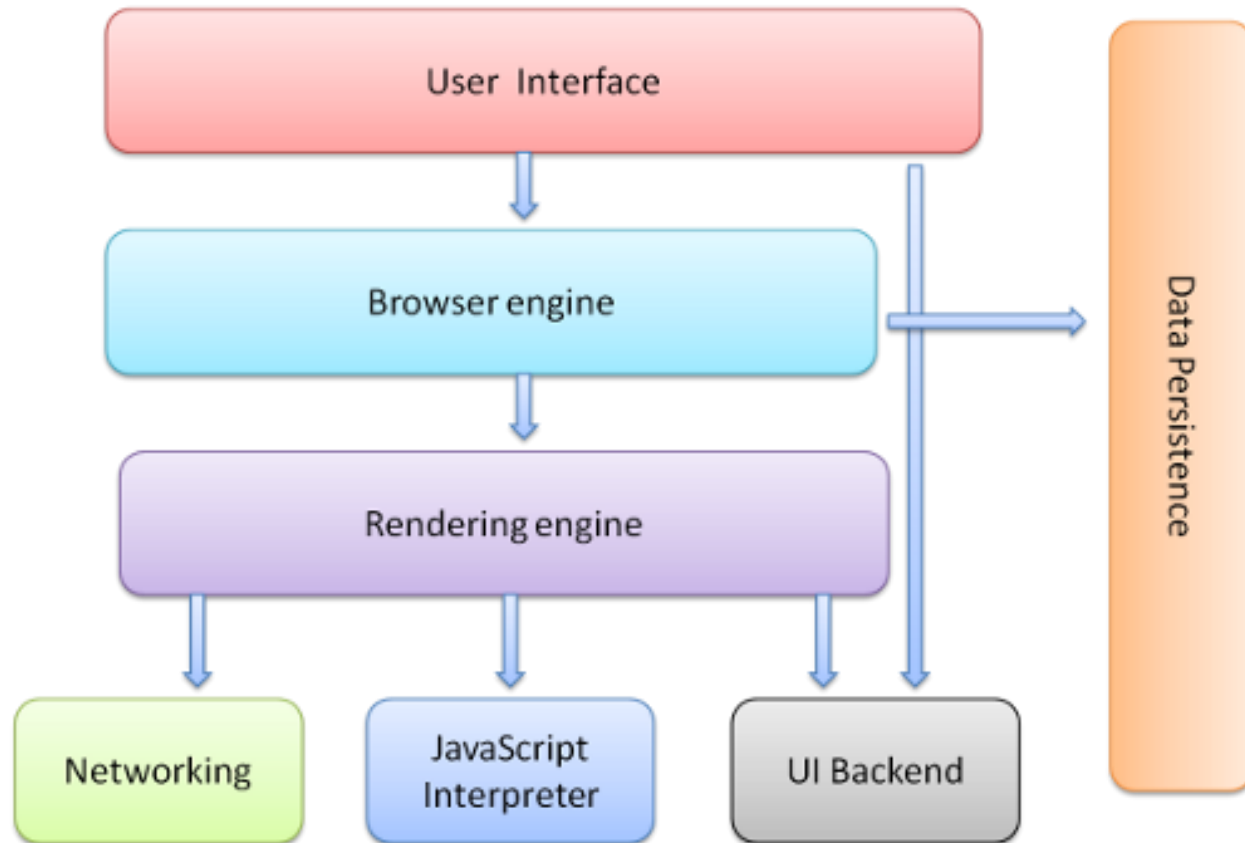
- DOM kann interaktiv verändert werden

`input2.value = this.value`

- Events können an Elemente DOM gehängt werden (haben wir bereits ohne das DOM wissen angesehen)

`button5.addEventListener("click", ...)`

# Document Object Model



# Document Object Model

- DOM Standard **API**
  - DOM Baum (document), Node Typen und White-Space
  - DOM API Anfragen und Response Typen
  - DOM Manipulation
  - DOM Events

