Einleitung

In diesem Buch geht es darum, Computern Anweisungen zu geben. Computer sind heutzutage so alltäglich geworden wie Schraubenzieher, allerdings deutlich komplizierter. Deshalb ist es nicht immer einfach, sie auch wirklich das tun zu lassen, was man will.

Wenn Sie Ihren Computer für eine übliche und klar umrissene Aufgabe einsetzen möchten, z. B. um E-Mails anzuzeigen oder Berechnungen wie mit einem Taschenrechner durchzuführen, können Sie einfach die entsprechende Anwendung öffnen und loslegen. Für besondere Aufgaben gibt es dagegen möglicherweise noch keine Anwendung.

An dieser Stelle kommt die *Programmierung* ins Spiel. Dabei handelt es sich um den Vorgang, ein *Programm* zu erstellen, also eine Folge genauer Anweisungen, die dem Computer sagen, was er tun soll. Da Computer stumpfsinning und pedantisch sind, ist Programmierung im Grunde zunächst mühselig und frustrierend.

Wenn Sie jedoch darüber hinwegkommen und vielleicht sogar Freude an einer Denkweise in strengen Bahnen finden, die auch eine stumpfe Maschine versteht, kann Programmierung lohnenswert sein. Denn damit lassen sich in Sekunden Dinge erledigen, die sonst ewig dauern würden. Sie bietet eine Möglichkeit, Ihr Werkzeug, den Computer, Aufgaben ausführen zu lassen, die er zuvor nicht beherrschte. Und außerdem ist es eine hervorragende Übung für abstraktes Denken.

Programmierung erfolgt mithilfe einer *Programmiersprache*. Dabei handelt es sich um eine künstliche Sprache, die dazu dient, Computern Anweisungen zu erteilen. Es ist schon bemerkenswert, dass sich die effektivste Möglichkeit zur Kommunikation mit Computern, die wir erfunden haben, so stark an die Art und Weise anlehnt, wie wir miteinander kommunizieren. Ebenso wie in menschlichen Sprachen können auch in Computersprachen Wörter und Ausdrücke kombiniert werden, um Ideen auszudrücken.

Textschnittstellen wie die BASIC- und DOS-Eingabeaufforderungen der 80er und 90er bildeten einst die Hauptmethode für die Kommunikation mit Computern. Mittlerweile wurden sie größtenteils durch grafische Schnittstellen ersetzt,

die leichter zu erlernen sind, aber weniger Freiheiten bieten. Die Computersprachen jedoch sind immer noch vorhanden. Sie müssen nur wissen, wo Sie danach zu suchen haben. Eine dieser Sprachen, nämlich JavaScript, ist in jeden modernen Webbrowser eingebaut und steht daher auf fast jedem Gerät zur Verfügung.

Dieses Buch soll Ihnen helfen, sich so weit mit dieser Sprache vertraut zu machen, dass Sie sie für nützliche und unterhaltsame Zwecke einsetzen können.

Programmierung

Neben JavaScript werde ich auch die Grundlagen der Programmierung erklären. Programmieren ist schwer. Die Grundregeln sind zwar einfach und deutlich, aber die nach diesen Regeln aufgebauten Programme geraten gewöhnlich so vielschichtig, dass sie ihre eigenen Regeln und eine eigene Komplexität aufweisen. In gewissem Sinne bauen Sie ein Labyrinth, in dem Sie sich selbst auch verirren können.

Bei der Lektüre dieses Buches werden Sie sich hin und wieder furchtbar frustriert fühlen. Wenn Programmierung ganz neu für Sie ist, gibt es viel Stoff zu verdauen. Eine Menge dieses Stoffs wird dann auf eine Weise miteinander kombiniert, die es erfordert, zusätzliche Verbindungen herzustellen.

