Einführung in JScript (JavaScript)

Inhaltsverzeichnis

Einführung in JScript für HTML-Seiten

1. Allgemeines über JScript:

1.2. Was ist JavaScript?

JavaScript ist eine *Script-Sprache* mit dem Zweck, u. a. Datenmanipulationen innerhalb von Webseiten clientseitig zu ermöglichen. Sie ist eine Script-Sprache, die hauptsächlich für das *DOM-Scripting* in Web-Browsern eingesetzt wird. Script-Sprachen sind Programmiersprachen, die meist nur für kleinere Anwendungen verwendet werden. Scripts werden vielfach zur *Laufzeit* von einem *Interpreter* ausgeführt. Solche Anwendungen sind in der Regel *HTML-Anwendungen*.

1.3. Was kann JavaScript?

- ✓ JavaScript gibt HTML-Designern ein Programmierwerkzeug.
- ✓ JavaScript kann dynamisch Text in HTML-Seiten einfügen.
- ✓ JavaScript kann auf Ereignisse reagieren und HTML-Elemente lesen und schreiben.
- ✓ JavaScript kann für Eingabevalidierung herangezogen werden.
- ✓ JavaScript kann den verwendeten Browser bestimmen.
- ✓ JavaScript kann Cookies generieren.

2. Einbinden von JavaScript / JScript in eine HTML- / ASPX-Seite

2.1. Erstellen einer HTML-Anwendung

Im Visual Studio kann auch eine reine HTML-Anwendung erstellt werden (es gibt möglicherweise mehrere Wege, hier sei dieser anhand folgenden Beispiels angeführt). Die Anwendung (Abb. 1) besteht aus einem Text <h2>, einer Text-Box und einem Schalter *Text ausgeben*. Die Betätigung des Schalters gibt den Inhalt der Text-Box in einer Message-Box (Abb. 2) aus.

Abb. 1: HTML-Anwendung

Abb. 2: Ausgabe des Textes mittels Msg-Box

Abb. 3: Bei Start der Anwendung.

2.1.1. Erstellen einer HTML-Anwendung im Visual Studio

Es wird erwartet, dass der Schüler die Eigenschaften der HTML-Tags (Input, TextArea, Image, Select, etc.) eigenständig erarbeitet!

1. *Erstellen der HTML-Datei:* Starte das Visual Studio und wähle aus der rechten Spalte die Zeile: *Continue without code* → Es erscheint nun ein leeres Projekt. Weiter mit *File – New –File:* Wähle aus der Palette eine *HTML Page*. Die HTML-Page erscheint nun in der IDE unter dem Namen *HTMLPage1.html*. Im Solution Explorer ist sie unter *Solution 'Solution1'* vermerkt. Dieser Name kann ebenso verändert werden, z.B.: *StartPage!*
2. *Speichern der HTML-Datei:* Ändern wir diese HTML-Datei durch Einfügen von *HTML-Tags* etc., muss vor dem Start diese Seite gespeichert werden: *File → Save StartPage As …* . Wähle dazu einen entsprechenden Speicherort und gib der Seite einen sinnvollen Namen, beispielsweise *Home.html*.
3. *Starten der Seite:* Die Seite wird durch *Drücken der rechten Maustaste* und Auswahl *„View in Browser"* im Browser aufgerufen – alternativ dazu *„Ctrl + Shift + W"*.

4. *Implementieren der Funktionalität:* Füge entsprechend Abb. 1 eine Überschrift <h2>, eine Textbox und einen Button hinzu. Ändere ihre *IDs* auf eine sinngebende Bezeichnung und ebenso ihre *values* (siehe eingerahmten blauen Kasten).

```
<head>
  <title>Exa01 Juli 12</title>
  <script type="text/javascript">
    alert("Hallo HTBLA Kaindorf");
    function onGibTextAus(text){
      alert(text);
    }
  </script>
</head>
```

```
<body>
  <form action="Home.html">
    <h2>Ausgabe eines Textes in einer Message-Box</h2>
    <input id="hTbIO" type="text" value="Testmessage" dir="rtl" /><br/>
    <input id="hBtMess" type="button" value="Text ausgeben"
           onclick="onGibTextAus(form.hTbIO.value)" />
  </form>
</body>
```

Erklärungen:

- ✓ Das Attribut `action` des `<form>`-Tags nennt die HTML-Seite, deren Inhalt des `<form>`-Tags ausgeführt wird, hier also von der Seite *Home.html*. Dieses Attribut ist nicht unbedingt notwendig; wesentlich aber, wenn es mehrere Seiten gibt.
- ✓ Die Attribute im HTML-Tag `<input id="hBtMess" type="button" value="Text ausgeben" onclick="onGibTextAus …>` bedeuten:
  - o `type`: Bezeichnet eine Reihe von Controls, die gewählt werden können (hier ein Button, sonst Textfeld darüber, siehe Toolbox!).
  - o `value`: Beschriftung des Buttons, Text im Feld, etc.
  - o `onclick`: HTML-Event. Nennt die aufzurufende Methode des *onClick*-Ereignisses. Dieser Event kann im `<head>`-Teil implementiert werden
  - o Sämtliche Tags innerhalb dieser `<form>` können über ihren `id.value` bzw. `name.value` angesprochen werden: **`form.hTbIO.value`**
  - o Weitere Attribute sind `name`, `class`, etc. sowie verschiedene Methoden (siehe unten).