## DOM Baum

- Globales Objekt document: DOM wichtige Eigenschaften / Methoden
  - E: body (body Element-Node)
  - M: getElementById(<idStr>)->Node
  - M: getElementsByTagName(<tagStr>) ->NodeList auch möglich in modernen Browsern:
    - M: querySelector()/querySelectorAll
- Objekt-Typ: Node wichtige Eigenschaften / Methoden
  - E: nodeType, childNodes, parentNode, nodeValue
  - M: appendChild(<childNodes>),  
removeChild(<childNodes>)

## DOM Baum

- Wichtige Node Interfaces:
  - Element-Node attributes, id, this.remove(), addEventListener(L)
  - Text-Node: (auch Whitespace) isElementContentWhitespace()
  - Comment-Node
- Collection-Typen:
  - NodeList (Achtung kein Array)
  - NamedNodeMap (Map)





## DOM API

- DOM Anfragen d: document, n: node
  - d.getElementById(<idStr>) -> Node

```
document.getElementById('p1')  
▶ <p id="p1">...</p>
```

- d.getElementsByTagName(<tagStr>)->NdLst  
ACHTUNG: Antworten sind live

```
document.getElementsByTagName('em')  
[ <em>with emphasized text</em>,  
  <em>with emphasized text</em>]
```

- n.getElementsByTagName(<tagStr>)->NdLst

```
> document.body.childNodes[5].getElementsByTagName('em')  
< [ <em>with emphasized text</em>]
```

## DOM API

- `querySelector()` / `querySelectorAll()`  
ACHTUNG Antworten sind NICHT live
- NodeList API:
  - `myNodeList.length`
  - `myNodeList[i]` oder `myNodeList.item(i)`
  - Iteration

```
for (var i = 0; i < myNodeList.length; ++i) {  
    var item = myNodeList[i]; ...  
}
```

## DOM API

- Weitere häufiger genutzte DOM Anfragen (und Navigation)

n.<HTML-Attribute> -> value

n.textContent -> Text aller Child-Nodes (ganzer Baum)

NICHT innerText

n.innerHTML -> HTML aller Child-Nodes (ganzer Baum)

n.firstChild / n.firstElementChild

n.nextSibling / n.nextElementSibling

# JavaScript

# DOM API

```
> b2 = document.getElementById('button2');  
< <button id="button2" value="valueText">Click Me2</button>  
> b2.childNodes[0].nodeValue  
< "Click Me2"  
> b2.innerText  
< "Click Me2"  
> p1=document.getElementById('p1');  
< ▶ <p id="p1">...</p>  
> p1.innerText  
< "paragraph with emphasized text and more"  
> p1.innerHTML  
< "paragraph <em>with emphasized text</em> and more"  
> p1.childNodes  
< [ "paragraph ", <em>with emphasized text</em>, " and more"]  
> p1.firstChild  
< <em>with emphasized text</em>  
> h1=document.body.firstChild;  
< <h1>DOM API Demo</h1>  
> h1.nextElementSibling  
< <em>with emphasized text</em>
```

ACHTUNG: Nicht nutzen! Siehe Browser Support

[←](#)
[→](#)
[C](#)
[www.quirksmode.org/dom/html/](#)

Contents of this table

See also the [key](#) to my compatibility tables.

