Bildungszentrum Uster
Höhere Berufsbildung
Uster

HBU

Unterlagen

JavaScript

Tutorial zu JavaScript

Inhaltsverzeichnis

1. Einführung

JavaScript ist eine Skript Sprache, welche in erster Linie in den verschiedenen Web Browser verwendet wird. Die Sprache wurde von Berdan Eich entwickelt und wurde 1995 in den Netscape Navigator integriert. Alle gängigen Browser übernahmen JavaScript als Client-Seitige Sprache für Web Seiten. Der Kern von JavaScript ist als ECMA Skript standardisiert.

JavaScript ist eine dynamisch typisierte Sprache und erlaubt die objektorientierte, prozedurale und funktionale Programmierung. Trotz des Namens hat die Sprache nichts mit Java zu tun. JavaScript wird üblicherweise nicht kompiliert sondern interpretiert.

JavaScript wird für die Client-Seitige Programmierung von Web Seiten verwenden. Folgende Anwendungsgebiete stehen im Vordergrund:

- Client-Seitige Berechnungen
- Manipulation vom Document Object Model (DOM)
- Senden und empfangen von Daten ohne die Seite neu zu laden
- Animationen
- Aufwändige Designs von Web Seiten

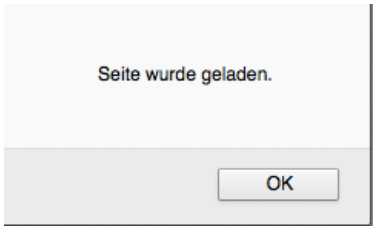
JavaScript läuft in einer Sandbox. Somit sind JavaScript Programme abgeschottet und es besteht keine Möglichkeit, auf das Dateisystem vom Client Computer zuzugreifen. Jede Applikation wird isoliert betrachtet und die Persistierung von Daten ist nur über Cookies möglich. Netzwerkzugriffe sind zum ursprünglichen Host erlaubt.

2. Integration von JavaScript

JavaScript Programmcode kann sowohl direkt in der HTML Seite oder in dedizierten externen JavaScript Dateien abgelegt werden. Der folgende Programmcode zeigt ein erstes Beispiel. Beim Laden der HTML Seite wird in einem PopUp Fenster der Text „Seite wurde geladen“ angezeigt.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>JavaScript</title>
</head>
<body onload="alert('Seite wurde geladen.');">
  <h1>JavaScript</h1>
</body>
</html>
```

Direkt nach dem Laden erscheint die folgende Meldung:



In diesem Beispiel wurde der JavaScript Code direkt dem EventHandler „onload“ im HTML <body> Element hinzugefügt. Dieser Code wird ausgeführt, sobald das HTML Dokument geladen wird. Weitere Events und deren Funktionalitäten werden später behandelt.

Umfangreicherer Code wird wie folgt dargestellt und im HTML Script Element abgelegt.

```
<script type="text/javascript">
  alert('Seite wurde geladen.');
</script>
```

Das <script> Element kann sich sowohl im <head> als auch im <body> Bereich eines HTML Dokumentes befinden. Als type muss „text/javascript“ angegeben werden. Mit dem <script> Element können auch externe JavaScript Dateien eingebunden werden. Man kann sich diese Anweisung als Inkludieren von JavaScript Dateien vorstellen.

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

Im „src“ Attribut kann eine Datei relativ oder mit einer URL referenziert werden.

3. Variablen und Arrays

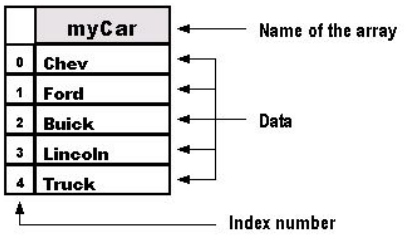
Variablen und Arrays sind nicht typisiert. Sie werden mit dem Schlüsselwort var deklariert. Auch wenn in JavaScript der Strichpunkt nicht zwingend ist, empfiehlt es sich, diesen konsequent immer zu machen.

Das folgende Skript zeigt die Deklaration von einfachen Variablen.

```
var s = "Hello World";
var i = 46;
var d = 46.46;
var l = s.length;
```

Arrays sind in JavaScript eine geordnete Sammlung von Werten. Jeder Wert wird Element bezeichnet und jedes Element hat eine nummerische Position im Array. Diese Position nennt man Index. Auch Arrays sind nicht typisiert. Das heisst, die Elemente in einem Array müssen nicht vom selben Typ sein. Der Index ist Nullbasiert. Das heisst, das erste Element hat die Position 0. Die Grösse eines Arrays muss nicht angegeben werden. JavaScript passt diese automatisch an.

Das folgende Bild verdeutlicht den Aufbau von Arrays.



Comparison of an array to a column of data

In JavaScript gibt es verschiedene Möglichkeiten, ein Array zu kreieren und mit Werten zu füllen. Das folgende Code Beispiel zeigt verschiedenen Methoden auf.

```
var myCars=new Array();
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";

var myCars=new Array("Saab","Volvo","BMW");

var myCars=["Saab","Volvo","BMW"];
```

Die drei Varianten sind semantisch äquivalent und haben je nach Verwendungszweck ihre Berechtigung. Die erste Variante ist wohl am gebräuchlichsten und entspricht auch der Syntax wie sie in zahlreichen anderen Programmiersprachen verwendet wird.

Auf ein Element wird mit der folgenden Syntax zugegriffen.

```
var car = myCars[1];
```

4. Spezielle Objekte

4.1 Strings

Text wird als String bezeichnet. Strings werden mit Hilfe einfacher oder doppelter Anführungszeichen deklariert. Mit der Deklaration wird ein String Objekt erzeugt. String Objekte sind nicht veränderbar. Jede Modifikation erzeugt ein neues Objekt. Die folgende Tabelle zeigt die wichtigsten Methoden, welche auf dem String Objekt definiert sind.

Methode	Beschreibung
charAt()	Gibt den Buchstaben an der angegebenen Stelle zurück.
charCodeAt()	Gibt den Unicode an der angegebenen Stelle zurück.
concat()	Verbindet zwei oder mehr Strings und gibt diese als neuen String zurück.
fromCharCode()	Konvertiert Unicode Codes in Buchstaben.
indexOf()	Gibt die Position vom ersten Vorkommen des angegebenen Strings zurück.
lastIndexOf()	Gibt die Position vom letzten Vorkommen des angegebenen Strings zurück.
match()	Gibt die Matches von einem Regular Expression zurück.
replace()	Ersetzt die Matches von einem Regular Expression mit dem Substring.
search()	Gibt die Position vom Match eines Regular Expressions zurück.
slice()	Gibt einen Substring zurück. Negative Werte für die Selektion von der rechten Seite.
split()	Splittet ein Array an der angegebenen Stelle und gibt ein Array zurück.
substr()	Gibt einen Substring zurück.
substring()	Gibt einen Substring zurück.
toLowerCase()	Konvertiert den String in Kleinbuchstaben.
toUpperCase()	Konvertiert den String in Grossbuchstaben.
valueOf()	Gibt den primitive Type vom String zurück.

Der folgende Programm Code zeigt einige Beispiele mit diesen Methoden.

