Moderne Webtechnologien I (WT 1)
Übung: JavaScript

WiSe 23/24

Prof. Dr. C. Köhn
Daniela Böing

24. Oktober 2023

Website:
https://www.hochschule-bochum.de/fbe/fachgebiete/institut-fuer-informatik/labor-fuer-medienkommunikation-internet-und-robotik/

Hochschule Bochum
Bochum University
of Applied Sciences

MIRO LAB

Inhaltsverzeichnis

1 JavaScript - Teil 1

1.1 Einführung

JavaScript (kurz *JS*) ist eine von HTML unabhängige, interpretierte, typenlose Programmiersprache mit Objektorientierung, die eine dynamische Steuerung von Aussehen und Inhalt von (WWW-)Dokumenten ermöglicht. Sie wird, ebenso wie HTML und CSS, clientseitig ausgewertet. Javascript wird zur Laufzeit, also beim Laden der Internetseite interpretiert. Hierbei muss der Browser über ein entsprechendes Interpreter-Modul verfügen, welches die JavaScript-Anweisungen versteht und umsetzt. Daraus folgt jedoch auch, dass JavaScript oftmals von Browser zu Browser unterschiedlich interpretiert wird und daher ausführlich getestet werden sollte.

1.2 JavaScript verwenden

Grundsätzlich ist auch eine Integration von JavaScript-Bereichen in HTML-Code möglich, was hier jedoch nicht genauer betrachtet werden soll. Sie werden Beispiele dafür noch sehen.

Eine Datei *nachricht.js* im selben Verzeichnis wie die HTML-Datei kann dabei wie folgt eingebunden werden:

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

Das Skript wird zum Zeitpunkt bzw. an der Stelle der Einbindung ausgeführt (dies wird später noch wichtig).

Übung 1:

Erzeugen Sie eine Datei *nachricht.js* mit folgendem Inhalt:

```
1 //Jede Menge JavaScript-Anweisungen
2 alert("Hallo Welt!");
3 window.document.write("<p>Innerhalb des JavaScripts</p>");
```

Erzeugen Sie anschließend eine einfache HTML-Datei, in der Sie die Datei einbinden, und öffnen Sie diese im Browser.

Die Funktion `alert(''Hallo Welt!'');` ruft eine Pop-Up-Box mit dem Text `''Hallo`

`Welt''` und einem Button zur Bestätigung auf, während `window.document.write(text)` den entsprechenden Text an die „aktuelle“ Stelle schreibt.

1.3 Debugging

Das Debugging ist ein sinnvolles Hilfsmittel, um Fehler im Quellcode (oder sogar im laufenden Programm) aufzudecken. In den meisten Browser können Sie die sog. Entwicklerkonsole mit (STRG)+(SHIFT)+(I) öffnen und dann auf den Tab „Console“ klicken - diese und der „Inspector“ werden im Weiteren noch hilfreich, also lassen Sie diese gerne für die weiteren Übungen offen.

Übung 2:

Kopieren Sie den folgenden Code in eine HTML-Datei. Vor Ausführung der Website: Welche Ausgaben sind auf der Website sichtbar? Welche Ausgaben sind überhaupt für Besucher lesbar?

```
1  <html>
2  <head>
3    <title>Darstellung der Ausfuehrreihenfolge</title>
4    <script type="text/javascript">
5      document.write("Ausgabe A");
6    </script>
7  </head>
8  <body>
9    <script type="text/javascript">
10     alert("Ausgabe D");
11   </script>
12   <p>Ausgabe B</p>
13   <script type="text/javascript">
14     console.log("Ausgabe C");
15   </script>
16 </body>
17 </html>
```

Öffnen Sie nun die Seite im Browser. Überprüfen Sie Ihre Überlegungen und finden Sie heraus, in welcher Reihenfolge die Anweisungen ausgeführt werden (Hinweis: dies ist zum Teil abhängig vom Browser, zum Beispiel Chrome, Firefox und Edge).

Übung 3:

Schreiben Sie ein neues Skript, das ein Pop-Up mit dem Text „Hallo Welt“ aufruft und „Fehler!“ in der Konsole ausgibt.

Übung 4:

Ändern Sie die `alert`-Anweisung in `olert`. Was passiert im Browser, was in der Entwicklerkonsole?

