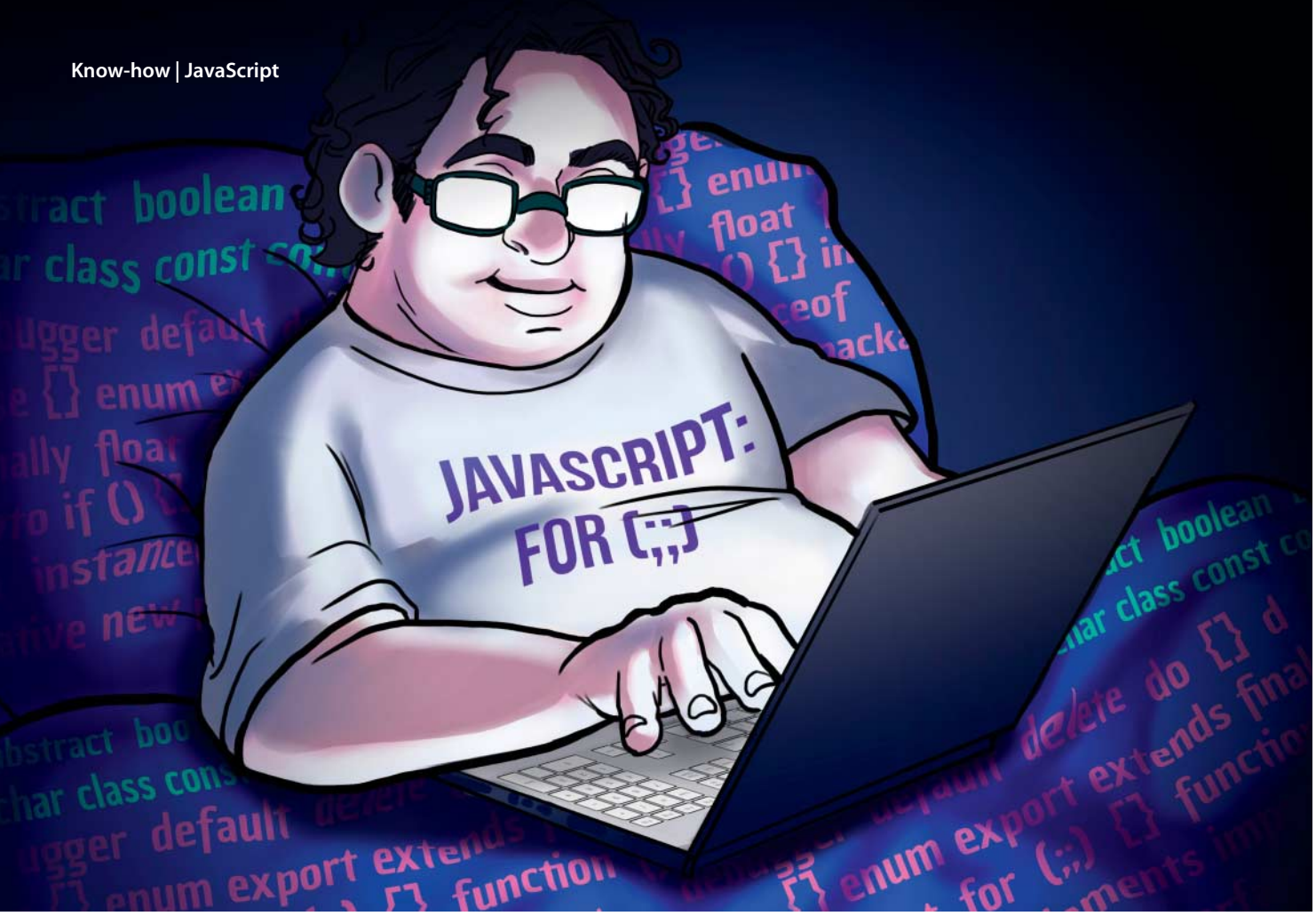


Herbert Braun

Tiefer einsteigen

Mit JavaScript Programmieren lernen, Teil 2

Objekte, Callbacks und ternäre Operatoren – das hört sich erst mal kompliziert an. Aber wenn Sie dem zweiten Teil unseres JavaScript-Einsteigerkurses Schritt für Schritt folgen, erschließt sich fast wie von selbst, was man damit anstellen kann.

Der erste Teil dieses Programmierkurses handelte von Variablen, Strings, Zahlen, Boole'schen Werten, Arrays, Anweisungen, Operatoren, Funktionsaufrufen, Bedingungen, Schleifen, regulären Ausdrücken und Kommentaren [1]. Dieser Teil führt in Objekte, die Programmierung (anonymer) Funktionen, Rückgabewerte sowie Gültigkeitsbereiche ein und stellt weitere Operatoren vor.

