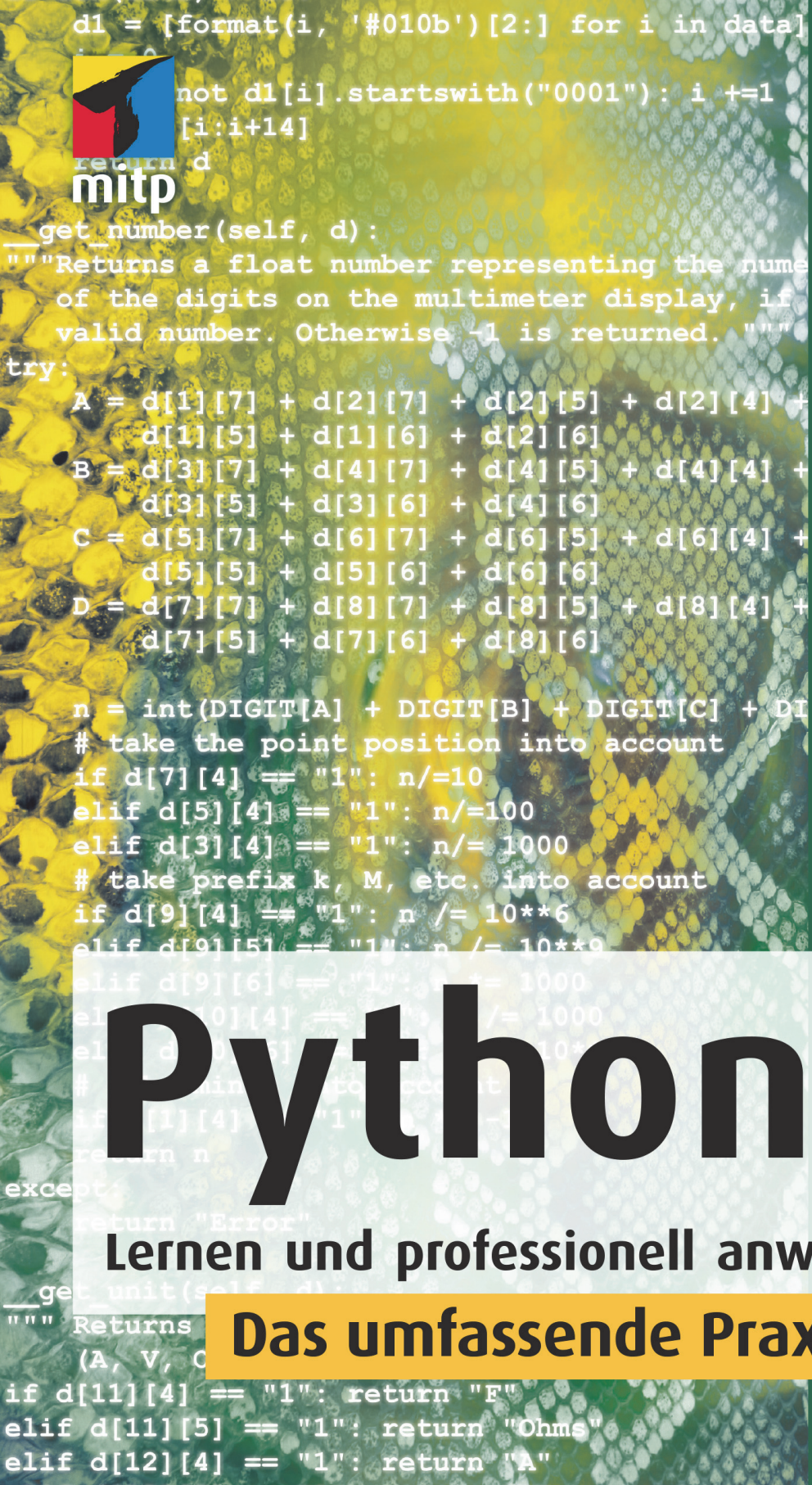

mitp

Michael Weigend

9., erweiterte Auflage

Python 3

Lernen und professionell anwenden

Das umfassende Praxisbuch

Inhaltsverzeichnis

Einleitung

Warum Python?

Es gibt triftige Argumente für die Verwendung der Programmiersprache Python.