Dazu müssen Sie sich anstrengen. Wenn Sie Schwierigkeiten haben, dem Text zu folgen, dann gehen Sie nicht vorschnell mit sich selbst ins Gericht. Mit Ihnen ist alles in Ordnung – Sie dürfen nur nicht aufgeben! Legen Sie eine Pause ein, lesen Sie etwas erneut und achten Sie darauf, die Beispielprogramme und Übungen nachzuvollziehen und zu begreifen. Lernen ist harte Arbeit. Aber alles, was Sie lernen, gehört zu Ihnen und macht das weitere Lernen einfacher.

> »Wenn Aktionen unrentabel werden, sammeln Sie Informationen; wenn Informationen unrentabel werden, legen Sie sich schlafen.«
>
> – Ursula K. LeGuin, *The Left Hand of Darknes* (auf Deutsch erschienen als *Winterplanet* und als *Die linke Hand der Dunkelheit*)

Ein Programm ist vieles zugleich: ein von einem Programmierer geschriebener Text; ein Datenpaket im Arbeitsspeicher des Computers, wo es Aktionen desselben steuert – und damit die lenkende Kraft, die dafür sorgt, dass der Computer das macht, was er tut. Alle Analogien, die Programme mit vertrauten Objekten des Alltags vergleichen, greifen zu kurz. Ein oberflächlicher Vergleich ist der mit einer Maschine: Viele einzelne Teile arbeiten zusammen, damit das Ganze läuft, und wir müssen Möglichkeiten finden, um diese Teile zu verzahnen, damit sie ihren Beitrag zum Funktionieren der Gesamtheit leisten können.

Ein Computer ist eine physische Maschine, die als Wirt solcher immateriellen Maschinen dient. Für sich allein genommen können Computer lediglich ganz einfache Dinge tun. Sie sind nur deshalb so nützlich, weil sie das mit unglaublich hoher Geschwindigkeit erledigen. Ein Programm kann nun auf raffinierte Art und Weise enorme Mengen solcher einfachen Aktionen kombinieren, um so äußerst komplizierte Aufgaben zu erfüllen.

Ein Programm ist ein Gedankenkonstrukt. Seine Erstellung kostet nichts, es ist gewichtslos und wächst rasch, während Sie die Tastatur bearbeiten. Wenn Sie nicht aufpassen, können Größe und Komplexität eines Programms rasch unkontrolliert zunehmen, sodass schließlich selbst die Person, die es geschrieben hat, den Überblick verliert und es nicht mehr richtig versteht. Das größte Herausforderung der Programmierung besteht darin, die Programme unter Kontrolle zu halten. Wenn ein Programm funktioniert, ist es schön. Die Kunst der Programmierung ist die Fähigkeit, die Komplexität im Zaum zu halten. Großartige Programme sind schlicht – also in all ihrer Komplexität möglichst einfach gestaltet.

Manche Programmierer sind der Meinung, dass sie dieser Komplexität am besten Herr werden, indem sie nur einige wenige, gut verstandene Techniken für ihre Programme verwenden. Dazu haben sie strenge Regeln (»empfohlene Vorgehensweisen«, auch »Best Practices«) aufgestellt, die vorschreiben, welche Form Programme haben sollen, und achten sorgfältig darauf, nicht aus dieser eng begrenzten Sicherheitszone auszubrechen.

Das ist jedoch nicht nur langweilig, sondern auch ineffektiv. Neue Probleme erfordern oft neue Lösungen. Programmierung ist eine junge Disziplin, die sich immer noch rasant weiterentwickelt, und sie ist vielgestaltig genug, um Platz für sehr unterschiedliche Vorgehensweisen zu bieten. Es gibt viele furchtbare Fehler, die einem bei der Programmgestaltung unterlaufen können. Sie sollten diese Fehler ruhig machen, damit Sie sie verstehen. Ein Gespür dafür, wie ein gutes Programm auszusehen hat, entwickeln Sie durch praktische Anwendung. Aus einem Satz von Regeln lernen Sie das nicht.