```
var s = "Hello World";
var l = s.length;
// l == 11;
var substring = s.substring(6, 11);
//substring == "World"

var i = s.indexOf("lo");
// i == 3

var u = s.toUpperCase();
// u == "HELLO WORLD"
```

4.2 Math Objekt

Das Math Objekt stellt gebräuchliche mathematische Funktionen zur Verfügung und wird in den folgenden Beispielen verwendet. Die folgende Tabelle zeigt die wichtigsten Funktionen und deren Bedeutung.

Methode	Beschreibung
abs(x)	Gibt den absoluten Wert zurück.
acos(x)	Gibt den ArcCos zurück.
asin(x)	Gibt den ArcSin zurück.
atan(x)	Gibt den ArcTan zurück.
ceil(x)	Rundet auf zur nächsten Ganzzahl.
cos(x)	Gibt den Cosinus zurück.
exp(x)	Gibt den Exponent zurück.
floor(x)	Rundet zur nächsten Ganzzahl ab.
log(x)	Gibt den Logarithmus zurück.
max(x,y,z,...,n)	Gibt den grössten Wert zurück.
min(x,y,z,...,n)	Gibt den kleinsten Wert zurück.
pow(x,y)	Berechnet die Potenz.
random()	Gibt eine Zufallszahl zwischen 0 und 1 zurück.
round(x)	Rundet zur nächsten Ganzzahl.
sin(x)	Gibt den Sinus zurück.
sqrt(x)	Gibt die Quadratwurzel zurück.
tan(x)	Gibt den Tangens zurück.

Der folgende Programm Code zeigt einige Beispiele mit dem Math Objekt.

```
var res1 = Math.floor(1.22);
// res1 == 1

var res2 = Math.ceil(1.22);
// res2 == 2

var res3 = Math.floor(Math.random()*10);
// e.g. res3 == 4

var res4 = Math.max(1,2,3);
// res4 == 3

var res5 = Math.sqrt(256);
// res5 == 16
```

4.3 Date Objekt

Das Date Objekt gehört nicht zum Core von JavaScript wird aber von allen gängigen Browsern zur Verfügung gestellt. Damit lässt sich das aktuelle Datum und die Zeit in verschiedenen Formaten ermitteln. Die folgende Tabelle zeigt eine Übersicht der wichtigsten Methoden.

Methode	Beschreibung
getDate()	Monatstag ermitteln
getDay()	Wochentag ermitteln
getFullYear()	volles Jahr ermitteln
getHours()	Stundenteil der Uhrzeit ermitteln
getMilliseconds()	Millisekunden ermitteln
getMinutes()	Minutenteil der Uhrzeit ermitteln
getMonth()	Monat ermitteln
getSeconds()	Sekundenteil der Uhrzeit ermitteln
getTime()	Zeitpunkt ermitteln
getTimezoneOffset()	Zeitzonenabweichung der Lokalzeit ermitteln
getUTCDate()	Monatstag von UTC-Zeit ermitteln
getUTCFullYear()	volles Jahr von UTC-Zeit ermitteln
getUTCHours()	Stundenteil der UTC-Uhrzeit ermitteln
getUTCMilliseconds()	Millisekunden Teil der UTC-Uhrzeit ermitteln
getUTCMinutes()	Minutenteil der UTC-Uhrzeit ermitteln
getUTCMonth()	Monat von UTC-Uhrzeit ermitteln
getUTCSeconds()	Sekundenteil der UTC-Uhrzeit ermitteln
getYear()	Jahr ermitteln
parse()	Millisekunden seit dem 01.01.1970, 0:00:00 Uhr ermitteln
setDate()	Monatstag setzen
setFullYear()	volles Jahr setzen
setHours()	Stunden der Uhrzeit setzen
setMilliseconds()	Millisekunden setzen
setMinutes()	Minuten der Uhrzeit setzen
setMonth()	Monat setzen
setSeconds()	Sekunden der Uhrzeit setzen
setTime()	Datum und Uhrzeit setzen
setUTCDate()	Monatstag für UTC-Zeit setzen
setUTCFullYear()	volles Jahr für UTC-Zeit setzen
setUTCHours()	Stunden der UTC-Zeit setzen
setUTCMilliseconds()	Millisekunden der UTC-Zeit setzen
setUTCMinutes()	Minuten der UTC-Zeit setzen
setUTCMonth()	Monat für UTC-Zeit setzen
setUTCSeconds()	Sekunden der UTC-Zeit setzen
setYear()	Datum und Uhrzeit setzen
toGMTString()	Zeitpunkt in UTC-Format konvertieren
toLocaleString()	Zeitpunkt in lokales Format konvertieren
UTC()	UTC-Zeit seit dem 01.01.1970 ermitteln

5. Sprachelemente

5.1 Boolesche Ausdrücke und Logische Operatoren

Bedingungen werden als boolesche Ausdrücke formuliert. Ein Boolescher Ausdruck ergibt nach der Auswertung immer true (wahr) oder false (falsch). Verschiedene Ausdrücke können mit Logischen Operatoren verknüpft werden. Die wichtigsten Operatoren sind UND (&&), ODER (||) und die Negation (!).

Die Ausdrücke können beliebig komplex und verschachtelt werden. Es lohnt sich, die Ausdrücke für die einfachere Lesbarkeit zu klammern. Grundsätzlich gilt die Regel UND vor ODER analog zur Regel aus der Mathematik Punkt vor Strich. (Multiplikation vor Addition)

Die folgende Wahrheitstabelle zeigt die Verknüpfung mit UND und ODER.

a	b	UND (a && b)	ODER (a || b)
true	true	true	true
true	false	false	true
false	false	false	false
false	true	false	true

Die folgende Tabelle zeigt die Operatoren für den Vergleich von Variablen und Werten. Mehrere direkte Vergleiche können mit UND und ODER verknüpft werden.

Operator	Beschreibung
Gleich (==)	True, wenn beide Werte gleich sind.
Ungleich (!=)	True, wenn die beiden Werte nicht gleich sind.
Grösser (>)	True, wenn der erste Wert grösser ist als der zweite.
Grösser-Gleich (>=)	True, wenn der erste Wert grösser oder gleich ist wie der zweite.
Kleiner (<)	True, wenn der erste Wert kleiner ist als der zweite.
Kleiner-Gleich(<=)	True, wenn der erste Wert kleiner oder gleich ist wie der zweite.

5.2 Bedingungen

Bedingungen werden als IF Block realisiert. Dieser gibt es in 3 Ausprägungen. Die erste Variante enthält nur einen Konditionalen Block. Dieser wird ausgeführt, wenn die Bedingung wahr ist.