Er knüpft direkt an den ersten Artikel an. An dessen Ende entstand ein kleines Skript, welches das längste Wort und die durchschnittliche Wortlänge eines Textes ermittelt (siehe Listing auf der nächsten Seite).

Im Anschluss gaben document.write()-Anweisungen die Anzahl der Wörter (words.length), deren durchschnittliche Länge (wordLengthAverage) und das längste Wort aus. Der c't-Link am Artikelende führt zum Skript und den für diesen Teil relevanten Code.

Eine Durchschnittsberechnung kann für weitere statistische Auswertungen eingesetzt werden. Vielleicht wäre es dazu sinnvoll, sie in eine wiederverwendbare Funktion zu stecken – und dabei gleich den Schönheitsfehler zu beseitigen, dass die errechnete Gleitkommazahl mit einem Punkt statt des im Deutschen üblichen Kommas ausgegeben wird.

Durchschnittlich

Diese Funktion bräuchte zwei durch Kommas getrennte Argumente: die Summe und die Anzahl der Teile. Die selbstgebaute Funktion wird nach derselben Art aufgerufen wie eine in JavaScript integrierte Funktion:

```
document.write(
  'Durchschnittliche Wortlänge: ' +
  average(wordsLengthTotal, words.length) +
  ' Zeichen'
);
```

Vor dem Aufruf muss die Funktion bereits definiert worden sein – das machen Sie so:

```
var average = function(sum, number) {
  // Funktionscode ...
}
```

Man definiert eine Funktion also wie eine Variable. Der Unterschied besteht darin, dass man der Variablen keinen festen Wert zuweist, sondern eine Funktion.

Dem Schlüsselwort function folgen zwei Klammerpaare: das erste rund, das zweite geschweift. In den runden Klammern muss stehen, unter welchen Variablennamen Sie die übergebenen Funktionsargumente verwenden möchten. Ob diese Variablennamen die gleichen sind wie beim Aufruf der Funktion, spielt keine Rolle, denn die Argumente befinden sich in einem anderen Gültigkeitsbereich. Die runden Klammern sind Pflicht, selbst wenn dazwischen nichts steht und die Funktion keine Argumente erhält. Eine alternative Form derselben Definition ist:

```
function average(sum, number) {
  // Funktionscode ...
}
```

Der eigentliche Code landet in beiden Fällen in den geschweiften Klammern:

```
var average = function(sum, number) {
  var av = sum / number;
  av = av.toFixed(1);
  av = av.replace('.', ',');
}
```

Die zweite Zeile berechnet wie gehabt den Durchschnitt aus den übergebenen Werten und weist diesen der Variable av zu. Anschließend stutzt toFixed() die Zahl der Nachkomma-

stellen zusammen. Die String-Funktion replace() erwartet als Argumente ein Suchmuster als String oder regulären Ausdruck und einen Ersetzungs-String. Das Ergebnis der Ersetzung gibt replace() an den Aufrufer zurück. Im Beispiel landet es also in der Variablen av.

Abgekapselt

Wenn Sie diese Funktion und Ihren Aufruf wie oben in den Code einbauen, führt das allerdings nicht zum erhofften Erfolgserlebnis: „Durchschnittliche Länge: undefined Zeichen" antwortet der Browser. Die Berechnung ist korrekt, aber die Funktion rückt das Ergebnis in der Variablen av nicht heraus. Vielleicht klappt der Aufruf, wenn man die in der Funktion definierte Variable av ausgibt:

```
var average = function(sum, number) {
  var av = sum / number;
  // ...
}
average(wordsLengthTotal, words.length);
console.log(av);
```

Wieder ist undefined die Ausgabe auf der Konsole. Merkwürdigerweise verhält sich die Maschine so, als hätte sie von av noch nie gehört. Neuer Anlauf:

```
var av;
var average = function(sum, number) {
  av = sum / number;
  // ...
}
average(wordsLengthTotal, words.length);
console.log(av);
```

Siehe da, diesmal funktionierts: Jetzt ändert die Funktion den Wert der außerhalb von ihr definierten Variable, weil Sie in der Funktion das Wörtchen var vor av = … weggelassen haben.

Sie haben soeben beobachtet, was es mit dem Gültigkeitsbereich von Variablen (Scope) auf sich hat. Jede Programmiersprache hat dafür ihre eigenen Regeln; die von JavaScript sind ziemlich schlicht:

– Eine Variable, die in einer Funktion definiert wurde, gilt nur in dieser.
– Eine außerhalb einer Funktion definierte Variable lässt sich in der Funktion benutzen. Mit einer Ausnahme: Definiert die betreffende Funktion mit var eine gleichnamige Variable, ersetzt sie die außerhalb definierte. Es gilt dann nur noch die innerhalb definierte.

