JavaScript

Einbinden von JavaScript in HTML-Seiten: 3 Varianten

HTMLwithJS.html | sote2.js | Insert title here

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="ISO-8859-1">
5      <title>Insert title here</title>
6      <script type="text/javascript" src="sote2.js"></script>
7      <script type="text/javascript">
8          //alert("JavaScript Pop-up");
9
10     </script>
11 </head>
12 <body>
13     <h1>Überschrift</h1>
14
15     <script type="text/javascript">
16         alert("JavaScript im Body");
17     </script>
18 </body>
19 </html>
```

1. Referenz auf externe Datei

2. Im Head deklariert

3. Im Body deklariert

JavaScript wird gestartet, entweder wenn die Seite geladen wird und der Code in keiner Funktion steht oder wenn eine Funktion z.B. über einen EventHandler ausgelöst wird.

HTMLwithJS.html | sote2.js | Insert title here

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="ISO-8859-1">
5      <title>Insert title here</title>
6      <script type="text/javascript" src="sote2.js"></script>
7      <script type="text/javascript">
8          function hallo() {
9              alert("JavaScript Pop-up");
10         }
11
12     </script>
13 </head>
14 <body>
15     <h1>Überschrift</h1>
16     <p onmouseover="hallo()">Ein Text</p>
17     <input type="button" value="Kopf" onclick="hallo()">
18
19 </body>
20 </html>
```

EventHandler

EventHandler: Wenn Button geklickt wird, wird die Funktion "hallo()" aufgerufen

Funktion kann mit Übergabeparameter aufgerufen werden:

```
onclick="hallo('Nachricht')"
```

Liste an EventHandler

alphabetisches Verzeichnis

A

- abort (Abbruch beim Laden der Seite, unvollständig geladen)
- activate (ausgelöst ↔ *click*)
- afterprint (nach dem Druckereignis)
- animationend (CSS-animation beendet)
- animationiteration (CSS-animation wiederholt)
- animationstart (CSS-animation begonnen)

B

- beforeprint (vor Auslösen des Druckereignis)
- beforeunload
- blur (beim Verlassen)

C

- canplay (bereit zum Abspielen)
- canplaythrough (bereit zum Abspielen bis zum Ende)
- change (bei erfolgter Änderung)
- click (beim Anklicken)
- contextmenu (wenn die rechte Maustaste ein Kontextmenü anfordert)
- copy (in Zwischenablage kopiert)
- cuechange
- cut (ausgeschnitten)

https://wiki.selfhtml.org/wiki/JavaScript/DOM/Event/%C3%9Cbersicht

Syntax von JavaScript

Gemeinsamkeiten mit Java

- Zeilen / Statement (optional) mit Semicolon
- Klammerung
- Kontrollstrukturen (if, for, while) ähnlich
- Kommentare //, /* ... */

Deklaration von Variablen

- a = 2;
- var a = 2; //Zahl (int)
- var a = "2"; //String
- var vieleNamen = new Array(); vieleNamen[0] = "Hans"; vieleNamen[1] = 2.3;

Es gibt bereits deklarierte Objekte z.B. Array, String, Boolean, Object, ...

Funktion

```
function NAME (Parameter1, Parameter, ...) {...}
```

Beispiel:

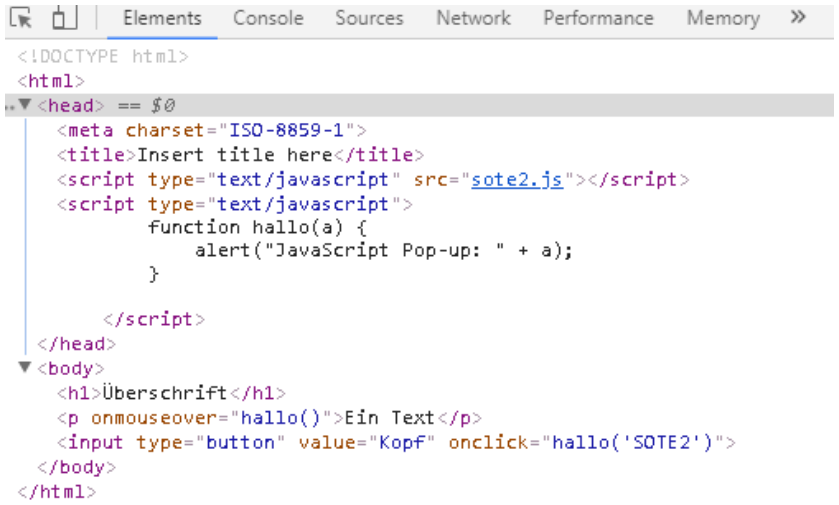
```
function add (a, b) {
 var c = a + b;
 return c;
}
```

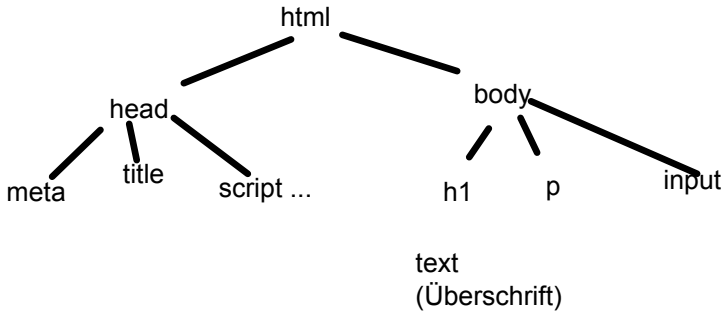
Keine deklarierten Rückgabe und Übergabe-Datentypen!