2.1.2. Implementieren der Java-Script Funktionen

Der 2. Teil betrifft die Funktionalität der Seite an sich. Durch Drücken des Schalters soll die Information aus der Text-Box in einer Message-Box ausgegeben werden (Abb. 2). Dazu verwenden wir das HTML-`onclick`-Ereignis, das die **Java-Script**-Funktion `onGibTextAus(…)` aufruft.

Events in HTML-Tags bedienen sich der Java-Script-Funktionen. D.h. wir benötigen einen Teil in der Seite, die eindeutig als Java-Script-Sektion ausgewiesen ist. Der Tag

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

weist diesen Teil als Java-Script-Sektion aus, die eigene Variablen, Funktionen und Objekte verwaltet und auf die die HTML-Tags in der `<body>`-Sektion zugreifen können. Daneben gibt es sogenannte *Java-Script Popup Boxen* (`alert, confirm, prompt` – siehe unten), die im Java-Script-Teil auch direkt aufgerufen werden können. Alle anderen, selbstdefinierten Methoden haben das Schlüsselwort `function` davor stehen und werden über Events oder andere Funktionen aufgerufen. Parameter haben **keinen Typ**!

Erklärungen zum Beispiel unten:

- ✓ Mit den Funktionen **`alert(), confirm()`** können Texte in den *Popup-Boxen* ausgegeben werden.

✓ Mit HTML-Tag `<input id="hBtMess" type="button" value="Text ausgeben"`
`onclick="onGibTextAus(form.hTbIO.value)"`
wird eine Funktion namens `onGibTextAus(…)` aufgerufen. Die aufzurufende Methode kann Parameter übergeben (muss aber nicht!).

Funktionsparameter sind nicht statisch typisiert. In der Funktion soll daher geprüft werden, von welchem Typ die Variable ist.

```
<script type="text/javascript">
  alert("Hallo Students …");        // Aufruf beim Öffnen der Seite
  confirm("TEST CONFIRM");          // Aufruf beim Öffnen der Seite

  function onGibTextAus(text){
    alert(text);                    // Aufruf über die Eventfunktion
  }
</script>
```

2.2. Auslagern von Java-Script in eine externe Datei

In der Regel schreibt man JScript-Code in eine eigene Datei. Diese hat die Endung ***.js*** und bindet diese Datei in die HTML-Datei ein. Dazu gehe in den *Solution Explorer*,

rechter Mausklick auf die Solution – Add – New Item – Text File/JavaScript File. Ändere den Namen und den Typ dieser Datei auf Extern.js → Add. Damit ist die JScript-Datei im Explorer sichtbar. Ab VS 2012 kann man automatisch eine JScript-Datei auswählen!

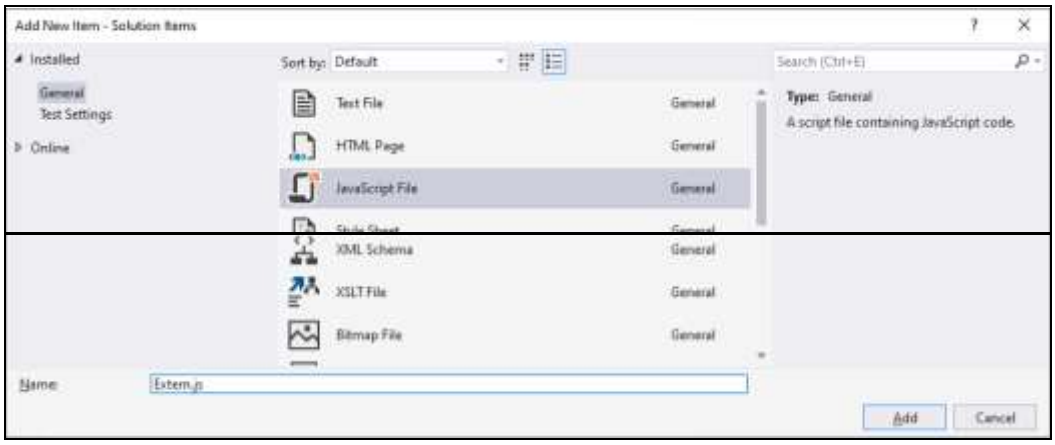

Abb. 4: Erstellen einer externen JScript-Datei

Das Einbinden des JScript-Files erfolgt meist im `<head>`-Teil einer HTML Seite mit dem Attribut **`src`**:

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

Inhalt der externen JScript-Date *Extern.js*:

```
// JavaScript source code
alert("Hallo Dear Students At Ingenium Education");
var isOK = confirm("I agree");
if (isOK == true) {
    function onGibTextAus(text) {
        alert(text);
    }
}
```

Die Definition von Funktionen erfolgt wie im Beispiel dargestellt. Übergabeparameter werden, wie in C#, in die runden Klammern geschrieben, haben auch hier keinen Datentyp!

