O'REILLY®

JavaScript Programmierung

von Kopf bis Fuß

Nehmen Sie sich vor typischen JavaScript-Fallstricken in Acht

Ein Lernbuch zur JavaScript-Programmierung

Starten Sie Ihre neue Karriere in einem Kapitel

Vermeiden Sie peinliche Fehler bei der Typumwandlung

Trainieren Sie Ihre Denkmuskulatur mit 120 Rätseln und Übungen

Erfahren Sie, warum alles, was Ihre Freunde zu Funktionen und Objekten wissen, vermutlich falsch ist

Eric Freeman & Elisabeth Robson

Deutsche Übersetzung von Jørgen W. Lang

Der Inhalt (in der Übersicht)

Der Inhalt (jetzt ausführlich)

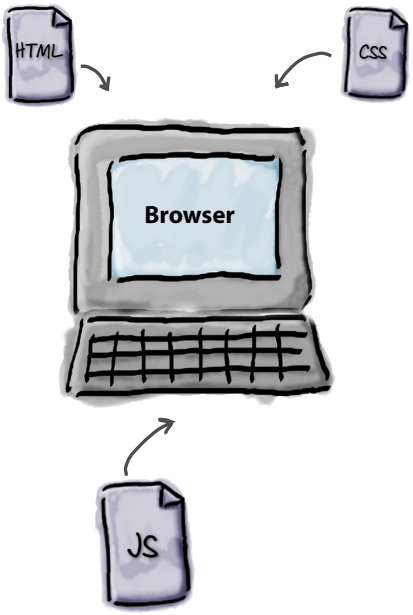
Einführung

Ihr Gehirn und JavaScript. *Sie* versuchen, etwas zu lernen, und Ihr *Hirn* tut sein Bestes, damit das Gelernte nicht *hängen bleibt*. Es denkt nämlich: »Wir sollten lieber Platz für wichtigere Dinge lassen, z. B. für das Wissen darüber, welche Tiere einem gefährlich werden könnten oder dass es eine ganz schlechte Idee ist, nackt Snowboard zu fahren.« Tja, wie schaffen wir es nun, Ihr Gehirn davon zu überzeugen, dass Ihr Leben davon abhängt, etwas über JavaScript-Programmierung zu wissen?

Ein Sprung ins kalte JavaScript-Wasser

1 Zeit für nasse Füße

JavaScript verleiht Superkräfte. Es ist **die Programmiersprache** des Webs. Mit JavaScript bekommen Ihre Webseiten nämlich **Verhalten**. Das heißt: keine trockenen, langweiligen statischen Seiten mehr, die einfach nur dasitzen und Sie anstarren. Mit JavaScript erreichen Sie Ihre Benutzer direkt. Sie können auf interessante Events reagieren, Daten aus dem Web einbinden, Diagramme direkt in Ihren Seiten erstellen und vieles mehr. Haben Sie JavaScript einmal verstanden, können Sie vollkommen neue Verhaltensweisen für Ihre Benutzer erstellen.

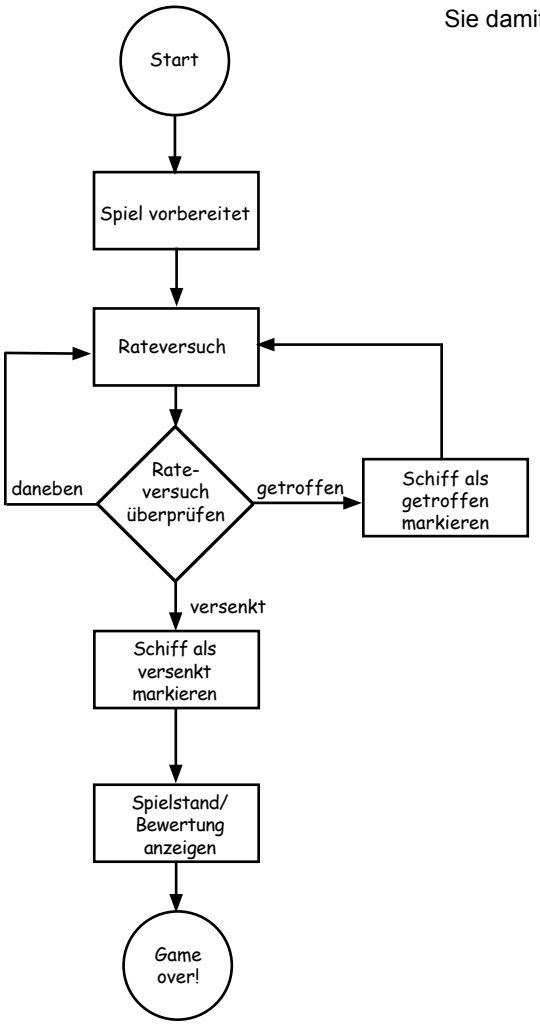


Richtigen Code schreiben

2

Die nächsten Schritte

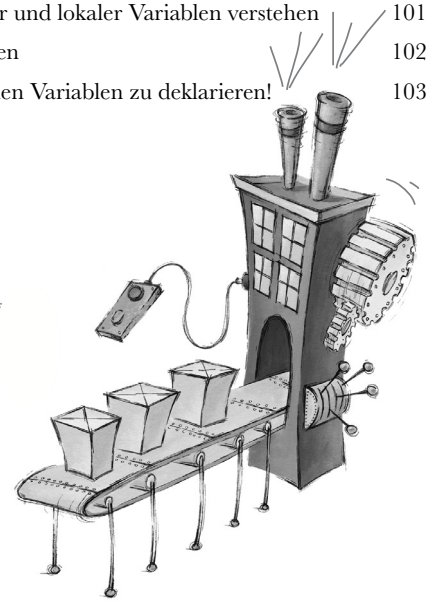
Mittlerweile kennen Sie schon Variablen, Typen, Ausdrücke und noch mehr. Sie wissen bereits ein paar Dinge über JavaScript. Das heißt, Sie können schon **richtigen Code** schreiben. Code, der etwas Interessantes tut – Code, den andere Leute benutzen wollen. Jetzt fehlt Ihnen nur noch die **richtige Erfahrung** beim Schreiben von Code. Und das wollen wir hier und jetzt ändern. Wie? Indem wir kopfüber ins kalte Wasser springen und ein komplettes JavaScript-Spiel programmieren. Klingt ehrgeizig, aber wir lassen Sie nicht hängen. Auf geht's, lassen Sie uns direkt loslegen. Und wenn Sie damit eine neue Firma gründen wollen – nur zu! Der Code gehört Ihnen.

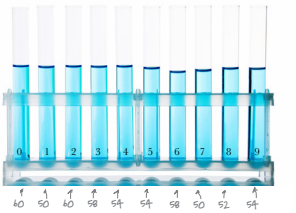


Einführung in Funktionen

3 Jetzt funkt's

Hier kommt Ihre erste Superkraft. Inzwischen können Sie schon ein wenig programmieren. Daher wollen wir Ihnen jetzt das Konzept der **Funktionen** vorstellen. Funktionen geben Ihnen die Macht, Code zu schreiben, der unter verschiedensten Umständen **wiederverwendet** werden kann. Dieser Code ist deutlich besser **wartbar**, er kann **abstrahiert** und mit einem einfachen Namen versehen werden. Dadurch können Sie die Komplexität ignorieren und sich auf die wichtigen Dinge konzentrieren. Funktionen sind ein wichtiger Schritt auf Ihrem Weg vom Skripter zum Programmierer und der Schlüssel zum Programmierstil von JavaScript. In diesem Kapitel zeigen wir Ihnen die Grundlagen: die Mechanik, die ganzen kleinen Details, die zum Verständnis von Funktionen wichtig sind. Danach werden wir Ihre Kenntnisse im Laufe dieses Buchs immer mehr erweitern und vertiefen. *Jetzt* wollen wir aber erst einmal solides Fundament schaffen.





Etwas Ordnung in die Daten bringen

4 Arrays

JavaScript besteht nicht nur aus Zahlen, Strings und Boolesche Werten. Bisher haben Sie in Ihrem JavaScript-Code nur primitive Datentypen – einfache Strings, Zahlen und Boolesche Werte wie »Fido«, 23 und true – benutzt. Damit lässt sich schon eine Menge anstellen, aber irgendwann müssen Sie mit **mehr Daten** arbeiten. Das könnten alle Artikel in einem Warenkorb sein oder alle Titel einer Playlist oder eine Liste von Sternen und ihre scheinbare Größe oder ein kompletter Produktkatalog. Dafür brauchen wir etwas mehr *Wumms*. Der Datentyp der Wahl für geordnete Daten dieser Art ist das JavaScript-**Array**. In diesem Kapitel zeigen wir Ihnen, wie Sie Ihre Daten in einem Array ablegen, es weitergeben und damit arbeiten können. Später in diesem Buch werden Sie weitere Möglichkeiten kennenlernen, Ihre **Daten zu strukturieren**; jetzt wollen wir uns aber erst mal mit Arrays befassen.

Objekte verstehen

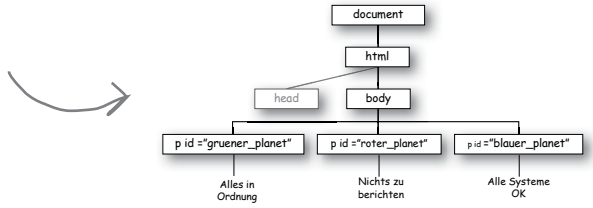
5 Ein Ausflug nach Objectville

Bis jetzt haben Sie in Ihrem Code nur »primitive« Datentypen und Arrays benutzt. Die Programmierung sind Sie bisher eher **prozedural** angegangen um mit einfachen Anweisungen, Bedingungen, for-/while-Schleifen und Funktionen, die nicht gerade objektorientiert waren. Eigentlich waren sie *überhaupt nicht* objektorientiert. Wir haben zwar hier und da, ohne es zu wissen, ein paar Objekte benutzt, aber eigene Objekte haben Sie noch nicht geschrieben. Jetzt ist es Zeit, diese langweilige prozedurale Stadt zu verlassen und selbst ein paar **Objekte** zu erstellen. In diesem Kapitel werden Sie lernen, wie Sie sich das Leben deutlich erleichtern können, zumindest was die **Programmierung** angeht. (Im selben Buch auch noch Ihrem modischen Geschmack auf die Sprünge zu helfen, ist dann doch ein bisschen viel verlangt.) Eine kleine Warnung: Haben Sie die Objekte einmal kennengelernt, wollen Sie nicht wieder zurück. Schicken Sie uns eine Postkarte, wenn Sie angekommen sind.

Interaktion mit der Webseite

6 Das DOM kennenlernen

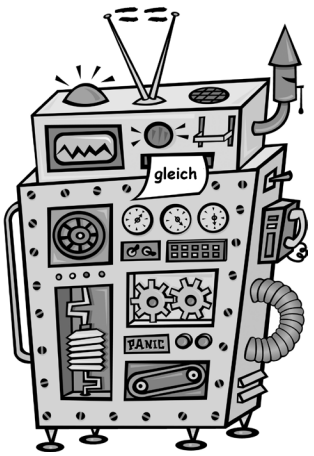
Ihre JavaScript-Fähigkeiten haben große Fortschritte gemacht. Mittlerweile haben Sie sich von einem Anfänger zu einem Skripter, ja sogar zu einem echten **Programmierer** entwickelt. Es fehlt aber noch etwas. Damit sich Ihre JavaScript-Kenntnisse wirklich entfalten können, brauchen Sie eine Möglichkeit, mit der Webseite, die Ihren Code enthält, zu interagieren. Nur so ist es möglich, **dynamische** Seiten zu schreiben, die reagieren, antworten und sich nach dem Laden automatisch aktualisieren. Wie funktioniert diese Interaktion? Über das **DOM**, das **Document Object Model**. Wir werden uns das DOM in diesem Kapitel Stück für Stück vornehmen und sehen, wie es zusammen mit JavaScript dazu bewegt werden kann, Ihrer Seite ein paar neue Tricks beizubringen.



Typen, Gleichheit, Umwandlung und der ganze Rest

7 Seriöse Typen

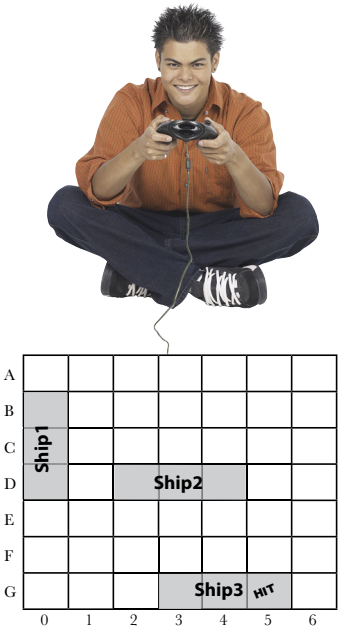
Jetzt wollen wir einmal ernsthaft über Typen reden. Eine der guten Seiten von JavaScript ist, dass Sie auch ohne viel Detailwissen schon sehr weit kommen können. Aber um **die Sprache wirklich zu meistern**, die Gehaltserhöhung zu bekommen und es im Leben richtig zu etwas bringen, müssen Sie sich extrem gut mit **Typen** auskennen. Wissen Sie noch, was wir am Anfang über JavaScript gesagt haben? Dass es keine auf dem Silbertablett servierte, durch akademische Kreuzgutachten überprüfte Sprache ist? Das stimmt wohl, aber das akademische Leben hat Steve Jobs und Bill Gates nicht von ihrem Erfolg abgehalten, und JavaScript hält es auch nicht ab. Das bedeutet, dass JavaScript … nun ja … kein besonders durchdachtes Typensystem besitzt und wir noch ein paar **Eigenheiten** bemerken werden. Aber machen Sie sich keine Sorgen! In diesem Kapitel werden wir die Sache in den Griff bekommen, und Sie werden schnell in der Lage sein, den peinlichen Momenten, die Ihnen mit Typen begegnen können, aus dem Weg zu gehen.



Die Einzelteile zusammenfügen

8 Eine App schreiben

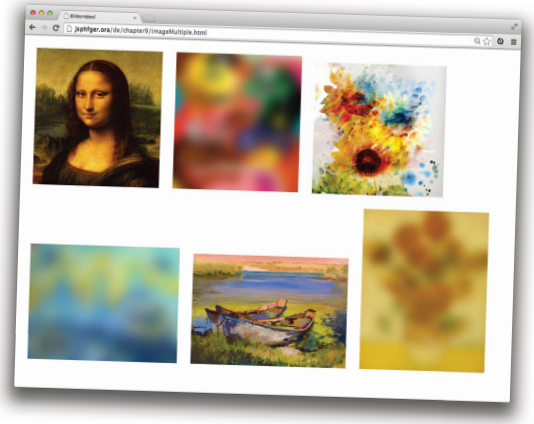
Greifen Sie sich Ihren Werkzeugkasten. Damit meinen wir den Werkzeugkasten mit all Ihren neuen Programmierkenntnissen, dem Wissen über das DOM und sogar mit etwas HTML und CSS. In diesem Kapitel bringen Sie alles zusammen und erstellen Ihre erste richtige **Webapplikation** – keine **albernen Kinderspiele** mit nur einem Schiff, das auf einer Zeile versteckt ist. Jetzt bauen wir **das komplette Spiel**: ein schönes Spielfeld mit mehreren Schiffen und Benutzereingaben direkt auf der Webseite. Die nötige Seitenstruktur erstellen wir mit HTML, visuelle Stile schreiben wir in CSS, und mit JavaScript programmieren wir das Verhalten für das Spiel. Machen Sie sich bereit: Dies ist ein richtiges Entwicklungskapitel, in dem wir ernsthaften Code schreiben.



