Herbert Braun

Hallo, Web!

Mit JavaScript Programmieren lernen, Teil 1

Wer logisch denkt und Zugang zu einem Computer hat, kann auch programmieren lernen. Für Einsteiger eignet sich JavaScript besonders gut, weil die Hürden so niedrig sind: Ein Browser und ein Texteditor genügen bereits als Arbeitsumgebung.

Will man mit dem Programmieren anfangen, ist JavaScript aus mehreren Gründen eine gute Wahl. Keine andere Programmiersprache ist so weit verbreitet: Einst für simple Webseitenmanipulationen angetreten, übernimmt JavaScript immer vielfältigere Aufgaben. Möglich wird das durch schnellere Browser und ihre wachsende Funktionsvielfalt. Außerdem sind die Einstiegshürden minimal: Ein Browser als Laufzeitumgebung und Debugger, ein gewöhnlicher Editor zum Programmieren, mehr braucht es nicht.

Insbesondere Windows-Anwender sollten sich nicht mit dem im Betriebssystem enthaltenen Minimal-Editor zufriedengeben. Gute kostenlose Text- und Programmierwerkzeuge sind zum Beispiel Notepad++ oder jEdit. Sie können mit mehreren Dateien umgehen, den Code farbig markieren, besitzen fortgeschrittene Such- und Ersetzungsfunktionen und lassen sich mit Plug-ins erweitern. Viele Webentwickler schwören auf Sublime Text, das allerdings zirka 50 Euro kostet. Windows-Anwender bekommen mit Microsofts kostenlosem Visual Studio Express für das Web eine exzellente Entwicklungsumgebung, sogar mit eingebautem Webserver.

Grob gesagt zerfällt JavaScript in zwei Komponenten: den eigentlichen Sprachkern, der unter dem Namen ECMAScript spezifiziert ist, und die Schnittstellen zum Browser, die Skripte mit Daten versorgen und über die sie etwas ausgeben oder eine Webseite modifizieren. Vor allem diese Schnittstellen – gern als APIs (Application Programming Interface) bezeichnet – sind es, über die sich die Sprache weiterentwickelt. Es wird nicht ganz ohne APIs gehen, aber im Mittelpunkt steht im Folgenden der Sprachkern: Variablen, Operatoren, Bedingungen, Schleifen, Funktionen, Arrays und reguläre Ausdrücke.

Egal ob C++, PHP, Java, Python oder Ruby: Was die Kernbestandteile angeht, folgen die meisten gängigen Programmiersprachen ähnlichen Konzepten, trotz vieler Unterschiede im Detail. Beherrscht man eine davon, hat man beim Erlernen einer anderen einen großen Vorteil. JavaScript hat übrigens keinerlei Beziehung zu Java – der Name war seinerzeit eine Art PR-Gag.

Achtung, Code

Als erstes JavaScript-Projekt soll eine Webseite entstehen, die einfache statistische Analysen an einem Text durchführt und beispielsweise herausfindet, welches darin das längste Wort ist.

JavaScript wurde dafür gemacht, im Kontext einer Webseite zu laufen (auch wenn es inzwischen weitere Einsatzgebiete dafür gibt). Sie brauchen also für den Anfang ein HTML-Gerüst:

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <title>JavaScript für Einsteiger</title>
      <meta charset="utf-8"/>
  </head>
```

```
  <body>
    <h1>JavaScript für Einsteiger</h1>
  </body>
</html>
```

Der <body>-Teil schreibt eine Überschrift in die Seite; der <head> enthält Meta-Informationen, insbesondere den in der Fensterleiste angezeigten Titel.

Speichern Sie diesen Code zum Beispiel unter dem Dateinamen jsdemo.html und rufen Sie diese Datei dann über die Funktion „Datei öffnen“ im Browser auf. Das Ergebnis: eine unspektakuläre Seite, die nichts tut. Das ändert sich gleich.

Um dieser Seite JavaScript hinzuzufügen, schreiben Sie Folgendes vor das schließende </head>:

```
<script type="text/javascript">
  alert("Hallo, JavaScript!");
</script>
```

Das Tag <script> kennzeichnet Skriptcode, der getreu den HTML-Vorschriften mit seinem MIME-Typ "text/javascript" ausgewiesen wird. Wenn Sie nun die Seite aufrufen, erscheint eine kleine Hinweis-Box mit der Nachricht „Hallo, JavaScript!“.

alert() ist eine in den Browser eingebaute Funktion, auf die Sie immer zugreifen können. Was in den Klammern hinter einem Funktionsnamen steht, sind die Funktionsargumente – das, was Sie der Funktion übergeben.

In diesem Fall handelt es sich beim Funktionsargument um einen Text – oder in der Sprache der Programmierer ausgedrückt: einen String. Strings sind immer in Paare von Anführungszeichen eingeschlossen. Ob Sie doppelte (") oder einfache Anführungszeichen (') nehmen, ist egal, es müssen nur vorne und hinten die gleichen sein. Die typografisch verschönerten Anführungszeichen, die ein Textverarbeitungsprogramm produziert, funktionieren hier nicht.

Am Ende der Zeile steht ein Strichpunkt; dieser schließt eine Anweisung ab. In diesem Fall hätte es zwar auch ohne Strichpunkt geklappt – aber gewöhnen Sie sich frühzeitig an, ihn hinter jede Anweisung zu setzen, denn dieses Versäumnis ist einer der häufigsten Anfängerfehler.

Der Browser führt das Skript genau an der Stelle aus, wo Sie es in den Code einbetten: Die Überschrift erscheint erst, wenn Sie die Hinweis-Box wegklicken. Verschieben Sie nun den Codeblock mit dem <script>-Tag vor das schließende </body>. Laden Sie die Seite im Browser nach dem Speichern im Editor neu, das geht am einfachsten mit F5. Diesmal erscheint die Überschrift bereits vor dem Klick auf „OK“.

```
var text = document.querySelector('article').innerHTML;
var words = text.split(/[.,;!?»«"'\s-]+/);
var longestWord = ''; // längstes Wort
var wordsLengthTotal = 0; // Summe der Wortlängen (für Durchschnitt)

// Schleife: durchläuft alle Elemente des Arrays
for (var i = 0; i < words.length; i++) {
  wordsLengthTotal += words[i].length;
  if (words[i].length > longestWord.length) {
    console.log(i + ' ' + words[i] + ' (' + words[i].length + ')');
    longestWord = words[i];
  }
}

// Ausgabe des Ergebnisses
document.write('<hr/><p>Dieser Text hat eine Länge von ' + text.length + '
    Zeichen.</p>');
var wordLengthAverage = (wordsLengthTotal / words.length).toFixed(1);
document.write('<p>Das längste Wort lautet <q>' + longestWord + '</q> und
```

Durch einen Programmier-Editor sinnvoll eingefärbt, liest sich Code wesentlich leichter als Schwarz auf Weiß.

Outsourcing

Bevor Sie tiefer in JavaScript einsteigen, sollten Sie den Code in eine eigene Datei ausgliedern. Kopieren Sie dazu die alert(…)-Zeile in eine neue Datei, die Sie zum Beispiel „skript.js“ taufen. Diese Datei binden Sie nun in jsdemo.html ein:

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

Wenn Sie alles richtig gemacht haben (nicht vergessen: speichern und im Browser neu laden!), verhält sich die Seite ebenso wie zuvor.

Textlese

Nach diesen ersten Schritten geht es jetzt an das Projekt Textanalyse. Dazu braucht es erst einmal einen Text – wenn Sie gerade keinen finden, greifen Sie einfach bei www.gutenberg.org zu. Als ersten Versuch, den Text ins Dokument zu bekommen, schreiben Sie ihn in skript.js als String in eine Variable:

```
var text = 'Als Gregor Samsa eines Morgens ...';
```

Das Schlüsselwort var, gefolgt von einem Namen, definiert einen Platzhalter für die unterschiedlichsten Inhalte. Wählen Sie den Namen der Variablen so, dass Sie möglichst auch nach einem halben Jahr noch wissen, was damit gemeint war. Beim Namen erlaubt sind alle Buchstaben und Ziffern sowie der Unterstrich _, aber keine Ziffer am Anfang: var hans_peter_2014 oder (gängiger) var hansPeter2014 gehen also, var hans-peter-2014 oder var 2014HansPeter hingegen nicht. JavaScript unterscheidet nach Groß- und Kleinschreibung: Der Name hansPeter bezeichnet also etwas anderes als HansPeter.

Das auf var text folgende Gleichheitszeichen hat eine andere Bedeutung als bei 1 + 1 = 2: Es verleiht der Variablen einen Wert. Deshalb lautet seine korrekte Bezeichnung Zuweisungsoperator. Im Unterschied zum Ist-gleich dürfen Sie rechts und links nicht vertauschen.

Zweiter Versuch

Lästigerweise akzeptiert JavaScript keine Strings, die über mehrere Zeilen gehen – inakzeptabel für längere Texte. Also packen Sie den Text besser ins HTML:

```
<article style="white-space: pre-wrap">Als Gregor Samsa
  eines Morgens ...</article>
<script src="script.js"></script>
```

Die CSS-Angabe im style-Attribut sorgt dafür, dass der Browser den Text richtig umbricht. Im Skript greifen Sie wie folgt auf das erste <article>-Element zu:

```
var text = document.querySelector('article').innerHTML;
```

Die Funktion querySelector() extrahiert dessen Inhalt als String und schreibt diesen in die Variable text.

Wichtig ist hierbei, dass Sie das Skript im HTML-Dokument nach dem Text einbinden, sonst findet die obige Zeile nichts.

Um zu testen, ob alles geklappt hat, setzen Sie als zweite Zeile darunter:

```
alert(text);
```

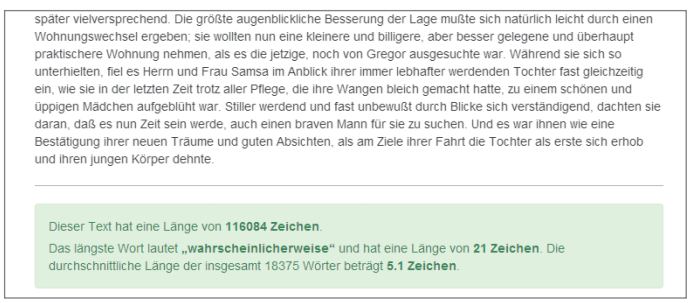
Voilà – in der Alertbox erscheint der gesamte Text (oder so viel, wie in die Box passt).

Zahlenspiele

Eine erste Frage an den Text könnte sein: Wie lang ist er? Diese Information steht in text.length. length ist eine Eigenschaft von Strings wie text. Sie enthält die Anzahl der Zeichen als Zahlwert. Sie können sie mit einem alert() sichtbar machen:

```
alert(text.length);
```

Im Browser sollte nach dem Neuladen die Hinweis-Box mit einer Zahl erscheinen.



später vielversprechend. Die größte augenblickliche Besserung der Lage mußte sich natürlich leicht durch einen Wohnungswechsel ergeben; sie wollten nun eine kleinere und billigere, aber besser gelegene und überhaupt praktischere Wohnung nehmen, als es die jetzige, noch von Gregor ausgesuchte war. Während sie sich so unterhielten, fiel es Herrn und Frau Samsa im Anblick ihrer immer lebhafter werdenden Tochter fast gleichzeitig ein, wie sie in der letzten Zeit trotz aller Pflege, die ihre Wangen bleich gemacht hatte, zu einem schönen und üppigen Mädchen aufgeblüht war. Stiller werdend und fast unbewußt durch Blicke sich verständigend, dachten sie daran, daß es nun Zeit sein werde, auch einen braven Mann für sie zu suchen. Und es war ihnen wie eine Bestätigung ihrer neuen Träume und guten Absichten, als am Ziele ihrer Fahrt die Tochter als erste sich erhob und ihren jungen Körper dehnte.

Dieser Text hat eine Länge von **116084 Zeichen**.

Das längste Wort lautet **„wahrscheinlicherweise“** und hat eine Länge von **21 Zeichen**. Die durchschnittliche Länge der insgesamt 18375 Wörter beträgt **5.1 Zeichen**.

Das fertige Projekt ermittelt statistische Daten eines Texts.

Der in den Browser eingebaute Debugger (hier zum Beispiel Chrome) erlaubt einen raschen Blick auf Variablen und gibt bei Fehlern erste Anhaltspunkte.

Ähnlich wie Eigenschaften lassen sich auch spezielle Funktionen für Strings aufrufen. Zum Beispiel können Sie ganz einfach herausfinden, an welcher Stelle der Autor zum ersten Mal ein kleines „x“ verwendet hat:

```
alert(text.indexOf('x'));
```

Wie bei alert() steht das Funktionsargument in Klammern. Das Ergebnis ist eine Zahl, die die Position des jeweiligen Suchstrings angibt – und wie es unter Programmierern üblich ist, beginnt die Zählung bei 0. Wenn in der Alertbox –1 steht, war die Suche erfolglos.

length oder indexOf() liefern Ganzzahlen (engl. integer) zurück. Gleitkommazahlen sind bekannt als „float“ und werden wie im englischen Sprachraum üblich mit Punkt statt Komma geschrieben (var pi = 3.14159);

Mit Zahlen kann JavaScript rechnen. Die dazu nötigen Operatoren sehen nicht nur aus wie im Mathe-Unterricht, sie verhalten sich auch so: 2 + 1 * 3 ergibt laut Punkt-vor-Strich-Regel nicht 9, sondern 5. Das Sternchen entspricht dem Multiplikationssymbol („mal“).

Wichtig ist der Unterschied von Zahlen zu Strings:

```
var frage = 3 + "2";
```

"2" sieht für Menschen wie eine Zahl aus, aber die Maschine behandelt sie als String. In manchen Programmiersprachen löst das Aufeinandertreffen unterschiedlicher Variablentypen einen Fehler aus; JavaScript wandelt stattdessen die Zahl 3 ebenfalls in einen String um. Diese stillschweigende Konvertierung von Zahl zu String passierte übrigens auch schon bei den letzten beiden Alert-Beispielen (etwa alert(text.length)).

Der Plus-Operator verbindet also zwei Strings. Deshalb hat frage den Wert "32".

Ausgaben

Hinweis-Boxen sind praktisch, nerven aber auf Dauer. Schließlich kann man auch direkt ins Dokument schreiben:

```
document.write("Textlänge: <i>" + text.length + "</i>");
```

Nach querySelector() kommt also schon wieder eine document-Funktion zum Einsatz. Der String, den Sie write() übergeben, kann wie im Beispiel oben auch HTML-Tags enthalten. Das Ergebnis steht dort im HTML-Dokument, wo Sie das Skript aufrufen.

Eine weitere Alternative zu alert() ist console.log():

```
console.log(text.length);
```

Wenn Sie diese Zeile in Ihr Skript einfügen, sehen Sie erst einmal kein Ergebnis. console.log() macht sich nur in den Entwicklerwerkzeugen bemerkbar, die Sie im Browser mit F12 aufrufen, manchmal auch mit Strg+Umsch+I oder über das Kontextmenü. Dort steht Ihnen eine umfangreiche Werkzeugkiste zur Verfügung, deren Leistungsfähigkeit sich nach und nach erschließt.

Vorerst ist der Reiter „Konsole“ interessant, in dem die console.log()-Ausgaben erscheinen sowie die Informationen darüber, an welcher Stelle und warum ein fehlerhaftes Skript abgebrochen wurde. Probieren Sie es aus und geben Sie folgende Anweisung in Ihr Skript oder direkt in die Browser-Konsole ein, bei der das Anführungszeichen am Ende des Strings fehlt:

```
alert("Fehler!);
```

„Unterminated string literal“ oder so ähnlich wird die Antwort auf diesen Tippfehler lauten, versehen mit einer Zeilennummer, wo er zu finden ist. Hätten Sie diesen Fehler versehentlich gemacht, wäre es mithilfe der Debugging-Werkzeuge ein Leichtes, die betreffende Stelle im Code zu finden.

Château d’If

Oft ist es notwendig, eine Variable zu vergleichen und den Code zu verzweigen: Wenn X diesen Wert hat, mach das. Falls Y kleiner als Z ist, tu dies, sonst jenes. In JavaScript sieht das so aus:

```
if (text.length > 99999) {
  alert('Ganz schön lang!');
}
```

Die Bedingung steht in runden Klammern, der auszuführende Code in geschweiften. Die Formatierung – also Umbrüche, Leerzeichen, Einrückungen – sind Geschmackssache, folgen im Beispiel aber gängigen Konventionen, die sich als übersichtlich bewährt haben (der Maschine ist das egal, aber ab und zu muss auch ein Mensch den Code lesen).

Die wichtigsten Vergleiche sind größer (>), kleiner (<), größer gleich (>=), kleiner gleich (<=) und ungleich (!=). Da das einfache Gleichheitszeichen (=) schon reserviert ist, nimmt man es zum Vergleich doppelt:

```
if (text.length == 0) {
  document.write("Nichts drin!");
}
```

Die Verwechslung von = und == stellt eine erstklassige Fehlerquelle dar – zumal dummerweise if (meineVariable = 0) syntaktisch ebenfalls korrekt ist, weshalb der Browser beim Ausführen nicht abbricht.

Wahr oder falsch

Eine Bedingung muss nämlich nicht unbedingt einen Vergleich enthalten, sondern lediglich einen Ausdruck, der wahr oder unwahr ergibt. Auf Sprache übertragen ist ein Ausdruck so etwas wie ein Satzteil, wohingegen die Anweisung (die mit Strichpunkt endet) dem Satz entspricht. Als wahr (true) gelten zum Beispiel folgende Ausdrücke:

```
1 < 2
"eins" != "zwei"
1
"1"
true
!false
meineVariable = 1
```

Unwahr (false) sind hingegen:

```
1 > 2
"eins" == "zwei"
0
""
false
!true
meineVariable = 0
```

Eine Zahl oder ein String ergeben true, wenn sie nicht gerade 0 oder einen leeren String als Wert haben. Außer Strings, Ganz- und Gleitkommazahlen gibt es auch Boole’sche Variablen, die nur true oder false sein können – die Entsprechung zum digitalen 1 und 0. Das vorangestellte Ausrufezeichen kennen Sie aus dem Vergleichsoperator != – und wie dort verkehrt es die Aussage ins Gegenteil, weshalb es auch „Not“-Operator (not, englisch für „nicht“) heißt.

Auch vieles, was man typischerweise als eigenständige Anweisung schreiben würde, besitzt einen Wert, zum Beispiel die Zuweisung; das obige Beispiel if (text.length = 0) gibt den zugewiesenen Wert 0 und damit false zurück.

Soll Code ausgeführt werden, falls eine Bedingung nicht zutrifft, dann schreiben Sie ihn in einen else-Block:

```
if (x == 1) {
  // ...
} else {
  // ...
}
```

Oder auch:

```
if (x == 1) {
  // ...
} else if (x == 2) {
  // ...
```

```
} else {
  // ...
}
```

So können Sie den Programmablauf verzweigen.

Der doppelte Schrägstrich leitet einen Kommentar ein. JavaScript ignoriert alles von dort bis zum Ende der Zeile. Im Beispiel oben soll // … nur symbolisieren, dass an dieser Stelle beliebiger Code stehen kann. Kommentare über mehrere Zeilen leiten Sie mit /* ein und beenden ihn mit */:

```
/* Kommentare dienen
   typischerweise der Dokumentation.
   Man kann mit ihnen aber auch
   Code-Abschnitte übergangsweise
   deaktivieren.
 */
```

Programmplan

Wie könnte nun ein Skript aussehen, um das längste Wort im Text herauszufinden? Nachdem der Text in eine Variable eingelesen ist – das können Sie schon –, müsste man ihn irgendwie in Wörter auftrennen und diese in eine Liste schreiben. Dann soll der Computer diese Liste durchgehen und sich merken, wenn ein Wort länger ist als das bisher längste. Eigentlich gar nicht so schwer – die Beschreibung der Funktion ist länger als ihr Code.

Der nächste Schritt ist also das Auftrennen des Textes. Dafür existiert eine weitere String-Funktion namens split():