```
if (BEDINGUNG) {
  ...
}
```

Die zweite Variante enthält zwei Blöcke. Der erste wird ausgeführt, wenn die Bedingung wahr ist. Der zweite (else-Teil) wird ausgeführt, wenn die Bedingung falsch ist.

```
if (BEDINGUNG) {
  ...
} else {
```

```
  ...
}
```

In der dritten Variante werden mehrere IF Blöcke kombiniert. Dabei wird immer nur ein Block ausgeführt. Sind verschiedene Bedingungen wahr, so wird der erste Block ausgeführt.

```
if (BEDINGUNG) {
  ...
} else if (BEDINGUNG) {
  ...
} else {
  ...
}
```

5.3 Schlaufen

JavaScript kennt verschiedenen Schlaufen Typen. Die für den Moment wichtigsten sind die For-Schlaufe und die While-Schlaufe.

Der folgende Programm Code zeigt eine For-Schlaufe. Sie wird verwendet um den Kontrollfluss mit einem Zähler zu steuern. Die Zählvariable wird üblicherweise mit „i“ bezeichnet.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>JavaScript</title>
  </head>
  <body>
    <script type="text/javascript">
    var summe = 0;
    for(var i = 0; i <= 10; i++){
      summe = summe + i;
    }
    alert(summe);
    </script>
  </body>
</html>
```

Das Programm deklariert eine Variable mit dem Namen summe und weist dieser den Wert 0 zu. Danach kommt die For-Schlaufe. Diese besteht aus drei Teilen.

1. Deklaration der Laufvariable
2. Bedingung für den Schleifenabbruch
3. Modifikation der Variable

In unserem Beispiel wird in 1) die Variable i definiert und ihr den Wert 0 zugewiesen. In 2) wird die Abbruchbedingung definiert. Die Schlaufe soll so lange wiederholt werden bis der Wert von i grösser als 10 ist. In 3) wird der Wert der Variable jeweils um 1 erhöht.

In jedem Schleifendurchgang wird i zum aktuellen Wert von Summe dazu addiert.

Nachdem die Schlaufe 11 Mal (i=0,1,...,10) durchlaufen wurde, wird der Wert der Summe in einer Alert Box ausgegeben.

Dasselbe Programm kann auch mit einer While-Schlaufe realisiert werden.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>JavaScript</title>
  </head>
  <body >
    <script type="text/javascript">
    var summe = 0;
    var i = 0;
    while(i <= 10){
      summe = summe + i;
       i = i + 1;
    }
    alert(summe);
    </script>
  </body>
</html>
```

Die While-Schlaufe enthält nur die Abbruch Bedingung. Somit müssen wir für unser Beispiel im Vorfeld eine Zählervariable i deklarieren und diese in jedem Schleifendurchgang um 1 hochzählen.

Grundsätzlich kann jede For-Schlaufe durch eine While-Schlaufe substituiert werden. For-Schlaufen werden bevorzugt verwendet, wenn eine Zählvariable gebraucht wird.

Mit den beiden Schlüsselwörter continue und break kann das Verhalten der Schlaufe beeinflusst werden. Mit continue wird zum nächsten Schleifendurchgang gesprungen. Mit einem break wird die Schlaufe verlassen.

Im folgenden Programm wird in einer While-Schlaufe gewürfelt bis wir eine 6 haben. Danach wird ausgegeben, wie oft gewürfelt wurde.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>JavaScript</title>
</head>
<body >
        <script type="text/javascript">
        var anzahlWuerfe = 0;
        while(true){
                anzahlWuerfe++;
```

```
                var zufallsZahl = Math.ceil(Math.random() * 6);
                if(zufallsZahl == 6){
                        break;
                }
        }
        alert(anzahlWuerfe);
        </script>
</body>
</html>
```

Zu Beginn wird die Variable anzahlWuerfe deklariert und auf 0 gesetzt. Danach kommt die Schlaufendeklaration mit der Abbruchbedingung true. Somit erzeugen wir eine Endlosschlaufe.

In jedem Schleifendurchgang wird die Variable anzahlWuerfe um eins erhöht. Die Schreibweise anzahlWuerfe++ ist eine Kurzform von anzahlWuerfe = anzahlWuerfe + 1.

Danach wird eine Zufallszahl zwischen 1 bis und mit 6 erzeugt. Math.random gibt eine Zahl zwischen 0 und 1 zurück. Wir multiplizieren diese mit 6 und runden auf die nächste Ganzzahl auf.

Nun folgt die Fallunterscheidung. Wir prüfen, ob die Variable zufallsZahl den Wert 6 enthält. Ist diese Bedingung wahr, wird mit break die Schlaufe abgebrochen. In allen anderen Fällen geht es weiter zum nächsten Schleifendurchgang.

5.4 Switch

Ein weiteres Element für die Steuerung des Kontrollflusses von Programmen ist die Switch Anweisung. Es wird ein Wert übergeben, anschliessend werden verschiedene Fälle aufgelistet. Zusätzlich kann mit dem Schlüsselwort „default“ der Fall bezeichnet werden, bei dem alle Eingaben behandelt werden, die sonst nicht aufgeführt sind.

Das folgende Beispiel zeigt den Gebrauch der Switch Anweisung. Als Anweisung wird der aktuelle Wochentag als Nummer genommen. Für die Nummern 1 – 7 weisen wir die Wochentage zu. In allen anderen Fällen wird der Wert „unbekannt“ zugewiesen.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>JavaScript</title>
</head>
<body>
<script type="text/javascript">
  var wochentag = "";
  var date = new Date();
  switch (date.getDay()) {
  case 1:
    wochentag = "Montag";
    break;
  case 2:
```

```
    wochentag = "Dienstag";
    break;
       // usw.
    case o:
      wochentag = "Sonntag";
      break;
    default:
      wochentag = "unbekannt";
  }
  alert(wochentag);
</script>
</body>
</html>
```

Im Weiteren ist das Schlüsselwort „break“ zu beachten. Damit wird aus der Switch Anweisung gesprungen. Fehlt diese Anweisung, wird mit dem nächsten „case“ Block weiter gemacht.

5.5 Funktionen

Eine Funktion bündelt eine Menge von Anweisungen und kann Parameter und einen Rückgabewert haben. Funktionen werden mit dem Schlüsselwort „function“ deklariert. Das folgende Beispiel zeigt eine Funktion, welche eine Addition ausführt. Sie hat die beiden Parameter a und b und gibt das Resultat der Addition zurück.

