JSON – JavaScript Object Notation

Im Unternehmenskontext spielt der Austausch von Daten zwischen unterschiedlichen Systemen eine große Rolle. Dabei müssen oft Technologie und Zuständigkeitsgrenzen überwunden werden. Die Liste der Formate, die in Projekten eingesetzt werden, ist lang und oft überleben die Formate sogar die Technologie, für die sie ursprünglich erfunden wurden.

ln den Jahren ab 1990 entwickelte sich XML (Extensible Markup Language) als das universelle Format für den Austausch von strukturierten Daten. Aus dem Alltag vieler Unternehmen ist es gerade im Umfeld von B2B (Business to Business) nicht mehr wegzudenken. Typische Einsatzbeispiele sind Produktkataloge, Bestellungen oder Prozesse. Es gibt viele gute Gründe, es einzusetzen. Die Struktur der Daten kann mit den Tags sehr flexibel und frei als Baum definiert werden. Über DTD (Document Typ Definition) und Schemata ist die Prüfung der Struktur und der Datentypen sehr einfach. Leider ist das Format etwas »geschwätzig« und bläht eine Nachricht schnell auf eine enorme Größe auf.

JSON

Eine der bekanntesten leichtgewichtigen Varianten für den Datenaustausch ist JSON (www.json.org/json-de.html). Der Name steht für JavaScript Object Notation. Dieses Textformat ist sehr kompakt, flexibel und sowohl für Menschen als auch Maschinen gut lesbar. Es wurde Anfang der 2000er Jahre entwickelt, um Daten zu speichern und auszutauschen, wobei die Daten einerseits für Menschen gut lesbar sind und andererseits gut in JavaScript verarbeitet werden können.

Jedes JSON-Dokument stellt gleichzeitig einen gültigen JavaScript-Ausdruck dar. Das Einlesen und die Interpretation sind im Handumdrehen erledigt. Davon abgesehen ist JSON in anderen Sprachen sehr gut nutzbar. Die offizielle JSON-Seite dokumentiert dies eindrucksvoll: Es gibt Parser für alle ernsthaften Programmiersprachen, selbst Urgesteine wie COBOL und RPG sind vertreten.

Die Regeln von JSON sind einfach.

- Es gibt immer einen Schlüssel und einen zugehörigen Wert,

Das Zeichen für die Schlüssel-Wert-Zuweisung ist nicht das „=-Zeichen“, sondern der Doppelpunkt. Das kennt man ja von CSS.

- Die Schlüssel-Wert-Kombinationen werden anders als bei CSS, mit einem **Komma** getrennt.
- Die Schlüssel und alle Zahlen als Werte werden ohne Anführungszeichen geschrieben. Texte werden in Anführungszeichen geschrieben, da sie ja Strings sind.
- Jede Eigenschaft muss eindeutig sein. Es darf jeder Schlüssel nur genau einmal vorkommen. (Zumindest pro Kontext).
- Leerzeichen können zur besseren Lesbarkeit für die Formatierung genutzt werden.
- Zwischen den Anführungszeichen ist keine Zeilenumbruch erlaubt – wie auch nicht bei JavaScript.

- Ein Schlüssel kann auch mehrere Werte haben, so wie ein Array oder eine Liste. Diese Werte werden mit einem Komma getrennt und stehen, um sie (als Liste) besser zu erkennen in eckigen Klammern.
- Um einen Datensatz besser zu definieren, also die gesamte Passage mit allen Infos, setzt man diesen in geschweifte Klammern.
- Hat man mehrere Passagen (Datensätze), muss man sie auch wieder als Liste darstellen und die daher mit Komma trennen – also zwischen den geschweiften Klammern.
- Alle Datensätze zusammen werden in eckige Klammern gesetzt.
- Nach dem letzten Element einer Liste, egal welche und wie viele Elemente die Liste hat, wird **kein Komma** mehr geschrieben. Manche Browser kommen sonst durcheinander.
- Der „Gesamt-Schlüssel“ für alle diese Datensätze gemeinsam muss ganz zu Beginn, vor die eckige Klammer, gestellt werden mit einem Doppelpunkt. In unserm Fall „situationen:“
- Damit man darauf auch zugreifen kann, wird alles in geschweifte Klammern gesetzt und mit einer Variablen versehen, hier beim Beispiel „var geschichte“.