3. Grundlagen der Sprache:

3.1. Deklaration einer Variable

Variablen werden in JavaScript mit dem Schlüsselwort **`var`** ***dynamisch typisiert.*** Hier werden ein Zahlenwert und ein String vereinbart:

```
var x = 12;
var strName = "someValue";
```

```
function showMessage()
{
    var strName = "someValue";
    alert("Dies ist eine externe Funktion : " + strName);
}
```

3.1.1. Eigenschaften:

- ✓ Wird eine Variable in einer Funktion deklariert, so ist sie nur in dieser Funktion sichtbar (lokale Variable).
- ✓ Wird eine Variable außerhalb einer Funktion deklariert, so ist sie in allen Funktionen verwendbar (globale Variable).
- ✓ Eine Variable, die innerhalb einer Funktion ohne das Schlüsselwort **`var`** vereinbart ist, hat ebenfalls globale Gültigkeit.
- ✓ Die Lebenszeit einer Variablen startet mit ihrer Deklaration und endet, wenn die Seite geschlossen wird.
- ✓ Variablen in JScript sind *dynamisch typisiert!* Z.B.: **`var x = 1; var str = "String";`** Deshalb sollen Parameter von Funktionen auf Typ überprüft werden. ZB. **`if (typeof x == „String")`**

3.2. JavaScript Popup-Boxen:

3.2.1. Alert-Box:

Ausgabe in einem eigenen „Popup".

```
alert("Eine Meldung");
```

3.2.2. Confirm-Box:

Muss mit "OK" oder "Abbrechen" bestätigt werden. OK ergibt *true*, Abbrechen ergibt *false*.

```
var result = confirm("Eine Meldung")
```

3.2.3. Prompt-Box:

Muss mit "OK" oder "Abbrechen" bestätigt werden, nachdem ein String eingegeben wurde. Wird abgebrochen, ist der Return-Wert **`null`**!

```
var result = prompt("Aufforderung","Defaultwert");
```

3.3. JavaScript Kontroll-Strukturen:

Achtung: Bei der `for`-Schleife muss die Laufvariable vor der Schleife vereinbart werden. Der Startwert in der Schleife beginnt mit einer einfachen Zuweisung!

Verzweigungen:

- ✓ `if, if-else`
- ✓ `switch – case`
- ✓ Bedingungen werden folgendermaßen spezifiziert:

```
var bedingung = x=="String"
```

Schleifen:

- ✓ `for`
- ✓ `while`

3.4 Felder in JavaScript (Arrays)

`Arrays` sind Instanzen der Klasse `Array` und erlauben folgende Initialisierungen bzw. Deklarationen. Folgende Initialisierungen sind möglich:

```
var myNames = new Array("Franz", "Hans", "Kurt", "Linda");
var arr = [1,2,3]: Initialisierung eines Feldes mit 3 Elementen
var arr = [1,,,,5]: Initialisierung eines Feldes mit leeren Elementen
```

Ein Feldelement kann Daten beliebigen Typs enthalten. Es kann auch selbst wieder Felder enthalten. Im Folgenden besteht das 2. Sub-Array (Teilfeld) aus String und Zahlen:

```
var cats = [ ["Names", "John", "Pun", "Max"], ["Ages", 16, 15, 14]];
```

`Arrays` sind *dynamisch* und *typenlos*. Mit der Eigenschaft `.length` kann die Länge festgestellt werden. Wichtige Methoden sind:

- ✓ `push()`: Hängt ein neues Element an das Ende des Feldes und gibt die neue Feldlänge zurück.
- ✓ `pop()`: Löscht das letzte Element aus dem Feld und gibt es zurück. Wenn das Feld leer ist, wird `undefined` zurückgegeben.
- ✓ `shift()`: Löscht das erste Element aus dem Feld und gibt es zurück. Wenn das Feld leer ist, wird `undefined` zurückgegeben.
- ✓ `toString()`: Gibt das Feld als String zurück.
- ✓ `splice(index, anz)`: Löscht Feldelemente ab Index `index` die folgenden `anz` Elemente.
- ✓ `forEach(function())`: For-each-Schleife über alle Elemente des Feldes, wobei die genannte Funktion ausgeführt wird.

Referenz: SelfHtml – Array bzw. MSDN-Hilfe(!)

Beispiel *Felder:* Die Anwendung in Abb. 5 besteht aus einem Textfeld, einem Select-Control und den 3 Schaltern *„Feld in ComboBox“, „Füge Element hinzu“, „Lösche Element“.* Mit dem *Start der Anwendung* soll ein Feld mit 3 Zufallszahlen zwischen 0 und 10 erstellt werden. Weitere Funktionalitäten sind über Schalter zu implementieren:

- ✓ *Feld in DropDownList:* Überträgt das Feld in das Select-Control (DropDownList = ComboBox) Abb. 6.
- ✓ *Füge Element hinzu:* Liest aus dem Textfeld den Wert („Test“) und fügt ihn dem Feld hinzu.
- ✓ *Lösche Element:* Löscht das letzte Element aus der Liste und gibt es in dem Textfeld aus.

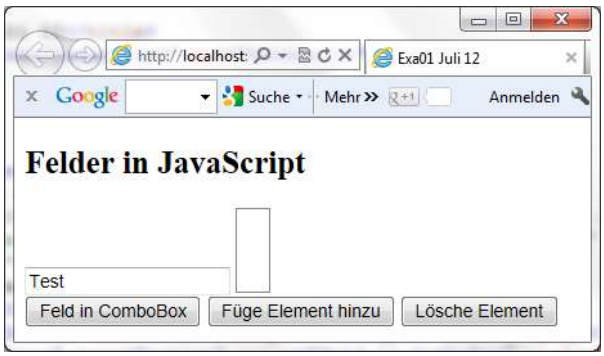

Abb. 5: Beispiel für Felder

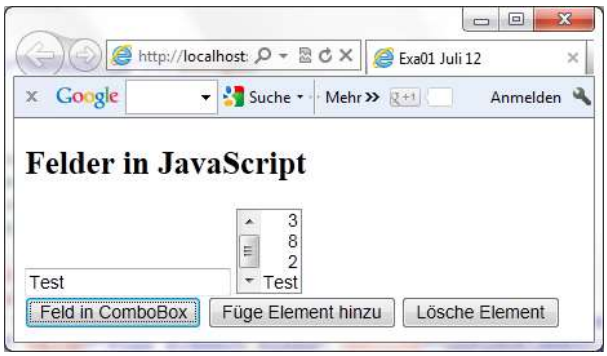

Abb. 6: Feld in das Select-Control

3.4.1. Spezielle Java-Script-Funktionen

Folgende Funktionen haben sich als nützlich herausgestellt:

- ✓ `Number(text):` Wandelt eine String-Zahl in eine Zahl um.
- ✓ `parseInt(zahl.toString(), 10):` Wandelt eine Zahl in eine Int-Zahl um.
- ✓ `replace(/[^a-zA-Z0-9 ]/g, ""):` Ersetzt alle Zeichen mit Ausnahme der hier genannten „a"- „z", „A" bis „Z", 0 bis 9 durch ein „Keinzeichen". Den Klammerausdruck nennet man *regular Expression*.
- ✓ `localStorage:` Damit können Daten von einer Seite auf die andere übertragen werden.
  Achtung: Ist ein HTML5-Feature und daher nicht bei allen Bowsern enthalten und ausführbar.
  - `localStorage.setItem("feld", feld):` Die Information `feld` wird auf den localStorage namens `"feld"` gespeichert.
  - `var feld = localStorage.getItem("feld"):` Die Information wird aus dem localStorage auf `feld` ausgelesen.
- ✓ `window.top.location`: Liest die Url aus.
- ✓ `isNan(<wert>):` Gibt an, ob der Wert <wert> eine Zahl ist oder nicht!
- ✓ `replace(".", ","):` Ersetzt den Punkt durch das Komma
- ✓ `toFixed(2):` Prints the number with 2 Kommas.
- ✓ `split(`x`):` Splittet den String nach `x` und gibt ein Feld zurück.
- ✓ `splice(4,3):` Löscht aus dem Feld ab dem 4. Element die folgenden 2.
- ✓ `parsInt(String s):` Wandelt String in int um
- ✓ `parseFloat(String s):` Wandelt String in float um.
- ✓ `remove(index):` Eine selektierte Option von einem select-Control wird folgendermaßen entfernt:

```
var optElem = document.getElementById("selTN");  // selTN = select-Control-ID
optElem.remove(indM);
```

HTML-Code

Im Folgenden ist der HTML-Code dargestellt. Die Eventmethoden beinhalten teilweise das Control, auf das im Java-Script-Code zugegriffen wird.

```
<h2>Felder in JavaScript</h2>
<input id="hTxIOID" type="text" value="Test" />
<select id="hSlDataID" dir="rtl" multiple="multiple" size="50px"></select><br />
<input id="hBtTrans" type="button" value="Feld in ComboBox"
       onclick="onFeldInsComboBox(form.hSlDataID)" />
<input id="hBtPushID" type="button" value="Füge Element hinzu"
       onclick="onPushElement(form.hTxIOID.value)" />
<input id="hBtPopID" type="button" value="Lösche Element"
       onclick="onPopElement(form.hTxIOID)" />
```

3.4.2. Implementieren der Java-Script Funktionen

Windows-Funktionen wie `window.onload` stehen zu Beginn der Java-Script-Datei und werden automatisch ausgeführt. Hier wird beim Laden der Datei eine Methode für die Erstellung eines Feldes mit 3 Werten aufgerufen. `feld` ist globale Variable!

```
window.onload = onErstelleFeld(3);
var feld;
```

Gleiches liefert der Aufruf der `onload`-Funktion im `<body onload ="onErstelleFeld(3)">` !

Weitere **window**-Methoden sind:

`onunload, onerror, onfocus, onresize, onscroll`

3.4.2.1. Erstellen von Zufallszahlen

Das Erstellen von Zufallszahlen kann man mit dem Zeitobjekt `Date()` steuern. Dieses Objekt bietet verschiedene Methoden, die man nutzen kann.

Die folgende Funktion ermittelt `anzahl` Zufallszahlen zwischen 0 und 10 (Eigenanalyse!)

```
function onErstelleFeld(anzahl) {
    feld = new Array();
    var i = 0;
    for (i = 0; i < anzahl; i++) {
        var basisZufZ = new Date().getMilliseconds();
        var zufZ = basisZufZ % 11;
        feld[i] = zufZ;
        // sleep() verzögert die Zeit um 5 Millisekunden für die Berechnung
        // eines neuen Zufallswertes.
        sleep(5); // oder setTimeout(5);
        // alternativ zu oben mit einem Zufallszahlengenerator
        var myInt = parseInt((Math.random()*90 + 10).toString(), 10);
    }
}
```

Die folgende Funktion `sleep()` implementiert eine Zeitverzögerung um `sleepTime` Millisekunden (Eigenanalyse!). Alternativ kann auch `setTimeout()` verwendet werden.

```
// Zeitfunktion für die Erstellung von Zufallszahlen
function sleep(sleepTime) {
    var aktTime = new Date().getTime();
    var testTime = new Date().getTime();
    while (aktTime < sleepTime + testTime) {
      aktTime = new Date().getTime();
    }
}
```

3.4.2.2. Feldoperationen

In der folgenden Funktion wird das letzte Element aus dem Feld entfernt und in das Quadrat dieser Zahl in das Textfeld ausgegeben. Für die Methoden der Feldoperationen ist hier beispielhaft die Methode onPopElement(wert) angegeben und erklärt. Die übrigen sind selbsterklärend!

`cntrl` als Übergabeparameter ist das Textfeld in der Form!