Asynchron programmieren

9 Events be-handeln

Nach diesem Kapitel werden Sie merken, dass Sie nicht mehr in Kansas sind. Bis jetzt haben Sie Code geschrieben, der typischerweise von oben nach unten ausgeführt wird. Sicher, Ihr Code ist vermutlich etwas komplexer und enthält ein paar Funktionen, Objekte und Methoden, aber irgendwann wird auch dieser Code ausgeführt. Es tut uns wirklich leid, wenn wir Ihnen das erst so spät im Buch sagen, aber **so schreibt man eigentlich keinen JavaScript-Code**. Stattdessen wird JavaScript meistens geschrieben, um auf **Ereignisse (Events) zu reagieren**. Was für Events? Alle möglichen! Das kann ein Mausklick auf Ihrer Seite, ankommende Daten aus dem Netzwerk, ein im Browser ablaufender Timer oder eine Änderung im DOM sein, um nur ein paar zu nennen. Eigentlich passieren **ständig** irgendwelche Events im Hintergrund des Browsers. In diesem Kapitel werden wir unsere Art, in JavaScript zu programmieren, neu überdenken und lernen, warum und wie Sie Code schreiben können, der auf Events reagiert.



Funktionen erster Klasse

10 Befreite Funktionen

Erst Funktionen machen Sie zum Rockstar. Jedes Handwerk, jede Kunst besitzt ein Schlüsselprinzip, das die mittelmäßigen Spieler von den großen Virtuosen unterscheidet. Bei JavaScript ist es das Verständnis von **Funktionen**. Funktionen sind ein wesentlicher Bestandteil von JavaScript und bieten viele Techniken für das **Design und die Organisation** von Code. Die Basis hierfür ist ein solides Wissen um die Verwendung von Funktionen. Funktionen auf diesem Niveau zu lernen, ist interessant und kann bewusstseinsverändernd wirken, also machen Sie sich bereit. Es ist ein bisschen, als würde Ihnen Willy Wonka eine Führung durch die Schokoladenfabrik geben. Beim Lernen von JavaScript-Funktionen werden Ihnen einige ziemlich wilde, verrückte, aber auch wundervolle Dinge begegnen.

Anonyme Funktionen, Geltungsbereiche und Closures

11 Seriöse Funktionen

Sie haben sich umfassend mit Funktionen befasst, aber es gibt noch mehr zu lernen. In diesem Kapitel gehen wir sogar noch weiter, jetzt kommt echter Hardcore. Wir zeigen Ihnen, wie man **wirklich** mit Funktionen **umgeht**. Dies wird kein superlanges Kapitel, aber es wird intensiv, und am Ende wird Ihr JavaScript ausdrucksstärker sein als je zuvor. Sie werden auch in der Lage sein, den Code Ihrer Mitarbeiter zu verstehen und eine Open Source-JavaScript-Bibliothek zu durchschauen. In diesem Kapitel behandeln wir nämlich einige häufige Code-Idiome und Konventionen, die bei Funktionen immer wieder vorkommen. Und wenn Sie noch nie von **anonymen Funktionen** oder **Closures** gehört haben, dann sind Sie hier mehr als richtig.

Fortgeschrittene Objektkonstruktion

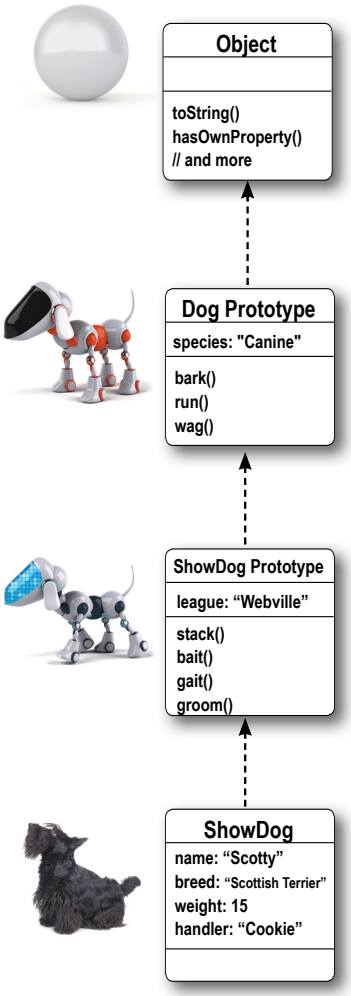
12 Objekte erstellen

Bisher haben wir unsere Objekte von Hand erstellt. Für jedes Objekt haben wir ein **Objektliteral** verwendet, um die einzelnen Eigenschaften anzugeben. Das ist in kleinem Rahmen auch kein Problem. Für ernsthaften Code brauchen wir aber etwas Besseres. Und da kommen **Objektkonstruktoren** ins Spiel. Mit Konstruktoren können wir die Objekte wesentlich einfacher erstellen, die außerdem alle nach der gleichen **Designschablone** aufgebaut sind. Das heißt, durch die Verwendung von Konstruktoren können wir sicherstellen, dass jedes Objekt die gleichen Eigenschaften besitzt und die gleichen Methoden enthält. Außerdem ist der mit Konstruktoren erstellte Objektcode wesentlich **kürzer** und deutlich weniger fehleranfällig, besonders wenn viele Objekte erstellt werden müssen. Legen wir also einfach mal los! In kürzester Zeit werden Sie so gut »konstruktorisch« sprechen, als seien Sie in Objectville groß geworden.

Prototypen verwenden

13 Extrastarke Objekte

Zu lernen, wie man Objekte erstellt, war erst der Anfang. Jetzt ist es Zeit, Ihren Objekten ein paar Muskeln wachsen zu lassen. Wir brauchen neue Möglichkeiten, **Beziehungen** zwischen den Objekten herzustellen und **Code** zwischen ihnen **auszutauschen**. Und wir benötigen die Fähigkeit, bestehende Objekte zu erweitern und aufzuwerten. Anders gesagt: Wir brauchen mehr Werkzeuge. In diesem Kapitel werden Sie sehen, dass JavaScript ein sehr mächtiges **Objektmodell** besitzt, aber auch, dass es sich von den üblichen objektorientierten Programmiersprachen unterscheidet. Anstelle des typischen klassenbasierten Objektmodells verwendet JavaScript ein mächtigeres **Prototypenmodell**, bei dem die Objekte voneinander erben oder das Verhalten anderer Objekte erweitern können. Wofür das gut sein soll, werden Sie gleich sehen. Auf geht's …



Anhang: Was übrig bleibt

14 Die Top Ten der Themen, die wir nicht behandelt haben

Wir sind ganz schön weit gekommen, und Sie haben das Buch fast durch. Wir werden Sie vermissen, aber bevor wir Sie gehen lassen, wollen wir Sie nicht ohne etwas zusätzliche Vorbereitung losschicken. Leider können wir nicht alles, was Sie noch wissen müssen, in diesem relativ kurzen Kapitel unterbringen. Ursprünglich hatten wir tatsächlich sämtliches notwendiges Wissen über JavaScript-Programmierung (das nicht bereits in den anderen Kapiteln behandelt wurde) hier untergebracht, indem wir die Schriftgröße ungefähr auf 0,00004 Punkt reduziert haben. Es passte alles rein, aber niemand konnte es lesen. Also haben wir das meiste wieder rausgeschmissen und die besten Sachen für den Top-Ten-Anhang behalten.

Das hier ist tatsächlich das Ende des Buchs. Das heißt, bis auf den Index natürlich (den Sie unbedingt lesen müssen!).

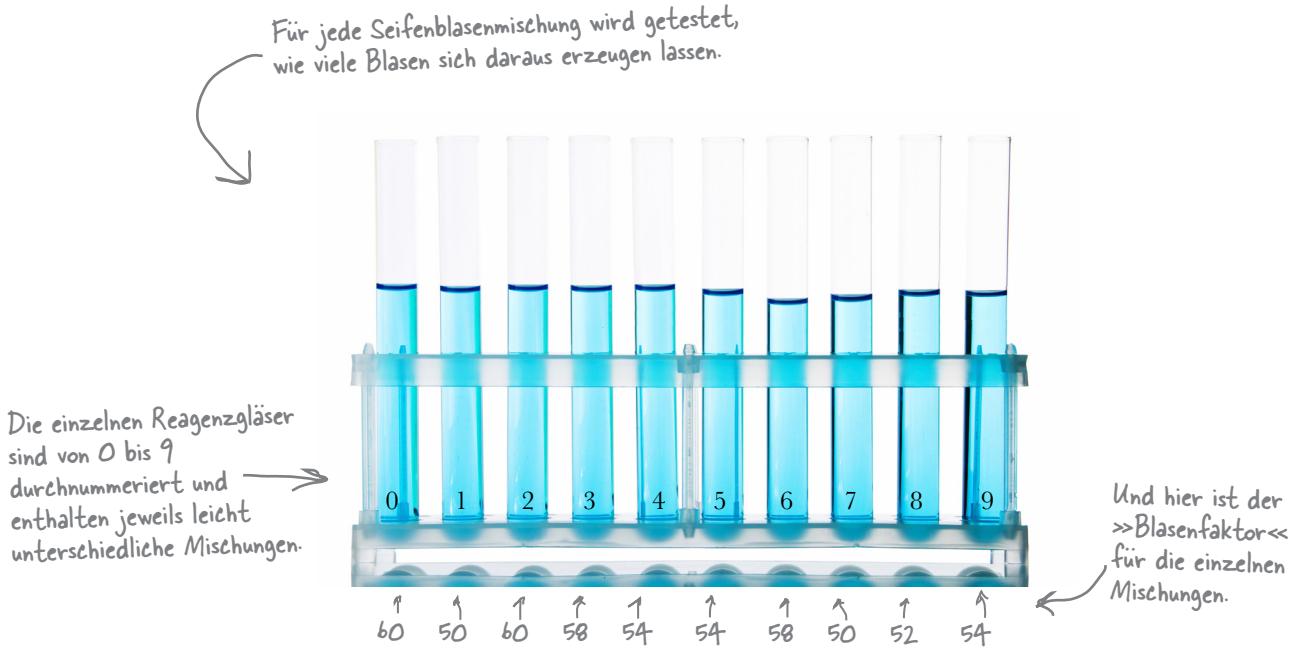
4 Etwas Ordnung in die Daten bringen

Arrays

JavaScript besteht nicht nur aus Zahlen, Strings und Booleschen Werten. Bisher haben Sie in Ihrem JavaScript-Code nur primitive Datentypen – einfache Strings, Zahlen und Boolesche Werte wie »Fido«, 23 und true – benutzt. Damit lässt sich schon eine Menge anstellen, aber irgendwann müssen Sie mit **mehr Daten** arbeiten. Das könnten alle Artikel in einem Warenkorb sein oder alle Titel einer Playlist oder eine Liste von Sternen und ihre scheinbare Größe oder ein kompletter Produktkatalog. Dafür brauchen wir etwas mehr *Wumms*. Der Datentyp der Wahl für geordnete Daten dieser Art ist das JavaScript-**Array**. In diesem Kapitel zeigen wir Ihnen, wie Sie Ihre Daten in einem Array ablegen, es weitergeben und damit arbeiten können. Später in diesem Buch werden Sie weitere Möglichkeiten kennenlernen, Ihre **Daten zu strukturieren**; jetzt wollen wir uns aber erst mal mit Arrays befassen.

Können Sie Bubbles-R-Us helfen?

Heute stellen wir Ihnen Bubbles-R-Us vor. Die unermüdliche Forschung der Firma stellt sicher, dass ihre Pusteringe und -maschinen die besten Seifenblasen machen. Heute wird der »Blasenfaktor« für mehrere Variationen der Seifenblasenmischung getestet. Man will herausfinden, wie viele Blasen eine bestimmte Mischung ergibt. Hier sind die Daten:



Sicher wollen Sie alle diese Daten zur Analyse in JavaScript abbilden. Allerdings sind das ziemlich viele Werte. Wie konstruieren Sie Ihren Code, damit er mit sämtlichen Werten umgehen kann?

Mehrere Werte in JavaScript darstellen

Sie wissen, wie einzelne Werte wie Zeichenketten (Strings), Zahlen und Boolesche Werte in JavaScript dargestellt werden. Aber wie funktioniert das bei *Gruppen* von Werten, z. B. allen Testergebnissen der zehn verschiedenen Mischungen? Für diesen Zweck gibt es JavaScript-*Arrays*. Ein Array ist ein JavaScript-Datentyp, der mehrere Werte auf einmal enthalten kann. Hier sehen Sie ein Array, das sämtliche Blasenfaktor-Messergebnisse enthält:

```
var scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54];
```

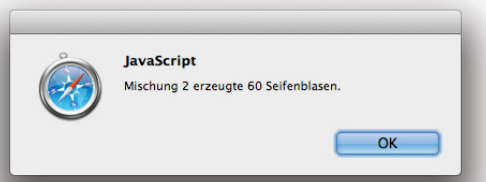
Hier wurden alle zehn Werte zu einem Array zusammengefasst und der Variablen scores zugewiesen.

Sie können alle Werte gemeinsam bearbeiten oder bei Bedarf auch auf die einzelnen Messergebnisse zugreifen. Sehen Sie mal hier:

Um auf ein Element in einem Array zuzugreifen, benutzen wir diese Syntax: der Variablenname des Arrays, gefolgt einem Paar eckiger Klammern, die den Index des Elements enthalten.

Arrays sind nullbasiert. Das heißt, die erste Mischung hat die Nummer 0, das passende Ergebnis ist unter scores[0] zu finden. Die dritte Mischung ist also Nummer 2, und das Ergebnis liegt in scores[2].

```
var solution2 = scores[2];
alert("Mischung 2 erzeugte" + solution2 + " Seifenblasen.");
```



Einer der Blaseologen von Bubbles-R-Us.

Wie Arrays funktionieren

Bevor wir Bubbles-R-Us weiterhelfen können, sollten wir uns zunächst ausführlich mit Arrays beschäftigen. Wie schon gesagt, können Sie Arrays zum Speichern *mehrerer* Werte verwenden (im Gegensatz zu Variablen, die immer nur einen Wert, z. B. eine Zahl oder einen String, enthalten können). Am häufigsten verwenden wir Arrays zum Gruppieren ähnlicher Dinge, wie einer Reihe von Blasenfaktoren, Eiscreme-Geschmacksrichtungen, Tagestemperaturen oder sogar Antworten auf eine Reihe von wahr/falsch-Fragen. Sobald Sie mehrere Werte gruppieren möchten, können zum Speichern ein Array erzeugen und bei Bedarf auf die einzelnen Werte zugreifen.

Ein Array erstellen

Hier zeigen wir Ihnen, wie Sie ein Array mit Eiscreme-Geschmacksrichtungen erstellen können. Das geht so:

Die einzelnen Elemente werden durch Kommata getrennt.

```
var flavors = ["Vanille", "Karamell", "Lavendel", "Schokolade", "Plätzchenteig"];
```

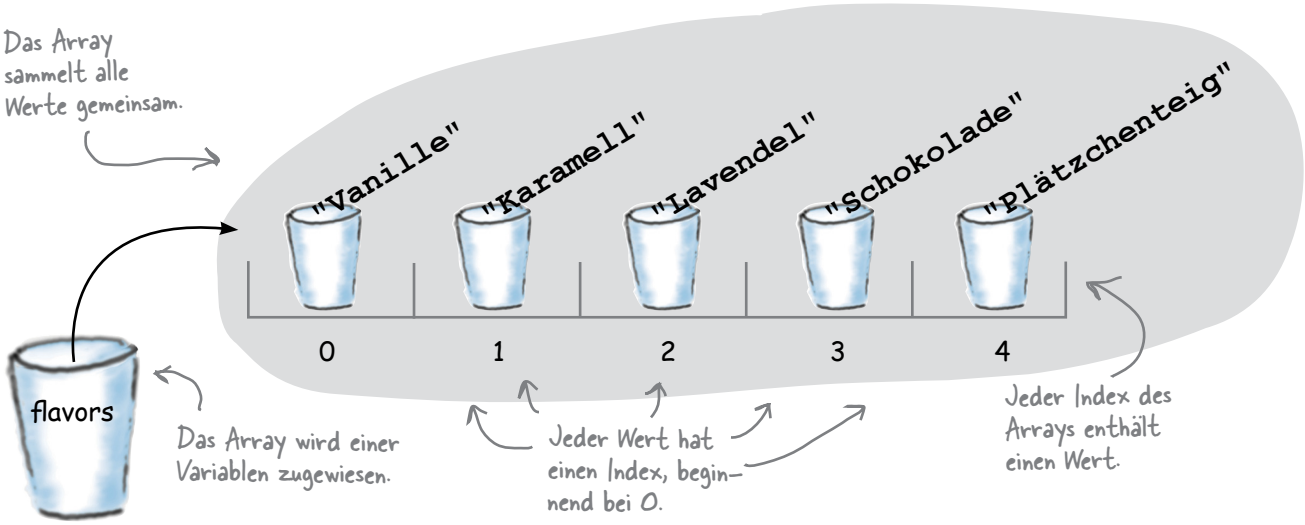
Wir weisen das Array einer Variablen namens flavors zu.