Beispiel:

```
 7 <body>
 8 <script>
 9     var geschichte = {
10         situationen: [
11             {
12                 id: 1,
13                 auswahlText: "Du betrittst die Klasse",
14                 text: "Du bist viel zu spät.",
15                 ziele: [2,3]
16             },
17             {
18                 id: 2,
19                 auswahlText: "Du gehst in die 3CK",
20                 text: "Der Professor ist schon da.",
21                 ziele: [4]
22             },
23         ]
24     };
25 </script>
```

Code:

```
<script>
  var geschichte = {
    situationen: [
      {
        id: 1,
        auswahlText: "Du betrittst die Klasse",
        text: "Du bist viel zu spät.",
        ziele: [2,3]
      },
      {
        id: 2,
```

```
            auswahlText: "Du gehst in die 3CK",
            text: "Der Professor ist schon da.",
            ziele: [4]
          },
      ]
    };
</script>
```

Hier ist eine etwas ältere Schreibweise. Die Anführungszeichen vor dem Doppelpunkt (bei den Schlüsseln) sind nicht mehr nötig:

```
{
  "ISBN": "3839154057",
  "Preis": 8.50,
  "Waehrung": "EURO",
  "Titel": "HirnSport.de - Rätsel für das tägliche Gehirnjogging",
  "Seiten": 120,
  "Lieferbar": true,
  "Kategorien": [ "Gehirnjogging", "Rätsel", "Freizeit" ],
  "Autor": {
    "Name": "Spindler",
    "Vorname": "Heiko",
    "Alter": 42
  }
}
```

Beachte: Autor hat eine eigene Unterliste.

Von JSON zu JavaScript:

Da wir hier die JSON-Daten innerhalb des <script>-Bereiches liegen, müssen sie nicht „übergeben“ werden. Das wäre etwas komplizierter, wenn sie auf einem entfernten Webserver liegen würden.

Nun folgt das Herauslesen aus der Variablen „geschichte“.

Für JavaScript ist JSON grundsätzlich ein „Objekt“. Deshalb verwendet man die bekannte objektorientierte Schreibweise um darauf zuzugreifen: Schreibweise mit Punkt

```
geschichte.situationen
```

„situationen“ wiederum hat seinen Inhalt in eckigen Klammern, daher ist es eine Liste bzw. ein Array. Genauso wie bei einem Array greift man daher auf diese Inhalte zu, nämlich mit Nummern von 0 beginnend. Das erste Element ist daher

```
geschichte.situationen[0)
```

Damit erhält man die komplette geschweifte Klammer von „id: 1“ bzw. ihren Inhalt.

Die einzelnen Teile erhält daraus erhält man folgenderweise:

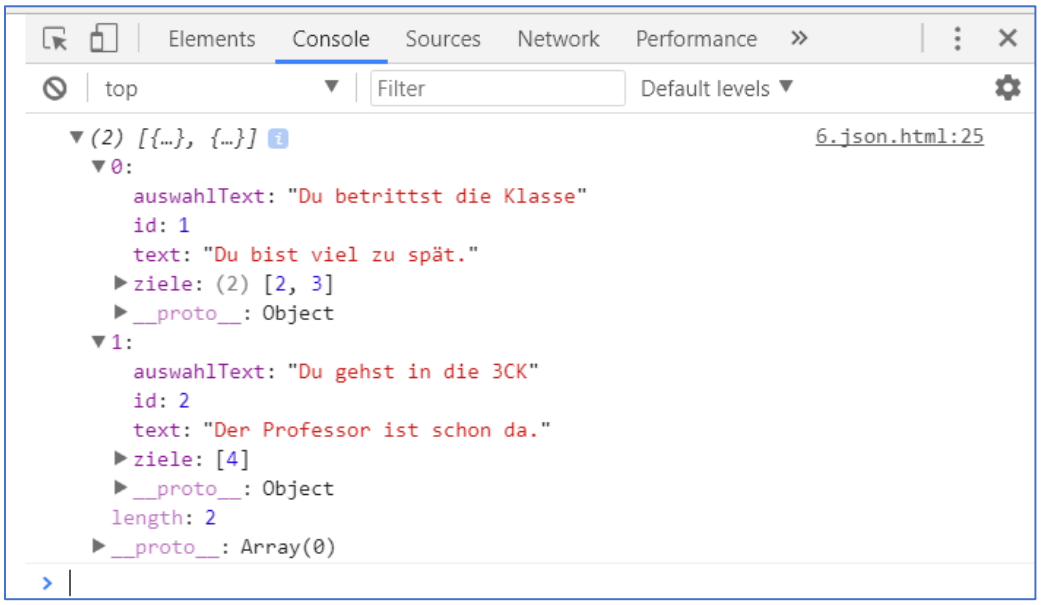
- Die ID mir geschichte.situationen[0].id
- Der Auswahltext geschichte.situationen[0].auswahlText
- Den Text geschichte.situationen[0].text
- Mit „geschichte.situationen[0].ziele“ bekammt man ein Array geliefert.

Ausgabe mit „console.log()“:

Im Gegensatz zu „alert“ erhält man genaue Informationen darüber, wie das jeweilige Element aufgebaut ist.

```
console.log(geschichte.situationen);
```

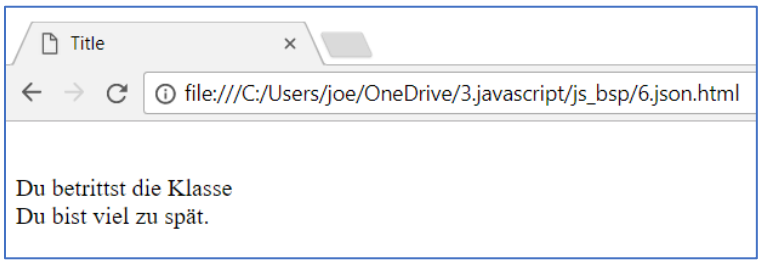
wird in das „script-Tag“ nach der Deklaration des JSON-Objektes geschrieben. In der Konsole des Browsers sieht man dann folgende genaue Aufstellung: - Drücke „F12“




```
24      };
25      console.log(geschichte.situationen);
26  </script>
27
28  </body>
```

Beispiel weiter:

Ziel: Es sollen zwei Texte aus dem ersten Element untereinander ausgegeben werden.



Dafür muss VOR dem Script-Block in einem <p> eine ID erstellt werden, in der später mit „innerHTML“ der Text geändert wird.

```
 7 ▼ <body>
 8       <p id="hierStehtText"> Text wird neu </p>
 9 ▼ <script>
10 ▼     var geschichte = {
```

Nach den JSON-Teil werden drei Variablen erstellt, wobei die ersten beiden den Text aus dem JSON-Format-Text auslesen. Die dritte fügt diese beiden Texte nur noch zusammen.

Darunter erfolgt mit „getElementById“ und „innerHTML“ der Austausch des Textes.

```
27      var ersterText = geschichte.situationen[0].auswahlText;
28      var text = geschichte.situationen[0].text;
29      var gesamterText = ersterText + '<br>' + text;
30      document.getElementById("hierStehtText").innerHTML = '<br>' +
        gesamterText +'<br>';
31  </script>
```

Kürzer wäre die Zeile 30 auch möglich – KURZFORM: „ID-Namen und innerHTML“

```
29      var gesamterText = ersterText + '<br>' + text;
30      hierStehtText.innerHTML = '<br>' + gesamterText +'<br>';
31  </script>
```

Übung:

Frage zusätzlich die beiden Texte ab, die aus der „id=2“ stammen und gib sie darunter aus. Nutze die Kurzform.

Quelle:

Stephan Elter, in: Programmieren lernen mit JavaScript;
Rheinwerk Verlag, Bonn, 2017, S. 236-259