JavaScript

Generell

1. JavaScript-Dateien sollen mit der Dateiendung .js versehen und ausgeliefert werden.
2. JavaScript-Code ist grundsätzlich in Dateien auszulagern, es sei denn, es handle sich um Code, der Benutzerabhängig ist, sich also mit jedem Aufruf, jeder Session verändert.

Validierung

Die Validierung von JavaScript-Code ist im Vergleich zu HTML oder CSS bedeutend komplexer. Folglich fehlen Standard-Tests, die JavaScript-Code als valide auszeichnen könnten.

Testing

Für JavaScript-Testing bieten sich folgende Tools an:

- JSLint - The JavaScript Code Quality Tool
  JSLint analysiert den JavaScript-Code und erkennt Fehler und allfällige Probleme.
- YUITest, JavaScriptMVC, Js-Unit und JSLitmus
  Unit-Testing für JavaScript

Integration in HTML

- Enthält die JavaScript-Datei keinen Code, der während des Ladens keinen Dokumenteninhalt erzeugen muss (document.write), sollte er möglichst ganz am Ende des HTML-Konstrukts eingebunden werden. Ist diese Platzierung nicht möglich, kann alternativ das Attribut defer="defer" gesetzt werden. Damit warten Browser nicht, bis die JavaScript-Datei geladen ist, um mit dem Verarbeiten der Seite fortzufahren. Diese Variante wird allerdings nicht von allen Browsern gleich unterstützt und sollte deshalb nur ausnahmsweise verwendet werden.
- Wenn unklar ist, ob JavaScript-Dateien am Ende des Body-Bereichs integriert werden kann, ist dieser im Head-Bereich des HTML-Konstrukts einzubinden.
- Innerhalb des Head-Bereich werden Script-Aufrufe zuletzt integriert, da die Reihenfolge eine direkte Auswirkung auf die Ladezeiten einer Webseite haben kann.
  Richtig:

```
<head>
    <link rel="stylesheet" type="text/css" href="/style.css" />
    <script src="/global.js"></script>
</head>
```

  Falsch:

```
<head>
<script src="/global.js"></script>
<link rel="stylesheet" type="text/css" href="/style.css" />
</head>
```

- JavaScript, der nicht ausgelagert werden kann, wird in einen CDATA-Bereich gesetzt:

```
<script type="text/javascript">
<![CDATA[
... JavaScript-Code ...
]]>
</script>
```

- Der Script-Tag wird auf jeden Fall mit dem Attribut type="text/javascript" versehen.
- Das Attribut wird "language" wird nicht mehr verwendet.
- Generell sollen möglichst wenig verschiedene JavaScript-Quellen eingebunden werden. Wenn möglich sind die verschiedenen Quellen für den produktiven Einsatz zusammenzufassen.
- JavaScript-Dateien sollen für den produktiven Einsatz minimiert werden. Tools:

    - shrinker - CSS & JavaScipt Compression
    - Online Javascript Compression Tool
    - YUI Compressor

- komprimierte Dateien erhalten die Kennung **.min** - die Orginal-Datei ohne Kennung befindet sich in der Regel im selben Verzeichnis.
  Beispiel:
    - /jscript/custom/slider.js
    - /jscript/Custom/slider.min.js

Unobstrusive JavaScript

- JavaScript wird grundsätzlich als optionale Technik betrachtet. Man kann nur in definierten Ausnahmefällen davon ausgehen, dass JavaScript vom Eingabegerät korrekt interpretiert werden kann.
- JavaScript wird unabhängig vom Eingabegerät programmiert.
- JavaScript-Interaktivität wird unabhängig vom verwendeten Eingabegerät definiert, so dass eine gleichwertige Nutzung über Maus, Tastatur möglich ist (onclick, onkeypress usw).
- JavaScript-Interaktivität wird erst nach dem Laden des Dokuments (onload) dynamisch hinzugefügt.
- Eine Vermischung von JavaScript mit HTML und CSS ist zu vermeiden.