Warum es auf die Sprache ankommt

In den Anfangstagen der Computer gab es noch keine Programmiersprachen. Programme sahen damals wie folgt aus:

```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```

Dieses Programm (für einen sehr einfachen, hypothetischen Computer) addiert die Zahlen von 1 bis 10 und gibt das Resultat aus: `1 + 2 + ... + 10 = 55`. Um die ersten Computer zu programmieren, musste man lange Reihen von Schaltern in die richtige Stellung bringen oder Löcher in Kartonkarten knipsen und in die Maschine einspeisen. Sie können sich vorstellen, wie mühselig und fehleranfällig das war. Selbst um einfache Programme zu schreiben, waren sehr viel Geschick und Disziplin erforderlich. Komplexe Programme waren damals geradezu unvorstellbar.

Allerdings verlieh die manuelle Eingabe dieser obskuren Muster aus Bits (Nullen und Einsen) den Programmierern das tiefe Gefühl, mächtige Zauberer zu sein. Das war schon viel wert, was die Zufriedenheit am Arbeitsplatz anging.

Jede Zeile des vorigen Programms bildet eine einzige Anweisung. Ins Deutsche übersetzt sehen diese Anweisungen wie folgt aus:

1. Speichere die Zahl 0 am Speicherort 0.
2. Speichere die Zahl 1 am Speicherort 1.
3. Speichere den Wert von Speicherort 1 im Speicherort 2.
4. Subtrahiere die Zahl 11 von dem Wert im Speicherort 2.
5. Wenn der Wert in Speicherort 2 die Zahl 0 ist, fahre mit Anweisung 9 fort.
6. Addiere den Wert von Speicherort 1 zu Speicherort 0.
7. Addiere die Zahl 1 zum Wert von Speicherort 1.
8. Fahre mit Anweisung 3 fort.
9. Gib den Wert von Speicherort 0 aus.

Das ist zwar schon besser lesbar als die Bit-Suppe, aber immer noch ziemlich unverständlich. Es wird besser, wenn wir für die Anweisungen und Speicherorte Namen statt Zahlen verwenden:

```
 Set "total" to 0.
 Set "count" to 1.
[loop]
 Set "compare" to "count".
 Subtract 11 from "compare".
 If "compare" is zero, continue at [end].
 Add "count" to "total".
 Add 1 to "count".
 Continue at [loop].
[end]
 Output "total".
```

Können Sie erkennen, wie das Programm abläuft? Die ersten beiden Zeilen geben zwei Speicherorten ihre Startwerte: `total` wird verwendet, um nach und nach das Ergebnis der Berechnung aufzubauen, während sich `count` jeweils merkt, welche Zahl wir uns gerade ansehen. Die Zeilen, in denen `compare` verwendet wird, sind wahrscheinlich die absonderlichsten. Das Programm prüft hier, ob `count` gleich 11 ist, um zu entscheiden, ob es die Ausführung beenden kann. Da unser hypothetischer Computer ziemlich primitiv ist, kann er nur prüfen, ob eine Zahl gleich 0 ist, und seine Entscheidung darauf gründen. Daher verwendet er den Speicherort `compare`, um den Wert von `count - 11` zu berechnen, und stellt den Vergleich damit an. Die nächsten beiden Zeilen addieren den Wert von `count` zu dem Ergebnis und setzen `count` um 1 herauf, solange dieser Wert noch ungleich 11 ist.

In JavaScript sieht das gleiche Programm wie folgt aus:

```
let total = 0, count = 1;
while (count <= 10) {
 total += count;
 count += 1;
}
console.log(total);
// → 55
```

Diese Version bietet einige weitere Verbesserungen. Vor allem haben wir es hier nicht mehr nötig, das Programm ausdrücklich anzuweisen, vor- oder zurückzuspringen. Darum kümmert sich das `while`-Konstrukt. Es setzt die Ausführung des dahinter stehenden Blocks (in geschweiften Klammern) so lange fort, wie die ihm übergebene Bedingung zutrifft. Diese Bedingung lautet `count <= 10`, was bedeutet: »`count` ist kleiner oder gleich 10.« Wir müssen hier nicht mehr einen Zwischenwert berechnen und mit 0 vergleichen, was ohnehin ein für das eigentliche Ziel unbedeutender Detailmechanismus war. Programmiersprachen sind unter anderem deshalb so leistungsfähig, weil sie uns solche uninteressanten Einzelheiten abnehmen. Am Ende des Programms, nach dem Abschluss des `while`-Konstruktors, sorgt die Operation `console.log` dafür, dass das Resultat ausgegeben wird.

Wenn uns die bequemen Operationen `range` und `sum` zur Verfügung stehen, die eine Zahlenmenge aus einem gegebenen Bereich zusammenstellen bzw. die Summe einer Menge von Zahlen berechnen können, sind wir in der Lage, das Programm wie folgt zu schreiben:

```
console.log(sum(range(1, 10)));
// → 55
```

Die Moral von der Geschichte ist, dass sich Programme auf verschiedene Weisen ausdrücken lassen – lang und kurz, unverständlich und verständlich. Die erste Version des Programms war extrem obskur, während die letzte schon fast wie Deutsch aussieht: »*Protokolliere* die *Summe* des *Bereichs* der Zahlen von 1 bis 10.« Wie sich Operationen wie `sum` und `range` definieren lassen, werden wir uns in den späteren Kapiteln noch ansehen.

Eine gute Programmiersprache unterstützt Programmierer, indem sie ihnen erlaubt, die groben Aktionen, die der Computer durchführen soll, zu beschreiben, und die Details wegzulassen. Sie stellt komfortable Bausteine bereit (wie `while` und `console.log`), ermöglicht die Definition eigener Bausteine (wie `sum` und `range`) und erleichtert es, diese Bausteine zu kombinieren.

Was ist JavaScript?

JavaScript wurde 1995 als eine Möglichkeit eingeführt, um Webseiten im Netscape Navigator Programme hinzuzufügen. Die Sprache wurde seitdem von allen anderen wichtigen grafischen Browsern übernommen. Sie hat moderne Webanwendungen möglich gemacht, mit denen Sie direkt arbeiten können, ohne nach jeder Aktion auf einen neuen Seitenaufbau warten zu müssen. JavaScript wird auch auf

herkömmlichen Webseiten eingesetzt, um verschiedenen Formen der Interaktivität und ausgeklügelte Funktionen zu bieten.

Beachten Sie jedoch, dass JavaScript so gut wie nichts mit der Programmiersprache Java zu tun hat. Der Name wurde eher aus Marketinggründen gewählt. Bei der Einführung von JavaScript wurde Java stark angepriesen und nahm an Beliebtheit zu. Jemand hielt es für eine gute Idee, auf der Erfolgswelle mitzuschwimmen. Bei dem Namen ist es dann geblieben.

Nach der Übernahme in andere Browser als Netscape wurde ein Normdokument verfasst, um zu beschreiben, wie sich JavaScript verhalten soll, damit auch tatsächlich immer die gleiche Sprache gemeint ist, wenn Software behauptet, JavaScript zu unterstützen. Dieser Standard wird nach der Organisation Ecma International, die die Normung vorgenommen hat, als ECMAScript bezeichnet. In der Praxis können die Begriffe ECMAScript und JavaScript synonym verwendet werden – es sind zwei Bezeichnungen für dieselbe Sprache.

Manche Personen werden Ihnen *fürchterliche* Dinge über JavaScript erzählen. Viele davon sind wahr. Als ich zum ersten Mal etwas in JavaScript schreiben musste, lernte ich die Sprache sehr schnell zu hassen. Sie akzeptierte fast alles, was ich eingab, deutete es dann aber völlig anders, als ich es gemeint hatte. Dies lag zu einem großen Teil natürlich auch daran, dass ich damals keine Ahnung hatte, was ich da eigentlich tat. Aber es ist auch das Symptom eines echten Problems: JavaScript ist absurd freizügig. Dahinter stand die Idee, dass es das Programmieren in JavaScript für Anfänger erleichtern würde. In Wirklichkeit aber erschwert dies, Probleme in Programmen zu finden, da das System Sie nicht darauf hinweist.

Diese Flexibilität hat jedoch auch ihre Vorteile. Sie erlaubt viele Techniken, die in strengeren Sprachen unmöglich sind. Wie Sie noch sehen werden (etwa in Kapitel 10), kann sie auch dazu genutzt werden, einige der Mängel von JavaScript auszubügeln. Nachdem ich die Sprache gründlich gelernt und eine Weile damit gearbeitet hatte, begann ich JavaScript sogar zu *mögen*.