`isNan(value)` prüft, ob das Element eine Zahl ist. Wenn nicht, gibt diese Funktion `false` zurück, sonst `true`!

```
// Feldelement löschen
function onPopElement(cntrl) {
    var elem = feld.pop();
    if (!isNaN(elem)) {
        elem = elem * elem;
        cntrl.value = elem;
    }
}
```

4. HTML-Controls

Hier sind in aller Kürze nun die wichtigsten HTML-Controls vorgestellt. Weitere Informationen sind der Hilfe über *SelfHTML* zu entnehmen!

`<select>`: Wird verwendet, um eine drop-down List zu erzeugen. Der `<option>`-Tag innerhalb eines `<select>`-Elements definiert die Optionen in der Liste. Der Zugriff auf ein `<option>`-Tag erfolgt über das `<form>`-Tag und deren Attribute name bzw. id des `<select>`-Tags:

`form.selectID.options[index].text:` Wert an der Stelle “index” im Select

`form.selectID.selectedIndex:` Index an der selektierten Option

`var` `op = document.createElement("option"):` Erstellt neues Option-Element

`form.selectID.options.add(op):` Fügt neues Option-Element an das Select-Control.

`form.selectedID.options.length=null:` Löscht alle Einträge in dem Select-Control.
`form.selectedID.options[index]=null:` Löscht das Element an der Stelle *index*.

`<textarea>:` Erzeugt ein Textfenster für lange Texte mit mehreren Zeilen in einem Formular. Breite und Höhe werden durch Attribute gesteuert.

`form.txtAreaID.value += feld[i] + "\n";:` Damit hängt man Text an einen bereits bestehenden Text in einer `<textarea>`.

`<input type="reset" …/>:` **Reset**-Button:: Setzt die Form in den Ausgangszustand zurück.
`<input type="text" …/>:` **Input**-Button:: Mit `id.value = „text“` wird der Wert hinzugefügt.

`<input type="submit" …/>:` **Submit**-Button:: Mit dem Absende-Button kann der Anwender das ausgefüllte Formular losschicken. Mit den Formulardaten geschieht dann das, was im einleitenden `<form>`-Tag mit dem Attribut `action` und `onsubmit` festgelegt wurde.

Im folgenden Beispiel soll die Url der anzusurfenden Page in einer TextArea angezeigt werden. Abb. 7 zeigt die Page, die mittels Submit aufgerufen wurde.

`<input type="radio" …/>:` **checked**:: prüft, ob der Schalter aktiviert ist.

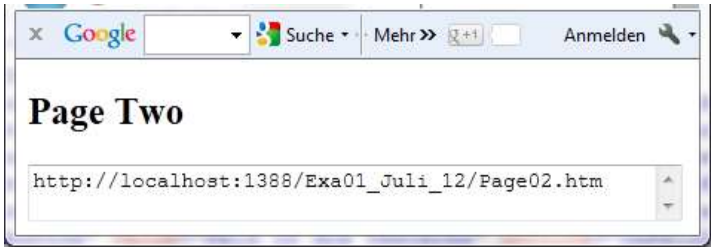

Abb. 7: Angesurfte Seite “Page Two”

Folgende Aktionen sind durchzuführen:

1. Definition des Submit-Buttons: Dieser Button löst nur die Submit-Funktion *validate()* im `<form>`-Tag aus:

```
<input id="submitID" type="submit" value="Textarea -> Page02.htm" />
```

2. Parameter im `<form>`-Tag spezifizieren: Damit werden die Parameter action für die Target-Page und die Funktion onsubmit für die auszuführende Funktion validate() definiert:

```
<form action="Page02.htm" onsubmit="return validate()">
```

3. Implementierung der JavaScript-Funktion validate(): Die Funktion true oder false zurück. Ist der Rückgabewert true, wird die neue Page angesteuert, sonst nicht!

```
// Falls im Feld ein alpha-numerische Zahl enthalten ist, wird
// true zurückgegeben, sonst false
function validate() {
    var i = 0;    var submitOK = false;
    for (i = 0; i < feld.length; i++) {
        if (isNaN(feld[i])) {
            submitOK = true;   break;
        }
    }
    return submitOK; }
```

4. Target-Page HTML: Im <body>-Tag wird die Funktion onFillData() aufgerufen. Sie führt die gewünschte Aufgabe beim Ansurfen dieser Page aus. Sie enthält die TextArea für die Ausgabe der Informationen (=Url).

```
<body onload="onFillData()" >
  <form id="form2" action="Page01.htm" >
    <h2>Page Two</h2>
    <textarea id="hTaGetDataID" cols="50" rows="2"></textarea>
    <div id="hDvGetDataID"></div>
  </form>
</body>
```

5. *Target-Page Java-Script:* Im <body>-Tag wird die Funktion onFillData() aufgerufen. Sie führt die gewünschte Aufgabe beim Ansurfen dieser Page aus. Sie enthält die TextArea für die Ausgabe der Informationen (=Url).

```
function onFillData() {
    var elem = form2.hTaGetDataID;
    var queryString = window.top.location;
    elem.value = queryString;
}
```

Beispiel *Schalter:* In der Anwendung Abb. 8 sollen die oben genannten Schalteroperationen durchgeführt werden. Sie besteht aus einem Textfeld, einer Select (DropDownList), einer TextArea für die IO, weiters einer Reihe von Schaltern und einem Image. Im Einzelnen:

Abb. 8: GUI zu Beispiel 2

1. Start der Anwendung: Beim Start der Anwendung soll ein Feld mit Zufallszahlen zwischen -10 und + 10 erstellt werden. Dieses Feld steht in der Folge zur Verfügung. Die Zufallszahlen werden dabei noch nicht angezeigt!