```
function plus(a, b) {
  return a + b;
}
```

Eine Funktion kann einen Wert zurückgeben. Der Rückgabewert wird mit dem Schlüsselwort „return“ zurückgegeben.

Funktionen sind Objekte und können Variablen zugewiesen werden. Das folgende Beispiel zeigt eine abstrakte Mathematik Funktion. Die ersten beiden Parameter sind Zahlen. Der dritte Parameter ist eine Funktion. Die Mathematik Funktion führt die übergebene Funktion mit den beiden Zahlen aus und zeigt das Resultat in einer Alert-Box.

```
function mathe(a,b, func){
  alert(func(a,b));
}

var plus = function(a,b){
  return a + b;
}

var mulitply = function(a,b){
  return a * b;
}
```

```
mathe(5,4,plus);
mathe(5,6,mulitply);
```

Eine Funktion kann wiederum eine andere Funktionen oder auch sich selbst aufrufen. Das folgende Beispiel zeigt eine Rekursive Funktion (eine Funktion, die sich selbst aufruft), welche die Fakultät (http://de.wikipedia.org/wiki/Fakultät_(Mathematik))berechnet.

```
function fakultaet(x){
  if(x > 1){
    return x * fakultaet(x-1);
  }
  return 1;
}
```

5.6 Events

Events werden Aufgrund von bestimmten Aktionen gefeuert. Eine Aktion kann sowohl von einem Benutzer also auch durch eine Zustandsänderung vom System ausgelöst werden. Sie werden im HTML Code als Attribute im Tag aufgeführt. In Hochkommas können JavaScript Anweisungen ausgeführt werden. Die folgende Tabelle zeigt eine Übersicht der wichtigsten Events.

Event	Beschreibung
onblur	Der Event wird gefeuert, wenn ein Element den Fokus verliert.
onchange	Der Event wird gefeuert, wenn der Inhalt von einem Element oder die Selektion ändert.
onclick	Der Event wird gefeuert, wenn der Benutzer mit der Maus auf ein Element klickt.
ondblclick	Der Event wird gefeuert, wenn der Benutzer mit der Maus auf ein Element doppel-klickt.
onfocus	Der Event wird gefeuert, wenn ein Element den Fokus bekommt.
onkeydown	Der Event wird gefeuert, wenn eine Taste gedrückt wird.
onkeypress	Der Event wird gefeuert, wenn eine Taste gedrückt wird.
onkeyup	Der Event wird gefeuert, wenn eine Taste losgelassen wird.
onload	Der Event wird gefeuert, wenn ein Objekt geladen wurde.
onmousedown	Der Event wird gefeuert, wenn die Maustaste gedrückt ist.
onmousemove	Der Event wird gefeuert, wenn der Benutzer die Maus über ein Element bewegt.
onmouseout	Der Event wird gefeuert, wenn die Maus ein Element verlässt.
onmouseover	Der Event wird gefeuert, wenn die Maus über ein Element bewegt wird.
onmouseup	Der Event wird gefeuert, wenn die Maustaste losgelassen wird.
onselect	Der Event wird gefeuert, wenn Text selektiert wird.
onsubmit	Der Event wird gefeuert, wenn ein Formular abgeschickt wird.

Das folgende Beispiel löst den onload Event aus, sobald das HTML Dokument fertig geladen ist.

```
<body onload="alert('Seite wurde geladen.');">
        <h1>JavaScript</h1>
</body>
```

Das folgende Beispiel löst den onclick Event aus, wenn der Benutzer auf den Knopf klickt.

```
<input type="button" value="Check" onclick="raten();" />
```

Das folgende Beispiel zeigt die Deklaration eines „onsubmit“ Events in einem Formular. Das Formular wird nur abgeschickt, wenn der Rückgabewert der angegebenen Funktion wahr ist. Somit können die Felder im Formular Client-Seitig vor dem Verschicken validiert werden.

```
<form onsubmit="return validiereFormular();">
```

6. Formular Validierung

Der folgende Code zeigt die Validierung der beiden Input Felder Name und Vorname. Die Validierungsregeln verlangen, dass keines der beiden Felder leer sein darf.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Formular Validieren</title>
<script type="text/javascript">
  function validiereFormular() {
    var nameElement = document.getElementById("name");
    var vornameElement = document.getElementById("vorname");

    return validiereElement(nameElement, "Name muss ausgefuellt werden.")
        && validiereElement(vornameElement, "Vorname muss ausgefuellt werden.");
  }

  function validiereElement(element, text) {
    if (element.value == "") {
      alert(text);
      return false;
    }
    return true;
  }
</script>
</head>
<body>
  <h1>Formular Validierung</h1>
  <form onsubmit="return validiereFormular();">
    <p>
      Name: <input type="text" value="" id="name" /><br />
      Vorname: <input type="text" id="vorname" value="" /><br />
      <input type="submit" />
    </p>
  </form>
</body>
</html>
```

Die Input Elemente werden jeweils mit einer ID versehen. Mit dem „onsubmit“ Event wird die Funktion validiereFormular() aufgerufen. Darin werden die beiden Input Elemente geholt und mit der Property value auf den Inhalt zugegriffen. Danach wird geprüft, ob die Felder nicht leer sind.

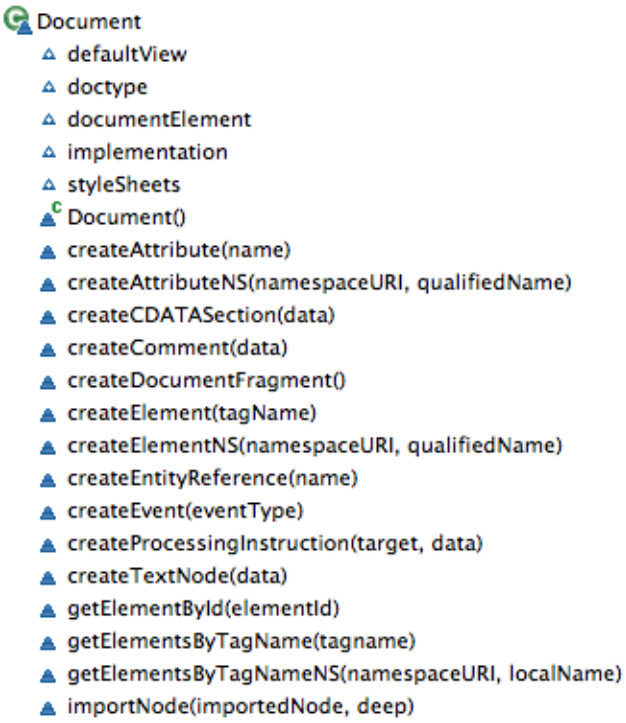
Die Funktion gibt true zurück, wenn beide Felder nicht leer sind. In allen anderen Fällen gibt die Funktion false zurück.