Es gibt verschiedene Versionen von JavaScript. ECMAScript Version 3 war während des Aufstiegs von JavaScript etwa zwischen 2000 und 2010 sehr weit verbreitet. In dieser Zeit wurde an der anspruchsvolleren Version 4 gearbeitet, bei der es einige radikale Verbesserungen und Erweiterungen geben sollte. Es erwies sich jedoch als politisch schwierig, eine lebendige, weiträumig genutzte Sprache so einschneidend zu verändern, weshalb die Arbeiten an Version 4 im Jahre 2008 eingestellt wurden. Stattdessen wurde 2009 die weniger ambitionierte Version 5 veröffentlicht, die nur einige nichtkontroverse Verbesserungen bot. 2015 kam die stark überarbeitete Version 6 heraus, die einige der für Version 4 geplanten Aspekte enthielt. Seitdem gab es jedes Jahr kleinere Aktualisierungen.

Da sich die Sprache weiterentwickelt, müssen die Browser mithalten. Wenn Sie einen älteren Browser verwenden, kann es sein, dass er nicht alle Funktionen unterstützt. Die Entwickler der Sprache achten sorgfältig darauf, keine Änderungen einzuführen, die bestehende Programme funktionsunfähig machen würden. Daher können neue Browser immer noch ältere Programme ausführen. In diesem Buch verwendete ich die JavaScript-Version von 2017.

Browser sind jedoch nicht die einzigen Plattformen, auf denen JavaScript eingesetzt wird. Auch einige Datenbanken wie MongoDB oder CouchDB nutzen JavaScript als Skripterstellungs- und Abfragesprache. Verschiedene Plattformen für die Desktop- und Serverprogrammierung, insbesondere das Projekt Node.js (um das es in Kapitel 20 geht), bieten eine Umgebung für die JavaScript-Programmierung außerhalb des Browsers.

Die Codebeispiele

Code ist der Text eines Programms. Die meisten Kapitel dieses Buches enthalten eine Menge Code. Ich bin der Meinung, dass das Lesen und Schreiben von Code unverzichtbar ist, um Programmieren zu lernen. Überfliegen Sie die Beispiele nicht einfach, sondern lesen Sie sie aufmerksam, um sie auch wirklich zu verstehen. Das kann zu Anfang langsam und verwirrend sein. Aber ich verspreche Ihnen, dass Sie den Bogen schon bald heraushaben werden. Das Gleiche gilt auch für die Übungen. Sie haben diese erst dann richtig verstanden, wenn Sie eine Lösung dafür geschrieben haben.

Ich empfehle Ihnen, Ihre Lösungen zu den Übungsaufgaben tatsächlich in einem JavaScript-Interpreter auszuprobieren. Dadurch erhalten Sie unmittelbar eine Rückmeldung, ob sie funktionieren, und werden hoffentlich auch dazu animiert, über die Aufgabenstellung hinaus noch weiter zu experimentieren.

Die einfachste Möglichkeit, den Beispielcode in diesem Buch auszuführen und damit zu experimentieren, besteht darin, ihn in der Online-Version des Buchs auf *https://eloquentjavascript.net* anzusehen. Dort können Sie auf die Codebeispiele klicken, um sie zu bearbeiten, auszuführen und sich die Ausgaben anzusehen. Um die Übungen zu bearbeiten, besuchen Sie *https://eloquentjavascript.net/code*. Dort erhalten Sie den Ausgangscode für alle Programmierübungen und können auch einen Blick auf die Lösungen werfen.