- Python ist einfach. Man könnte auch sagen minimalistisch. Auf Sprachelemente, die nicht unbedingt notwendig sind, wurde verzichtet. Mit Python kann man kurze Programme schreiben, die viel leisten.
- Python besitzt einen interaktiven Modus. Sie können einzelne Befehle direkt eingeben und ihre Wirkung beobachten. Python unterstützt das Experimentieren und Ausprobieren. Das erleichtert das Erlernen neuer Programmierkonzepte und hilft vor allem Anfängern bei den ersten »Gehversuchen«.
- Dennoch ist Python ist kein Spielzeug. Zusammen mit vielen Zusatzkomponenten, sogenannten Modulen, ist es eine sehr mächtige Programmiersprache.
- Python ist nichtkommerziell. Alle Software, die Sie benötigen, ist kostenlos und für jede Plattform verfügbar.
- Hinter Python steht eine wachsende internationale Community aus Wissenschaftlern und Praktikern, die die Sprache pflegen und weiterentwickeln.

Python 3

Im Jahre 2008 fand in der Python-Welt eine kleine Revolution statt. Python 3 wurde veröffentlicht. Eine neue Version, die mit den Vorgängerversionen 2.X nicht mehr kompatibel ist. Ein Programm, das z.B. in Python 2.5 geschrieben worden ist, läuft (in der Regel) nicht mehr mit einem Python-3-Interpreter. Das ist natürlich schade, war aber notwendig, weil es einige sehr tief gehende Änderungen gab. Doch das neue Python 3 ist noch konsistenter und führt zu schönerem Programmtext als die früheren Versionen.

An wen wendet sich dieses Buch?

Dieses Buch ist für jeden, der die Programmierung mit Python lernen möchte. Besondere Vorkenntnisse werden nicht erwartet. Für die hinteren Kapitel ist es allerdings hilfreich, wenn man sich mit HTML auskennt. Das Buch wendet sich sowohl an Anfänger als auch an Leserinnen und Leser, die bereits mit einer höheren Programmiersprache vertraut sind, und ihr Wissen erweitern und vertiefen wollen. Für Neulinge gibt es zahlreiche Passagen, in denen grundlegende Konzepte anschaulich erklärt werden. Insbesondere das erste Kapitel ist zum überwiegenden Teil eine allgemeine Einführung für diejenigen, die sich bisher noch nie ausführlicher mit der Computertechnik beschäftigt haben. Wenn Sie sich eher zu

den Fortgeschrittenen zählen, dürfen Sie getrost diese Textabschnitte überspringen und sich dem zuwenden, das Sie interessiert.

Auf der anderen Seite enthält das Buch auch Stellen, die eine Herausforderung darstellen. Einige Abschnitte tragen Überschriften, die mit *Hintergrund:* oder *Vertiefung:* beginnen. Sie enthalten Ausblicke und Hintergrundinformationen oder gehen vertiefend auf speziellere Aspekte der jeweiligen Thematik ein, die nicht jeden interessieren.

Generell ist der Theorieanteil dieses Buches gering. Die praktische Arbeit steht im Vordergrund. In der Regel ist es möglich, theoretische Passagen (wie die über formale Grammatiken) zu überspringen, wenn man nun gar nicht damit zurechtkommt. Alle wichtigen Dinge werden zusätzlich auch auf anschauliche Weise erklärt. Und Sie werden erleben, dass beim Nachvollziehen und praktischen Ausprobieren der Programmbeispiele auch zunächst schwierig erscheinende Konzepte verständlich werden. Lassen Sie sich also nicht abschrecken.

Inhalt und Aufbau

Im Zentrum steht die Kunst der Programmentwicklung nach dem objektorientierten Paradigma. Dabei machen wir einen Rundgang durch verschiedene Gebiete der Informatik. Wir werfen einen Blick hinter die Kulissen von Software-Systemen, die Sie als Anwender aus dem Alltag kennen. Wie gestaltet man eine grafische Benutzungsoberfläche? Wie funktioniert E-Mail? Wie programmiert man einen Chatroom? Darüber hinaus werden eine Reihe fundamentaler Ideen der Informatik angesprochen. Das Buch orientiert sich an den üblichen Curricula von Universitätskursen zur Einführung in die Programmierung. In vielen Fällen dürfte es deshalb eine sinnvolle Ergänzung zu einem Vorlesungsskript sein.

Dieses Buch ist so angelegt, dass man es von vorne nach hinten lesen kann. Wir fangen mit einfachen Dingen an und nachfolgende Kapitel knüpfen an den vorhergehenden Inhalt an. Idealerweise sollte jeder Begriff bei seiner ersten Verwendung erklärt werden. Doch lässt sich dieses Prinzip nur schwer in Perfektion umsetzen. Manchmal gehen wir von einem intuitiven Vorverständnis aus und erläutern die Begrifflichkeit erst kurz darauf ausführlich.