1.4 Grundlegende Syntax

Eine Anweisung in Javascript besteht immer aus einem Befehl, welcher optional mit einem Semikolon (`;`) abgeschlossen werden kann. Bei den Anweisungen kann es sich um Zuweisungen von Variablen, Funktionsaufrufen und dergleichen handeln.

```
1 var multiplikation = wert * wert;
```

Kommentare können entweder durch `//` (einzeilige Kommentare) oder mit `/* ... */` (mehrzeilige Kommentare) eingebunden werden.

```
1 /* Hierbei handelt es sich um einen
2    mehrzeiligen Kommentarblock!  */
3 var i;    // i hat den Wert 'undefined'
4 i = 2;    // i erhält den Wert 2
5 var gruss = "Hello World";
```

JavaScript ist eine typenlose Programmiersprache, es gibt also keine expliziten Datentypen, d.h. Sie können beliebige Werte zuordnen. Je nach Wertzuweisung oder Verwendung wird der gerade aktuelle Datentyp von JavaScript intern konvertiert, um das vermeintlich günstigste Ergebnis zu erhalten.

```
1 var wert1=3;
2 var wert2=5;
3 var summe;
4 summe = wert1+wert2; //wert1 und wert2 sind Zahlen, das Ergebnis
     summe wird ebenfalls als Zahl abgelegt.
5 //Ausgabe in die HTML-Datei und implizierte Konvertierung zur
     Textkette
6 document.write("Das Ergebnis: " + summe);
```

Übung 5:

Programmieren Sie obiges Beispiel. Welches Ergebnis erwarten Sie?

Was passiert, wenn Sie `wert2` ein `'x'` zuweisen?

Auch Arrays, Verzweigungen und Schleifen gibt es in JavaScript:

```
1  var wochentage = new Array('Montag', 'Dienstag', 'Mittwoch', '
       Donnerstag', 'Freitag', 'Samstag', 'Sonntag');
2
3  for (var i = 0; i <= wochentage.length; i++){
4    if (i%2 == 0){
5      console.log(wochentage[i]);
6    }
7  }
8  for(let tag of wochentage){
9    console.log(tag);
10 }
```

1.5 Document Object Model (DOM)

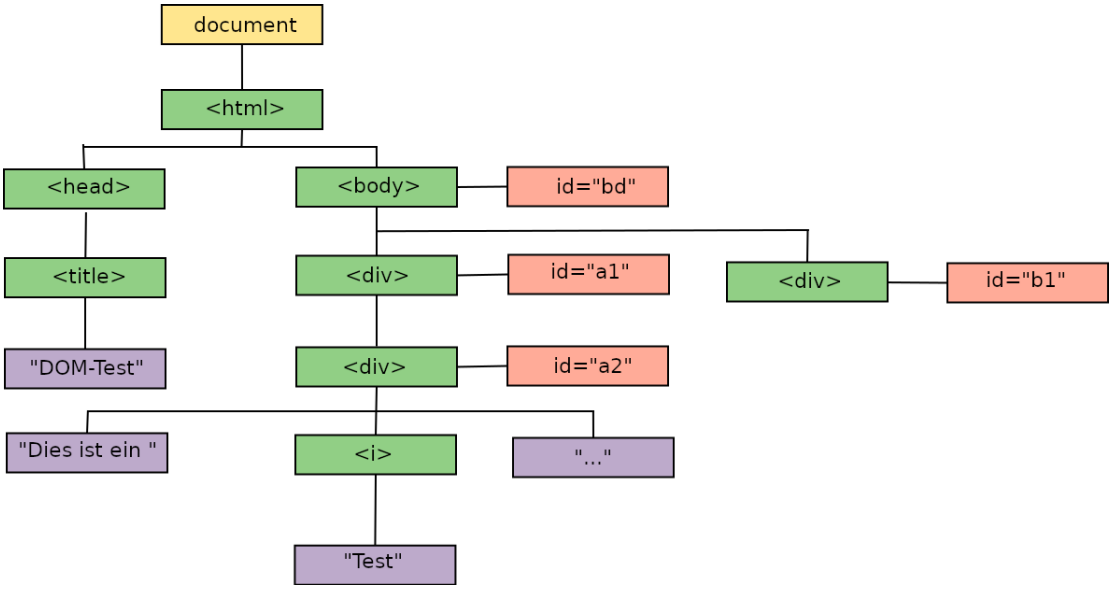
Da JavaScript-Programme, rein technisch betrachtet, Skripte sind, die von fremden Servern geladen und auf Ihrem Rechner ausgeführt werden, darf der JavaScript-Interpreter aus Sicherheitsgründen keine lesenden Zugriffe auf Ihr lokales Dateisystem erlauben. Ein typischer Verwendungszweck von JavaScript ist daher die Interaktion mit dem **DOM**.