2. Feld in ComboBox: Schreibt die Zahlen in die DropDownListe (=ComboBox)

3. Füge Element hinzu: Liest den Wert „Test“ aus dem Textfeld und fügt des dem Feld hinzu. Kontrolle durch Ausgabe des Feldes in einer Message-Box.

4. Lösche Element: Löscht das Feldelement an der letzten Stelle im Feld. Kontrolle durch Ausgabe des Feldes in einer Message-Box.

5. Lösche erstes Element: Löscht das erste Feldelement aus dem Feld. Kontrolle durch Ausgabe des Feldes in einer Message-Box.

6. Lese selektiertes Element: Selektiere ein Element in der DropDownListe und gib es in der Message-Box aus.

7. Feld in die TextArea: Schreibe das Feld in die TextArea, wobei die Elemente jeweils in einer eigenen Zeile stehen müssen!

8. Reset Formular: Was passiert beim Drücken dieses Schalters?

9. TextArea -> Page02.htm: Wechsle zur 2. Seite dieser Anwendung und gib den Inhalt des TextArea dort aus!

6. Exception-Handling in JavaScript:

Exception-Handling wird in JavaScript ähnlich abgewickelt wie in C#. Fehler können mit `throw` geworfen werden. Weitere Informationen sind zu entnehmen aus der Referenz: SelfHtml - try/catch bzw. MSDN-Hilfe.

Beispiel *Exception:* In der Anwendung Abb. 9 sind Zufallszahlen zwischen -10 und +10 zu erstellen und in ein Feld der Länge 10 zu speichern. Jeder ungerade Feldindex enthalte keinen Wert (Abb. 9).

Abb. 9: Anwendung zu Beispiel 3

Abb. 10: Werte < 0

Abb. 11: undefinierte Werte

1. Start der Anwendung: Beim Start der Anwendung soll ein Feld der Größe 10 mit Zufallszahlen zwischen -10 und + 10 erstellt werden, wobei die ungeraden Feldwerte leer bleiben sollen. Die Zufallszahlen werden dabei noch nicht angezeigt!

2. Feld in die DDL: Das Feld soll in die DDL geschrieben werden (Abb. 9).

3. Lese nur positive Zahlen: Liest die Feldwerte. Tritt eine Zahl < 0 auf, soll die Exception nach Abb. 10 geworfen werden, falls der Wert undefiniert ist, soll eine Exception nach Abb. 11 geworfen werden. Ist die Zahl positiv, soll diese Quadriert und eine Ausgabe in der Message-Box erfolgend: „Quadratur: 16“ z.B.

Ein Auszug aus der JavaScript-Funktion:

```
try {
  if (feld[i] < 0) throw feld[i] + " kleiner Null";
  var x = feld[i] * feld[i];
  if (isNaN(x)) throw feld[i] + " ist keine Zahl";
  alert("Quadratur: " + x);
}
catch (ex) {
  alert(ex);
}
```

Wie man in Abb. 9 erkennen kann, ist das 4. Element in der Liste selektiert und der Schalter Feld in DDL hat den Fokus. Das wird erreicht, indem nach Ablauf obiger Exception-Behandlung einfach die Methoden focus() und das Attribut selectedIndex gesetzt wurden:

```
form.hBtDisplayID.focus();
form.hSlDataID.selectedIndex = 3;
```

Was oben form.xxx betrifft, siehe dazu den nächsten Punkt!

Weitere Anmerkungen zur HTML-Seite: Dazu betrachten wir die HTML-Seite

```
<body onload="beimLaden()">
  <form>
    <h2>Exception Handling in JavaScript</h2>
    <select id="hSlDataID" size="10" ></select>
    <input id="hBtDisplayID" type="button" value="Feld in die DDL"
           onclick="onFeldInDieDDL(form.hSlDataID)" /><br />
    <input id="hBtExeptionID" type="button" value="Lese nur positive
           Zahlen" onclick="onZahlPositivFehler(form)" />
  </form>
</body>
```

1. *Parameter* **`form:`** Der `<form>`-Tag kann als Parameter übergeben werden. Damit wird ermöglicht, dass sämtliche Tags innerhalb dieser `<form>` über ihren `namen.value/text` oder `id.value/text` angesprochen werden können: `form.hBtDisplayID`

2. *Methoden* **`onfocus()`** und **`onselect():`** Diese Methoden beziehen sich ebenfalls auf den Namen eines Tags und fokusieren diesen bzw. selektieren den Wert in diesem Tag. Vorzugsweise werden hier Control-Tags angesprochen wie beispielsweise Textfelder, Buttons, Dropdownlisten, etc.

7. JavaScript Events:

Jedes Element einer Webseite unterstützt verschiedene Events, die den Aufruf eines Eventhandlers auslösen können. Mit Hilfe von Attributen der HTML-Tags können Events und Eventhandler definiert werden. Beispiele für Events sind:

- ✓ Mausklick
- ✓ Laden einer Webseite oder eines Bildes
- ✓ Die Maus betritt ein Element
- ✓ Absenden eines HTML-Formulars
- ✓ Tastatur Event

6.1. Übersicht über JavaScript Events

Die gebräuchlichsten sind:

- ✓ ***onload*** - Ein Ladevorgang wurde abgeschlossen: Das `onload`-Event wird ausgelöst, wenn der Benutzer eine Seite betritt. Es wird oft verwendet, um den Browsertyp zu erfragen und die angemessene Version der Webseite zu laden. Ebenso leistet es Dienste beim Verwalten von Cookies, wenn ein Benutzer die Seite betritt.
- ✓ ***onunload*** - Der Benutzer verlässt die Seite: Das `onunload`-Events wird ausgelöst, wenn der Benutzer eine Seite verlässt. Es wird oft verwendet, um Cookies zu verwalten, wenn ein Benutzer die Seite verlässt.
- ✓ ***onsubmit*** - Der Submit-Button eines Formulars wurde geklickt: Das `onsubmit`-Event wird verwendet um alle Felder eines HTML-Formulares zu validieren, bevor es gesendet wird.
- ✓ ***onblur*** - Ein Element verliert den Focus: Wird oft verwendet, um Benutzereingaben zu validieren.
- ✓ ***onchange*** - Der Inhalt eines Feldes verändert sich.
- ✓ ***onclick*** - Mausklick auf ein Element: Wird oft verwendet, um Benutzereingaben zu validieren.

- ✓ ***onfocus*** - Ein Element erhält den Focus: Wird oft verwendet, um Benutzereingaben zu validieren.
- ✓ ***onmousemove*** - Die Maus wird bewegt

Übersicht über JavaScript Events – hier die seltener verwendeten:

- ✓ ***onabort*** - Ein Ladevorgang wird unterbrochen
- ✓ ***ondblclick*** - Doppelklick auf ein Element
- ✓ ***onerror*** - Fehler bei einem Ladevorgang
- ✓ ***onkeydown*** - Eine Tastaturtaste wird gedrückt
- ✓ ***onkeypress*** - Eine Tastaturtaste ist gedrückt
- ✓ ***onkeyup*** - Eine Tastaturtaste wird losgelassen
- ✓ ***onmousedown*** - Ein Mausbutton wird gedrückt
- ✓ ***onmouseout*** - Die Maus verlässt ein Element
- ✓ ***onmouseover*** - Die Maus bewegt sich über einem Element
- ✓ ***onmouseup*** - Ein Mausbutton wird losgelassen
- ✓ ***onreset*** - Der Resetbutton eines Formulars wurde geklickt
- ✓ ***onresize*** - Die Größe eines Fensters oder Rahmens wurde verändert

4. Beispiel *Events:* In dieser Anwendung Abb. 9 sind nur einige wenige Events implementiert. Die Anwendung besteht aus einem Image, einer DDL, einem Textfeld und 2 Schaltern.

sind Zufallszahlen zwischen -10 und +10 zu erstellen und in ein Feld der Länge 10 zu speichern. Jeder ungerade Feldindex enthalte keinen Wert (Abb. 9). Das folgende Beispiel verwendet die Funktion `checkEmail()` immer dann, wenn der Benutzer den Inhalt des Eingabefeldes ändert:

Abb. 10: Anwendung zu Beispiel 4

Abb. 11: Durchgeführte Events

1. Start der Anwendung: Beim Start der Anwendung soll ein Feld der Größe 10 mit Zufallszahlen zwischen -10 und + 10 erstellt werden. Die Zufallszahlen werden dabei noch nicht angezeigt!

```
<body onload="beimLaden()">
…
</body>
```

JavaScript: Diese Funktion ist von oben übernommen.

```
function beimLaden() {
    feld = new Array();
    var i = 0;
    for (i = 0; i < 10; i++) {
        var zufZ = new Date().getMilliseconds();
        zufZ = (zufZ % 21) - 10;
        feld[i] = zufZ;
        sleep(5);
    }
```

2. Feld in die DDL – `onclick()`: Das Feld soll in die DDL geschrieben werden (Abb. 11).

```
<input id="hBtDisplayID" type="button" value="Feld in die DDL"
       onclick="onFeldInDieDDL(form.hSlDataID)" />
```

JavaScript: Diese Funktion ist von oben übernommen. Schreibt das Feld in die DDL:

```
function onFeldInDieDDL(elem) {
    var i = 0;
    for (i = 0; i < feld.length; i++) {
        var opt = document.createElement("option");
        elem.add(opt);
        opt.text = feld[i];
    }
}
```

3. Selektieren einer Zahl im DDL – `onchange()`: Nach dem Selektieren eines Feldwertes aus der DDL schreibt dieser Event die Zahl in das Textfeld (Abb. 11).

```
<select id="hSlDataID" size="10"
        onchange="onSelektierterWert(form.hSlDataID, form.hTxIO01)" >
</select>
```

JavaScript: Die JavaScript-Funktion hat hier 2 Parameter. Sie übernimmt einen selektierten Wert und schreibt ihn in das Textfeld.

```
function onSelektierterWert(form, elem) {
    var wert = form.options[form.selectedIndex].value;
    elem.value = wert;
}
```

4. Klick in das Textfeld – `onfocus()`: Durch einen Klick in das Textfeld wird die Summe der Werte aus dem Feld errechnet und in das Textfeld geschrieben. Das Ergebnis dieses Events ist nicht abgebildet.

```
<input id="hTxIO01" type="text"
    onfocus="onAddiere(this, 'addition')" />
```

JavaScript: Die JavaScript-Funktion hat hier 2 Parameter. Sie addiert alle Elemente des Feldes und gibt die Summe aus. Falls addition nicht gewählt wurde, bleibt das Feld frei!

```
function onAddiere(elem, name) {
    var sum;
    if (name == "addition") {
        sum = 0; var i = 0;
        for (i = 0; i < feld.length; i++) {
            sum += feld[i];         }
        elem.value = sum;
    } }
```