show page contents

Method or property	Internet Explorer					Firefox			Safari	Chrome			Opera		Yandex	
	7 and lower	8	9	10	11	32 Win	32 Mac	32 Linux	7	37 Win	37 Mac	37 Linux	24 Win	24 Mac	14 Win	14 Mac
<div>innerHTML</div> <div> <p>The HTML contained by a tag, as a string.</p> <p>Originally a Microsoft extension, innerHTML is so useful that all other browsers support it, too.</p> <p><a href="#">Test page</a></p> </div>	incomplete	almost	yes			yes			yes	yes			yes		yes	
<div> <div> <div>x.innerHTML</div> <div>x.innerHTML = "Let's &lt;u&gt;change&lt;/u&gt; it!"</div> <div>Get or set the HTML contained by node x.</div> <div> <ul style="list-style-type: none"> <li>If you change the innerHTML of a script tag the browser doesn't recognize the new script and you can't execute it. Don't do this.</li> <li>The order of the attributes on the HTML element may be off in IE. Not a big deal, but still.</li> <li>If you remove elements through innerHTML in IE, their content is wiped and only the element itself (i.e. opening and closing tags) remain. If you want to remove nodes that you may want to reinsert at a later time, use DOM methods such as removeChild().</li> <li>IE8 can't write the innerHTML of a &lt;style&gt; tag.</li> <li>IE8 returns UPPER CASE tags, even though the source is lower case.</li> <li>In IE9 and lower innerHTML refuses to work on tables and selects. Solve this by using pure DOM methods instead. See <a href="#">this explanation</a> of the table behaviour by innerHTML's inventor. I assume something similar goes for selects.</li> </ul> </div> </div> </div>																
<div>innerText</div> <div> <p>The text contained by a tag.</p> <p><a href="#">Test page</a></p> </div>	yes	no	yes			yes			yes	yes			yes		yes	
<div> <div> <div>x.innerText</div> <div>x.innerText = "Let's change it!"</div> <div>Get or set the text contained by node x. Any HTML tags in the node are ignored, though their text content is added to the innerText. If you set innerText all inner HTML elements are removed.</div> <div>Cross browser:</div> <div>var text = x.textContent    x.innerText</div> </div> </div>																
<div>insertAdjacentElement</div> <div> <p>Add a DOM node next to an existing node</p> </div>	probably	yes	no			yes			yes	yes			yes		yes	
<div> <div> <div>x.insertAdjacentElementL(position, DOM_node)</div> <div>Takes two arguments: the position (see below) and the DOM node to be inserted.</div> </div> </div>																

# DOM API

- Was ist der Log Output?

```
1 <!DOCTYPE html>
2 <html>
3 <head lang="en">
4   <meta charset="UTF-8">
5   <title>HTML Parsing Demo</title>
6   <script>
7     var outTxt;
8     outTxt = document.getElementById("p1").nextSibling.textContent; // -> _____
9     console.log(outTxt);
10    //outTxt = document.getElementById("p1")._____ // -> Text2.2
11
12    //...
15    console.log(outTxt);
16  </script>
17 </head>
18 <body>
19   <p id="p1">Text1.1<em>Text1.2</em>Text1.3</p>
20   <p>Text2.1<em>Text2.2</em>Text2.3</p>
21 </body>
22 </html>
```

**domDemo4HTMLParsing.html**



# DOM Manipulation

## DOM Manipulation (e: Element)

- e.removeChild(<childNodes>)

```
t1 = document.getElementById('text1');  
<input id="text1" type="text" value="replace text">  
document.body.removeChild(t1)  
<input id="text1" type="text" value="replace text">
```

- e.appendChild(<newDocNode>)
- e.insertAfter(<childNodes>, <newDocNode>)

```
> t1 = document.createElement('INPUT')  
t1.id = "text1";  
t1.value = "new text1";  
< "new text1"  
> document.body.appendChild(t1)  
< <input id="text1">
```

- e.textContent = "....."; e.innerHTML = "...."

```
> b2.innerText="**Click Me**"  
< "**Click Me**"
```

## DOM Manipulation

- Effizientes append von mehreren Elementen :  
"document fragment" nutzen

```
var df = document.createDocumentFragment();
songs.forEach(function (song) {
  var liElement = document.createElement("li");
  [...]
  df.appendChild(liElement);
});
document.getElementById("songs").appendChild(df);
```



# DOM Manipulation

## Properties (Auswahl)

- Node.childNodes
- ParentNode.childElementCount
- Node.firstChild
- ParentNode.firstElementChild
- Node.lastChild
- ParentNode.lastElementChild
- Node.localName
- Node.nextSibling
- Node.nodeName
- Node.nodeType
- Node.nodeValue
- Node.ownerDocument
- Node.parentElement
- Node.parentNode
- Node.previousSibling
- Node.textContent

## Methods

- Node.appendChild()
- Node.cloneNode()
- Node.compareDocumentPosition()
- Node.contains()
- Node.hasChildNodes()
- Node.insertBefore()
- Node.isEqualNode()
- Node.removeChild()
- Node.replaceChild()

# DOM Manipulation

## Properties (Auswahl)

- Document.activeElement
- Document.body
- Document.currentScript
- Document.documentURI
- Document.forms
- Document.head
- Document.height
- Document.lastModified
- Document.linkColor
- Document.location
- Document.onafterscriptexecute
- Document.onoffline
- Document.ononline
- Document.title
- Document.tooltipNode
- Document.URL
- Document.width