Das DOM ist ein Standard der W3C und erlaubt die Abbildung der Objekte einer Webseite in einer Hierarchie. So können browserunabhängige Skripte programmiert werden. Von den meisten Browsern wird wohl DOM Level 2 unterstützt (es gibt auch Level 3 und 4).

```
1  <html>
2  <head>
3    <title>DOM-Test</title>
```

```
</head>
<body id="bd">
  <div id="a1">
    <div id="a2">
      Dies ist ein <i>Test</i>...
    </div>
  </div>
  <div id="b1">
  </div>
</body>
</html>
```

Die vorgenannte Webseite würde als DOM-Baum wie folgt dargestellt werden (DOM-Level 2):



Sämtliche Tags werden als Objekte abgebildet (Näheres dazu in der nächsten Übung). Das Ziel ist der Zugriff auf die einzelnen Knoten (nodes). Als Wurzelelement steht dazu stets zunächst der Dokumentknoten (hier: gelb). Darunter können sich Elementknoten (hier: grün) befinden, die die HTML-Elemente der Seite beschreiben. Diese Elementknoten können Attributknoten (hier: rot) besitzen, die zum Beispiel Informationen zur ID oder der Klasse eines Elements enthalten. Als letztes Element gibt es Textknoten (hier: lila), die den tatsächlich angezeigten Text darstellen.

Bei der Arbeit mit dem DOM werden hauptsächlich Elementknoten abgerufen und manipuliert. Sie können die Elementknoten des DOM-Baums ansprechen mit Hilfe von drei Möglichkeiten:

- `getElementById()`
- `getElementsByClassName()`

- `getElementsByTagName()`

Hier in unserem Beispiel könnte man also definieren

```
1 var el = document.getElementById("a2");
```

`el` enthält nun eine Referenz auf das Objekt mit der ID `a2`.

Übung 6:

Testen Sie die folgenden Fragestellungen gerne direkt als JavaScript-Code und lassen Sie sich das Element z.B. in der Konsole ausgeben.

1. Wie würde der Aufruf für die andere Variante im Beispiel lauten, also `getElementsByTagName`?
2. Wie könnte ein Element mit der Klasse `c3` mit Hilfe von `getElementsByClassName()` abgerufen werden?

Mit JavaScript können nicht nur Elemente aus dem DOM abgerufen, sondern auch verändert, gelöscht oder hinzugefügt werden. Dazu kann die mittels vorgenannter Methoden erhaltene Referenz verwendet werden. Soll einfach der vom verwendeten Tag umschlossene Inhalt verwendet werden, kann das Attribut `innerHTML` verwendet werden:

```
1 var el = document.getElementById("test");
2 el.innerHTML = "<p>Hello World!</p>";
```

Dies sollte jedoch, vor allem auf produktiv eingesetzten Seiten, nicht verwendet werden, da so Schadcode auf die Seite gelangen kann.

Alternativ dazu kann die Methode `el.textContent` verwendet werden, die nur das Hinzufügen von Text ermöglicht (HTML-Elemente werden ignoriert). Darüber hinaus können die benötigten Knoten einzeln hinzugefügt werden. Die wichtigsten Methoden dafür sind:

- `createElement(html)`
- `createTextNode(text)`
- `appendChild(childNode)`, `removeChild(childNode)`
- `setAttribute(attributeName, attributeValue)`

Übung 7:

Rufen Sie eine Website auf, z.B. `https://www.google.de`. Öffnen Sie dann die Console und löschen Sie damit den `<body>`-Container (temporär). Der dafür notwendige Code ist auf jeder Seite einsetzbar.

Übung 8:

Erzeugen Sie eine HTML-Seite mit einem `div`-Container mit beliebiger ID. Schreiben Sie dann einen JavaScript-Code, der Folgendes macht:

- Die aktuelle Uhrzeit wird ausgelesen (ergibt einen Wert zwischen 0 und 23, Näheres zu dem `Date`-Objekt in der nächsten Übung):

```
1 var d = new Date();
2 var h = d.getHours();
```

- Wenn es zwischen 11 und 13 Uhr ist, soll zehnmal „Mittagspause!“ in den `<div>`-Container geschrieben werden, sonst einmalig „Noch kein Feierabend...“.

Übung 9:

Gegeben sei folgende Website:

```
<html>
<head>
  <title>JavaScript DOM</title>
</head>
<body>
  <div id="test"></div>
  <div class="test"></div>
</body>
</html>
```

und folgender JavaScript-Quelltext:

```
var el = document.getElementById("test");
var para = document.createElement("p");
var helloWorld = document.createTextNode("Hello World!");
para.setAttribute("class", "important");
para.appendChild(helloWorld);
el.appendChild(para);
```

Binden Sie den obigen Quelltext an passender Stelle in der HTML-Vorgabe ein und öffnen Sie die Seite im Browser. Warum ist es wichtig, an welcher Stelle die JavaScript-Datei eingebunden wird?

Übung 10:

Verändern Sie den JavaScript-Code nun so, dass auf den zweiten `<div>`-Container zugegriffen wird (ohne diesem eine ID zu vergeben) und fügen Sie eine Liste mit zwei „Hello World!“-Einträgen hinzu, ohne `innerHTML` zu verwenden. Die Liste soll zudem eine ID erhalten.