5. Feld ins DIV – `onclick()`: Schreibt das Feld in den `Div`-Tag (Abb. 11).

```
<input id="hBtSelWertID" type="button" value="Feld ins Div"
       onclick="onFeldInsDiv(form.divID)" />

<div id="divID"> </div>
```

JavaScript: Ausgabe des Feldes im `Div`-Tag. Ist nur mittels DOM möglich!

```
function onFeldInsDiv(elem) {
    var div = document.getElementById("divID");
    div.innerHTML = feld;
}
```

6. *Imageaktion* – `onmouseover()`: Bewegt man die Maus auf das Bild, erscheint ein zufällig anders Bild aus der einer Reihe von 5 Bildern. Die Bilder sind in diesem Fall im selben Verzeichnis wie die Anwendung (Abb. 11).

```
    <img id="bildID" alt="Testbild" src="bild1.jpg" width="150px"
         height="150px" onmouseover="onMoveBild(this.bildID)" />
```

JavaScript: Im Aufruf steht das Argument this.bildID. this bezieht sich hier auf den <form>-Tag und funktioniert nur so(?). Die Auswahl erfolgt zufällig aus einer Menge von 5 Bildern. Achtung: Der Zugriff auf das Bild-Attribut src ist nur über das DOM möglich!

```
function onMoveBild(elem) {
    var wert = new Date().getMilliseconds();
    wert = wert % 5 + 1;
    var bild = "bild" + wert + ".jpg";
    document.getElementById("bildID").setAttribute("src", bild);
}
```

7. JavaScript Objekte:

JavaScript unterstützt die Programmierung von Objekten. Objekte werden aber nicht in Form von Klassen repräsentiert, sondern in Form von Funktionen. Ein JavaScript-Objekt hat Eigenschaften und Methoden. Beispielsweise hat ein String-Objekt die Eigenschaft **`length`** und die Methode **`toUpperCase()`**:

```
function beimLaden() {
    var string = "Hallo 5. Klasse";
    alert("Laenge von String: " + string.length);
    alert("UpperCase von String: " + string.toUpperCase());
}
```

JavaScript besitzt unter anderem die folgenden eingebauten Objekte (nähere Informationen unter http://de.selfhtml.org/javascript/objekte/index.htm):

- ✓ String
- ✓ Date (siehe Zufallszahlen)
- ✓ Array (siehe oben)
- ✓ Boolean
- ✓ Math (siehe oben)

Zum Erzeugen eigener Objekte gibt es zwei Möglichkeiten:

- ✓ Erzeugen einer direkten Instanz von `Object`
- ✓ Erzeugen eines *Templates*

7.1. Instanz von Object

Objekte sind fest umgrenzte Datenelemente mit Eigenschaften und oft auch mit objektgebundenen Funktionen (Methoden).

5. Beispiel *Instanz als Objekt:* In dieser Anwendung Abb. 12 wird das Objekt Toni Gigler, 10. 3. 2000 erstellt und mit dem 2. Schalter ausgegeben (Abb. 13).

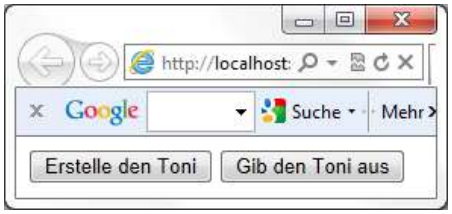

Abb. 12: Anwendung zu Beispiel 5

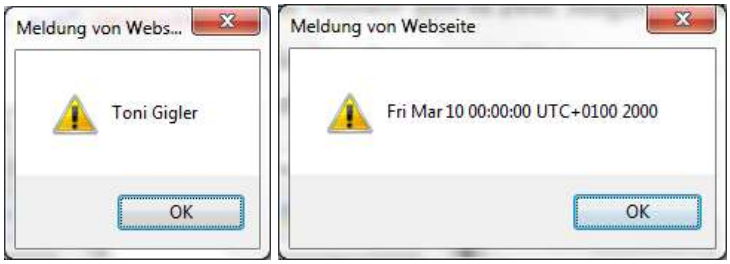

Abb. 13: Ausgabe des Toni

In der Java-Script-Datei definieren wir eine globale Variable person. Diese wird in der ersten Methode erstellt und in der 2. Methode ausgegeben

```
var person;

function onErstelleToni() {
    person = new Object();
    person.firstName = "Toni";
    person.lastName = "Gigler";
    person.birthDate = new Date("March 10, 2000");
    person.methode = myMethode; // hinzufügen der Methode myMethode
}
```

7.2. Instanz von Template

Ein Template ist nichts anderes als eine Funktion. In ihr definiert man die Membervariablen wie die Memberfunktionen.

6. Beispiel *Instanz als Template:* In dieser Anwendung wird das Objekt Toni Gigler, 10. 3. 2000 erstellt und über eine Funktion ausgegeben.

✓ Beim Laden wird das Objekt erstellt

```
function beimLaden() {
    Person("Toni", "Gigler", new Date("April 10, 1990"));
}
```

✓ Die Objektdefinition

```
function Person(vorname, nachname, alter) {
    this.vorname = vorname;
    this.nachname = nachname;
    this.alter = alter;
    this.showMe = zeigen; // Methoden sind JavaScript Funktionen
}
```

✓ Die Memberfunktion des Templates

```
function zeigen() {
    alert("Hi - I'm " + this.vorname + " " + this.nachname
      + "\nand I'm born on " + this.alter);
}
```

✓ Ausgabe der Person

```
function onGibToni() {
    alert(Person.showMe);
}
```