## Methods (Auswahl)

- Document.caretPositionFromPoint()
- Document.createAttribute()
- Document.createCDATASection()
- Document.createComment()
- Document.createElement()
- Document.createTextNode()
- Document.elementFromPoint()
- Document.evaluate()
- Document.getElementById()
- Document.getElementsByTagName()
- Document.getElementsByTagName()
- Document.getSelection()
- Document.hasFocus()
- Document.queryCommandSupported()
- Document.querySelector()
- Document.querySelectorAll()

# DOM Manipulation

## Basics

- SelfHTML  
<http://wiki.selfhtml.org/wiki/JavaScript/Objekte/DOM>
- MDN DOM Introduction  
[https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model/Introduction](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction)

## Weiterführend

- MDN DOM Developer Guide  
<https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM>
- MDN DOM Reference  
[https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model)

# DOM Manipulation

Weiterführend

- W3Schools zu DOM API

[http://www.w3schools.com/jsref/dom\\_obj\\_document.asp](http://www.w3schools.com/jsref/dom_obj_document.asp)

[http://www.w3schools.com/jsref/dom\\_obj\\_all.asp](http://www.w3schools.com/jsref/dom_obj_all.asp)

[http://www.w3schools.com/jsref/dom\\_obj\\_attributes.asp](http://www.w3schools.com/jsref/dom_obj_attributes.asp)

[http://www.w3schools.com/jsref/dom\\_obj\\_event.asp](http://www.w3schools.com/jsref/dom_obj_event.asp)

# Übungen

- Übung 1.1 selbständig lösen
- Übung 1.2 selbständig lösen
- Übung 1.3 selbständig lösen



Practice

## DOM Events

- Im Browser ist window das zentrale globale Object  
(In Node.js ist global das Global Object)  
Jede Zuweisung ohne vorherige Variablen-Deklaration erzeugt eine globale Variable, eine Eigenschaft in window können anderen globalen Objekte aufgelistet werden.  
Andere bekannte globale Objekte: window.document und window.document.body

## DOM Events

- Viele Benutzerinteraktionen und Zustandsänderungen im Browser werden vom Browser als EVENT an das window oder direkt an Elemente im DOM Baum geliefert. Das heisst der Browser ruft die vom Web-Seiten Autor gesetzten (`window.onload = myLoadEventHandler`) oder registrierten (`window.addEventListener("load", function (event){...})`) Event Handler auf wenn der entsprechende Zustand erreicht wurde (hier: die Seite ist fertig geladen)

## DOM Events

- Genauso wie Änderungen im Browserzustand werden auch Benutzerinteraktionen an die entsprechenden Elemente im DOM geliefert.  
Clickt ein Benutzer auf einen Button im HTML, so stellt der Browser sicher, dass alle gesetzten oder registrierten click-EventHandlerler aufgerufen werden.



## DOM Events

- GUI Programmierung im Browser unterscheidet sich damit nicht dramatisch von anderen GUI Frameworks. Bei allen modernen GUI Frameworks definieren User Interface Entwickler das Verhalten des GUIs dadurch, dass Sie für GUI-Komponenten Event Handler definieren und bei diesen Komponenten registrieren.

Im Gegensatz zu einem Java main Programm, ist damit die Hauptkontrolle nicht beim Code des Entwicklers, sondern beim GUI Framework. Diese Abgabe der Kontrolle wird auch das "Hollywood Prinzip" genannt ("don't call us, we call you")

## DOM Events

- Die Registrierung der DOM Events etc. haben wir bereits in der letzten Vorlesung abgehandelt
- Hier eine kurze Repetition

- HTML

```
<button name="button1"  
  onclick="console.log('clicked '+this.name  
  "e:"+event)">B1</button>
```

- Zuweisung EventHandler

```
buttonClickListener = function (event) { ... }  
button4.onclick = buttonClickListener;
```

- Registrierung des EventHandlers

```
button5.addEventListener("click", buttonClickListener);
```