Wollen Sie die Programme aus diesem Buch außerhalb der Begleitwebseite ausführen, ist etwas Vorsicht geboten. Viele der Beispiele sind eigenständig und sollten in jeder JavaScript-Umgebung funktionieren. Code in den hinteren Kapiteln ist jedoch oft für eine bestimmte Umgebung geschrieben (den Browser oder Node.js) und kann nur dort ausgeführt werden. Außerdem werden in einigen Kapiteln umfangreichere Programme erstellt, deren einzelne Teile voneinander und manchmal auch von externen Dateien abhängen. Die Sandbox auf der Webseite stellt Links zu Zip-Dateien mit allen Skript- und Datendateien bereit, die erforderlich sind, um den Code zu einem Kapitel auszuführen.

Übersicht über dieses Buch

Dieses Buch besteht aus drei Teilen. Die ersten zwölf Kapitel beschreiben die Sprache JavaScript, in den nächsten sieben geht es um Webbrowser und ihre Programmierung mit JavaScript, und die beiden letzten sind Node.js gewidmet, einer weiteren Programmierumgebung für JavaScript.

Über das ganze Buch verstreut sind fünf *Projektkapitel*, in denen umfangreiche Beispielprogramme beschrieben werden, um Ihnen einen Vorgeschmack auf die tatsächliche Programmierarbeit zu geben. In diesen Kapiteln programmieren wir einen Zustellroboter, eine Programmiersprache, ein Jump'n'Run-Spiel, ein Malprogramm und eine dynamische Webseite.

Der erste Teil des Buches beginnt mit vier Kapiteln, die die Grundstruktur von JavaScript vorstellen. Sie geben eine Einführung in Steuerstrukturen (z. B. das `while`-Konstrukt, das Sie bereits in dieser Einleitung kennengelernt haben), Funktionen (von Ihnen selbst geschriebene Bausteine) und Datenstrukturen. Damit sind Sie schon in der Lage, einfache Programme zu schreiben. Anschließend werden in den Kapiteln 5 und 6 Techniken eingeführt, um mithilfe von Funktionen und Objekten *abstrakteren* Code zu schreiben und die Komplexität zu reduzieren.

Nach einem ersten Projektkapitel geht es im ersten Teil des Buches mit Kapiteln über Fehlerbehandlung und -behebung, reguläre Ausdrücke (ein wichtiges Instrument für die Arbeit mit Text), Modularität (eine weitere Maßnahme gegen zu hohe Komplexität) und asynchrone Programmierung (Umgang mit Ereignissen, die einige Zeit dauern) weiter. Das zweite Projektkapitel schließt den ersten Teil ab.

Der zweite Teil – die Kapitel 13 bis 19 – beschreibt die Werkzeuge, auf die JavaScript im Browser Zugriff hat. Sie erfahren hier, wie Sie etwas auf dem Bildschirm darstellen (Kapitel 14 bis 17), wie Sie auf Benutzereingaben reagieren (Kapitel 15) und wie Sie über das Netzwerk kommunizieren (Kapitel 18). Auch in diesem Teil gibt es wieder zwei Projektkapitel.

Im Anschluss daran wird in Kapitel 20 Node.js beschrieben und in Kapitel 21 eine kleine Webseite damit erstellt.

In Kapitel 22 finden Sie schließlich einige Überlegungen dazu, wie Sie JavaScript-Programme optimieren können, um die Ausführung zu beschleunigen.

Schreibweisen

Text in einer `nichtproportionalen Schrift` kennzeichnet in diesem Buch Programmelemente – sowohl eigenständige Fragmente als auch »Zitate« aus einem in der Nähe abgedruckten Programm. Die Programme selbst (von denen Sie inzwischen ja schon einige gesehen haben) werden wie folgt dargestellt:

```
function factorial(n) {
  if (n == 0) {
    return 1;
  } else {
    return factorial(n - 1) * n;
 }
}
```

Um die erwartete Ausgabe eines Programms zu zeigen, wird sie manchmal hinter zwei Schrägstrichen und einem Pfeil angegeben:

```
console.log(factorial(8));
// → 40320
```

Viel Glück!