7. DOM

7.1 Zugriff auf HTML Elemente

JavaScript bietet verschiedene Methodem, um auf HTML Elemente im Document Object Model zuzugreifen.

Der Zugriff auf HTML Elemente erfolgt jeweils über die Variable document. Das Objekt Document enthält die folgenden Eigenschaften und Methoden:



Die performanteste Methode ist der Zugriff über die ID vom HTML Element. Moderne Browser erlauben dabei einen direkten Zugriff über die ID ohne dass der gesamte Document Object Model Baum traversiert werden muss.

Der folgende Code zeigt, wie auf ein Input Element über die ID zugegriffen und der Wert ausgelesen wird. Dafür wird die Methode getElementById verwendet. Anschliessend wird der Text in einer Alert-Box ausgegeben.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>JavaScript</title>
<script type="text/javascript">
  function zeigeText() {
    var inputElement = document.getElementById("eingabe");
    alert(inputElement.value);
```

```
  }
</script>
</head>
<body>
  <h1>JavaScript</h1>
  <p>
    <input type="text" id="eingabe" />
    <input type="button" value="Start" onclick="zeigeText();" />
  </p>
</body>
</html>
```

Eine weitere Methode ist der Zugriff über den Element Namen. So können beispielsweise alle P-Elemente in einem HTML Dokument geholt werden. Der Rückgabewert ist nicht mehr ein Element sondern eine Liste von HTML Elementen.

Der folgende Code ausschnitt zeigt eine Methode, welche einen Element Namen (z.B. P) entgegen nimmt und in einer Alert-Box die Anzahl Tags im Dokument ausgibt. Dafür wird die Methode getElementByTagName verwendet.

```
function countElements(elementName) {
  var nodeList = document.getElementsByTagName(elementName);
  alert(nodeList.length);
}
```

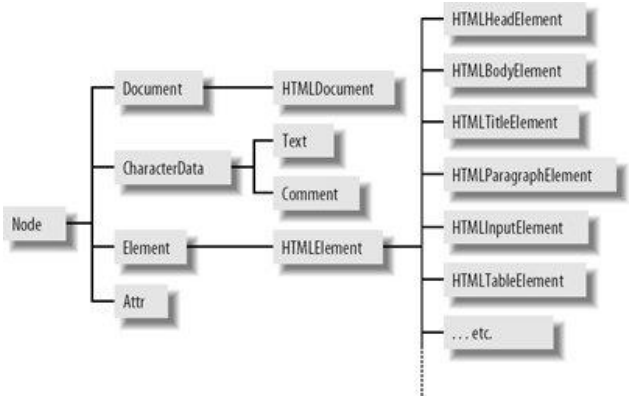
Analog dazu gibt es die beiden Methoden getElementByClassName, welche eine CSS Class entgegen nimmt und querySelectorAll, welche einen beliebigen CSS Selektor entgegennimmt.

Der folgende Code zeigt die Verwendung der Methode querySelectorAll.

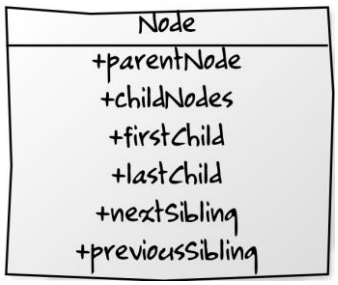
```
function countElements(elementName) {
  var nodeList = document.querySelectorAll(elementName);
  alert(nodeList.length);
}

countElements("#abc");
countElements("#abc > li");
```

7.2 Navigation im DOM

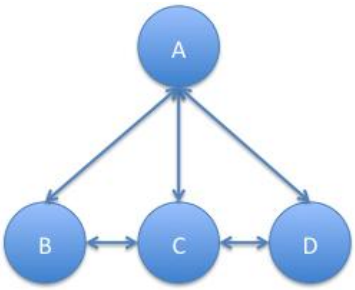
Für jedes HTML Element gibt es eine JavaScript Objekt welches das Element repräsentiert. Grundsätzlich sind diese Objekte wie im folgenden Bild dargestellt vom Objekt Node abgeleitet.



Für die Navigation werden die Properties auf dem Node Objekt verwendet. Das folgende Bild zeigt die relevanten Properties (Eigenschaften) auf dem Node Objekt.



Betrachten wir den folgenden Ausschnitt aus einem DOM.



Vom Knoten A können B (firstChild) und D (lastChild) direkt besucht werden. Die Knoten B, C, D sind in der Liste childNodes enthalten. Von den Knoten B, C und D kommt man jeweils über parentNode zum Knoten A. In C kann B über previousSibling, bzw. D über nextSibling erreicht werden.

7.3 Modifikation vom DOM

Der DOM kann nicht nur besucht, sondern auch modifiziert werden. Die folgende Tabelle zeigt die wichtigsten Methoden, um Knoten hinzuzufügen und zu entfernen.

Methode	**Beschreibung**

insertBefore(newChild, refChild)	Fügt einen Knoten vor dem angegebenen Knoten ein.
replaceChild(newChild, oldChild)	Ersetzt den angegeben Knoten.
removeChild(oldChild)	Entfernt den angegebenen Knoten.
appendChild(newChild)	Fügt einen Knoten am Ende dazu.
hasChildNodes()	Wahr, wenn der Knoten Kinder hat.
innerHTML(html)	Mit dieser Methode kann kompletter HTML Code an einen Knoten gehängt werden. Sämtliche Kinder werden damit überschrieben.
hasAttributes()	Wahr, wenn der Knoten Attribute hat.

Das folgende Code Beispiel enthält eine leere Liste. Mit jedem Knopfdruck wird der Liste ein Element am Ende dazu gefügt.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>JavaScript</title>
<script type="text/javascript">
  function neuerEintrag() {
    var ul = document.getElementById("list");
    var li = document.createElement("li");
    var textNode = document.createTextNode("ein Listen Eintrag");
    li.appendChild(textNode);
    ul.appendChild(li);
  }
</script>
</head>
<body>
<h1>JavaScript</h1>
<p>
  <input type="button" value="Start" onclick="neuerEintrag();" />
</p>
<ul id="list">
</ul>
</body>
</html>
```

Jedes Mal wenn auf den Knopf geklickt wird, wird zuerst die Liste mit der ID „list“ geholt. Danach wird ein neues Listenelement und ein neuer Text Knoten erzeugt. Der Text Knoten wird dem Listenelement dazu gefügt, das Listenelement der Liste.