# DOM Events

## domDemo5domEvents.html

```

window.onload = function () {
    function buttonClickListener () {
        console.log('(buttonClickListener) clicked button '+this.name);
    }
    var button4 = document.querySelector("#button4");
    button4.onclick = buttonClickListener; //only a single listener possible
    function buttonClickListenerE (event) {
        //Event interface is differnt with IE8 or older
        console.log('(buttonClickListener E) clicked button '+ event.target.name);
        event.stopPropagation();
    }
    var button5 = document.querySelector("#button5");
    button5.addEventListener("click", buttonClickListenerE);
    var buttonsArray = document.querySelectorAll("button");
    for (var i=0; i < buttonsArray.length-1; i++) {
        //does not work with IE8 or older
        buttonsArray[i].addEventListener("click", buttonClickListenerE);
    }
    // event bubbling
    function bodyBubblingListener (event) {
        console.log('(bodyBubblingListener) clicked body this = '+this+" event target =" +event.target.name)
    }
    document.body.addEventListener("click", bodyBubblingListener);
    // event capturing
    function bodyClickCaptureListener (event) {
        console.log('(bodyClickCaptureListener) captured clicked body this = '+this+" event target =" +event
        if (glasspane) {
            event.stopPropagation();
        }
    }
    document.body.addEventListener("click", bodyClickCaptureListener, true);
}

```

## DOM Events

- Der Click beim **Button 1** wird von folgenden Handlern bedient
  - Capture Event Handler (aber ohne Glass-Pane nicht aktiv)
  - Vom Inline Code für die onclick Eigenschaft (dies ist gibt schlechte Trennung von UI & Code). "this" ist der Button.
  - Vom explizit im For-Loop mit addEventHandler angehängten Event-Handler
  - NICHT vom BubblingEventHandler, da im buttonClickListenerE event.stopPropagation aufgerufen wurde.

## DOM Events

- Click-Handling beim **Button 2** unterscheidet sich von *Button 1* indem im aufgerufenen ButtonClickListener1 `this` nicht definiert ist (nicht übergeben)
- Click-Handling beim **Button 3** unterscheidet sich von *Button 2* indem im Aufruf von ButtonClickListener2 `"this"` als `eventTarget` übergeben wurde.
- Click-Handling beim **Button 4** unterscheidet sich von *Button 3* indem der EventHandler zu `button.onclick` zugewiesen wurde. Hier ist Button4 als `"this"` gesetzt.

## DOM Events

- Der Click-Handler beim **Button 5** wurde mit `AddEventHandler` hinzugefügt. Hier wird der auslösende Button am besten über `event.target` zugegriffen
- **Button 6** wird nicht vom `buttonClickListenerE` bedient. Daher kommt nun der Bubbling Event Handler zum Zug.

## DOM Event Bubbling

- Bei einer Benutzerinteraktion ruft der Browser nicht nur den Event Handler des eigentlichen TargetElements auf sondern auch alle Container diese Elements. Die Reihenfolge ist vom TargetElement nach aussen. Diese Prinzip nennt sich Event Bubbling.  
Dies erlaubt es einen Event Handler nur an einem Container anzuhängen. Über die Eigenschaft des Events **event.target** lässt sich stets auf das originale TargetElement zugreifen.

## DOM Event Bubbling

- Das Bubbling eines Events kann unterdrückt werden indem auf dem Event **event.stopPropagation()** aufgerufen wird. Weitere registrierte EventHandlerler auf dem TargetElement werden aber allfällig trotz Aufruf von event.stopPropagation() aufgerufen



# DOM Event Capturing

- Event Capturing wird selten genutzt. Bei der Registrierung des Listeners wird hierfür ein dritter (optionaler) Parameter von `addEventListener` auf `true` gesetzt. z.B.  
`document.body.addEventListener("click",  
bodyClickCaptureListener, true)`  
Die Phase des Event Capturing passiert VOR der Phase des Event Bubblings und geht von aussen nach innen.
- Mehr Details zur Vertiefung
  - <http://javascript.info/tutorial/bubbling-and-capturing>
  - <https://developer.mozilla.org/en-US/docs/Web/API/Event>
  - [https://developer.mozilla.org/en-US/docs/Web/API/Event/Comparison\\_of\\_Event\\_Targets](https://developer.mozilla.org/en-US/docs/Web/API/Event/Comparison_of_Event_Targets)