Sie beginnen das Array mit einem [-Zeichen ...

... und listen dann die einzelnen Array-Elemente auf.

Am Ende schließen wir das Array mit einem]-Zeichen.

Beim Anlegen eines Arrays wird jedes Element an einem bestimmten Platz oder *Index* innerhalb des Arrays abgelegt. Das erste Array-Element in scores ist »Vanille« mit dem Index 0, das zweite ist »Karamell« mit dem Index 1 und so weiter. Sie können sich das Array in etwa so vorstellen:



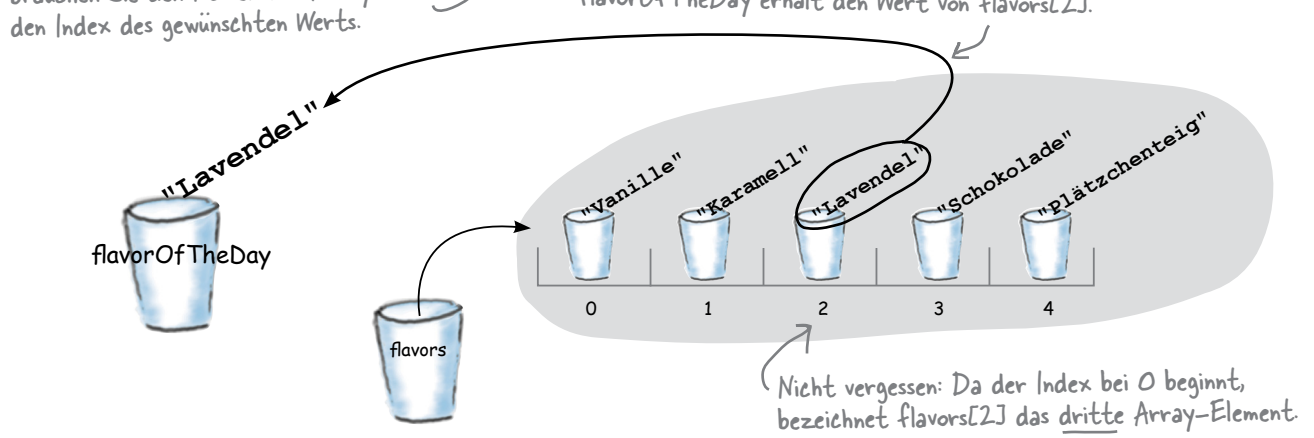
Auf Array-Elemente zugreifen

Jedes Array-Element hat seinen eigenen Index. Und das ist Ihr Schlüssel für den Zugriff und das Ändern der Werte in einem Array. Hierfür benutzen Sie einfach den Namen des Arrays, gefolgt vom Index des Elements in eckigen Klammern. Diese Schreibweise funktioniert überall dort, wo auch eine Variable funktioniert:

```
var flavorOfTheDay = flavors[2];
```

Hier wird der Wert des flavors-Arrays mit dem Index 2 (»Lavendel«) der Variablen flavorOfTheDay zugewiesen.

Um auf ein Array-Element zuzugreifen, brauchen Sie den Namen des Arrays und den Index des gewünschten Werts.



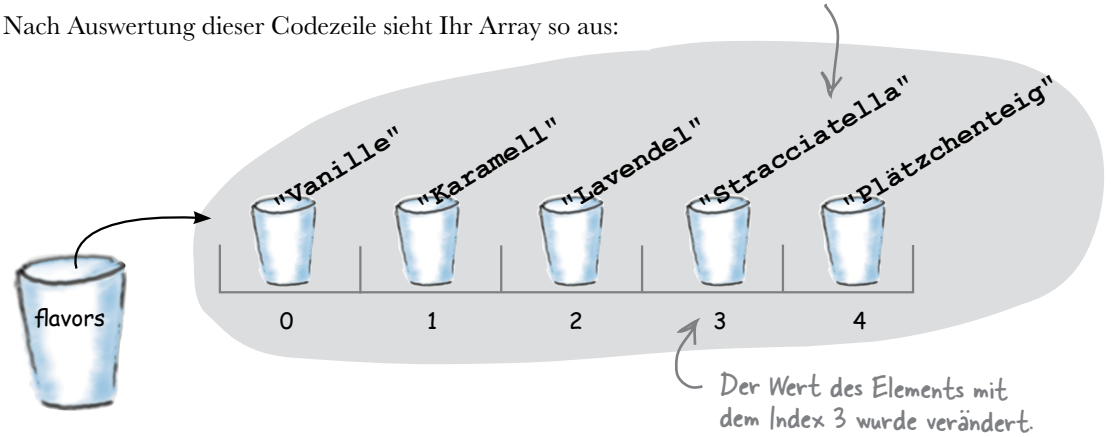
Einen Wert im Array aktualisieren

Anhand des Array-Indexes können Sie außerdem den Wert in einem Array ändern:

```
flavors[3] = "Stracciatella";
```

Hiermit wird der Wert des Elements mit dem Index 3 (zuvor »Schokolade«) mit dem neuen Wert »Stracciatella« überschrieben.

Nach Auswertung dieser Codezeile sieht Ihr Array so aus:



Wie groß ist das Array eigentlich?

Angenommen, jemand übergibt Ihnen ein schönes großes Array mit wichtigen Daten. Sie wissen, was es enthält, kennen aber vermutlich nicht die genaue Zahl der Elemente. Glücklicherweise besitzt jedes Array die Eigenschaft `length`. Über Eigenschaften (properties) und ihre Funktionsweise werden wir im folgenden Kapitel ausführlicher sprechen. Im Moment gilt: Eine Eigenschaft ist einfach ein mit dem Array verbundener Wert. Und so können Sie die Eigenschaft `length` einsetzen:

Jedes Array besitzt die Eigenschaft length, die die Anzahl der Array-Elemente enthält.

Um die Länge eines Arrays zu bestimmen, nehmen Sie den Namen des Arrays, setzen dann einen Punkt (.) und dann length.

```
var numFlavors = flavors.length;
```

Jetzt enthält numFlavors die Anzahl der Array-Elemente, hier sind es 5.

Spitzen Sie Ihren Bleistift

Das Array products enthält die Eiscreme-Sorten von Jenn und Berry's. Die Sorten wurden in der Reihenfolge ihrer Erstellung ergänzt. Vervollständigen Sie den Code, um herauszubekommen, welche Sorte *zuletzt* hinzugefügt wurde.

```
var products = ["Mousse au Chocolat",
     "Minze", "Kuchenstreusel", "Kaugummi"];
var last = _________________;
var recent = products[last];
```

Sie meinen, unsere seriöse Geschäftsanwendung aus Kapitel 1 war nicht seriös genug? Na gut. Dann probieren Sie es mal hiermit, wenn Ihr Chef das nächste Mal Resultate sehen will.

Versuchen Sie herauszufinden, was die brandneue Phras-O-Mat-App macht, bevor Sie weiterlesen ...

```
<!doctype html>
<html lang="de">
<head>
  <meta charset="utf-8">
  <title>Phras-O-Mat</title>
  <script>
    function makePhrases() {
      var words1 = ["Kundenfreundliche", "Wertsteigernde", "Marktorientierte",
                    "Fokussierte", "Angepasste"];
      var words2 = ["24/7", "mehrschichtige", "responsive", "mobile-first", "win-win"];
      var words3 = ["Benutzererfahrung", "Lösung", "Trendwende",
                    "Strategie", "Killer-App"];

      var rand1 = Math.floor(Math.random() * words1.length);
      var rand2 = Math.floor(Math.random() * words2.length);
      var rand3 = Math.floor(Math.random() * words3.length);

      var phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
      alert(phrase);
    }
    makePhrases();
  </script>
</head>
<body></body>
</html>
```

Der Phras-O-Mat

Der Phras-O-Mat ist das perfekte Werkzeug, um den Marketingslogan für Ihr nächstes Start-up zu generieren. Er hat bereits Perlen wie »Fokussierte 24/7-Strategie« und »Marktorientierte mehrschichtige Killer-App« hervorgebracht. Für die Zukunft rechnen wir mit weiteren großartigen Ergebnissen. Jetzt wollen wir sehen, wie das im Einzelnen funktioniert:

1. Zuerst definieren wir die Funktion `makePhrases`, die beliebig oft aufgerufen werden kann, um die gewünschten Phrasen zu erzeugen:

Wir definieren die Funktion makePhrases, die wir später aufrufen werden.

```
function makePhrases() {

}
makePhrases();
```

Sämtlicher Code für makePhrases steht hier. Wir werden gleich darauf eingehen.

Hier rufen wir makePhrases nur einmal auf. Wir könnten die Funktion aber auch mehrfach ansprechen, um weitere Phrasen zu erzeugen.

2. Anschließend können wir den Code für `makePhrases` schreiben. Zunächst erstellen wir drei Arrays. Sie enthalten die Wörter, aus denen nachher unsere Phrasen zusammengesetzt werden. Im nächsten Schritt wählen wir aus jedem Array ein zufälliges Wort aus, um daraus eine Phrase mit jeweils drei Wörtern zu bauen.

Für das erste Array legen wir eine Variable namens words1 an.

```
var words1 = ["Kundenfreundliche", "Wertsteigernde", "Marktorientierte",
"Fokussierte", "Angepasste"];
```

Das Array wird mit fünf Strings gefüllt. Hier ist Platz für Ihre Lieblingsschlagwörter.

```
var words2 = ["24/7-Strategie", "mehrschichtige", "responsive", "Mobile-
first-Lösung", "Win-win-Benutzererfahrung"];
var words3 = ["Benutzererfahrung", "Lösung", "Trendwende", "Strategie",
"Killer-App"];
```

Und hier haben wir noch zwei Arrays mit Wörtern, die den neuen Variablen words2 und words3 zugewiesen werden.

3. Jetzt erzeugen wir drei Zufallszahlen, eine für jedes zufällige Wort, das in unserer Phrase auftauchen soll. Aus Kapitel 2 wissen Sie, dass Math.random eine Zufallszahl zwischen 0 und 1 (exklusive der 1) erzeugt. Wenn wir diesen Wert mit der Länge des Arrays multiplizieren und das Ergebnis mit Math.floor abrunden, bekommen wir eine Zahl zwischen 0 und der Anzahl der Array-Elemente minus 1.

```
var rand1 = Math.floor(Math.random() * words1.length);
var rand2 = Math.floor(Math.random() * words2.length);
var rand3 = Math.floor(Math.random() * words3.length);
```

rand1 ist eine Zahl zwischen 0 und dem letzten Index von words1.

Genauso verfahren wir für rand2 und rand3.

4. Und nun erstellen wir unseren Marketingslogan, indem wir die drei Zufallswörter miteinander verbinden. Dazwischen lassen wir aus Gründen der Lesbarkeit etwas Platz.

Eine weitere Variable soll die fertige Phrase speichern.

Hier benutzen wir die Zufallszahlen, um bestimmte Indizes im Array anzusprechen.

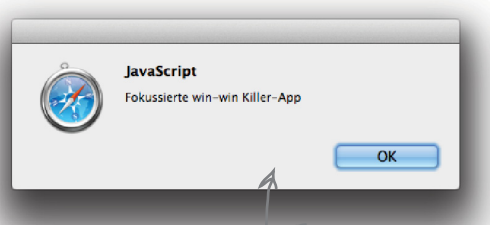
```
var phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
```

5. Fast fertig. Jetzt müssen wir die Phrase nur noch ausgeben. Wie üblich benutzen wir auch hier `alert()`.

```
alert(phrase);
```

6. Gut. Schreiben Sie die letzte Codezeile, werfen Sie noch einmal einen Blick darauf und freuen Sie sich über Ihre Leistung, bevor Sie die Seite im Browser aufrufen und die Phrasen zu testen.

Hier ist der fertige Slogan!



Endlose weitere Variationen bekommen Sie, indem Sie die Seite neu laden. (Na gut, nicht ganz, aber Spaß muss schließlich sein.)

Es gibt keine Dummen Fragen

F: Spielt die Reihenfolge eines Arrays eine Rolle?

A: Meistens schon, aber nicht immer. Im Bubbles-R-Us-Array ist die Reihenfolge sehr wichtig, weil der Index der Ergebnisse auch die Rangfolge bestimmt. Die Mischung Nummer 0 hat 60 Punkte und wird im Array-Element mit dem Index 0 gespeichert. Würden wir die Ergebnisse im Array vermischen, wäre das Experiment nichts mehr wert! In anderen Fällen ist die Reihenfolge der Elemente nicht wichtig, z. B. wenn Sie das Array zum Speichern von ungeordneten Zufallswörtern benutzen. Sollen die Wörter später alphabetisch geordnet werden, ist die Reihenfolge wieder wichtig. Es kommt also darauf an, was Sie vorhaben, meistens ist sie wichtig.

F: Wie viele Elemente kann ich in einem Array speichern?

A: Theoretisch können Sie unbegrenzt viele Elemente speichern. Die Obergrenze hängt nur vom Arbeitsspeicher des Computers ab. Jedes Array-Element benötigt ein wenig RAM. Vergessen Sie nicht, dass JavaScript im Browser ausgeführt wird und dass dieser nur eines von vielen Programmen ist, die auf Ihrem Rechner laufen. Wenn Sie das Array immer weiter füllen, ist der Speicher irgendwann voll. Je nachdem, welcher Art die Elemente sind, liegt die Obergrenze bei mehreren Tausend – wenn nicht sogar Millionen. So viele Elemente werden Sie vermutlich nur selten brauchen. Die Anzahl der Elemente hat außerdem Einfluss auf die Ausführungsgeschwindigkeit Ihrer Programme. Daher sollten Sie die Anzahl der Elemente nach Möglichkeit auf ein paar Hundert beschränken.

F: Können Arrays auch leer sein?

A: Das funktioniert tatsächlich, und wir werden gleich ein Beispiel dafür sehen. Um ein leeres Array zu erstellen, schreiben Sie einfach:

```
var emptyArray = [ ];
```

Leere Arrays können Sie auch später noch mit Elementen füllen.

F: Bisher haben wir nur Strings und Zahlen in einem Array gespeichert. Kann ein Array auch andere Dinge enthalten?

A: Ja, Sie können alle in JavaScript gültigen Werttypen in einem Array speichern, wie Zahlen, Strings, Boolesche Werte, andere Arrays und sogar Objekte (die wir im nächsten Kapitel kennenlernen).

F: Müssen alle Werte im Array den gleichen Typ haben?