Das folgende Code Beispiel traversiert das Komplette HTML Dokument und erzeugt eine verschachtelte Liste mit allen Tag Namen, die im HTML Dokument vorkommen.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>JavaScript</title>
<script type="text/javascript">
  function traverse(parent) {
    function rec(ul, parent) {
      if (parent.hasChildNodes()) {
        for (var i = 0; i < parent.childNodes.length; i++) {
          var child = parent.childNodes[i];
          var childUl = document.createElement("ul");
          var li = createLiNode(child.nodeName);
          rec(childUl, child);
          ul.appendChild(li);
          li.appendChild(childUl);
        }
      }
    }
    var ul = document.createElement("ul");
    rec(ul, parent);
    var originalUl = document.getElementById("list");
    originalUl.parentNode.replaceChild(ul, originalUl);
    ul.setAttribute("id", "list")
  }

  function createLiNode(nodeName) {
    var li = document.createElement("li");
    var textNode = document.createTextNode(nodeName);
    li.appendChild(textNode);
    return li;
  }
</script>
</head>
<body>
<h1>JavaScript</h1>
<p>
  <input type="button" value="Start" onclick="traverse(document.body);" />
  <span><span><span><span></span></span></span></span>
</p>
<ul id="list">
</ul>
</body>
</html>
```

Für die Traversierung wird die rekursive Methode rec aufgerufen. Damit werden jeweils alle Kinder besucht. Für jedes Kind wird diese Methode erneut aufgerufen.

7.4 Modifikation von CSS Regeln

Mit Hilfe von JavaScript können die CSS Regeln von einem Element modifiziert werden. Auf jedem HTML Element gibt es die Property style, welche die einzelnen CSS Regeln enthält. Dabei werden dieselben Attribute unterstützt wie wenn die CSS Regeln direkt deklariert werden. Wichtig ist, dass die Teilworte der CSS Attribute nicht mit Bindestrichen zusammengefügt werden sondern mit Hilfe der Camel Case Notation. Das bedeutet, dass jedes Teilwort mit einem Grossbuchstaben beginnt.

Die folgende Tabelle zeigt die JavaScript Namen für die wichtigsten CSS Attribute.

JavaScript-Angabe	Kurzbeschreibung
background	Hintergrundbild
backgroundColor	Hintergrundfarbe
backgroundImage	Hintergrundbild
backgroundRepeat	Wallpaper-Effekt
border	Rahmen
borderBottom	Rahmen unten
borderBottomColor	Rahmenfarbe unten
borderBottomStyle	Rahmenart unten
borderBottomWidth	Rahmendicke unten
borderColor	Rahmenfarbe
borderLeft	Rahmen links
borderLeftColor	Rahmenfarbe links
borderLeftStyle	Rahmenart links
borderLeftWidth	Rahmendicke links
borderRight	Rahmen rechts
borderRightColor	Rahmenfarbe rechts
borderRightStyle	Rahmenart rechts
borderRightWidth	Rahmendicke rechts
borderStyle	Rahmenart
borderTop	Rahmen oben
borderTopColor	Rahmenfarbe oben
borderTopStyle	Rahmenart oben
borderTopWidth	Rahmendicke oben
borderWidth	Rahmendicke
bottom	Position von unten
clear	Fortsetzung bei Textumfluss
color	Textfarbe
cursor	Mauszeiger
display	Sichtbarkeit (ohne Platzhalter)
cssFloat	Textumfluss
font	Schrift
fontFamily	Schriftart
fontSize	Schriftgröße
fontStyle	Schriftstil
fontWeight	Schriftgewicht
height	Höhe eines Elements
left	Position von links
letterSpacing	Zeichenabstand
lineHeight	Zeilenhöhe
listStyle	Listendarstellung
listStyleImage	Grafik für Aufzählungslisten
listStylePosition	Listeneinrückung
listStyleType	Darstellungstyp der Liste
margin	Abstand/Rand
marginBottom	Abstand/Rand unten
marginLeft	Abstand/Rand links
marginRight	Abstand/Rand rechts
marginTop	Abstand/Rand oben
maxHeight	Maximalhöhe
maxWidth	Maximalbreite
minHeight	Mindesthöhe
minWidth	Mindestbreite
overflow	übergroßer Inhalt
padding	Innenabstand
paddingBottom	Innenabstand unten
paddingLeft	Innenabstand links
paddingRight	Innenabstand rechts

paddingTop	Innenabstand oben
pageBreakAfter	Seitenumbruch danach
pageBreakBefore	Seitenumbruch davor
position	Positionsart
right	Position von rechts
textAlign	Ausrichtung
textDecoration	Textdekoration
textIndent	Texteinrückung
textTransform	Text-Transformation
top	Position von oben
verticalAlign	vertikale Ausrichtung
visibility	Sichtbarkeit (mit Platzhalter)
width	Breite eines Elements
wordSpacing	Wortabstand
zIndex	Schichtposition bei Überlappung

Das folgende Beispiel vergrössert mit jedem Knopfdruck die Schriftgrösse vom Titel.

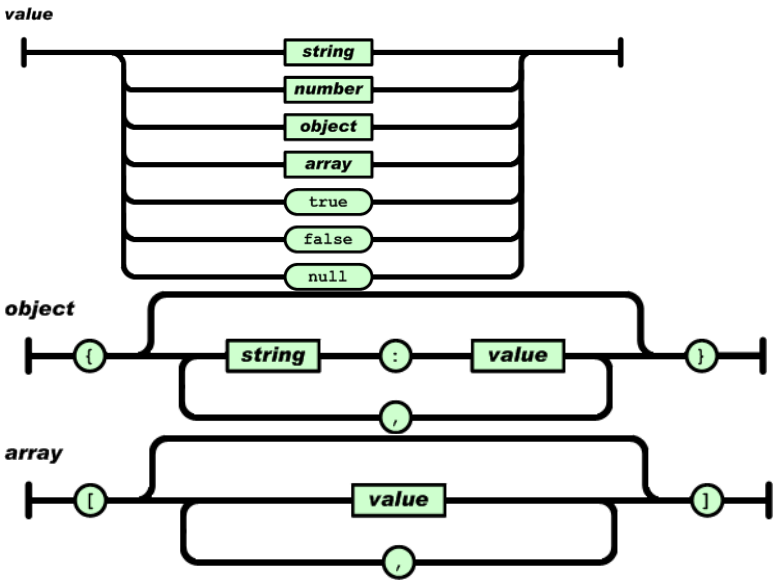
```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>JavaScript</title>
<script type="text/javascript">
  var groesse = 16;
  function aendereSchriftgroesse(element, size) {
    groesse = groesse + 2;
    var titel = document.getElementById('titel')
    titel.style.fontSize = groesse + "px";
  }
</script>
</head>
<body>
  <h1 id='titel' style="font-size: 16px;">JavaScript</h1>
  <p>
    <input type="button" value="Klick" onclick="aendereSchriftgroesse();" />
  </p>
  <ul id="list">
  </ul>
</body>
</html>
```

Dem Titel wird mit dem CSS Attribut font-size die Grösse 16 zugewiesen. Wird auf den Knopf gelickt, wird die Grösse um 2 erhöht. Das Titel Element wird geladen und das Attribut fontSize (in Camel Case) auf dem Style Objekt wird neu gesetzt.

8. JSON Datenaustauschformat

JSON steht für JavaScript Object Notation und ist ein leichtgewichtiges Datenaustauschformat welches aus den Literalen der JavaScript Sprache besteht. JSON ist von Mensch und Maschine gut lesbar und inzwischen gibt es für alle aktuellen Programmiersprachen Bibliotheken für den Import und Export. Das Datenaustauschformat ist unter www.json.org definiert.

JSON besteht aus den 6 Typen Strings, Zahlen, Booleans, Objekte, Listen und Null. Die folgenden Grafiken definieren die Syntax von JSON.



Das folgende Stück Code zeigt die Definition einer Person in einem einzelnen Objekt.