Im Unterschied zu vielen anderen Sprachen gelten diese Regeln nur in Funktionsblöcken, nicht in anderen Code-Blöcken wie zum Beispiel bei if:

```
if (true) {
  var x = 123;
}
```

Die Variable x lässt sich auch außerhalb des Blocks lesen und verändern.

Nebenwirkungen

Sie haben jetzt einen Weg kennengelernt, Ergebnisse aus einer Funktion nach außen zu tragen – und zwar den indirekten, der eine zuvor vorhandene Variable verändert. Es geht auch direkt, indem Sie einen Rückgabewert für die Funktion festlegen:

```
var average = function(sum, number) {
  // hier Code wie oben ...
  return av;
}
```

Mit return bricht die Funktion sofort ab und gibt den dahinter angegebenen Wert an den Aufrufer zurück. return ist keine Funktion, sondern eine Anweisung, die nur innerhalb einer Funktion auftreten darf. Nun können Sie das Ergebnis der Funktion einer Variablen übergeben:

```
var myAverage = average(x, y);
```

Sie können sie auch direkt ausgeben:

```
document.write(
  'Durchschnitt: ' + average(x, y)
);
```

In einer Funktion wie oben mit außerhalb definierten Variablen zu arbeiten, kann sinnvoll sein, aber üblicherweise ist es gute Praxis, eine Funktion im mathematischen Sinne zu nutzen, also den direkten Weg über Rückgabewerte zu nehmen. Die meisten der in JavaScript eingebauten Funktionen arbeiten auch so.

Statistisch

Von Statistikern hört man oft, dass der Median aussagekräftiger ist als der Durchschnitt. Dabei handelt es sich um den in der Mitte einer sortierten Zahlenreihe befindlichen Wert. 1, 2, 9 hat den Durchschnittswert 4, aber den Median 2. Das in ein Programm zu fassen, kann ja wohl nicht so schwer sein …

Um herauszufinden, welches der oder die mittleren Werte sind, muss die Funktion erst mal alle Werte bekommen. Sie brauchen also ein Array, das die Längen aller Wörter enthält. Da im vorhandenen Code ohnehin bereits eine Schleife alle Wörter durchläuft, ist das Array schnell angelegt:

```
var wordLengths = [];
for (var i = 0; i < words.length; ++i) {
  wordLengths.push(words[i].length);
  // ...
}
```

```
var text = document
  .querySelector('article').innerHTML;
var words = text.split(/[.,;!?«»"'\s-]+/);
var longestWord = '';
var wordsLengthTotal = 0;
for (var i = 0; i < words.length; i++) {
  wordsLengthTotal += words[i].length;
  if (words[i].length > longestWord.length) {
    longestWord = words[i];
  }
}
var wordLengthAverage =
  (wordsLengthTotal / words.length).toFixed(1);
```

Das Skript aus dem ersten Teil ermittelt statistische Daten zu einem vorgegebenen Text.

Die eckigen Klammern definieren ein leeres Array. In der Schleife hängt die push()-Funktion bei jedem Durchlauf einen neuen Wert an das Ende des Arrays an, hier die Länge des aktuellen Wortes. Nach dem letzten Schleifendurchlauf enthält wordLengths so viele Zahlen, wie der Text Wörter hat. Dieses Array wird später der Parameter für eine neue Funktion sein:

```
var median = function(values) {
  // ...
}
```

Zur Ermittlung des Medians muss das übergebene Array sortiert werden. Hierfür bringt JavaScript eine Funktion namens sort() mit. Sie sortiert das Array aufsteigend und gibt es zurück. Testen Sie mit console.log(), ob alles funktioniert wie geplant:

```
var median = function(values) {
  values.sort();
  console.log(values);
}
```

Unerfreuliche Überraschung: Die ausgegebene Liste ist in der Reihenfolge 1, 10, 10, 11 … angeordnet. Standardmäßig sortiert sort() nämlich lexikografisch, selbst wenn alle enthaltenen Array-Bestandteile Zahlen sind – unpassend für die gestellte Aufgabe.

Anonym

Um die Aufgabe zu lösen, müssen Sie wissen, dass die Funktion sort() beim Sortieren immer zwei Elemente miteinander vergleicht. Sie vertauscht deren Positionen immer dann, wenn das Sortierkriterium nicht erfüllt ist. Am Ende ist das Array sortiert.

sort() nimmt ein solches Sortierkriterium als optionalen zweiten Parameter entgegen. In Code formuliert:

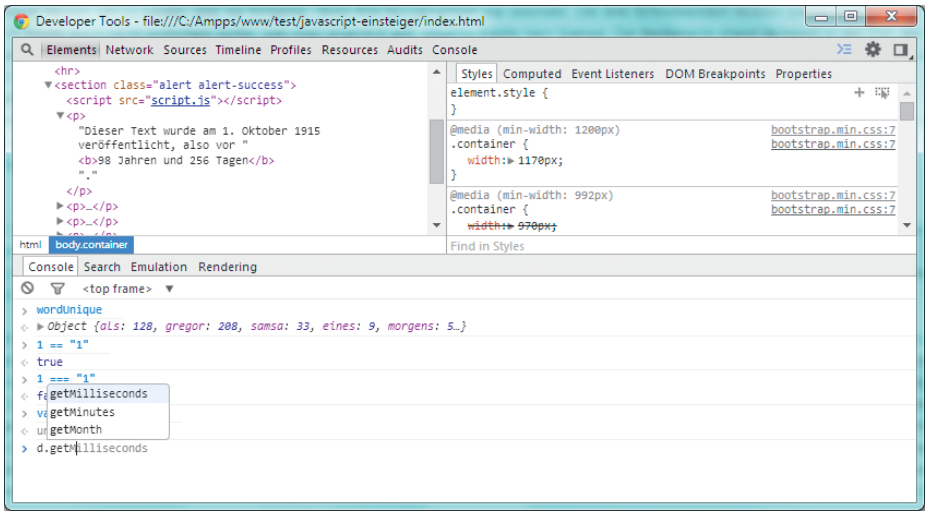
```
values.sort(function(a, b) {
  // a und b umsortieren?
});
```

Eine Funktion als Funktionsargument, und dann noch ohne Namen, also anonym – das sieht erst einmal komisch aus, ist aber letztlich auch nichts anderes als bei String, Zahl oder Array. Schwierig ist allenfalls, die Klammern in der richtigen Reihenfolge zu schließen. Vielleicht finden Sie es daher übersichtlicher, statt mit einer anonymen Funktion mit einer Variablen zu arbeiten:

```
var numsort = function(a, b) {
  // a und b umsortieren?
};
var median = function(values) {
  values.sort(numsort);
}
```

Beachten Sie, dass die Funktion hier ohne das übliche Klammernpaar referenziert wird – andernfalls würde sort() nämlich nicht wie erforderlich die Funktion selbst bekommen, sondern deren Rückgabewert.

sort() erwartet von der Funktion mit dem Sortierkriterium als Rückgabewert eine Zahl. Ist diese kleiner oder gleich 0, bleibt die Rei-



Nicht alles muss man im Skript-Code testen: Das Debugger-Werkzeug des Browsers lädt mit seiner Code-Vervollständigung zum Ausprobieren und Kennenlernen der Sprache ein.

henfolge der verglichenen Elemente wie zuvor, sonst vertauscht sort() sie. Also:

```
var numsort = function(a, b) {
  return a - b;
};
```

Steht vorne eine größere Zahl, gibt der Algorithmus einen positiven Wert zurück und veranlasst die Umsortierung.

Modulo

Aus dem sortierten Array gilt es, den oder die mittleren Werte herauszufinden. Bei einer ungeraden Länge hat das mittlere Element von values den Index (values.length – 1) / 2 (nicht vergessen: Indexnummern beginnen mit 0); bei einer geraden Anzahl von Elementen ist beim Median der Mittelwert aus den beiden mittleren Zahlen gefragt, also denen mit den Indexnummern values.length / 2 und values.length / 2 – 1.

Für diese Fallunterscheidung erweist sich der Modulo-Operator % als nützlich. Er ermittelt bei einer Division den Rest. 10 % 3 ergibt 1, ein Rest von 0 käme etwa bei 21 % 7 heraus. Bei allen ungeraden Zahlen gilt demnach x % 2 == 1. Damit können Sie die Median-Funktion zusammensetzen:

```
var median = function(values) {
  values.sort(numsort);
  if (values.length % 2 == 1) {
    return values[(values.length - 1) / 2];
  }
  else {
    var tmp = values[values.length / 2 - 1] +
      values[values.length / 2];
    return average(tmp, 2);
  }
};
```

Bei ungerader Array-Länge ermittelt das Skript für die if-Bedingung den Wert 1, also wahr. In diesem Fall kann die Funktion das Ergebnis sofort zurückgeben. Bei gerader Länge muss sie den Durchschnitt aus zwei benachbarten Werten bilden, was sie mit der zuvor entstandenen average()-Funktion erledigt.

Rückruf

Funktionen, die an eine andere Funktion übergeben werden, um sie darin zu gegebener Zeit aufzurufen, bezeichnet man als Callback-Funktion oder kurz Callback. Ein einfaches Beispiel ist die Array-Funktion forEach (callback): Sie ruft für jedes Array-Element die Funktion callback() auf. Dabei übernimmt sie den Array-Wert und dessen Indexnummer als Parameter. Damit können Sie die im ersten Artikel verwendete for-Schleife über alle Wörter ersetzen. Statt

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

schreiben Sie

```
words.forEach(function(word, i) {
  wordLengths.push(word.length);
});
```

Ob man die for-Schleife oder Array.forEach() bevorzugt, ist eine Geschmacksfrage.