Im vorderen Teil des Buches finden Sie an verschiedenen Stellen Hinweise zum Programmierstil und zu typischen Fehlern. Am Ende jedes Kapitels gibt es Übungsaufgaben, die in der Regel nach Schwierigkeitsgrad sortiert sind. Einige Programmieraufgaben sind so komplex, dass man sie (insbesondere als Anfänger) eigentlich gar nicht eigenständig lösen kann. Sie sind dann eher als Erweiterung gedacht und es wurde ins Kalkül gezogen, dass Sie »mogeln« und während der Bearbeitung in die Lösung gucken.

Unterkapitel, deren Überschriften mit dem Wort »Vertiefung« beginnen, wenden sich an besonders interessierte Leser und können in der Regel übersprungen werden.

Der vordere Teil des Buches befasst sich mit den grundlegenden Konzepten der Programmierung mit Python. Herausgestellt werden die syntaktischen Besonderheiten gegenüber anderen Programmiersprachen. Sie finden an verschiedenen Stellen Hinweise zum Programmierstil und zu typischen Fehlern. Angesprochen werden unter anderem folgende Punkte:

- Aufbau von Anweisungen in einem Python Programm
- Umgang mit der Standard-Entwicklungsumgebung IDLE

- Standard-Datentypen
- Modellieren mit Datenstrukturen: Tupel, Listen, Dictionaries, Mengen
- Kontrollstrukturen: Wiederholungen, Verzweigungen, Abfangen von Ausnahmen (try ... except)
- Funktionen: Arten von Parametern, Voreinstellungen, Lambda-Ausdrücke, Rekursion, Docstrings
- Ein- und Ausgabe: Dateien, pickle
- Konzepte der Objektorientierung: Klassen, Objekte, Vererbung, statische Methoden, Polymorphie, Properties
- Techniken der objektorientierten Modellierung: Analyse (OOA) und Design (OOD), UML, Objekt- und Klassendiagramme, Assoziationen
- Modularisieren
- Verarbeitung von Zeichenketten: String-Methoden, Codierung und Decodierung, Formatierung, reguläre Ausdrücke, Sprachsynthese, Chat-Bots
- Systemfunktionen: Schnittstelle zum Betriebssystem, Datum und Zeit
- Grundprinzipien der Gestaltung von grafischen Benutzungsoberflächen mit tkinter: Widgets, Event-Verarbeitung, Layout, Threads
- Debugging-Techniken

Im hinteren Teil des Buches werden die Kapitel immer spezieller. Hier kommen dann gelegentlich auch Module von Drittanbietern ins Spiel, die nicht zur Standardinstallation von Python gehören (z.B. PIL, PyQt, NumPy). Sie müssen erst heruntergeladen und installiert werden. Zu diesen spezielleren Themen gehören:

- Internet-Programmierung: CGI-Skripte, WSGI, Webserver, E-Mail-Clients
- Datenbanken und XML
- Testen und Performance-Analyse: doctest, unittest
- Benutzungsoberflächen für Multimedia-Anwendungen mit PyQt: Video-Player, Webbrowser, Kalender
- Wissenschaftliches Rechnen mit NumPy und SciPy: Arrays, Vektoren und Matrizen, digitale Bildbearbeitung, Datenvisualisierung, lineare Gleichungssysteme, Integralrechnung
- Parallele Datenverarbeitung: Prozesse und Synchronisation, Queues, Pipes, Pools
- Messdaten eines externen digitalen Multimeters erfassen und verarbeiten
- Webentwicklung mit Django.

Hinweise zur Typographie

Achten Sie beim Lesen auf den Schrifttyp. Formale Texte, wie Python-Programmtext, Funktions- und Variablennamen, Operatoren, Grammatik. Regeln, Zahlen und mathematische Ausdrücke, werden in einem Zeichenformat mit fester Breite gesetzt. Beispiele:

```
x = y + 1
print()
```

In solchen formalen Texten tauchen gelegentlich Wörter auf, die kursiv gesetzt sind. Hierbei handelt es sich um Platzhalter, die man nicht Buchstabe für Buchstabe aufschreibt, sondern z.B. durch Zahlen oder andere Zeichenfolgen ersetzt. Beispiel:

```
range(zahl)
```

Hier bezeichnet *zahl* eine (ganze) Zahl. Ein korrekter Aufruf der Funktion `range()` lautet z.B. `range(10)`, während `range(zahl)` zu Problemen führen kann.