A: Nicht unbedingt, obwohl das die *übliche Vorgehensweise* ist. Im Gegensatz zu anderen Sprachen gibt es in JavaScript keine Regel, nach der alle Elemente vom gleichen Typ sein müssen. Allerdings sollten Sie bei verschiedenen Werttypen im gleichen Array besonders vorsichtig sein. Wir sagen Ihnen auch, warum: Angenommen, Sie haben ein Array mit den Werten [1, 2, »fido«, 4, 5]. Was passiert beispielsweise, wenn Sie Code schreiben, der überprüft, ob die Elemente größer als 2 sind? Spätestens bei »fido« gibt es Probleme. Um sicherzugehen, dass Ihr Code wie gewünscht funktioniert, müssen Sie neben dem numerischen Test überprüfen, ob der Wert den richtigen Typ hat. Das ist natürlich möglich (wie wir später in diesem Buch sehen werden), aber allgemein ist es einfacher und sicherer, wenn alle Elemente des Arrays den gleichen Werttyp haben.

F: Was passiert, wenn Sie versuchen, mit einem zu kleinen oder zu großen Index auf das Array zuzugreifen (z. B. kleiner als 0)?

A: Angenommen, Sie haben dieses Array:

```
var a = [1, 2, 3];
```

und versuchen auf a[10] oder a[-1]. Dann ist das Ergebnis in beiden Fällen undefined. Entweder müssen Sie also sicherstellen, dass nur gültige Indizes für den Array-Zugriff verwendet werden, oder Sie müssen überprüfen, ob der Rückgabewert nicht undefined ist.

F: Mit dem Index 0 kann ich auf das erste Element eines Arrays zugreifen. Wie komme ich aber an das letzte Element im Array? Muss ich immer genau wissen, wie viele Elemente es enthält?

A: Sie können die Eigenschaft length verwenden, um auf das letzte Array-Element zuzugreifen. Sie wissen ja bereits, dass das Ergebnis von length um eins größer ist als die Anzahl der Elemente, oder? Um auf das letzte Element im Array zuzugreifen, könnten Sie also schreiben:

```
myArray[myArray.length - 1];
```

JavaScript ermittelt die Anzahl der Elemente im Array, subtrahiert 1 und verwendet das Ergebnis als Index. Hat Ihr Array beispielsweise 10 Elemente, wird – wie gewünscht – auf das Element mit dem Index 9 zugegriffen. Wir benutzen diesen Trick andauernd, um auf das letzte Element eines Arrays zuzugreifen, dessen Länge wir nicht kennen.

Inzwischen bei Bubbles-R-Us ...

Der Bubbles-R-Us-CEO

Hallo, schön, dass Sie da sind. Wir haben gerade eine ganze Reihe neuer Tests durchgeführt. Sehen Sie sich mal diese Ergebnisse an! Ich brauche Ihre Hilfe, die Resultate zu verstehen. Können Sie mir nach der Skizze unten bitte ein passendes Programm schreiben?

Was wir programmieren sollen.

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];
```

Neue Testergebnisse.

Bubbles-R-Us

Ich brauche diesen Bericht, um schnell entscheiden zu können, welche Seifenblasenmischung wir herstellen sollen. Können Sie das für mich programmieren?

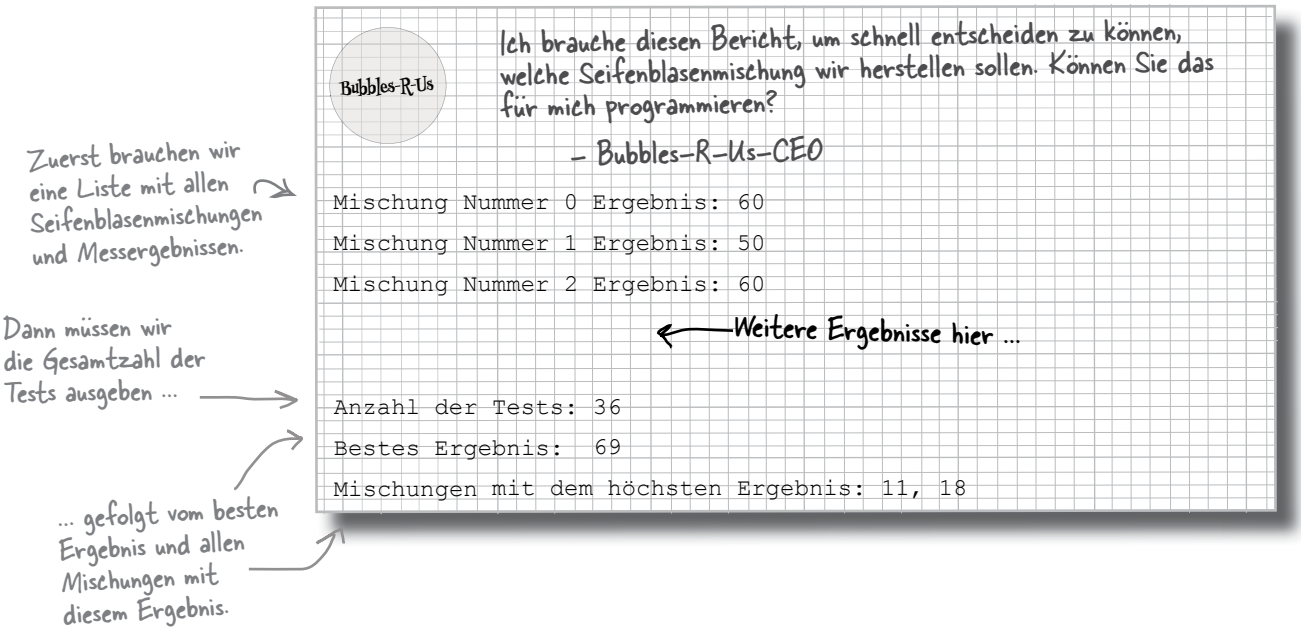
– Bubbles-R-Us-CEO

```
Mischung Nummer 0 Ergebnis: 60
Mischung Nummer 1 Ergebnis: 50
Mischung Nummer 2 Ergebnis: 60
```

← Weitere Ergebnisse hier ...

```
Anzahl der Tests: 36
Bestes Ergebnis:  69
Mischungen mit dem höchsten Ergebnis: 11, 18
```

Lassen Sie uns die Vorgaben des CEO etwas genauer untersuchen:



KOPF-NUSS

Nehmen Sie sich ein bisschen Zeit, einen Weg für die Erstellung dieses Berichts zu finden. Behandeln Sie jeden Punkt im Bericht einzeln und überlegen Sie, wie Sie ihn strukturieren und die Ergebnisse ausgeben würden. Schreiben Sie Ihre Notizen in diesen Kasten.

Gespräche unter Büronachbarn

Judy: Zu Beginn sollen wir alle Messergebnisse und die Nummer der dazugehörigen Mischung ausgeben.

Joe: Und die Nummer der Mischung entspricht dem Array-Index für das Ergebnis, richtig?

Judy: Ja, das ist vollkommen richtig.

Frank: Nicht ganz so schnell, bitte. Wir müssen also für jedes Ergebnis seinen Index entsprechend seiner Nummer und das dazugehörige Ergebnis ausgeben.

Judy: Genau. Und das Ergebnis ist einfach der entsprechende Wert im Array.

Joe: Für die Seifenblasenmischung Nummer 10 ist der Wert also in `scores[10]`.

Judy: Richtig.

Frank: Das sind aber ganz schön viele Ergebnisse. Wie schreiben wir denn den Code, um die alle einzeln auszugeben?

Judy: Iteration, mein Freund.

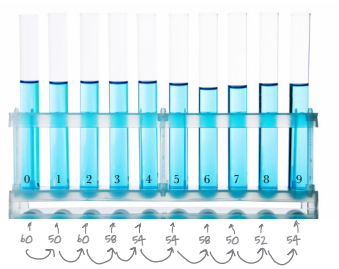
Frank: Oh, du meinst so was wie eine while-Schleife?

Judy: Korrekt. Wir führen einfach eine Schleife über alle Werte von null bis zum Wert von length – ich meine natürlich von length minus eins – aus.

Joe: Langsam kann ich mir das vorstellen. Lasst uns etwas Code schreiben. Ich glaube, ich weiß, was zu tun ist.

Judy: Klingt gut. Auf geht's! Mit dem restlichen Bericht beschäftigen wir uns danach.

Über ein Array iterieren



Ihr Ziel ist eine Ausgabe wie diese hier:

```
Mischung Nummer 0 Ergebnis: 60
Mischung Nummer 1 Ergebnis: 50
Mischung Nummer 2 Ergebnis: 60
.
.
.
Mischung Nummer 35 Ergebnis: 44
```

Die Ergebnisse 3 bis 34 stehen hier. Wie retten einfach ein paar Bäume (oder Bits, je nachdem, welche Version dieses Buchs Sie lesen).

Wir machen das, indem wir das Ergebnis mit dem Index 0 ausgeben, das Gleiche tun wir für die Indizes 1, 2, 3 und so weiter bis zum letzten Index im Array. Wie man eine while-Schleife benutzt, wissen Sie ja schon. Wir wollen einmal sehen, wie Sie damit die Ergebnisse ausgeben können:

Und gleich danach zeigen wir Ihnen eine bessere Methode ...

```
var scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51, 69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18, 41, 53, 55, 61, 51, 44];

var output;

var i = 0;

while (i < scores.length) {

  output = "Mischung Nummer " + i + " Ergebnis: " + scores[i];

  console.log(output);

  i = i + 1;

}
```

Diese Variable benutzen wir in der Schleife, um einen String für die Ausgabe zu erzeugen.

Eine Variable, die den aktuellen Index enthält.

Die Schleife läuft, solange der Index kleiner ist als die Länge des Arrays.

Dann erzeugen wir einen String für die Ausgabezeile. Er enthält die Nummer der Mischung (den Index) und das entsprechende Ergebnis.

Danach benutzen wir console.log, um den String auszugeben.

Schließlich inkrementieren wir den Index um eins, bevor die Schleife erneut durchlaufen wird.

Codemagneten

Mit diesem Code überprüfen wir, welche Eiscreme-Sorten Kaugummi enthalten. Wir hatten die Codeschnipsel am Kühlschrank bereits richtig angeordnet, aber dann sind sie heruntergefallen. Es ist Ihre Aufgabe, sie wieder richtig zusammenzusetzen. Vorsicht: Einige Magneten werden nicht gebraucht. Überprüfen Sie Ihre Antwort am Ende dieses Kapitels, bevor Sie weiterlesen.

```
while (i < hasBubbleGum.length)
```

```
{
```

```
}
```

```
i = i + 2;
```

```
i = i + 1;
```

```
}
```

```
var i = 0;
```

```
{
```

```
{
```

```
if (hasBubbleGum[i])
```

```
while (i > hasBubbleGum.length)
```

```
var products = ["Mousse au Chocolat",
                "Minze", "Kuchenstreusel",
                "Kaugummi"];
```

```
var hasBubbleGum = [false,
                    false,
                    false,
                    true];
```

```
console.log(products[i] +
    " enthält Kaugummi");
```

Bringen Sie die Magneten hier in die richtige Reihenfolge.

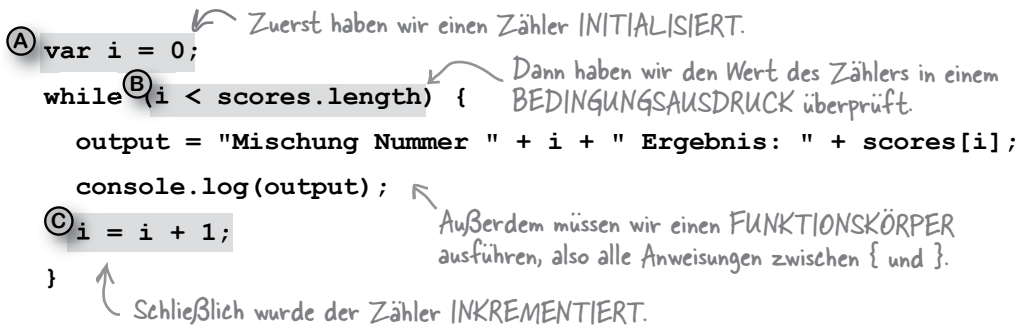
Hier sehen Sie die erwartete Ausgabe.

JavaScript-Konsole

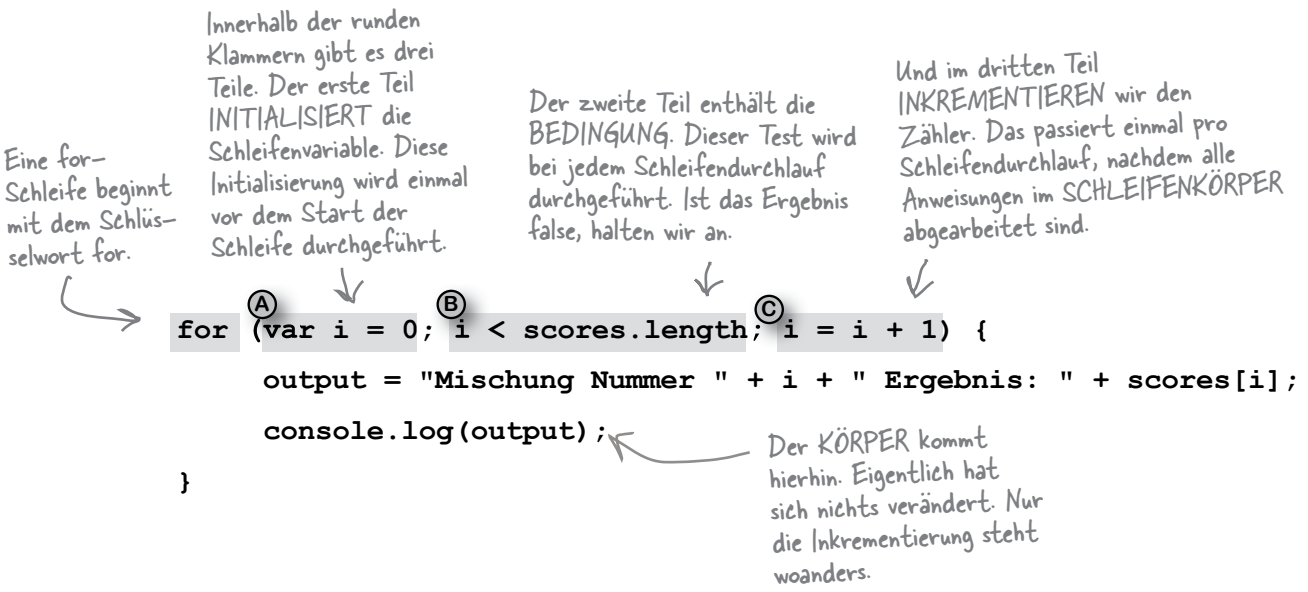
```
Kaugummi enthält Kaugummi
```

Moment mal! Es gibt noch eine bessere Methode, über ein Array zu iterieren.

Wir müssen uns wirklich bei Ihnen entschuldigen. Jetzt sind wir tatsächlich schon im vierten Kapitel und haben Ihnen die for-Schleife immer noch nicht vorgestellt. Im Prinzip ist die for-Schleife einfach ein Cousin der while-Schleife. Eigentlich machen beide das Gleiche, nur das die Benutzung der for-Schleife etwas bequemer ist. Sehen Sie sich einmal die while-Schleife an und überlegen Sie, wie sie auf eine for-Schleife abgebildet werden kann.



Und jetzt wollen wir sehen, wie eine for-Schleife das Ganze wesentlich einfacher machen kann:



Spitzen Sie Ihren Bleistift

Schreiben Sie den Codemagnetencode (zwei Seiten vorher) neu, sodass er anstelle der while- eine for-Schleife benutzt. Um zu sehen, wie die Positionen auf die for-Schleife abgebildet werden, können Sie sich die einzelnen Teile der while-Schleife auf der vorigen Seite noch einmal ansehen.