Objekte

Weitere Fragen bezüglich des Textes könnten sein: Welches ist das meistgenutzte Wort? Welches Wort kommt nur einmal vor? Wie viele unterschiedliche Wörter gibt es? Um diese Fragen zu beantworten, muss man wieder eine Schleife über alle Wörter laufen lassen.

Der bisher erarbeitete Code ist ein gutes Fundament für die Programmierung der Suche nach den Worthäufigkeiten. Beginnen Sie mit der Umwandlung aller Wörter in Kleinbuchstaben mit der String-Funktion toLowerCase(), um am Satzanfang groß geschriebene Wörter nicht gesondert zu erfassen:

```
var text = ...
var words = text.split(...);
words.forEach(function(word) {
  var lcWord = word.toLowerCase();
  // ...
});
```

Am einfachsten legen Sie die Wörter und ihre Häufigkeiten in einem Objekt ab. Wie das Array ist auch ein Objekt eine Sammelvariable, nur dass die einzelnen Elemente nicht nummeriert und sortiert, sondern benannt sind. Arrays bestehen aus Elementen, Objekte hingegen aus Eigenschaften. Der schnellste Weg, ein neues Objekt anzulegen, sieht so aus:

```
var wordUnique = {};
```

Statt eckigen Klammern wie beim Array stehen hier geschweifte.

Innerhalb der Schleife beziehungsweise Callback-Funktion genügen zwei Zeilen zum Zählen der Wörter:

```
words.forEach(function(word) {
  // ...
  if (wordUnique[lcWord] == undefined)
    wordUnique[lcWord] = 0;
  ++wordUnique[lcWord];
});
```

Wie beim Array können Sie die Eigenschaften eines Objekts mit objekt[eigenschaft] ansprechen. Wenn es eine Eigenschaft nicht gibt (das betreffende Wort also noch nicht vorkam), lautet ihr Wert undefined. In diesem Fall setzt der obige Code den Wert auf 0.

Vielleicht vermissen Sie die geschweiften Klammern um wordUnique[lcWord] = 0. Diese sind nicht notwendig, wenn der if-Block nur aus einer einzelnen Anweisung besteht. Abschließend erhöht der Operator ++ den Zähler um 1.

Was dabei herausgekommen ist, zeigt Ihnen die Eingabe von wordUnique in der Browser-Konsole:

```
{als: 128, gregor: 208, samsa: 33, eines: 9, morgens: 5,
3623 weitere ...}
```

Die durch Doppelpunkt aufgeteilten Name-Wert-Paare stehen durch Kommas getrennt in geschweiften Klammern. Auf diese Weise können Sie Objekte auch im Code angeben. Der Schlüssel braucht dabei keine Anführungszeichen, wenn im Text nicht gerade ein reserviertes Wort wie if oder function vorkommt. Diese Schreibweise in Form eines sogenannten Objektliterals ist auch Grundlage des verbreiteten Datenformats JSON (JavaScript Object Notation).

Eigenschaften

Auf der Konsole können Sie schon einmal bequem herausfinden, wie oft ein bestimmtes Wort vorkommt: Geben Sie dort beispielsweise wordUnique["der"] ein.

Das gleiche Ergebnis zeigt wordUnique.der an. Diese einfachere Schreibweise können Sie verwenden, wenn der Name der Eigenschaft auch ein Variablennamen sein könnte, also nur aus den Buchstaben a bis z und

A bis Z, Zahlen, Unterstrich sowie Dollar-Zeichen besteht. Beispiele: „Mann“ erfüllt dieses Kriterium, weshalb Sie wordUnique.Mann schreiben können. Der String „naiv-optimistisch“ enthält ein Minus-Zeichen, das als Subtraktionsoperator interpretiert wird, weshalb der Ausdruck wordUnique.naiv-optimistisch nicht die Eigenschaft naiv-optimistisch referenziert, sondern nur naiv, und davon den Wert der Variablen optimistisch subtrahiert. Um tatsächlich die Eigenschaft "naiv-optimistisch" anzusprechen, müssen Sie also wordUnique["naiv-optimistisch"] schreiben.

In wordUnique haben alle Eigenschaften Integer-Werte. Doch nicht alle Werte von Objekteigenschaften gehören wie Zahlen, Strings, Boole-Werte oder undefined zu den sogenannten primitiven Werten. In einem Objekt können auch Funktionen, Arrays oder andere Objekte unterkommen:

```
var ehefrau = {
  name: "Anja",
  alter: 32,
  verheiratet: true,
  kinder: ["Tim", "Tom"],
  auto: {
    farbe: "rot",
    ps: 85
  },
  gruessen: function() {
    alert("Hallo!");
  }
};
```

Eine ziemlich übersichtliche Art, eine komplexe Datenstruktur zu notieren, oder? ehefrau.kinder[1] verweist auf Tom, ehefrau.auto.ps entspricht 85, ehefrau.gruessen() gibt „Hallo!“ aus.