In Programmtexten sind wichtige Passagen fett gedruckt, damit man sie schneller finden kann.

Programmbeispiele

Das Buch enthält zahlreiche Programmbeispiele, die zum Ausprobieren, Nachmachen und Weiterentwickeln ermuntern sollen. Sie können alle Skripte und einige zusätzliche Dateien als ZIP-Archiv von der Website des mitp-Verlages herunterladen. Der URL ist:

`http://www.mitp.de/0544`

Klicken Sie im Kasten DOWNLOADS auf den Link PROGRAMMBEISPIELE.

Außerdem sind die Programmbeispiele in einem GitHub-Repository veröffentlicht. URL:

`https://github.com/mweigend/python3/`

Weitere Hinweise zum Download finden Sie im Anhang C.

Beim Design der Beispiele wurde darauf geachtet, dass sie möglichst kurz und übersichtlich sind. Häufig sind die Skripte Spielzeugversionen richtiger Software, die man im Alltag zu sinnvollen Dingen nutzen kann. Sie sind Modelle – etwa so wie Häuser aus Legosteinen Modelle richtiger Häuser sind. Sie sind auf das Wesentliche reduziert und sollen nur bestimmte Aspekte verdeutlichen. Sie genügen deshalb nicht den Qualitätsanforderungen, die man üblicherweise an professionelle Software stellt, aber sie dienen vielleicht als Anregung und Inspiration für eigene Projekte.

Kapitel 1

Grundlagen

Bitte noch etwas Geduld! Im ersten Kapitel bleibt der Computer noch ausgeschaltet. Hier wird zunächst eine anschauliche Vorstellung von einigen Grundideen der Programmierung vermittelt. Sie helfen, den Rest des Buches besser zu verstehen. Im Mittelpunkt stehen folgende Fragen:

- Was sind Programme und Algorithmen?
- Worin unterscheiden sich Programmierparadigmen?
- Was ist die Philosophie der objektorientierten Programmierung?

1.1 Was ist Programmieren?

Es ist eigentlich ganz einfach: Programmieren ist das Schreiben eines Programms. Nun gibt es den Begriff »Programm« auch in unserer Alltagssprache – fernab von jeder Computertechnik. Sie kennen Fernseh- und Kinoprogramme, planen ein Programm für Ihre Geburtstagsparty, genießen im Urlaub vielleicht Animationsprogramme (sofern Sie nichts Besseres zu tun haben) und lesen als gewissenhafter Staatsbürger vor den Bundestagswahlen Parteiprogramme. In diesen Zusammenhängen versteht man unter einem Programm eigentlich recht unterschiedliche Dinge: Ein Parteiprogramm ist so etwas wie ein strukturiertes Konzept politischer Ziele, ein Kinoprogramm ein Zeitplan für Filmvorstellungen und ein Animationsprogramm ein Ablauf von Unterhaltungsveranstaltungen.

In der Informatik – der Wissenschaft, die hinter der Programmiertechnik steht – ist der Begriff Programm natürlich enger und präziser gefasst. Allerdings gibt es auch hier unterschiedliche Sichtweisen.

Die älteste und bekannteste Definition basiert auf dem Begriff *Algorithmus*. Grob gesprochen ist ein Algorithmus eine Folge von Anweisungen (oder militärisch formuliert: Befehlen), die man ausführen muss, um ein Problem zu lösen. Unter einem Programm versteht man in dieser Sichtweise einen Algorithmus,

- der in einer Sprache geschrieben ist, die auch Maschinen verstehen können (Programmiersprache), und
- der das Verhalten von Maschinen steuert.

Daraus folgt: Wer ein Computerprogramm schreibt, muss zumindest zwei Dinge tun:

- Er oder sie muss einen Algorithmus erfinden, der in irgendeiner Weise nützlich ist und zum Beispiel bei der Lösung eines Problems helfen kann.
- Der Algorithmus muss fehlerfrei in einer Programmiersprache formuliert werden. Man spricht dann von einem Programmtext.

Ziel einer Programmentwicklung ist korrekter Programmtext.