```
var products = ["Mousse au Chocolat",
                "Minze", "Kuchenstreusel",
                "Kaugummi"];

var hasBubbleGum = [false,
                    false,
                    false,
                    true];

var i = 0;
while (i < hasBubbleGum.length)
{
  if (hasBubbleGum[i])
  {
    console.log(products[i] +
        " enthält Kaugummi");
  }
  i = i + 1;
}
```

Hier kommt Ihr Code hin.

Dies ist der übliche HTML-Kram, den wir für eine Webseite brauchen. Hier steht nur das, was für die Erstellung unseres Skripts nötig ist.

```
<!doctype html>
<html lang="de">
<head>
  <meta charset="utf-8">
  <title>Seifenblasen-Testlabor</title>
  <script>
    var scores = [60, 50, 60, 58, 54, 54,
                  58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51,
                  69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18,
                  41, 53, 55, 61, 51, 44];

    var output;

    for (var i = 0; i < scores.length; i = i + 1) {

        output = "Mischung Nummer " + i +
                    " Ergebnis: " + scores[i];

        console.log(output);

    }

  </script>
</head>
<body></body>
</html>
```

Hier ist das Array mit unseren Testergebnissen.

Dies ist die for-Schleife, mit der wir über das Array mit den Testergebnissen iterieren.

Bei jedem Schleifendurchlauf erzeugen wir einen String. Dieser enthält den Wert von i, also die Nummer der Mischung, sowie den Wert von scores[i], also das entsprechende Messergebnis.

(Wir verteilen den String hier auf zwei Zeilen. Das ist in Ordnung, solange Sie zwischen den Anführungszeichen, die den String umgeben, keinen Zeilenumbruch einfügen. Hier haben wir es nach dem Verkettungsoperator (+) gemacht, was in Ordnung ist. Geben Sie den Code genau so ein, wie Sie ihn hier sehen.)

Danach geben wir den String auf der Konsole aus, und das war's! Zeit, das Berichtsprogramm zu testen.

Den Seifenblasen-Report Probe fahren

Sichern Sie den Code als »bubbles.html« und öffnen Sie die Datei in Ihrem Browser. Sorgen Sie dafür, dass die Konsole zu sehen ist (möglicherweise müssen Sie die Seite nach dem Aktivieren der Konsole neu laden). Dann können Sie sich den gerade für Bubbles-R-Us erzeugten Bericht in all seiner Schönheit ansehen.

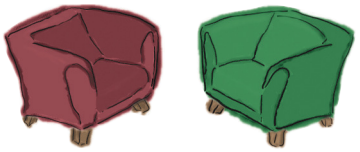
Genau das, was der CEO wollte.

Es ist schön, alle Messergebnisse im Bericht zu sehen. Die besten Ergebnisse sind aber immer noch schwer zu finden. Wir müssen auch die anderen Anforderungen an den Bericht integrieren, um das Auffinden des Gewinners zu erleichtern.

JavaScript-Konsole

```
Mischung Nummer 0 Ergebnis: 60
Mischung Nummer 1 Ergebnis: 50
Mischung Nummer 2 Ergebnis: 60
Mischung Nummer 3 Ergebnis: 58
Mischung Nummer 4 Ergebnis: 54
Mischung Nummer 5 Ergebnis: 54
Mischung Nummer 6 Ergebnis: 58
Mischung Nummer 7 Ergebnis: 50
Mischung Nummer 8 Ergebnis: 52
Mischung Nummer 9 Ergebnis: 54
Mischung Nummer 10 Ergebnis: 48
Mischung Nummer 11 Ergebnis: 69
Mischung Nummer 12 Ergebnis: 34
Mischung Nummer 13 Ergebnis: 55
Mischung Nummer 14 Ergebnis: 51
Mischung Nummer 15 Ergebnis: 52
Mischung Nummer 16 Ergebnis: 44
Mischung Nummer 17 Ergebnis: 51
Mischung Nummer 18 Ergebnis: 69
Mischung Nummer 19 Ergebnis: 64
Mischung Nummer 20 Ergebnis: 66
Mischung Nummer 21 Ergebnis: 55
Mischung Nummer 22 Ergebnis: 52
Mischung Nummer 23 Ergebnis: 61
Mischung Nummer 24 Ergebnis: 46
Mischung Nummer 25 Ergebnis: 31
Mischung Nummer 26 Ergebnis: 57
Mischung Nummer 27 Ergebnis: 52
Mischung Nummer 28 Ergebnis: 44
Mischung Nummer 29 Ergebnis: 18
Mischung Nummer 30 Ergebnis: 41
Mischung Nummer 31 Ergebnis: 53
Mischung Nummer 32 Ergebnis: 55
Mischung Nummer 33 Ergebnis: 61
Mischung Nummer 34 Ergebnis: 51
Mischung Nummer 35 Ergebnis: 44
```

Kamingespräche



Heute Abend: **Die while- und die for-Schleife beantworten die Frage: »Wer ist wichtiger?«**

Die WHILE-Schleife

Was? Machen Sie Witze? Hallo? Ich bin *das* Schleifenkonstrukt *schlechthin* in JavaScript. Ich bin doch nicht mit einem albernen Zähler verheiratet. Ich kann mit jeder beliebigen Bedingung verwendet werden. Hat eigentlich irgendjemand mitgekriegt, dass ich in diesem Buch zuerst gezeigt wurde?

Und das ist noch so eine Sache. Haben Sie gemerkt, dass die FOR-Schleife überhaupt keinen Sinn für Humor hat? Na ja, wenn wir alle den ganzen Tag nur diese hirnzermarternden Iterationen durchführen müssten, wären wir auch so.

Oh, das ist ja wohl so gut wie unmöglich.

Dieses Buch sagt, dass FOR- und WHILE-Schleifen mehr oder weniger das Gleiche sind. Wie kann das dann sein?

Die FOR-Schleife

Ihr Ton gefällt mir ganz und gar nicht!

Echt süß. Aber haben Sie denn gemerkt, dass die Programmierer in neun von zehn Fällen eine FOR-Schleife benutzen?

Mal ganz davon abgesehen, dass die Iteration, zum Beispiel über ein Array mit einer festen Anzahl von Elementen, mit einer WHILE-Schleife einfach als schlecht und ungeschickt gilt.

Ha! Sie geben also zu, dass Sie mir mehr gleichen, als Sie anfangs gesagt haben. Ich sage Ihnen auch, warum …

Die WHILE-Schleife

Na, das ist ja richtig nett und praktisch von Ihnen. Und überhaupt: Für die meisten Iterationen wird doch überhaupt kein Zähler gebraucht. Das meiste ist doch Zeug wie:

```
while (answer != "zweiundvierzig")
```

– versuchen Sie das mal mit einer FOR-Schleife!

Ha! Ich kann kaum glauben, dass das überhaupt funktioniert.

Perlen vor die Säue.

Nicht nur besser, sondern auch hübscher.

Die FOR-Schleife

Wenn Sie eine WHILE-Schleife benutzen, müssen Sie den Zähler in verschiedenen Anweisungen initialisieren und inkrementieren. Gibt es viele Änderungen am Code, kann es schnell passieren, dass versehentlich eine der Anweisungen an der falsche Stelle landet oder ganz gelöscht wird. Und dann wird es richtig übel. Bei einer FOR-Schleife ist dagegen alles sauber und gut sichtbar in der FOR-Anweisung verpackt, ohne dass etwas verändert werden oder verloren gehen kann.

Okay:

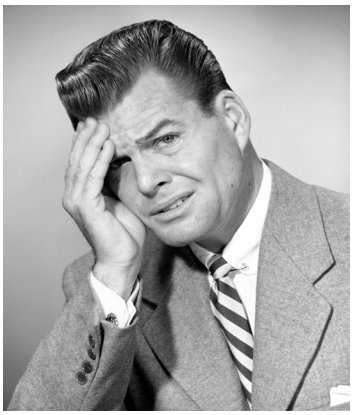
```
for (;answer != "zweiundvierzig";)
```

Tut es aber.

Mehr haben Sie nicht drauf? Mit einer allgemeinen Bedingung können Sie besser umgehen?

Oh, Entschuldigung. Ich wusste nicht, dass das hier ein Schönheitswettbewerb ist.

Wir müssen mal wieder über Ihren Wortschatz reden …



Sie haben inzwischen eine Menge Code geschrieben, der so aussieht:

Angenommen, myImportantCounter hat den Wert null.

Hier nehmen wir die Variable und erhöhen sie um eins.

```
myImportantCounter = myImportantCounter + 1;
```

Nach dieser Anweisung ist der Wert von myImportantCounter um eins größer.

Tatsächlich kommt diese Anweisung so häufig vor, dass es dafür in JavaScript eine Abkürzung gibt. Sie wird Postinkrement-Operator genannt und ist trotz ihres ungewöhnlichen Namens ganz einfach. Anhand des Postinkrement-Operators können wir die oben stehende Codezeile auch so schreiben:

Schreiben Sie einfach >>++<< hinter den Variablennamen.

```
myImportantCounter++;
```

Nach dieser Anweisung ist der Wert von myImportantCounter um eins höher als vorher.

Natürlich würde etwas fehlen, wenn es nicht auch einen Postdekrement-Operator gäbe. Diesen können Sie verwenden, um den Wert einer Variablen um eins zu verringern, wie hier:

Schreiben Sie einfach >>--<< direkt hinter den Variablennamen.

```
myImportantCounter--;
```

Nach dieser Anweisung ist der Wert von myImportantCounter um eins kleiner als vorher.

Und warum sagen wir Ihnen das gerade jetzt? Weil der Postinkrement-Operator häufig zusammen mit `for`-Schleifen benutzt wird. Wir wollen unseren Code mit seiner Hilfe etwas verschönern …

Eine Neufassung der for-Schleife mit dem Postinkrement-Operator

Wir wollen die Schleife neu schreiben und prüfen, ob der Code das Gleiche tut wie vorher:

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

for (var i = 0; i < scores.length; i++) {

    var output = "Mischung Nummer " + i +
                 " Ergebnis: " + scores[i];

    console.log(output);

}
```

Wo wir die Variable zuvor >>von Hand<< inkrementiert haben, benutzen wir jetzt den Postinkrement-Operator.

Schnelle Probefahrt

Es ist Zeit für eine kleine Probefahrt, um sicherzustellen, dass die Änderung mit dem Postinkrement-Operator auch richtig funktioniert. Speichern Sie Ihre Datei als »bubbles.html« und rufen Sie sie in Ihrem Browser auf. Es sollte der gleiche Bericht erscheinen wie vorhin auch.

JavaScript-Konsole

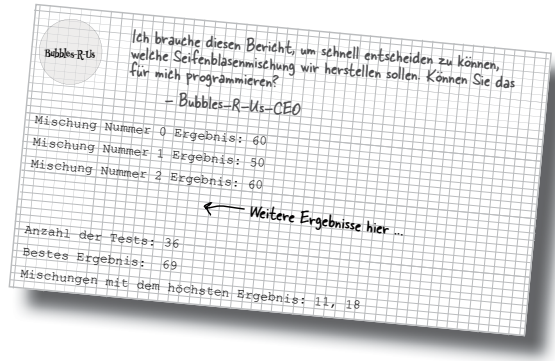
```
Mischung Nummer 0 Ergebnis: 60
Mischung Nummer 1 Ergebnis: 50
Mischung Nummer 2 Ergebnis: 60
 ...
Mischung Nummer 34 Ergebnis: 51
Mischung Nummer 35 Ergebnis: 44
```

Der Bericht sieht genau gleich aus.

Wir retten ein paar Bäume und zeigen Ihnen nur ein paar Ergebnisse, auch wenn alle da sind.

Gespräche unter Büronachbarn – Fortsetzung ...

Es werden alle Ergebnisse für unsere Seifenblasenmischungen angezeigt. Jetzt müssen wir nur noch den Rest des Berichts erzeugen.



Judy: Zuerst müssen wir die Anzahl der Tests ermitteln. Das ist leicht, hierfür können wir einfach die Länge des Arrays scores benutzen.

Joe: Oh richtig, und wir müssen das beste Ergebnis herausfinden – und die passenden Mischungen dazu.

Judy: Genau, aber die letzte Aufgabe ist die schwerste. Fangen wir einfach mal mit dem höchsten Ergebnis an.

Joe: Klingt nach einem guten Startpunkt.

Judy: Hierfür brauchen wir einfach eine Variable, die während der Schleifendurchläufe den Höchststand speichert. Am besten beginnen wir mit etwas Pseudocode:

```
DECLARIERE eine Variable namens highScore und setze sie auf 0.
FOR (Schleife): var i=0; i < scores.length; i++
    ZEIGE das Ergebnis für die aktuelle Mischung score[i]
    FALLS (if) scores[i] > highScore
        SETZE highScore = scores[i];
    ENDE FALLS
ENDE FOR (Schleife)
ZEIGE den Wert von highScore
```

Eine Variable für den Höchststand einbauen.

Bei jedem Schleifendurchlauf überprüfen wir, ob es ein höheres Ergebnis gibt. In diesem Fall ist dies der neue Wert für unseren Höchststand.

Nach der Schleife geben wir den Höchststand einfach aus.

Joe: Oh, schön. Und dafür waren nur ein paar zusätzlichen Zeilen Code nötig.

Judy: Bei jedem Schleifendurchlauf wird überprüft, ob das aktuelle Ergebnis größer ist als der Wert von `highScore`. In diesem Fall ist das der neue Höchststand. Nach dem Ende der Schleife geben wir den Höchststand einfach aus.

Spitzen Sie Ihren Bleistift

Jetzt können Sie den Pseudocode zum Finden des besten Ergebnisses implementieren. Füllen Sie hierfür die Lücken im unten stehenden Code. Danach können Sie Ihren Code im Browser testen, indem Sie die Daten in »bubbles.html« aktualisieren und die Seite neu laden. Überprüfen Sie Ihre Ergebnisse in der Konsole und schreiben Sie Ihren Höchststand und die Anzahl der Tests auf die leeren Zeilen in unserem Konsolenfenster unten auf dieser Seite. Vergleichen Sie Ihre Ergebnisse mit der Antwort am Ende dieses Kapitels, bevor Sie weiterlesen.