Das letzte Beispiel dürfte Sie an eine Syntax erinnern, die Sie schon öfter gesehen haben: Ob in document.querySelector(…), console.log(…) oder words.forEach(…) – die meisten eingebauten Funktionen hängen an einem übergeordneten Objekt. Solche Funktionen bezeichnet man als Objektmethoden.

querySelector() ist also eine Methode des document-Objekts, log() eine des console-Objekts. Aber words mit seiner forEach()-Methode war doch ein Array, und word.toLowerCase() bezog sich auf einen String?

Tatsächlich ist ein Array auch ein Objekt, eben vom Typ Array. Eine Variable wie wordUnique ist ein Objekt vom Typ Object.

Objektbesichtigung

Nun stecken also die Worthäufigkeiten im Objekt wordUnique – aber wie findet man darin die häufigsten und die einmaligen Wörter?

Dazu müssen Sie einmal komplett wordUnique durchblättern. Das einfachste Verfahren ist eine for-in-Schleife, die alle Eigenschaften des Objekts durchläuft:

```
for (var key in wordUnique) {
  // ...
}
```

key enthält den Namen der Eigenschaft, also das Wort; wordUnique[key] verweist auf die Anzahl.

Der Rest fällt nicht schwer:

```
var wordCount = 0,
  usedOnce = [],
  mostPopular = {count: 0};
for (var key in wordUnique) {
  ++wordCount;
  if (wordUnique[key] === 1)
    usedOnce.push(key);
  if (wordUnique[key] > mostPopular.count) {
    mostPopular.word = key;
    mostPopular.count = wordUnique[key];
  }
});
```

Es empfiehlt sich aus Gründen der Übersichtlichkeit, die Variablen wie im obigen Beispiel en bloc zu definieren. So müssen Sie nicht jedes Mal var xyz = 123; schreiben, sondern können nach einmaligem var eine kommagetrennte Liste folgen lassen – je nach Bedarf mit oder ohne Wertzuweisungen.

Anders als bei Arrays lässt sich die Anzahl der Objekteigenschaften nicht per .length auslesen, weshalb mit ++wordCount bei jedem Durchlauf der Zähler um eins erhöht wird. Kommt das Wort nur einmal vor, so kopiert die zweite Zeile innerhalb der Schleife dieses in das Array usedOnce.

Der Code initialisiert die Eigenschaft count des Objekts mostPopular mit dem Wert 0. Jedes Mal, wenn ein Wort häufiger vorkommt als in mostPopular.count vermerkt, aktualisiert die Callback-Funktion die Werte der Eigenschaften word und count.

Vielleicht sind Sie über die drei Gleichheitszeichen bei if (wordUnique[key] === 1) gestolpert. JavaScript nimmt es bei Vergleichen mit den Variablentypen nicht so genau: Wenn Sie auf der Konsole den Ausdruck 2 == "2" eingeben, evaluiert der Interpreter diesen als wahr, obwohl der Vergleich zwischen Zahl und String eigentlich scheitern sollte. Einen typsicheren Vergleich erhalten Sie erst mit drei Gleichheitszeichen: 2 === "2" ergibt false. Typsicherheit bedeutet hier, dass der Vergleich nur dann wahr ergeben kann, wenn außer dem Wert auch die Typen der verglichenen Werte gleich sind.

Für das Beispielprojekt wäre derlei Genauigkeit nicht notwendig, aber anderswo kann Ihnen das schwierige Fehlersuchen ersparen. Darum: Wenn Sie sicherstellen wollen, dass 2 immer nur gleich 2 ist, aber niemals gleich "2", dann verwenden Sie den typsicheren Vergleich mit === beziehungsweise seine Negierung !==. Für Größer- und Kleiner-Vergleiche gibt es keine typsicheren Entsprechungen.

Nun können Sie mit document.write() die Ergebnisse Ihrer Berechnungen ausgeben: mostPopular.word und mostPopular.count enthalten Informationen zum meistgenutzten Wort, wordCount die Anzahl unterschiedlicher Wörter, usedOnce.length die der nur einmal verwendeten.

Zufällig

Die Liste usedOnce können Sie jetzt sicher ohne Hilfe komplett ausgeben. Doch kann sie sehr lang werden, denn in vielen Texten kommt jedes zweite Wort nur einmal vor. Vielleicht wäre es eine gute Idee, von diesen Exoten nur eine zufällige Auswahl hinzuschreiben?