1.2 Hardware und Software

Ein Computer ist eine universelle Maschine, deren Verhalten durch ein Programm bestimmt wird. Ein Computersystem besteht aus Hardware und Software. Ersteres ist das englische Wort für »Eisenwaren« und meint alle Komponenten des Computers, die man anfassen kann – Arbeitsspeicherbausteine, Prozessor, Peripheriespeicher (Festplatte, Diskette, CD), Monitor, Tastatur usw. Software dagegen ist ein Kunstwort, das als Pendant zu Hardware gebildet wurde. Mit Software bezeichnet man die Summe aller Programme, die die Hardware steuern.

Man kann die gesamte Software eines Computers grob in zwei Gruppen aufteilen:

Das *Betriebssystem* regelt den Zugriff auf die Hardware des Computers und verwaltet Daten, die im Rechner gespeichert sind. Es stellt eine Umgebung bereit, in der Benutzer Programme ausführen können. Bekannte Betriebssysteme sind Unix, MS Windows oder macOS. Python-Programme laufen unter allen drei genannten Betriebssystemen. Man nennt sie deshalb portabel.

Anwendungs- und Systemsoftware dient dazu, spezifische Probleme zu lösen. Ein Textverarbeitungsprogramm z.B. unterstützt das Erstellen, Verändern und Speichern von Textdokumenten. Anwendungssoftware ist also auf Bedürfnisse des Benutzers ausgerichtet, während das Betriebssystem nur für ein möglichst störungsfreies und effizientes Zusammenspiel der verschiedenen Komponenten des Computersystems sorgt.

Ein Computersystem wird häufig durch ein Schichtenmodell wie in Abbildung 1.1 beschrieben. Die unterste Schicht ist die Computer-Hardware, darüber liegt das Betriebssystem und zuoberst befinden sich schließlich die Anwendungs- und Systemprogramme, die eine Benutzungsschnittstelle enthalten. Nur über diese oberste Software-Schicht kommunizieren Menschen mit einem Computersystem.

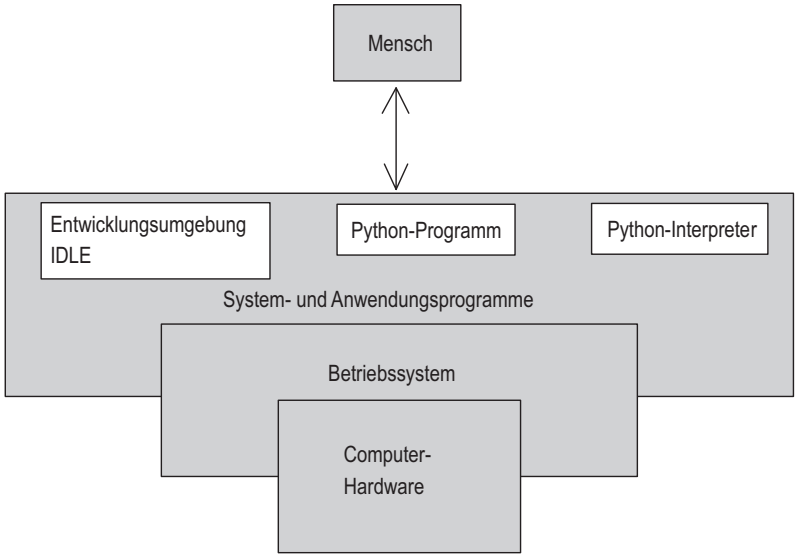

Abb. 1.1: Komponenten eines Computer-Systems

Wenn Sie ein Python-Programm schreiben, entwickeln Sie vor allem Anwendungssoftware. Dabei verwenden Sie eine Systemsoftware, zum Beispiel die integrierte Entwicklungsumgebung IDLE. Ausgeführt wird das Programm mithilfe einer weiteren Systemsoftware, nämlich dem Python-Interpreter. Dieser »liest« den Python-Programmtext Zeile für Zeile und beauftragt das Betriebssystem (eine Schicht tiefer), bestimmte Dinge zu tun – etwa eine Zahl auf den Bildschirm zu schreiben.