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

var highScore = _____;
var output;
for (var i = 0; i < scores.length; i++) {
    output = "Mischung Nummer " + i + " Ergebnis: " + scores[i];
    console.log(output);
    if (_____ > highScore) {
        _____ = scores[i];
    }
}
console.log("Anzahl der Tests: " + _____);
console.log("Bestes Ergebnis: " + _____);
```

Füllen Sie die leeren Zeilen, um den hier stehenden Code zu vervollständigen ...

... und ergänzen Sie dann die Leerzeilen mit Ihren Ergebnissen von der Konsole.

JavaScript-Konsolew

```
Mischung Nummer 0 Ergebnis: 60
Mischung Nummer 1 Ergebnis: 50
Mischung Nummer 2 Ergebnis: 60
 ...
Mischung Nummer 34 Ergebnis: 51
Mischung Nummer 35 Ergebnis: 44
Anzahl der Tests: _____
Bestes Ergebnis: _____
```

Jetzt haben Sie's fast geschafft! Es müssen nur die Mischungen mit dem besten Ergebnis gesammelt und ausgegeben werden. Vergessen Sie nicht, dass es mehr als einen Sieger geben kann.

»Mehr als einen« – hmmm. Was benutzen wir, wenn wir mehr als ein Element speichern müssen? Ein Array natürlich. Wir würden gern über das scores-Array iterieren, um alle Ergebnisse zu finden, die dem Höchststand entsprechen. Diese sollen dann in einem weiteren Array gespeichert werden, das wir schließlich in unserem Bericht ausgeben. Geht das? Aber sicher! Hierfür müssen Sie allerdings erst lernen, wie man ein leeres Array anlegt und ihm neue Elemente hinzufügt.

Jetzt fehlt nur noch dieser Teil.

Bubbles-R-Us

Ich brauche diesen Bericht, um schnell entscheiden zu können, welche Seifenblasenmischung wir herstellen sollen. Können Sie das für mich programmieren?

– Bubbles-R-Us-CEO

```
Mischung Nummer 0 Ergebnis: 60
Mischung Nummer 1 Ergebnis: 50
Mischung Nummer 2 Ergebnis: 60

                          ← Weitere Ergebnisse hier ...
Anzahl der Tests: 36
Bestes Ergebnis:  69
Mischungen mit dem höchsten Ergebnis: 11, 18
```

Ein neues Array anlegen (und Elemente hinzufügen)

Bevor wir den Code fertigstellen, wollen wir sehen, wie man ein Array erstellt und ihm neue Elemente hinzufügt. Wie man ein Array erzeugt, das bereits Elemente enthält, wissen Sie schon:

```
var genres = ["80s", "90s", "Electronic", "Folk"];
```

Das hier nennt man ein Array-Literal, da wir direkt (»wörtlich«) schreiben, was das Array enthalten soll.

Sie können die ursprünglichen Elemente aber auch weglassen und auf diese Weise ein leeres Array erzeugen:

```
var genres = [];
```

Ein neues Array mit der Länge null, bereit für neue Taten.

Auch das hier ist ein Array-Literal. Es hat einfach (noch) keinen Inhalt.

Sie wissen außerdem, wie Sie einem Array neue Werte hinzufügen. Hierfür weisen Sie einem Element mit einem bestimmten Index einfach einen Wert zu:

```
var genres = [];
genres[0] = "Rockabilly";
genres[1] = "Ambient";
var size = genres.length;
```

Hier wird ein neues Array-Element mit dem Inhalt »Rockabilly« erzeugt.

Und hier erstellen wir ein zweites Element, das den String »Ambient« enthält.

Hier bekommt die Variable size den Wert 2, die aktuelle Länge unseres Arrays.

Wenn Sie einem Array neue Elemente hinzufügen, müssen Sie darauf achten, welchem Index etwas zugewiesen wird. Ansonsten ist Ihr Array möglicherweise »lückenhaft« (z. B. ein Array mit Elementen bei den Indizes 0 und 2, aber nicht bei 1). Ein lückenhaftes Array ist nicht unbedingt etwas Schlechtes, erfordert aber besondere Aufmerksamkeit. Es gibt jedoch noch eine andere Methode, einem Array Elemente hinzuzufügen, ohne dass Sie sich um den Index kümmern müssen, und zwar per `push`:

```
var genres = [];
genres.push("Rockabilly");
genres.push("Ambient");
var size = genres.length;
```

Erzeugt ein neues Element mit dem nächsten verfügbaren Index (in diesem Fall 0) und weist ihm den Wert »Rockabilly« zu.

Erzeugt ein weiteres neues Element mit dem nächsten verfügbaren Index (hier 1) und weist ihm den Wert »Ambient« zu.

Es gibt keine Dummen Fragen

F: Die for-Anweisung enthält im ersten Teil eine Variablendeklaration und -initialisierung. Sie haben gesagt, Variablen sollen am Anfang deklariert werden. Warum nicht auch hier?

A: Das ist richtig. Es gilt als gute Vorgehensweise, Variablen am Anfang zu definieren (am Anfang der Datei, wenn die Variablen global sind, oder am Anfang einer Funktion, wenn sie lokal sind). Es gibt allerdings Situationen, in denen es sinnvoller ist, Variablen dort zu deklarieren, wo sie verwendet werden, und die for-Anweisung ist so ein Fall. Typischerweise benutzen Sie eine Schleifenvariable wie i nur zum Iterieren. Sobald die Schleife beendet ist, wird die Variable nicht mehr gebraucht. Daher hilft die Deklaration »an Ort und Stelle« in diesem Fall, unseren Code sauber zu halten.

F: Was bedeutet die Syntax myarray.push(Wert) genau?

A: Zugegeben, wir haben Ihnen ein kleines Geheimnis vorenthalten. In JavaScript ist ein Array eine spezielle Art Objekt. Wie Sie im folgenden Kapitel lernen werden, kann ein Objekt bestimmte Funktionen besitzen, mit denen Sie das Objekt bearbeiten können. Stellen Sie sich push am besten als eine Funktion vor, die etwas mit myarray anstellen kann. In unserem Fall fügt die Funktion dem Array das als Argument für push übergebene Element hinzu. Wenn Sie also schreiben:

```
genres.push("Metal");
```

rufen Sie die Funktion push mit dem String-Argument »Metal« auf. Daraufhin fügt push das Argument als neues Element mit dem übergebenen Wert am Ende des Arrays genres ein.

F: Können Sie mir das mit dem lückenhaften Array noch einmal genau erklären?

A: Ein lückenhaftes Array (sparse array) ist einfach ein Array, das nicht an jedem Index einen Wert enthält. Die Elemente dazwischen fehlen. Ein lückenhaftes Array lässt sich ganz einfach anlegen:

```
var sparseArray = [ ];
sparseArray[0] = true;
spraseArray[100] = true;
```

In diesem Beispiel enthält sparseArray nur zwei Werte, und zwar an den Indizes 0 und 100 (beide true). Obwohl das Array nur zwei Elemente enthält, ist die Länge des Arrays (der Rückgabewert von length) 101.

F: Angenommen, ich habe ein Array mit der Länge 10 und füge nun am Index 10000 ein neues Element ein. Was passiert dann mit den Indizes zwischen 10 und 9999?

A: Die übrigen Array-Indizes erhalten den Wert undefined. Wie Sie sich erinnern, ist undefined der Wert, den eine noch nicht initialisierte Variable erhält. Sie können sich das vorstellen, als erzeugten Sie 9989 Variablen, die aber nicht initialisiert werden. Beachten Sie, dass auch diese Variablen Arbeitsspeicher verbrauchen. Daher sollten Sie lückenhafte Arrays nur mit gutem Grund erstellen.

F: Angenommen, ich iteriere über ein Array, dessen Elemente teilweise den Wert undefined haben. Dann wäre es doch sinnvoll, das vor der Verwendung der Werte zu überprüfen, oder?

A: Wenn Sie vermuten, das Array könnte lückenhaft sein oder einen undefinierten Wert enthalten, sollten Sie das sehr wahrscheinlich vor der Verwendung überprüfen. So können Sie sicherstellen, dass ein Element an einem bestimmten Index auch tatsächlich einen definierten Wert enthält. Soll der Wert einfach nur auf der Konsole ausgegeben werden, ist das nicht so schlimm. In den meisten Fällen wollen Sie den Wert aber anderweitig verwenden, zum Beispiel in einer Berechnung. Der Versuch, undefined in dieser Situation zu benutzen, führt möglicherweise zu einem Fehler, zumindest aber wohl zu unerwartetem Verhalten. Mit folgendem Code können Sie testen, ob ein Array-Element den Wert undefined hat:

```
if (myarray[i] == undefined)
{
    ...
}
```

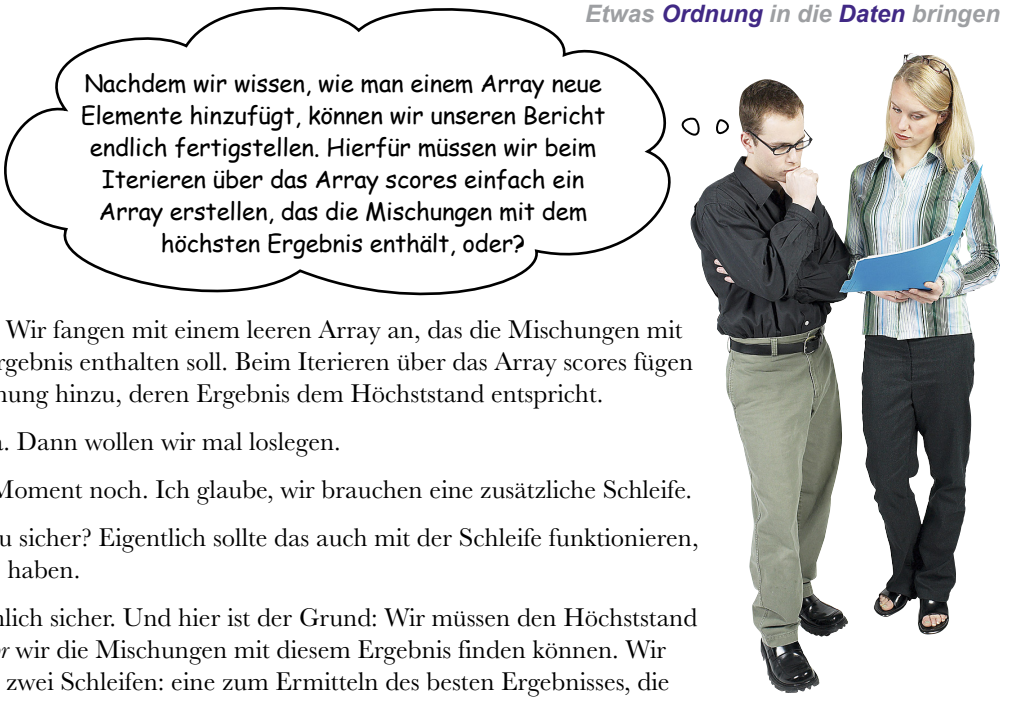
Beachten Sie, dass undefined nicht mit Anführungszeichen umgeben wird (weil es ein eigenständiger Wert ist, wie z. B. true oder false).

F: Bisher haben wir nur literale Arrays erstellt. Gibt es noch eine andere Methode, ein Array anzulegen?

A: Ja, vielleicht ist Ihnen schon einmal diese Syntax begegnet:

```
var myarray = new Array(3);
```

Diese Anweisung erzeugt ein neues Array mit drei leeren Elementen (also ein Array der Länge 3, das aber noch keine Werte besitzt). Sie können das Array nun wie üblich füllen, indem Sie den Indizes 0, 1 und 2 von myarray Werte zuweisen. Bis dahin haben die Elemente den Wert undefined. Ein auf diese Weise erstelltes Array ist das Gleiche wie ein Array-Literal. In der Praxis wird Ihnen deutlich häufiger die literale Syntax begegnen. Daher werden wir im weiteren Verlauf dieses Buchs ebenfalls die literale Schreibweise verwenden. Und machen Sie sich im Moment noch keine Sorgen um die Details der Syntax (wie »new« oder die Großschreibung von Array). Wir werden später genauer darauf eingehen.



Judy: Genau. Wir fangen mit einem leeren Array an, das die Mischungen mit dem besten Ergebnis enthalten soll. Beim Iterieren über das Array scores fügen wir jede Mischung hinzu, deren Ergebnis dem Höchststand entspricht.

Frank: Prima. Dann wollen wir mal loslegen.

Judy: Einen Moment noch. Ich glaube, wir brauchen eine zusätzliche Schleife.

Frank: Bist du sicher? Eigentlich sollte das auch mit der Schleife funktionieren, die wir bereits haben.

Judy: Ja, ziemlich sicher. Und hier ist der Grund: Wir müssen den Höchststand ermitteln, *bevor* wir die Mischungen mit diesem Ergebnis finden können. Wir brauchen also zwei Schleifen: eine zum Ermitteln des besten Ergebnisses, die wir bereits geschrieben haben, und eine zweite, mit der wir alle Mischungen finden, die dieses Ergebnis besitzen.

Frank: Oh, ich verstehe. Und in der zweiten Schleife vergleichen wir die einzelnen Ergebnisse mit dem Höchststand. Passen die Werte zusammen, fügen wir den Index der Mischung dem Array für die Mischungen mit dem besten Ergebnis hinzu.

Judy: Genau! Auf geht's.

Spitzen Sie Ihren Bleistift

Können Sie die Schleife schreiben, mit der wir die Ergebnisse mit dem Höchststand vergleichen? Probieren Sie es vor dem Umblättern erst einmal selbst. Auf der folgenden Seite zeigen wir Ihnen die Lösung und testen sie auch gleich.

Die Variable highScore enthält das beste gefundene Ergebnis. Das können Sie im unten stehenden Code verwenden.

```
var bestSolutions = [];
for (var i = 0; i < scores.length; i++) {



}
```

Hier ist das neue Array, in dem wir die Mischungen mit dem besten Ergebnis speichern wollen.

Und hier kommt Ihr Code hin.

Spitzen Sie Ihren Bleistift
Lösung

Können Sie die Schleife schreiben, um alle Mischungen zu finden, deren Wert dem Höchststand entspricht? Hier ist unsere Lösung.

Auch hier beginnen wir, indem wir ein neues Array mit allen Mischungen anlegen, deren Ergebnis dem Höchststand entspricht.

Danach führen wir eine Schleife über das gesamte scores-Array aus, um die Elemente zu finden, deren Wert dem Höchststand entsprechen.

```
var bestSolutions = [];

for (var i = 0; i < scores.length; i++) {
    if (scores[i] == highScore) {

        bestSolutions.push(i);

    }
}

console.log("Mischungen mit dem besten Ergebnis: " + bestSolutions);
```

Bei jedem Schleifendurchlauf vergleichen wir das Ergebnis beim Index i mit dem Höchststand. Sind die Werte gleich, fügen wir den Index mithilfe von push dem Array bestSolutions hinzu.

Danach können wir die Mischungen mit dem besten Ergebnis ausgeben. In diesem Fall benutzen wir console.log. Wir könnten auch eine weitere Schleife bauen, um die Array-Elemente einzeln auszugeben. Glücklicherweise erledigt console.log das aber auch so für uns (und wenn Sie sich die Ausgabe ansehen, werden Sie feststellen, dass zwischen den Array-Elementen sogar automatisch Kommata eingefügt werden!).

KOPF-NUSS

Sehen Sie sich noch einmal den Code in der »Spitzen Sie Ihren Bleistift«-Übung an. Was wäre, wenn es push plötzlich nicht mehr gäbe? Könnten Sie diesen Code auch ohne push schreiben? Probieren Sie es hier mal aus:

Probefahrt für den endgültigen Bericht

Jetzt können Sie den Code für die Ausgabe der Mischungen mit dem besten Ergebnis in die Datei »bubbles.html« einbauen und einen weiteren Probelauf durchführen. Den kompletten JavaScript-Code finden Sie unten:

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

var highScore = 0;
var output;
for (var i = 0; i < scores.length; i++) {
    output = "Mischung Nummer " + i + " Ergebnis: " + scores[i];
    console.log(output);
    if (scores[i] > highScore) {
        highScore = scores[i];
    }
}
console.log("Anzahl der Tests: " + scores.length);
console.log("Bestes Ergebnis: " + highScore);

var bestSolutions = [];
for (var i = 0; i < scores.length; i++) {
    if (scores[i] == highScore) {
        bestSolutions.push(i);
    }
}
console.log("Mischungen mit dem besten Ergebnis: " + bestSolutions);
```

Und die Gewinner sind ...

Die Mischungen Nummer 11 und Nummer 18 haben beide das beste Ergebnis: 69! Damit sind dies die besten Mischungen, die bei unseren Tests herausgekommen sind.

JavaScript-Konsole

```
Mischung Nummer 0 score: 60
Mischung Nummer 1 score: 50
 ...
Mischung Nummer 34 score: 51
Anzahl der Tests: 36
Bestes Ergebnis: 69
Mischungen mit dem besten Ergebnis: 11,18
```

Sie haben recht, das sollten wir. Da Sie die Verwendung von Funktionen gerade erst gelernt haben, war es uns wichtiger, dass Sie zunächst die Grundlagen von Arrays kennen, bevor wir die Funktionen in der Praxis anwenden. Davon abgesehen sollten Sie immer überlegen, welche Teile Ihres Codes Sie in eine Funktion auslagern können. Außerdem sind Funktionen eine gute Möglichkeit, die Arbeit, die Sie in die Entwicklung Ihrer Seifenblasenberechnungen gesteckt haben, auch anderen Entwicklern zur Verfügung zu stellen, damit diese das Rad nicht neu erfinden müssen.