Für Zufälle ist random() zuständig, eine Methode, die zum in JavaScript eingebauten Objekt Math gehört. Sie produziert eine Gleitkommazahl, die größer gleich 0 und kleiner als 1 ist. Probieren Sie das ruhig mal auf der Konsole aus. Folgende Rechenvorschrift errechnet gleichmäßig über das Array verteilte Zufallszahlen:

```
Math.floor(Math.random() * usedOnce.length)
```

Die Funktion Math.floor() wandelt eine Gleitkommazahl in die nächstniedrige Ganzzahl um; die Nachkommastellen schneidet sie dazu einfach ab. Die vom obigen Ausdruck produzierten Werte reichen demnach von einschließlich 0 bis usedOnce.length – 1.

Dieser Zufallszahlengenerator steckt zusammen mit der Ausgabe in einer Schleife. Um zu verhindern, dass random() zufällig mehrmals den gleichen Array-Eintrag auswählt, speichern Sie die bereits gewählten in ein Array:

```
var randoms = [];
while (randoms.length < 5) {
  if (randoms.length === usedOnce.length)
    break;
  var random =
    Math.floor(Math.random() * usedOnce.length);
  if (randoms.indexOf(random) === -1) {
    document.write('<li>' +
      usedOnce[random] + '</li>');
    randoms.push(random);
  }
}
```

Die while-Schleife soll fünfmal durchlaufen werden, sofern usedOnce ausreichend Wörter enthält. Andernfalls sorgt die Anweisung break dafür, dass der Interpreter die Schleife vorzeitig verlässt.

Die zweite if-Bedingung überprüft, ob der von random() ausgewürfelte Wert schon in randoms enthalten ist. Dabei kommt die Array-Methode indexOf() zum Einsatz, die den Array-Index mit dem betreffenden Wert zurückgibt. Falls er nicht enthalten ist, lautet das Ergebnis –1. Nur dann gibt document.write() das entsprechende Wort aus und hängt die Zufallszahl an die Liste randoms an.

JavaScript heute

Zum Abschluss verraten Sie dem Nutzer spaßeshalber noch, wie alt der Text in Jahren und Tagen ist. Dazu erzeugen Sie zunächst ein Datumsobjekt mit dem Entstehungstag des Textes:

```
var pubdate = new Date("1915-10-01");
```

Bisher haben Sie Objekte mit {} und Array-Objekte mit [] initialisiert, also ganz anders als im obigen Beispiel. Tatsächlich ist die bisher genutzte Form myObject = {} eine Abkürzung für myObject = new Object() und myArray = [] eine für myArray = new Array(). Der Standardweg, um ein Objekt zu generieren, führt über den Operator new und den Objektnamen, zum Beispiel Object, Array, String oder eben Date.

pubdate ist somit ein Objekt vom Typ Date und kennt damit dessen Methoden, die den Umgang mit Datum und Uhrzeit erleichtern, zum Beispiel:

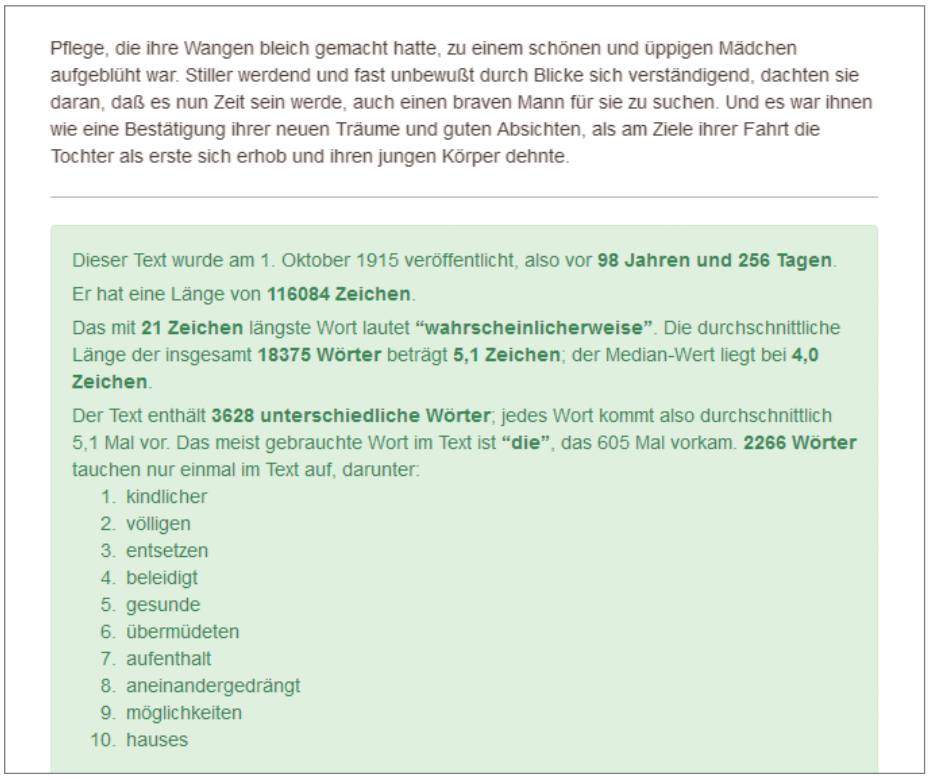
```
document.write(
  pubdate.getDate() + '.' +
  pubdate.getMonth() + '.' +
  pubdate.getFullYear()
);
```