# JavaScript

# Übungen

- Übung 1.4 selbständig lösen
- Übung 1.5 selbständig lösen
- Übung 1.6 selbständig lösen
- Übung 1.7 selbständig lösen



Practice

# **JavaScript Fehlerbehandlung**

# Fehlerbehandlungsarten

- Fehler in JavaScript können auf zwei Arten abgefangen werden:
  - `window.onerror` Funktion zuweisen
  - `try {...} catch` Block
- Generell gilt, dass nur Laufzeitfehler so abgefangen werden können.
- Syntaxfehler führen **IMMER** zu einem Abbruch des Scripts.

## Window.onerror

- Es wird eine Funktion angegeben die bei einem Fehler aufgerufen werden soll.
- Die Funktion ist immer global für das ganze Fenster aktiv.

```
window.onerror = Fehlerbehandlung; // Zuweisung  
function Fehlerbehandlung (Nachricht, Datei, Zeile) {  
    Fehler = "Fehlermeldung:\n" + Nachricht + "\n" + Datei + "\n" + Zeile;  
    zeigeFehler();  
    return true;  
}
```

```
function zeigeFehler () {  
    alert(Fehler);  
}
```

## Try...catch

- Wie in den meisten modernen Sprachen kann die Fehlerbehandlung auch mittels try {...} catch gemacht werden

```
try {  
    if (x == 2) {  
        throw "Wert 2 nicht erlaubt";  
    } else {  
        x= x+10;  
    }  
} catch (e) {  
    alert(e );  
} finally {  
    // Wird immer am Schluss ausgeführt  
}
```

## Tipps

- Fehler die «erwartet» werden, sollten immer mit try/catch behandelt werden.
- Unerwartete Fehler können mit window.onerror mindestens protokolliert werden.
- Fehler werden in der JS Konsole des Webbrowser protokolliert
- Es ist technisch möglich solche Fehler auch zurück an den Server zu senden, bringt aber selbst wieder zusätzliche Fehlerquellen mit sich.

# Übungen

- Übung 1.8 selbständig lösen



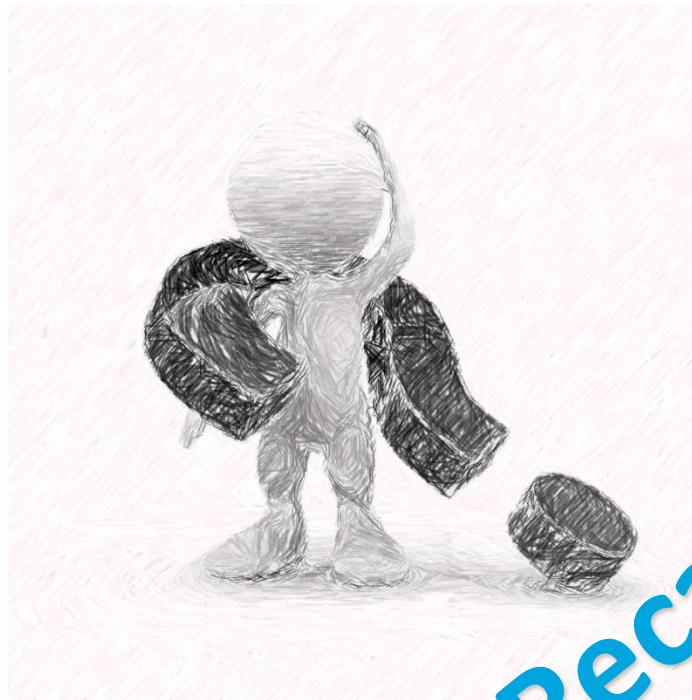


## Selbststudium

- Hier gibt es zwei online Tutorials um sich noch tiefer Vertraut mit JavaScript zu machen
- Diese beiden Tutorials sollen zur Vertiefung des Stoffes durchgearbeitet werden
- <https://www.codecademy.com/tracks/javascript>
- <https://www.udacity.com/course/javascript-basics--ud804>



**Practice**



**Recap**