```
var person = {
  "name": "Kant",
  "vorname": "Immanuel",
  "geburtstag": "22.04.1724"
}

alert(person.name);
```

In geschweiften Klammern wird ein Objekt mit den Eigenschaften name, vorname und geburtstag erzeugt. Dieses Objekt wird der Variable person zugewiesen. Der rechte Teil der Zuweisung entspricht der JSON Darstellung. Auf die Eigenschaften von einem Objekt kann über die Punktnotation zugegriffen werden.

Das folgende Beispiel zeigt eine Liste mit 2 Objekten. Grundsätzlich müssen die Listenelemente nicht vom selben Typ sein.

```
var personen = [{
    "name": "Kant",
    "vorname" : "Immanuel"
```

```
  }, {
    "name" : "Schopenhauer",
    "vorname" : "Arthur",
}];

alert(person[0].name);
alert(person[1].name);
```

In eckigen Klammern wird eine Liste (Array) erzeugt. Die einzelnen Elemente in der Liste sind in diesem Fall Objekte, welche Personen darstellen. Zugewiesen wird die JSON Datenstruktur der Variable personen. Auf die Listenelemente wird über den Index zugegriffen. Auf die Objekte wird mit der Punktnotation zugegriffen.

Wird JSON als Datenaustauschformat verwendet, so wird ein JSON String verschickt oder empfangen. Dieser muss beim empfangen geparst werden. Dafür gibt es in JavaScript mehrere Möglichkeiten.

Das folgende Beispiel verwendet die eval Methode von JavaScript, mit der zur Laufzeit beliebiger JavaScript Code ausgeführt werden kann.

```
var jsonString = '{"name": "Kant", "vorname":"Immanuel"}';
var person = eval('('+jsonString+')');
alert(person.name);
```

Diese Methode funktioniert in jedem Browser welcher JavaScript unterstützt. Der Nachteil ist, dass damit ein erhebliches Sicherheitsrisiko besteht. Der JSON String könnte beliebige andere JavaScript Anweisungen enthalten. Diese werden ohne Kontrolle ausgeführt. Diese Methode eignet sich somit nur, wenn man der Datenquelle Vertrauen kann.

Eine zweite Möglichkeit ist die Verwendung der JSON Bibliothek. Auf aktuellen Browsern ist diese ein fester Bestandteil. Die Bibliothek kann aber auch von der JSON Webseite (www.json.org) heruntergeladen und in der Webseite eingebunden werden. Die Bibliothek stellt sicher, dass nur JSON Code interpretiert wird.

Das folgende Beispiel zeigt das Parsen eines JSON Strings mit der JSON Bibliothek.

```
var s = '{"name": "Kant", "vorname":"Immanuel"}';
var person = JSON.parse(s);
alert(person.name);
```

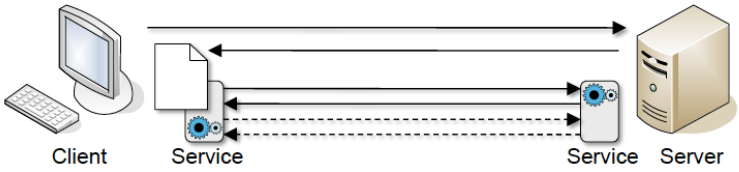
Die folgende Datenstruktur zeigt die Zugsverbindungen. Sie enthält generelle Informationen und die einzelnen Teilverbindungen, welche als Liste der Eigenschaft connectionParts zugewiesen werden.

```
{
  "from" : "Bern",
  "to" : "Zürich HB",
  "depDate" : "09.02.2012 04:21",
  "arrDate" : "09.02.2012 06:00",
  "products" : "IR, S3",
```

```
  "changes" : "1",
  "connectionParts" : [
  { "from" : "Bern",
    "to" : "Lenzburg",
    "depDate" : "09.02.2012 04:21",
    "arrDate" : "09.02.2012 05:04",
    "depPlatform" : "5",
    "arrPlatform" : "2",
    "product" : "IR 1903"
  },
  { "from" : "Lenzburg",
    "to" : "Z\u00fcrich\nHB",
    "depDate" : "09.02.2012 05:24",
    "arrDate" : "09.02.2012 06:00",
    "depPlatform" : "2",
    "arrPlatform" : "23\/24",
    "product" : "S3 18315
  }]
}
```

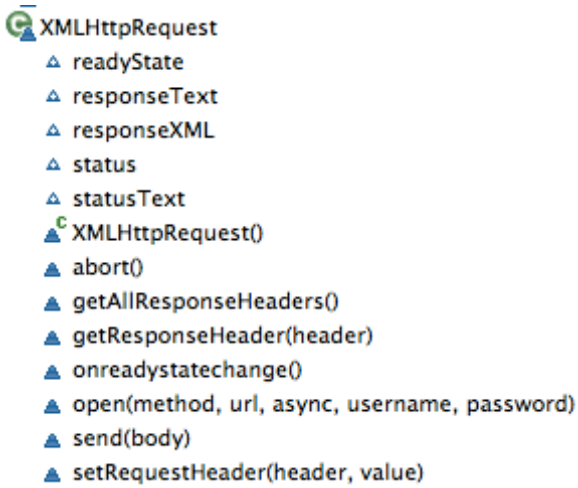
9. AJAX

AJAX steht für Asynchronous JavaScript and XML. Es ist keine Technologie sondern eine Anwendung von HTML, CSS und JavaScript und partielle Update von Webseiten zu realisieren. Das bedeutet, dass nach dem Laden einer Webseite gewisse Teile davon ersetzt oder ergänzt werde ohne dass die Seite neu geladen wird. AJAX ist nicht auf XML beschränkt. Heutzutage wird für den Datentransfer oft JSON verwendet.



Der Client lädt eine Seite. Danach werden mittels JavaScript ein oder mehrere Services auf einem Server aufgerufen. Dazu wird wiederum eine HTTP Verbindung verwendet. Die Daten werden vom Client empfangen, geparst und auf der Webseite visualisiert.

Aktuelle Browser stellen das XMLHttpRequest Objekt zur Verfügung. Damit können HTTP Anfragen an den Ursprungshost (der Host, von dem der JavaScript Code geladen wurde) gesendet werden. Die folgende Grafik zeigt die Eigenschaften und Methoden von diesem Objekt.



Für einen Request wird das folgende Code Stück verwendet. Es wird erwartet, dass das empfangene Dokument im JSON Format gesendet wird.

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
  if (this.readyState !== 4) {
    return false;
  }
  if (this.status !== 200) {
    alert("Error...”); return false;
  }
```

```
  var result = eval('(' + xhr.responseText + ')');
};
xhr.open('GET', 'data.json');
xhr.send('');
```