Deshalb wollen wir uns den Code für die Seifenblasenmischungen noch einmal vornehmen und ihn in eine Reihe von Funktionen *refaktorieren*. Das heißt, wir organisieren den Code so um, dass er lesbarer und wartbarer wird. Dabei nehmen wir keine Änderungen an seiner Funktionsweise vor. Wenn wir fertig sind, macht der Code also genau das Gleiche wie jetzt auch, er ist aber wesentlich besser organisiert.

Eine schnelle Codeüberprüfung ...

Lassen Sie uns einen Blick auf den bisher geschriebenen Code werfen und herausfinden, welche Teile wir in Funktionen auslagern wollen:

Hier sehen Sie den Bubbles-R-Us-Code.

```
<!doctype html>
<html lang="de">
<head>
  <meta charset="utf-8">
  <title>Seifenblasen-Testlabor</title>
  <script>
    var scores = [60, 50, 60, 58, 54, 54,
                  58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51,
                  69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18,
                  41, 53, 55, 61, 51, 44];

    var highScore = 0;
    var output;

    for (var i = 0; i < scores.length; i++) {
      output = "Mischung Nummer " + i + " Ergebnis: " + scores[i];
      console.log(output);
      if (scores[i] > highScore) {
        highScore = scores[i];
      }
    }
    console.log("Anzahl der Tests: " + scores.length);
    console.log("Bestes Ergebnis: " + highScore);

    var bestSolutions = [];

    for (var i = 0; i < scores.length; i++) {
      if (scores[i] == highScore) {
        bestSolutions.push(i);
      }
    }

    console.log("Mischungen mit dem besten Ergebnis: " + bestSolutions);
  </script>
</head>
<body> </body>
</html>
```

Die Ergebnisse (im Array scores) wollen wir nicht in einer Funktion deklarieren, da sie für jede Benutzung der Funktion unterschiedlich sind. Stattdessen werden wir die einzelnen Ergebnisse als Argumente an die Funktionen übergeben. So können die Funktionen beliebige Ergebnisse verarbeiten.

Im ersten Codeabschnitt werden die einzelnen Ergebnisse ausgegeben, und gleichzeitig wird das beste Ergebnis im Array ermittelt. Diesen Teil könnten wir in eine printAndGetHighScore-Funktion (ausgebenUndHoechststandErmitteln) auslagern.

Den zweiten Codeabschnitt benutzen wir, um anhand des Höchststands die besten Seifenblasenmischungen herauszubekommen. Das könnten wir in eine getBestResults-Funktion auslagern.

Die printAndGetHighScore-Funktion schreiben

Den Code für die Funktion `printAndGetHighScore` haben wir bereits geschrieben. Um daraus eine Funktion zu machen, müssen wir nur wissen, welche Argumente übergeben werden sollen und ob irgendwelche Rückgabewerte gebraucht werden.

Es scheint sinnvoll, das Array scores zu übergeben, denn so können wir die Funktion auch für andere Arrays mit Messergebnissen verwenden. Außerdem wollen wir den berechneten Höchststand zurückgeben, damit der aufrufende Code weitere interessante Dinge damit anstellen kann (schließlich brauchen wir das Ergebnis, um die besten Mischungen zu ermitteln).

Oh, und noch etwas: Oftmals wollen Sie, dass Ihre Funktion genau *eine Sache* wirklich gut macht. Hier sind es zwei: Wir geben die Ergebnisse im Array aus, und gleichzeitig berechnen wir den Höchststand. Vielleicht sollten wir das auf zwei Funktionen aufteilen. So einfach, wie die Dinge sind, werden wir dieser Versuchung aber fürs Erste noch widerstehen. In einer professionellen Umgebung würden wir sicher darüber nachdenken, zwei Funktionen (z. B. `printScores` und `getHighScore`) daraus zu machen. Hier bleiben wir bei einer Funktion. Nun wollen wir mal mit dem Refaktorieren beginnen:

Unsere neue Funktion erwartet ein Argument: das scores-Array.

```
function printAndGetHighScore(scores) {
    var highScore = 0;
    var output;
    for (var i = 0; i < scores.length; i++) {
        output = "Mischung Nummer " + i + " Ergebnis: " + scores[i];
        console.log(output);
        if (scores[i] > highScore) {
            highScore = scores[i];
        }
    }
    return highScore;
}
```

Dieser Code macht das Gleiche wie vorher. Tatsächlich SIEHT er genau gleich AUS. Aber jetzt verwendet er den Parameter scores und nicht mehr die globale Variable gleichen Namens.

Außerdem haben wir eine Codezeile hinzugefügt, die den Höchststand (in der lokalen Variablen highScore) an den aufrufenden Code zurückgibt.

Den Code mit der Funktion printAndGetHighScore refaktorieren

Jetzt müssen wir den übrigen Code so umbauen, dass er unsere neue Funktion auch benutzt. Hier fügen wir einfach einen Aufruf der Funktion ein und weisen der globalen Variable `highScore` den Rückgabewert von `printAndGetHighScore` zu:

```
<!doctype html>
<html lang="de">
<head>
  <title>Seifenblasen-Testlabor</title>
  <meta charset="utf-8">
  <script>
    var scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51, 69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18, 41, 53, 55, 61, 51, 44];

    function printAndGetHighScore(scores) {
        var highScore = 0;
        var output;
        for (var i = 0; i < scores.length; i++) {
            output = "Mischung Nummer " + i + " Ergebnis: " + scores[i];
            console.log(output);
            if (scores[i] > highScore) {
                highScore = scores[i];
            }
        }
        return highScore;
    }

    var highScore = printAndGetHighScore(scores);
    console.log("Anzahl der Tests: " + scores.length);
    console.log("Bestes Ergebnis: " + highScore);

    var bestSolutions = [];

    for (var i = 0; i < scores.length; i++) {
      if (scores[i] == highScore) {
        bestSolutions.push(i);
      }
    }

    console.log("Mischungen mit dem besten Ergebnis: " + bestSolutions);
  </script>
</head>
<body> </body>
</html>
```

Unsere neue Funktion wartet nur darauf, benutzt zu werden.

Und hier rufen wir die Funktion einfach auf und übergeben ihr als Argument das Array scores. Ihren Rückgabewert weisen wir der globalen Variablen highScore zu.

Jetzt noch diesen Code zu einer Funktion refaktorieren und die nötigen Änderungen am übrigen Code durchführen.

Spitzen Sie Ihren Bleistift

Den nächsten Schritt wollen wir gemeinsam gehen. Das Ziel ist eine Funktion, die ein Array mit den Seifenblasenmischungen erstellt, deren Messergebnis dem Höchststand entspricht (und weil es mehr als eine Mischung geben kann, verwenden wir hier ein Array). Dieser Funktion übergeben wir das scores-Array und zusätzlich die zuvor mit der Funktion printAndGetHighScore berechnete Variable highScore. Stellen Sie den unten stehenden Code fertig. Die Lösung finden Sie auf der nächsten Seite – aber nicht abgucken! Versuchen Sie es zuerst allein, damit Sie auch etwas lernen.

Hier sehen Sie für alle Fälle noch mal den Originalcode.

```
var bestSolutions = [];
for (var i = 0; i < scores.length; i++) {
    if (scores[i] == highScore) {
        bestSolutions.push(i);
    }
}
console.log("Mischungen mit dem besten Ergebnis: " + bestSolutions);
```

Wir haben schon mal angefangen. Allerdings brauchen wir Ihre Hilfe, um die Funktion fertigzustellen.

```
function getBestResults(__________, ____________) {
    var bestSolutions = ______;
    for (var i = 0; i < scores.length; i++) {
        if (__________ == highScore) {
            bestSolutions.__________;
        }
    }
    return _______________;
}

var bestSolutions = _____________(scores, highScore);
console.log("Mischungen mit dem besten Ergebnis: " + bestSolutions);
```

Alles zusammengenommen ...

Sobald Sie mit der Refaktorierung des Codes fertig sind, sollten Sie Ihre Änderungen in »bubbles.html« speichern. Das fertige Programm soll jetzt aussehen wie unten gezeigt. Wenn Sie die Seite im Browser neu laden, sollte das Ergebnis so aussehen wie vorher. Aber Ihr Code ist jetzt besser organisiert und wartbarer. Erstellen Sie testweise Ihr eigenes scores-Array und versuchen Sie, den Code wiederzuverwenden!

```
<!doctype html>
<html lang="de">
<head>
  <meta charset="utf-8">
  <title>Seifenblasen-Testlabor</title>
  <script>
    var scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51, 69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18, 41, 53, 55, 61, 51, 44];

    function printAndGetHighScore(scores) {
        var highScore = 0;
        var output;
        for (var i = 0; i < scores.length; i++) {
            output = "Mischung Nummer " + i + " Ergebnis: " + scores[i];
            console.log(output);
            if (scores[i] > highScore) {
                highScore = scores[i];
            }
        }
        return highScore;
    }

    function getBestResults(scores, highScore) {
        var bestSolutions = [];
        for (var i = 0; i < scores.length; i++) {
            if (scores[i] == highScore) {
                bestSolutions.push(i);
            }
        }
        return bestSolutions;
    }

    var highScore = printAndGetHighScore(scores);
    console.log("Anzahl der Test: " + scores.length);
    console.log("Bestes Ergebnis: " + highScore);

    var bestSolutions = getBestResults(scores, highScore);
    console.log("Mischungen mit dem besten Ergebnis: " + bestSolutions);

  </script>
</head>
<body> </body>
</html>
```

Hier sehen Sie die neue Funktion getBestResults.

Den Rückgabewert der Funktion verwenden wir, um die Mischungen mit dem besten Ergebnis anzuzeigen.

Gute Arbeit! Jetzt fehlt nur noch eine Sache … können Sie auch herausfinden, welche Seifenblasenmischung am kosteneffektivsten ist? Mit diesen zusätzlichen Informationen übernehmen wir mit Sicherheit den gesamten Seifenblasenmarkt. Hier haben Sie ein Array mit den Kosten für jede Mischung, das Sie für diese Aufgabe benutzen können.

Hier ist das Array. Jede Kostenangabe entspricht der Seifenblasenmischung mit dem gleichen Index im Array scores.

```
var costs = [.25, .27, .25, .25, .25, .25,
             .33, .31, .25, .29, .27, .22,
             .31, .25, .25, .33, .21, .25,
             .25, .25, .28, .25, .24, .22,
             .20, .25, .30, .25, .24, .25,
             .25, .25, .27, .25, .26, .29];
```

Was ist hier zu tun? Es geht darum, die beste Seifenblasenmischung zu finden. Hierfür brauchen wir die Mischungen mit dem besten Testergebnis und müssen hieraus diejenige mit den geringsten Kosten auswählen. Zum Glück haben wir das Array `costs`, das in seiner Struktur dem `scores`-Array entspricht. Das heißt, die Kosten für die Mischung mit dem Index 0 (in `scores`) finden wir im `costs`-Array ebenfalls beim Index 0 (.25). Die Kosten für die Mischung mit dem Index 1 finden wir im `costs`-Array beim Index 1 (0.27) und so weiter. Für jedes Messergebnis finden Sie die entsprechenden Kosten im Array `costs` am gleichen Index. Man spricht in diesem Fall auch von *parallelen* Arrays.

Die Arrays scores und costs sind parallel, da es für jedes Messergebnis in scores eine entsprechende Kostenangabe in costs mit dem gleichen Index gibt.

```
var costs = [.25, .27, .25, .25, .25, .25, .33, .31, .25, .29, .27, .22, ..., .29];
var scores = [60,  50,  60,  58,  54,  54,  58,  50,  52,  54,  48,  69,  ..., 44];
```

Die Kosten für die Mischung mit dem Index 0 haben ebenfalls den Index 0 …

Das Gleiche gilt für die übrigen Werte in den Arrays costs und scores.

Judy: Na ja, das beste Messergebnis kennen wir ja schon.

Frank: Außerdem haben wir jetzt zwei Arrays. Wie bringen wir beide zur Zusammenarbeit?

Judy: Ich bin ziemlich sicher, dass wir eine einfache Schleife schreiben können, die das scores-Array noch einmal durchläuft und die Elemente heraussucht, die dem Höchststand entsprechen.

Frank: Ja, das könnte ich machen. Aber was dann?

Judy: Jedes Mal, wenn wir ein Ergebnis finden, das dem besten Messergebnis entspricht, testen wir, ob es die geringsten Kosten aufweist.

Frank: Ah, ich verstehe. Wir bräuchten eine Variable, die den Index des »kosteneffektivsten besten Messergebnisses« speichert – ein ziemlich dicker Brocken.

Judy: Genau. Und nachdem wir das gesamte Array durchlaufen haben, finden wir in der Variablen den Index des Elements, das nicht nur das beste Messergebnis hat, sondern auch am wenigsten kostet.

Frank: Was machen wir, wenn zwei Elemente die gleichen Kosten haben?

Judy: Da müssen wir wohl eine Entscheidung treffen. Ich würde sagen, das zuerst gefundene Element ist der Gewinner. Das geht natürlich auch noch komplizierter, aber solange der CEO nichts anderes sagt, bleiben wir dabei.

Frank: Das ist ja auch schon schwer genug. Ich glaube, wir sollten die Sache vor dem Programmieren wieder im Pseudocode skizzieren.

Judy: Einverstanden. Bei der Arbeit mit Array-Indizes kann es schnell zu Schwierigkeiten kommen. Machen wir's so. Auf lange Sicht geht es sowieso schneller, wenn man vorher ordentlich plant.

Frank: In Ordnung. Ich mache mal einen ersten Anlauf …

Ich bin mir ziemlich sicher, dass ich den Pseudocode gut hinbekommen habe. Jetzt können Sie ihn in richtigen Code übersetzen. Vergessen Sie nicht, Ihre Lösung zu überprüfen.

```
FUNCTION GETMOSTCOSTEFFECTIVESOLUTION (SCORE, COSTS, HIGHSCORE)
DEKLARIERE eine Variable mit dem Namen cost und setze sie auf 100.
DEKLARIERE eine Variable mit dem Namen index.
FOR (Schleife): var i=0; i < scores.length; i++
    FALLS (if) die Mischung bei score[i] das beste Ergebnis hat
        FALLS (if) der aktuelle Wert von cost größer ist als die Kosten der Seifenblasenmischung
        DANN
            SETZE den Wert von index auf den Wert von i
            SETZE den Wert von cost auf die Kosten der Seifenblasenmischung
        ENDE FALLS (if)
    ENDE FALLS (if)
ENDE FOR (Schleife)
RÜCKGABE VON index
```

```
function getMostCostEffectiveSolution(scores, costs, highScore) {
```

← Übersetzen Sie hier den Pseudocode in JavaScript.

```
}
var mostCostEffective = getMostCostEffectiveSolution(scores, costs, highScore);
console.log("Mischung Nummer " + mostCostEffective + " ist am kosteneffektivsten");
```

DER GEWINNER: MISCHUNG NUMMER 11

Das letzte von Ihnen geschriebene Codestück hat sehr dabei geholfen, den WAHREN Gewinner zu ermitteln. Wir haben jetzt die Mischung, die die meisten Seifenblasen bei den niedrigsten Kosten erzeugt. Herzlichen Glückwunsch dazu, dass Sie einen Haufen Daten in etwas verwandelt haben, mit dem Bubbles-R-Us echte Geschäfte machen kann.