getDate(), getMonth() und getFullYear() geben Monatstag, Monat und Jahr des jeweiligen Datums aus. Allerdings ist aus dem 1. Oktober merkwürdigerweise der 1.9. geworden, denn getMonth() zählt ab 0. Für die numerische Ausgaben des Monats ist das ungünstig, bei der Arbeit mit Arrays hingegen praktisch:

```
var months = ['Januar', 'Februar', ...];
document.write(months[pubdate.getMonths()]);
```

Array-Eintrag Nummer 9 enthält den Oktober; die Ausgabe ist also korrekt. Um die numerische Ausgabe zu korrigieren, addieren Sie einfach 1 zur Monatszahl:

Mit ungefähr 80 Zeilen JavaScript-Code kann man bereits eine Menge über einen Text herausfinden.



```
document.write(
  pubdate.getDate() + '.' +
  (1 + pubdate.getMonth()) + '.' +
  pubdate.getFullYear()
);
```

Der einfachste Weg, um die Differenz zwischen zwei Daten zu ermitteln, führt über die Umrechnung in Millisekunden. Dafür gibt es die Methode getTime():

```
var datediff = Date.now() - pubdate.getTime();
```

Um das heutige Datum herauszufinden, könnten Sie new Date() ohne Argument aufrufen und getTime() folgen lassen. Noch schneller geht es mit der Date-Methode now(), die Millisekunden zurückgibt.

Nun errechnen Sie die Differenz in Tagen:

```
datediff /= 1000 * 60 * 60 * 24;
datediff = Math.floor(datediff);
```

Die Uhrzeit in pubdate steht implizit auf 0 Uhr, wenn man sie wie oben weglässt, weshalb das aktuelle Datum abgerundet werden muss. Schließlich rechnen Sie die Tage in Jahre um:

```
var years = Math.floor(datediff / 365.2425);
var days = Math.round(datediff % 365.2425);
```

Der Modulo-Operator liefert den Rest an Tagen. Die Annahme von durchschnittlich 365,2425 Tagen pro Jahr ist eine passable Näherung; für mehr Genauigkeit müsste man die exakte Folge von Schaltjahren einbeziehen. Nun fehlt nur noch die Ausgabe:

```
document.write(
  "<p>Dieser Text erschien vor " +
  (years ? years + " Jahren und " : "") +
  days + " Tagen.</p>"
);
```

In den inneren runden Klammern sehen Sie einen sogenannten ternären Operator im Einsatz. Das ist der einzige Operator, der drei Operanden benötigt:

```
document.write(years ? years + " Jahre" : "");
```

ist eine praktische Abkürzung für:

```
if (years) {
  document.write(years + " Jahre");
}
else {
  document.write("");
}
```

Die Bedingung endet auf einem Fragezeichen. Trifft sie zu, gibt der Operator den Mittelteil zwischen Fragezeichen und Doppelpunkt zurück, andernfalls den Schlussteil hinter dem Doppelpunkt. Konkret bedeutet das: Beträgt der Wert von years 0, fügt der Ausdruck einen leeren String ein (also nichts), andernfalls die Anzahl der Jahre.

Bestandsaufnahme

Zu diesem Zeitpunkt sollten Sie Funktionen und Objekte einsetzen können sowie die Bedeutung von Rückgabewerten, Callbacks, anonymen Funktionen und Gültigkeitsbereichen kennen. Die wichtigsten JavaScript-Konstrukte lassen sich mit Abstrichen auf andere Programmiersprachen übertragen. Gewappnet mit einer guten Dokumentation wie [2] sollten Sie nun einfache Projekte programmieren können. Und das ist auch der beste Weg, die Feinheiten und praktischen Probleme auszuloten, die in den zahlreichen Objekten und Funktionen sowie in der grundlegenden Syntax stecken.

Im nächsten Teil dieses Kurses werden Sie noch mehr über die Arbeit mit Objekten erfahren. (ola)

Literatur

[1] Herbert Braun, Hallo, Web!, Mit JavaScript Programmieren lernen, Teil 1, c't 15/14, S. 166

[2] Mozillas JavaScript-Dokumentation (MDN): https://developer.mozilla.org/en-US/docs/Web/JavaScript

www.ct.de/1416168 c't