Es gibt bereits vordefinierte Funktionen :

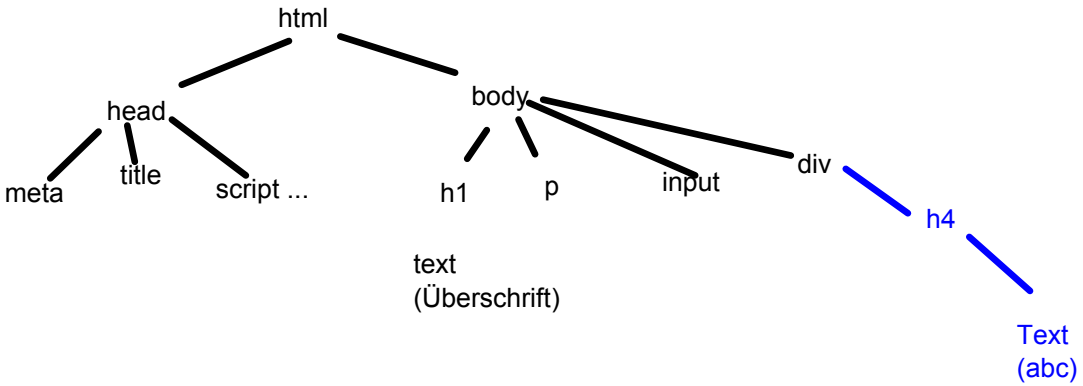
- alert(ZEICHENKETTE)
- eval(ZEICHENKETTE); wertet eine Formel aus "3*4 + 7"
- escape(ZEICHENKETTE): Sonderzeichen oder Leerzeichen umwandeln z.B. Leerzeichen in %20
- Typ-Umwandlung
  - > parseInt()
  - > Number()
  - > String()

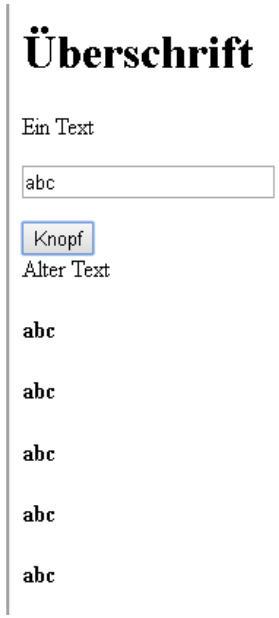
Manipulieren von HTML-Seiten mit JavaScript





DOM-Baum (Document Object Model)






```
function manipulate() {
    var derWert = document.getElementById("unserinput").value;
    alert("Der Wert im TextFeld ist " + derWert);

    //1. Variante: Direkt HTML-Code in Element einfügen (ersetzen)
    document.getElementById("austausch").innerHTML = derWert;

    //2. Variante: Neue Knoten erzeugen und einfügen
    var knoten = document.createElement("h4");
    var knoteninhalt = document.createTextNode(derWert);
    knoten.appendChild(knotenInhalt);
    document.getElementById("austausch").appendChild(knoten);
}
```

Manipulation des DOM-Baums: Knoten (h4) mit Inhalt wird an das div-Element angehängt.

JavaScript: Klassen und Funktionen

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="ISO-8859-1">
    <title>Insert title here</title>
    <script type="text/javascript">
        //1. Variante, um Objekt zu erzeugen
        var student1 = new Object();
        student1.name="Alex";
        student1.matrikel=123;

        //2. Variante, um Objekte zu erzeugen
        var student2 = {name:"Steffi", matrikel:456};

        //3. Variante, um Objekte zu erzeugen
        function Student (derName, dieMatrikel) {
            this.name = derName;
            this.matrikel = dieMatrikel;
        }

        var student3 = new Student("Tobi", 789);

        function ausgeben() {
            alert("Der Student 1 heißt " + student1.name + "\n" +
                  "Der Student 2 heißt " + student2.name + "\n" +
                  "Der Student 3 heißt " + student3.name );
        }
```

Klasse/Funktion deklarieren

Instanzierung

Auslesen von Instanz-variablen

```
var student4 = new Object();
student4.name="Alex";
student4.matrikel=123;
student4.immatrikulieren = function (datum) {
    this.datum = datum;
    alert("Neues Datum ist " + datum);
}

student4.immatrikulieren("2018");
```

Attributen kann man nicht nur Werte (z.B. Zahlen, Strings) zuweisen, sondern auch Funktionen.

Hier wird ausgegeben "Neues Datum ist 2018"

JSON (JavaScript Object Notation)

```
{
    "Vorname": "Lea",
    "Matrikel": 123
    "Noten": [1.3, 1.0, 2],
    "Auto": {
        "Typ": "Audi",
        "PS": "100"
    }
}
```

Erlaubte Datentypen

- String
- Zahlen
  - > Ganzzahlen
  - > Gleitkommazahlen 1.23
  - > Wiss. Notation: 1.2E3
- Bool: true, false
- Null
- Object
- Array

```
<student>
    <Vorname>Lea</Vorname>
    <Matrikel>123</Matrikel>
</student>
```

Gemeinsamkeiten mit XML

- Plain-Text, menschenlesbar
- Hierarchische Struktur

Unterschiede mit XML

- JSON ist "schlanker"
- JSON kennt nur vordefinierte Datentypen, XML erlaubt die Definition eigener
- JSON kennt Arrays
- JSON lässt sich direkt in JavaScript