Wenn Sie und wir uns ein bisschen ähnlich sind, dann wollen Sie jetzt wissen, was die Mischung Nummer 11 enthält. Suchen Sie nicht weiter; der CEO von Bubbles-R-Us hat entschieden, Ihnen das Rezept als Belohnung für Ihre ganze unbezahlte Arbeit zu überlassen.
Unten sehen Sie das Rezept für die Seifenblasenmischung Nummer 11. Geben Sie Ihrem Gehirn Gelegenheit, die Arrays in Ruhe zu verarbeiten, indem Sie ein paar Seifenblasen blasen, bevor Sie zum nächsten Kapitel übergehen. Oh, und vergessen Sie nicht die »Punkt für Punkt«-Liste und das Kreuzworträtsel, bevor Sie weiterlesen!

Seifenblasenmischung Nummer 11

4 Esslöffel grüne Seife (Schmierseife)

1 Liter Wasser

4 Esslöffel Glycerin (aus der Apotheke)

ANWEISUNG: Mischen Sie die Zutaten in einer großen Schüssel, und dann viel Spaß!!

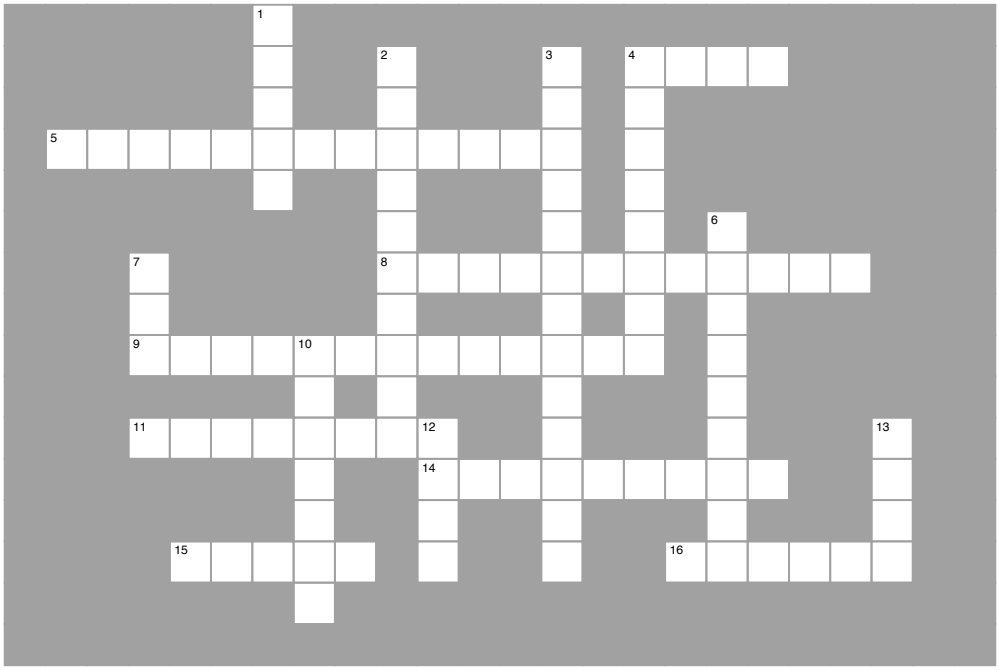
Probieren Sie das AUF JEDEN FALL zu Hause aus!

Punkt für Punkt

- Arrays sind **Datenstrukturen** für geordnete Daten.
- Ein Array enthält eine Reihe von Elementen, jeweils mit einem eigenen **Index**.
- Arrays benutzen einen nullbasierten Index, bei dem das erste Element den Index 0 erhält.
- Alle Arrays besitzen die Eigenschaft **length**. Sie enthält die Anzahl der Array-Elemente.
- Über den Index können Sie auf einzelne Array-Elemente zugreifen. Auf das Element mit dem Index 1 im Array myArray (das zweite Element) erhalten Sie beispielsweise per myArray[1] Zugriff.
- Versuchen Sie, auf ein nicht existentes Array-Element zuzugreifen, ist der Rückgabewert undefined.
- Weisen Sie einem bereits vorhandenen Element einen Wert zu, wird der ursprüngliche Wert überschrieben.
- Die Zuweisung zu einem noch nicht existierenden Element sorgt dafür, dass dieses Element angelegt wird.
- Ein Array-Element kann Werte beliebigen Typs enthalten.
- Die Werte in einem Array müssen nicht vom gleichen Typ (String, Zahl usw.) sein.
- Benutzen Sie die **Schreibweise für Array-Literale**, um ein neues Array anzulegen.
- Mit folgendem Code können Sie ein leeres Array anlegen:
  `var myArray = [ ];`
- Um über ein Array zu iterieren, wird üblicherweise eine **for-Schleife** verwendet.
- Die for-Schleife kombiniert Variableninitialisierung, eine Bedingung und das Inkrementieren der Variablen in einer gemeinsamen Anweisung.
- Die while-Schleife wird eher benutzt, wenn Sie nicht wissen, wie oft die Schleife durchlaufen werden soll, und die Schleife so lange laufen soll, bis die Bedingung erfüllt ist.
- Lückenhafte Arrays können auftreten, wenn sich im Array undefinierte Elemente befinden.
- Mit dem **Postinkrement**-Operator (++) können Sie den Wert einer Variablen um eins erhöhen.
- Mit dem **Postdekrement**-Operator (--) können Sie den Wert einer Variablen um eins verringern.
- Mit **push** können Sie ein neues Element in ein Array einfügen.

JavaScript-Kreuzworträtsel

Helfen Sie Ihrem Gehirn mit diesem Kreuzworträtsel, Arrays zu verinnerlichen.



WAAGERECHT

4. Wie viele Seifenblasenmischungen hatten das höchste Ergebnis?

5. Ein Array mit undefinierten Werten bezeichnet man auch als _____ Array.

8. Funktionen können helfen, Ihren Code besser zu _____.

9. Durch _____ können Sie Ihren Code lesbarer und wartbarer machen.

11. Arrays eignen sich gut zum Speichern von _____ Werten.

14. Diesen Wert erhält ein Array-Element, wenn Sie selbst keinen angeben.

15. Jeder Wert in einem Array wird an einem bestimmten _____ abgelegt.

16. Beim Iterieren über ein Array benutzen wir meistens die Eigenschaft _____, um zu ermitteln, wann wir anhalten müssen.

SENKRECHT

1. Um auf ein Array-Element zuzugreifen, verwenden Sie dessen _____ in eckigen Klammern.

2. Wer dachte, er hätte die beste Seifenblasenmischung?

3. Mit diesem Operator inkrementieren wir eine Schleifenvariable.

4. Um einen Wert in einem Array zu ändern, müssen Sie dem Element nur einen neuen Wert _____.

6. Ein Array ist eine _____ Datenstruktur.

7. In der Regel wird eine _____ -Schleife benutzt, um über ein Array zu iterieren.

10. Der letzte Index eines Arrays ist immer um eins _____ als die Länge (length) des Arrays.

12. Der Index des ersten Array-Elements ist _____.

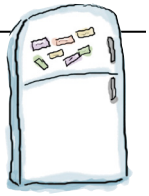
13. Mithilfe von _____ kann man am Ende eines Arrays ein neues Element hinzufügen.

Spitzen Sie Ihren Bleistift
Lösung

Das Array products enthält die Eiscreme-Sorten von Jenn und Berry's. Die Sorten wurden in der Reihenfolge ihrer Erstellung ergänzt. Vervollständigen Sie den Code, um herauszubekommen, welche Sorte *zuletzt* hinzugefügt wurde. Hier ist unsere Lösung.

```
var products = ["Mousse au Chocolat", "Minze", "Kuchenstreusel", "Kaugummi"];
var last = products.length - 1;
var recent = products[last];
```

Um den Index des letzten Elements zu bekommen, subrahieren wir 1 von der Länge (length) des Arrays. Die Länge ist 4, also ist der Index des letzten Elements 3.



Codemagneten, Lösung

Mit diesem Code überprüfen wir, welche Eiscreme-Sorten Kaugummi enthalten. Wir hatten die Codeschnipsel am Kühlschrank bereits richtig angeordnet, aber dann sind sie heruntergefallen. Es ist Ihre Aufgabe, sie wieder richtig zusammenzusetzen. Vorsicht: Einige Magneten werden nicht gebraucht. Hier ist unsere Lösung.

```
var products = ["Mousse au Chocolat",
                "Minze", "Kuchenstreusel",
                "Kaugummi"];

var hasBubbleGum = [false,
                    false,
                    false,
                    true];

var i = 0;

while (i < hasBubbleGum.length)

{

if (hasBubbleGum[i])

{

console.log(products[i] +
    " contains bubble gum");

}

i = i + 1;

}
```

Bringen Sie diese Magneten wieder in die richtige Reihenfolge.

Überflüssige Magneten.

```
{

i = i + 2;

while (i > hasBubbleGum.length)
```

Hier sehen Sie die erwartete Ausgabe.

JavaScript-Konsole

```
Kaugummi enthält Kaugummi!
```

Spitzen Sie Ihren Bleistift
Lösung

Schreiben Sie den Codemagnetencode (zwei Seiten vorher) neu, sodass er anstelle der while- eine for-Schleife benutzt. Um zu sehen, wie die Positionen auf die for-Schleife abgebildet werden, können Sie sich die einzelnen Teile der while-Schleife auf der vorigen Seite noch einmal ansehen.

```
var products = ["Mousse au Chocolat",
                "Minze", "Kuchenstreusel",
                "Kaugummi"];
var hasBubbleGum = [false,
                    false,
                    false,
                    true];
var i = 0;
while (i < hasBubbleGum.length)
{
if (hasBubbleGum[i])
{
console.log(products[i] +
    " enthält Kaugummi");
}
i = i + 1;
}
```

Hier kommt Ihr Code hin.

```
var products = ["Mousse au Chocolat",
                "Minze", "Kuchenstreusel",
                "Kaugummi"];
var hasBubbleGum = [false,
                    false,
                    false,
                    true];
for (var i = 0; i < hasBubbleGum.length; i = i + 1) {
    if (hasBubbleGum[i]) {
        console.log(products[i] + " enthält Kaugummi");
    }
}
```

Spitzen Sie Ihren Bleistift
Lösung

Jetzt können Sie den Pseudocode zum Finden des besten Ergebnisses implementieren. Füllen Sie hierfür die Lücken im unten stehenden Code. Danach können Sie Ihren Code im Browser testen, indem Sie die Daten in »bubbles.html« aktualisieren und die Seite neu laden. Überprüfen Sie Ihre Ergebnisse in der Konsole und schreiben Sie Ihren Höchststand und die Anzahl der Tests auf die leeren Zeilen in unserem Konsolenfenster unten auf dieser Seite. Hier ist unsere Lösung.

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

var highScore = 0;
var output;
for (var i = 0; i < scores.length; i++) {
    output = "Mischung Nummer " + i + " Ergebnis: " + scores[i];
    console.log(output);
    if ( scores[i] > highScore) {
        highScore = scores[i];
    }
}
console.log("Anzahl der Tests: " + scores.length );
console.log("Bestes Ergebnis: " + highScore );
```

Füllen Sie die leeren Zeilen, um den hier stehenden Code zu vervollständigen ...

... und ergänzen Sie dann die Leerzeilen mit Ihren Ergebnissen von der Konsole.

JavaScript-Konsole

```
Mischung Nummer 0 Ergebnis: 60
Mischung Nummer 1 Ergebnis: 50
Mischung Nummer 2 Ergebnis: 60
 ...
Mischung Nummer 34 Ergebnis: 51
Mischung Nummer 35 Ergebnis: 44
Anzahl der Tests: 36
Bestes Ergebnis: 69
```

Lösung zur Übung

Hier ist unsere Lösung für die Funktion getMostCostEffectiveSolution. Sie übernimmt ein Array mit Messergebnissen, ein Array mit den Kosten und einen Höchststand. Basierend auf diesen Informationen, ermittelt die Funktion den Index der Seifenblasenmischung mit dem besten Ergebnis und den niedrigsten Kosten. Probieren Sie Ihren gesamten Code in »bubbles.html« einmal aus, um zu sehen, ob wir die gleichen Ergebnisse haben.

Die Funktion getMostCostEffectiveSolution übernimmt zwei Arrays mit Messergebnissen und den Kosten sowie den Höchststand.

```
function getMostCostEffectiveSolution(scores, costs, highscore) {
    var cost = 100;
    var index;

    for (var i = 0; i < scores.length; i++) {
        if (scores[i] == highScore) {
            if (cost > costs[i]) {
                index = i;
                cost = costs[i];
            }
        }
    }
    return index;
}
var mostCostEffective = getMostCostEffectiveSolution(scores, costs, highScore);
console.log("Mischung Nummer " + mostCostEffective + " ist am kosteneffektivsten");
```

Die Variable cost speichert die Mischung mit den geringsten Kosten ...

... und den Index der Mischung mit den niedrigsten Kosten in der Variablen index.

Wir initialisieren die Variable cost mit einer hohen Zahl und reduzieren sie, sobald wir eine Mischung mit geringeren Kosten (und dem Höchststand) finden.

Wie zuvor iterieren wir über das Array scores ...

... und überprüfen, ob das Messergebnis dem Höchststand entspricht.

In diesem Fall überprüfen wir die Kosten. Sind die aktuellen Kosten höher als die Kosten der Mischung, haben wir eine günstigere Mischung gefunden. Wir identifizieren die Mischung (über ihren Array-Index) und speichern sie als die bisher geringsten Kosten in der Variablen cost.

Ist die Schleife durchlaufen, wird der Index der Mischung mit den geringsten Kosten in der Variablen index gespeichert, damit wir sie an den aufrufenden Code zurückgeben können.

Und schließlich geben wir den Index (entsprechend der Nummer der Seifenblasenmischung) auf der Konsole aus.

JavaScript-Konsole

```
Mischung Nummer 0 Ergebnis: 60
Mischung Nummer 1 Ergebnis: 50
Mischung Nummer 2 Ergebnis: 60
 ...
Mischung Nummer 34 Ergebnis: 51
Mischung Nummer 35 Ergebnis: 44
Anzahl der Tests: 36
Bestes Ergebnis: 69
Mischungen mit dem besten Ergebnis: 11,18
Mischung Nummer 11 ist am kosteneffektivsten
```

Der abschließende Bericht: Mischung Nummer 11 ist der Gewinner. Sie hat den größten Blasenfaktor bei den geringsten Kosten.

BONUS: Wir könnten diese Funktionalität auch mit dem Array bestSolutions implementieren. Dadurch müssten die Ergebnisse nicht erneut durchlaufen werden. Denken Sie daran, dass das Array bestSolutions die Indizes der Mischungen mit dem besten Ergebnis bereits enthält. In diesem Code würden wir die Elemente in bestSolutions verwenden, um über ihre Indizes auf das Array costs zuzugreifen und so die Kosten direkt zu vergleichen. Dieser Code wäre etwas effizienter, aber schlechter lesbar und weniger verständlich. Falls es Sie interessiert, finden Sie die Lösung im Beispielcode zu diesem Buch unter der Adresse http://examples.oreilly.de/german_examples/hfjavascriptprogger/Beispielcode.zip.

JavaScript-Kreuzworträtsel, Lösung

Helfen Sie Ihrem Gehirn mit diesem Kreuzworträtsel, Arrays zu verinnerlichen.

				[1]I															
				N			[2]B				[3]P		[4]Z	W	E	I			
				D			L				O		U						
[5]L	U	E	C	K	E	N	H	A	F	T	E	S		W					
				X			S				T		E						
							E				I		I		[6]G				
		[7]F					[8]O	R	G	A	N	I	S	I	E	R	E	N	
		O					L				K		E		O				
		[9]R	E	F	A	[10]K	T	O	R	I	E	R	E	N		R			
					L		G				E				D				
		[11]M	E	H	R	E	R	E	[12]N			M				N			[13]P
					I			[14]U	N	D	E	F	I	N	E	D			U
					N			L			N				T				S
			[15]I	N	D	E	X		L			T			[16]L	E	N	G	T	H
					R