```
var words = text.split(' ');
```

split() säbelt den Text an jedem Leerzeichen auseinander – genauer gesagt, den Text in einer Kopie von text, denn die ursprüngliche Variable bleibt erhalten. Die Leerzeichen selbst fallen beim Auftrennen unter den Tisch.

Auf der linken Seite der Zuweisung steht eine Art von Variable, den Sie bisher noch nicht kennengelernt haben. Sie kann eine nummerierte Liste von Daten speichern und heißt Array.

Um Arrays zu verstehen, schauen Sie sich erstmal den Inhalt des Arrays words mit console.log(words) an – alert() oder document.write() funktionieren bei Arrays nicht. Vermutlich werden Sie Wörter wie „verwandelt.“ oder „Augen.¶»Was“ darin finden (das ¶ steht für einen Zeilenumbruch) – klar, ein Text besteht ja nicht nur aus Wörtern und Leerzeichen, sondern auch aus Satzzeichen und Absätzen. Aber wie soll ein Suchstring die alle finden?

Gar nicht – dafür braucht es ein Suchmuster, einen sogenannten regulären Ausdruck. Dieses extrem nützliche Werkzeug finden Sie zum Beispiel auch in der Suchfunktion besserer Texteditoren oder im Linux-Werkzeug grep. split() nimmt als Argument nicht nur einen String entgegen, sondern auch einen regulären Ausdruck. Wenn Sie einen englischsprachigen Text gewählt haben, genügt folgende Anweisung:

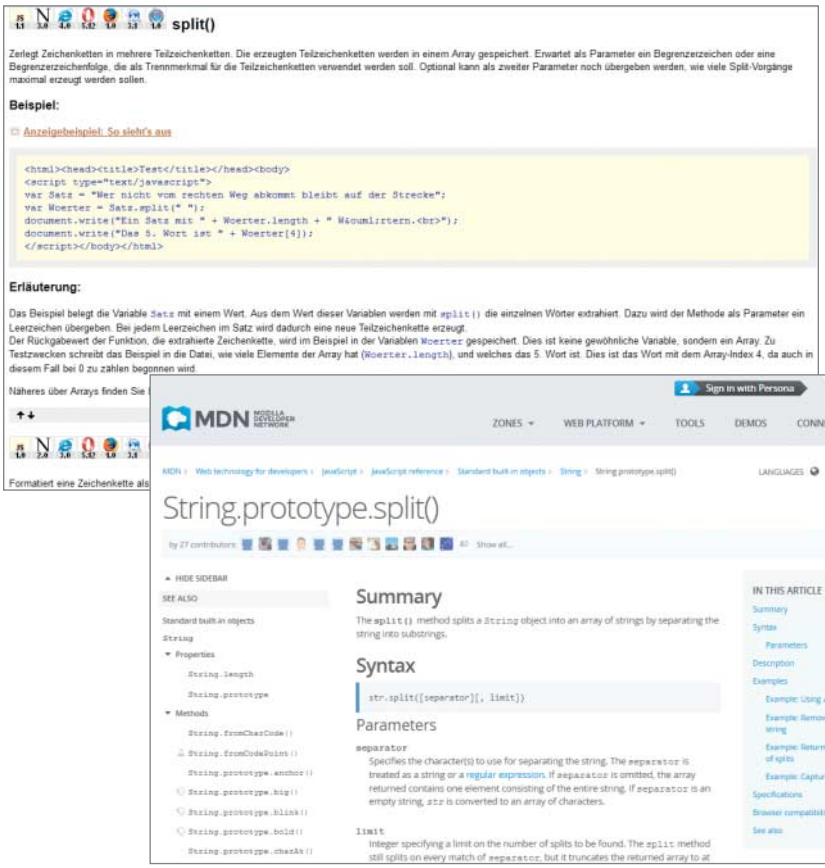
```
var words = text.split(/\W+/);
```

Reguläre Ausdrücke stehen nicht zwischen Anführungszeichen, sondern zwischen Schrägstrichen. Statt nach festgelegten Zeichen kann man nach vordefinierten Zeichenklassen suchen. So bezeichnet etwa \d eine beliebige Ziffer, \w ein sogenanntes Wortzeichen (Buchstaben, Ziffern und Unterstrich) und \W ein Nichtwortzeichen, also alles andere. Das Pluszeichen steht für eines oder mehrere der vorgenannten Zeichen. Folgen zum Beispiel ein Punkt, ein Anführungs- und ein Leerzeichen aufeinander, trennt das Skript nur einmal statt dreimal.

Dummerweise zählt JavaScript auch Umlaute zu den Nichtwortzeichen. So muss man im Deutschen mit einer selbstgebauten Zeichenklasse operieren, zum Beispiel:

```
var words = text.split(/[.„;!?"'»«\s]+/);
```

Keine Panik, so etwas schreibt sich leichter, als es sich liest. Die eckigen Klammern fassen alle darin enthaltenen Zeichen zu einer Klas-



Die deutschsprachige Referenz SelfHTML hat schon vielen Webentwicklern bei den ersten Gehversuchen geholfen, ist aber nicht mehr auf dem neuesten Stand. Mozillas Developer-Website ist aktueller, aber leider auch größtenteils auf Englisch.

se zusammen: diverse Interpunktionszeichen sowie „Whitespace“ – also Leerzeichen, Umbrüche und Tabs, die von der Zeichenklasse \s abgedeckt werden.

Damit sind die Fähigkeiten regulärer Ausdrücke gerade einmal angerissen, doch für das aktuelle Projekt haben sie ihren Job vorerst erledigt.

Listig

Sämtliche Wörter des Texts stecken nun also in einer Variablen namens words vom Typ Array (Feld, Liste). Wie so ein Array funktioniert, verstehen Sie besser, wenn Sie eines explizit anlegen:

```
var einArray = [
  "eins",
  "zwei",
  "drei"
];
```

Die einzelnen Array-Einträge werden durch Kommas getrennt und in eckige Klammern gesperrt. Die Schreibweise über mehrere Zeilen dient nur der Übersichtlichkeit – Sie können diese fünf Zeilen auch zu einer zusammenfassen. Ein Array kann Werte beliebigen Typs enthalten.

Und wie kommt man nun zum Beispiel an den Eintrag mit dem Inhalt „zwei“? So:

```
alert(einArray[1]);
```

Die Array-Einträge sind nummeriert, beginnend mit 0. Sie können auf einen dieser Einträge zugreifen, indem Sie dessen Indexnummer in eckigen Klammern hinter den Array-Namen setzen.

Wie Strings besitzen auch Arrays eine Eigenschaft namens length. Sie bezeichnet die Anzahl der Einträge. einArray.length hat also den Wert 3.

Mit Schleife

Nun braucht es ein Konstrukt, das alle Einträge im Array words nacheinander abklappert – eine Schleife. Schleifen sehen so ähnlich aus wie Bedingungen:

```
var index = 0;
while (index < 5) {
  console.log(index);
  index = index + 1;
}
```

Im Unterschied zu Bedingungen führt der Browser den Codeblock einer Schleife immer wieder aus, so lange die Bedingung (hier: index < 5) zutrifft. In dem Moment, in dem der Zähler index den Wert 5 erreicht, bricht die Schleife ab. In der Konsole erscheinen also die Zahlen von 0 bis 4.

Sollten Sie die Zeile index = index + 1; vergessen, kehrt der Browser nie mehr aus der Schleife zurück, sodass Sie den betreffenden Tab schließen müssen. Deshalb hat sich eine weniger fehleranfällige Kurzschreibweise durchgesetzt, die for-Schleife:

```
for (var index = 0; index < 5; index++) {
  console.log(index);
}
```

In den runden Klammern einer for-Schleife stehen durch Strichpunkte getrennt die drei Bestandteile, die für eine Schleife notwendig sind: eine vor dem ersten Durchlauf ausgeführte Anweisung, üblicherweise zur Deklaration einer Variablen; die Bedingung; und schließlich eine Anweisung, die der Browser nach jedem Schleifendurchlauf und vor dem neuerlichen Evaluieren der Bedingung ausführt. index++ ist die Kurzfassung von index = index + 1.

Damit haben Sie alle Werkzeuge beisammen, um das Array Element für Element durchzusehen:

```
for (var i = 0; i < words.length; i++) {
  console.log(words[i]);
}
```

Die Obergrenze für den Zähler ist die Länge des Arrays. Beträgt dessen Länge 10 Elemente, so hat das letzte die Indexnummer 9, wonach sich die Schleife beendet.

Größenvergleich

Da Sie ja schon wissen, wie Sie Werte vergleichen können, dürfte Ihnen der Rest nicht besonders schwer fallen: Bei jedem Schleifendurchlauf vergleichen Sie das jeweilige Array-Element mit dem aktuell längsten Wort, das Sie in einer Variablen zwischenlagern. Das Wort, das nach dem Ende der Schleife in dieser Variablen steht, ist das längste. Und so sieht das gesamte Skript aus:

```
var text = document.querySelector('article').innerHTML;
var words = text.split(/[.,;!?»«"'\s-]+/);
var longestWord = '';
for (var i = 0; i < words.length; i++) {
  if (words[i].length > longestWord.length) {
    longestWord = words[i];
  }
}
document.write('<p>Das mit ' + longestWord.length +
 ' Zeichen längste Wort lautet: <q>' + longestWord +
 '</q>.');
```

Die in Zeile 3 initialisierte Variable ist der Platzhalter für das längste Wort. Die Variable ist anfangs ein leerer String und hat somit die Eigenschaft length (mit dem Wert 0). Da das Array words nur Strings enthält, können Sie in Zeile 5 die Länge des aktuellen Elements words[i] herausfinden und mit der bisherigen längsten in der Variablen longestWord vergleichen.

Sie können statt größer (>) auch größer gleich (>=) als Operator benutzen; in diesem Fall enthält longestWord bei mehreren gleich langen längsten Wörtern nicht das erste, sondern das letzte. Um zu sehen, wie die Änderungen von longestWord im Detail abgelaufen sind, fügen Sie in den if-Block ein:

```
console.log(i + ': ' + words[i] + ' (' +
  words[i].length + ')');
```

Das gibt auf der Konsole zum Beispiel aus:

```
1: Als (3)
2: Gregor (6)
5: Morgens (7)
7: unruhigen (9)
```

Werk- und Spielzeug

Mit dem bisher vorgestellten Programmierwerkzeug lässt sich schon allerhand anstellen. Ganz einfach ist es zum Beispiel, statt des längsten Worts den längsten Satz herauszufinden – dazu müssen Sie den regulären Ausdruck so anpassen, dass er nach Satzzeichen plus Leerraum (Whitespace) auftrennt:

```
var sentences = text.split(/[.!?]\s+/);
```

Wie steht es mit der durchschnittlichen Wortlänge? In einer ersten Näherung könnte man `text.length` durch `words.length` teilen (beachten Sie die verschiedenen Bedeutungen von `length`). Das wäre aber nicht sehr genau, da die ganzen Leer- und Satzzeichen in die Berechnung einflössen. Aber da das Skript ohnehin schon das Array durchläuft, kann es bei der Gelegenheit auch gleich die Wortlängen addieren:

```
var wordsLengthTotal = 0;
for (var i = 0; i < words.length; i++) {
  wordsLengthTotal += words[i].length;
  // restlicher Code wie oben
}
```

Ähnlich wie bei `longestWord` initialisieren Sie eine Zahlenvariable mit dem Wert 0. Bei jedem Schleifendurchlauf zählen Sie die Länge des aktuellen Wortes hinzu; dafür setzen Sie den Operator += ein, der Addition und Zuweisung kombiniert. Analog funktionieren übrigens etwa auch -= oder *=.

Nach dem Ende der Schleife berechnen Sie den Durchschnittswert wie folgt:

```
var wordLengthAverage = wordsLengthTotal /
  words.length;
document.write('Durchschnittliche Wortlänge: ' +
  wordLengthAverage);
```

Eine kleine Schönheitskorrektur ist aber noch fällig: JavaScript führt Berechnungen auf 16 Nachkommastellen genau aus. So exakt muss es niemand wissen. Der eleganteste Weg, die Nachkommastellen zu stutzen, ist die Funktion `toFixed()`, die für alle Zahlen zur Verfügung steht. Sie rundet die Zahl auf die übergebene Anzahl von Nachkommastellen und gibt das Ergebnis als String zurück:

```
var wordLengthAverage = (wordsLengthTotal /
  words.length).toFixed(1);
```

Vertiefung

Sie haben jetzt einige der grundlegenden Konstrukte kennengelernt, die praktisch alle Programmiersprachen gemein haben: Variablen und Werte unterschiedlicher Typen, Operatoren für Zuweisungen, Berechnungen und Vergleiche, Bedingungen und Schleifen, Kommentare – und haben sogar einen flüchtigen Blick auf reguläre Ausdrücke geworfen.

Der wohl beste Weg, dieses Wissen zu vertiefen, ist Herumexperimentieren – gute Dokumentation findet sich im Web, ebenso wie fertige Codeschnipsel, die Sie studieren, anpassen und nutzen können [1].

Fortgeschritteneren Dinge wie dem Programmieren eigener Funktionen oder dem Umgang mit Objekten widmet sich ein Artikel in einem der folgenden Hefte. (ola)

Literatur

[1] JavaScript-Dokumentation: http://de.selfhtml.org/javascript (veraltet, aber anschaulich mit Beispielen); https://developer.mozilla.org/de/docs/JavaScript (aktuell und umfassend, überwiegend englischsprachig)

www.ct.de/1415166 c't