Zuerst wird eine neue Instanz vom XMLHttpRequest Objekt erzeugt. Danach wird der Eigenschaft onreadystatechange eine Funktion zugewiesen. Diese wird immer ausgeführt, wenn sich der ReadyState der Anfrage ändert. Dieser Status kann die folgenden Werte annehmen.

Konstante	Bedeutung
USENT (0)	Open() wurde nicht aufgerufen
OPENED (1)	Open() wurde aufgerufen
HEADERS_RECEIVED (2)	Header Daten wurden empfangen
LOADING (3)	Die Body Daten werden empfangen
DONE (4)	Die Antwort ist komplett

Für uns ist der Status Done (4) von Interesse. Wenn die Daten komplett geladen wurden, sollen sie interpretiert und verarbeitet werden. Deshalb wird als erstes geprüft, ob der readyState 4 ist. Ist dem nicht so, wird aus der Methode gesprungen. Danach wird geprüft, ob die HTTP Anfrage erfolgreich war. Ist dies der Fall, wird der JSON String mit der eval Methode geparst und der Variable result zugewiesen. Nun könnten die Daten in dieser Methode mittels JavaScript im HTML Dokument visualisiert werden.

Mit der open Methode wird angegeben, wohin und mit welcher HTTP Methode die Anfrage geschickt werden soll. Mit der Methode send wird die Anfrage verschickt.

Abfragen mit dem XMLHttpRequest Objekt funktionieren nur mit echten HTTP Abfragen. Das heisst, dass für die lokalen Tests ein Webserver verwendet werden muss. Ein sehr einfacher Webserver ist Mongoose (http://code.google.com/p/mongoose/). Mongoose gibt es für verschiedene Plattformen. Am einfachsten wird die ausführbare Datei in dasselbe Verzeichnis gelegt wo auch die HTML Dateien sind. Sobald der Webserver gestartet wird, sind die HTML Seiten über die URL http://localhost:8080 erreichbar. Weitere Informationen zu Mongoose sind auf dessen Webseite zu finden.

Das folgende Beispiel zeigt eine Webseite, welche auf Knopfdruck Städte aus einem JSON Dokument lädt und in einer Liste anzeigt. Das JSON Dokument mit den Städten hat die folgende Struktur:

```
[{
        "name":"Luzern",
        "land":"Schweiz",
        "code":"ch"
}, {
        "name":"Bern",
        "land":"Schweiz",
        "code":"ch"
},{
        "name":"Chur",
        "land":"Schweiz",
```

```
        "code":"ch"
},{
        "name":"Zug",
        "land":"Schweiz",
        "code":"ch"
}, {
        "name":"Berlin",
        "land":"Deutschland",
        "code":"de"
}]
```

Die Webseite lädt dieses Dokument und zeigt es in einer Liste an. Der folgende Code zeigt die Webseite.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>JavaScript</title>
<script type="text/javascript">
  function ladeStaedte() {
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function() {
      if (this.readyState !== 4) {
        return false;
      }
      if (this.status !== 200) {
        alert("Fehler beim Laden vom JSON Dokument.");
        return false;
      }
      var staedte = JSON.parse(xhr.responseText);
      staedteHinzufuegen(staedte);
    }
    xhr.open('GET', 'staedte.json');
    xhr.send('');
  }

  function staedteHinzufuegen(staedte) {
    var ul = document.getElementById("staedte");
    for ( var i = 0; i < staedte.length; i++) {
      var stadt = staedte[i];
      var li = document.createElement("li");
      var textNode = document.createTextNode(stadt.name);
      li.appendChild(textNode);
      ul.appendChild(li);
    }
  }
</script>
```

```
</head>
<body>
  <h1>JavaScript</h1>
  <ul id="staedte">
  </ul>
  <p>
    <input type="button" value="Laden" onclick="ladeStaedte();" />
  </p>
</body>
</html>
```

Wird der Knopf gedrückt, wird ein XMLHttpRequest Objekt zusammengestellt. Abgefragt wird das JSON Dokument staedte.json. Das Dokument wird geparst und an die Methode staedteHinzufügen übergeben. Diese Methode fügt pro Stadt ein Eintrag zur Liste hinzu.

10. Referenzen

Die folgende Liste gibt eine Übersicht von JavaScript Büchern. Für Fortgeschrittene empfehle ich dringend die Lektüre von 4), 5) und 6). Wer einen einfachen Einstieg mag, dem hilft sicher 3) weiter.

1. Selfhtml – JavaScript/DOM, Deutsch, Stefan Münz
   http://de.selfhtml.org/javascript/

2. JavaScript Tutoria, Englisch, W3Schools,
   http://www.w3schools.com/js/

3. Head First JavaScript, English/Deutsch, O'Reilly
   http://headfirstlabs.com/books/hfjs/

4. JavaScript: The definitive Guide, Englisch, O'Reilly
   http://shop.oreilly.com/product/9780596000486.do

5. JavaScript: The good Parts, Englisch, O'Reilly
   http://shop.oreilly.com/product/9780596517748.do

6. JavaScript Patterns, Englisch, O'Reilly
   http://shop.oreilly.com/product/9780596806767.do

7. Eloquent JavaScript - A Modern Introduction to Programming, Englisch, Marijn Haverbeke
   http://eloquentjavascript.net/

8. JavaScript, Deutsch, Galileo OpenBooks
   http://openbook.galileocomputing.de/javascript/index.htm

Eigene Notizen

Eigene Notizen

Kontaktieren Sie uns. Wir beraten Sie gerne individuell zu Ihrer Ausbildung am Bildungszentrum Uster.

Höhere Berufsbildung Uster
Berufsschulstrasse 1
8610 Uster
Telefon +41 44 943 64 22
info@hbu.ch
www.hbu.ch

Ein Angebot der Berufsfachschule Uster und der Höheren Fachschule Uster