Formatierung

- Sauber formatierter Quellcode erleichtert das Arbeiten. Dazu gehört das Strukturieren mit Zeileneinzügen. Ob dafür Spaces oder Tabs verwendet werden, spielt zur Zeit keine Rolle. Zu bevorzugen sind generell 4 Leerzeichen für einen einfachen Einzug. Tabs sind zwar in der Eingabe schneller und brauchen weniger Speicherplatz, allerdings können bei der Ersetzung von Tabs immer wieder Probleme auftreten.
- Vermeide Zeilen, die länger als 80 Zeichen sind. Wenn ein Ausdruck nicht auf einer Zeile untergebracht werden kann, ist es ratsam, das Statement auf mehrere Zeilen aufzuteilen. Der Umbruch sollte dabei nach einem Operator eingefügt werden. Die folgende Zeile soll um eine weitere Einheit eingerückt werden.
  Beispiel:

  ```
  function test() {
  var t;
  t = 15 +
  23;
  return t;
  }
  ```

- Abkürzungen sind zu vermeiden, sprich optionale Klammern und Semikolon sind zu setzen.
  gut:

  ```
  if(irgendwas){
  x = false;
  }
  ```

  schlecht:

  ```
  if(irgendwas)
  x = false
  ```

- Ein bekanntes Tool zur Formatierung von Quellcode ist der Javascript beautifier

Variablen

- Variablen werden zu Beginn jeder Funktion deklariert, jede Variable erhält eine eigene Zeile. Die Variablen werden in alphabetischer Reihenfolge notiert.
  Beispiel:

```
var i; //counter
var email; // Platzhalter fuer E-Mail-Adresse
var isOK = false; // Boolean fuer Pruefresultat
```

  Generell ist auf globale Variablen zu verzichten, um Konflikte mit anderen Scripten zu vermeiden. Werden globale Variablen verwendet, sollen sie in Grossbuchstaben geschrieben werden.

Funktionen

- Als generelle Schreibweise für Funktionen hat sich folgende Notation etabliert:

```
function foo() {
return true;
}
```

- Zur Vermeidung von gleichnamigen Funktionen sind Funktionssammlungen anzulegen:

```
var bhs_nav = {
over: function () {
alert('over');
},
out: function () {
alert('out');
}
};
```

  Die Funktion 'over' gehört nun zur Bibliothek 'bhs_nav' und kann wie folgt aufgerufen werden:

```
bhs_nav.over();
```

Namensgebung

- Variablen und Funktionen enthalten nur folgenden Zeichen: a-z,A-Z,0-9,_ - Umlaute sind nicht erlaubt.
- Bezeichnungen beginnen nicht mit _ oder einer Ziffer, sondern in der Regel mit einem Kleinbuchstaben.
- Bezeichnungen sind kurz, sprechend und treffend zu gestalten, ohne ausschweifend zu werden. CamelCase-Notation ist bei längeren Bezeichnungen erwünscht aus Gründen der Lesbarkeit.
- Globale Variablen werden in Grossbuchstaben geschrieben.

Kommentare

- Verwende einzeilige Kommentare innerhalb des Codes:

```
// Berechnung des Geburtstages.
geb = kompliziertеformel * pi();
// oder
geb = kompliziertеformel * pi(); // Berechnung des Geburtstages
```

- Verwende mehrzeilige Kommentare zum temporären Auskommentieren von Code während der Entwicklungsphase oder für längere Kommentare, beispielsweise erklärende Informationen zum gesamten Script.

externe Quellen

- JavaScript CheatSheets
- JavaScript Quick Reference
- jQuery 1.3 CheatSheet
- JavaScript bei selfhtml
- JavaScript bei w3schools.com

Guidelines

- Google JavaScript Style Guide
- JavaScript Style Guide

Regular Expressions

- RegEx visualieren online
- RegEx-Tool, Air-Applikation