1.3 Programm als Algorithmus

Ein Algorithmus ist eine Anleitung zur Lösung einer Aufgabe. Es besteht aus einer Folge von Anweisungen, die so präzise formuliert sind, dass sie auch von einem völlig Unkundigen rein mechanisch ausgeführt werden können. Sie kennen Algorithmen aus dem Alltag:

- Kochrezept
- Anleitung zur Mund-zu-Mund-Beatmung in einer Erste-Hilfe-Fibel
- Gebrauchsanweisung für die Benutzung einer Bohrmaschine

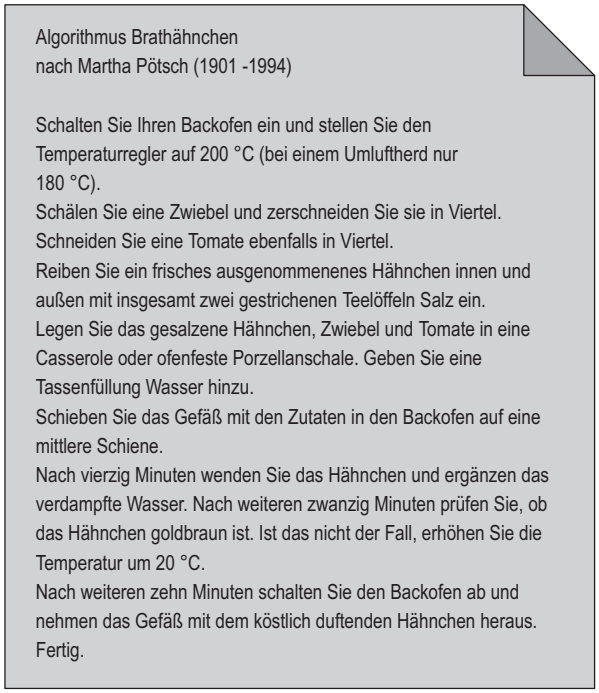

Abb. 1.2: Natürlichsprachlich formulierter Algorithmus zur Zubereitung eines Brathähnchens, entwickelt von Martha Pötsch aus Essen

Abbildung 1.2 zeigt einen äußerst effizienten Algorithmus zur Zubereitung eines Brathähnchens (Vorbereitungszeit: eine Minute). Wenn auch das Rezept wirklich sehr gut ist (es stammt von meiner Großmutter), so erkennt man dennoch an diesem Beispiel zwei Schwächen umgangssprachlich formulierter Alltags-Algorithmen:

- Sie beschreiben die Problemlösung meist nicht wirklich vollständig, sondern setzen voraus, dass der Leser, d.h. die den Algorithmus ausführende Instanz, über ein gewisses Allgemeinwissen verfügt und in der Lage ist, »Beschreibungslücken« selbstständig zu füllen. So steht in dem Kochrezept nichts davon, dass man die Backofentür öffnen und schließen muss. Das versteht sich von selbst und wird deshalb weggelassen.
- Sie enthalten ungenaue Formulierungen, die man unterschiedlich interpretieren kann. Was heißt z.B. »goldbraun«?

Auch ein Computerprogramm kann man als Algorithmus auffassen. Denn es »sagt« dem Computer, was er zu tun hat. Damit ein Algorithmus von einem Computer ausgeführt werden kann, muss er in einer Sprache formuliert sein, die der Computer »versteht« – einer Programmiersprache. Im Unterschied zu »natürlichen« Sprachen, wie Deutsch oder Englisch, die sich in einer Art evolutionärem Prozess im Laufe von Jahrhunderten entwickelt haben, sind Programmiersprachen »künstliche« Sprachen. Sie wurden von Fachleuten entwickelt und sind speziell auf die Formulierung von Algorithmen zugeschnitten.

1.4 Syntax und Semantik

Eine Programmiersprache ist – wie jede Sprache – durch Syntax und Semantik definiert. Die *Syntax* legt fest, welche Folgen von Zeichen ein Programmtext in der jeweiligen Sprache ist. Zum Beispiel ist

```
a = 1 ! 2
```

kein gültiger Python-Programmtext, weil die Python-Syntax vorschreibt, dass in einem arithmetischen Ausdruck zwischen zwei Zahlen ein Operator (z.B. `+`, `-`, `*`, `/`) stehen muss. Das Ausrufungszeichen `!` ist aber nach der Python-Syntax kein Operator.

Dagegen ist die Zeichenfolge

```
print("Schweinebraten mit Klößen")
```

ein syntaktisch korrektes Python-Programm. Die Syntax sagt aber nichts darüber aus, welche Wirkung dieses Mini-Programm hat. Die Bedeutung eines Python-Programmtextes wird in der *Semantik* definiert. Bei diesem Beispiel besagt die Semantik, dass auf dem Bildschirm die Zeichenkette `Schweinebraten mit Klößen` ausgegeben wird.

1.5 Interpreter und Compiler

Python ist eine höhere Programmiersprache. Es ist eine künstliche Sprache für Menschen, die Algorithmen formulieren wollen. Mit einer höheren Programmiersprache lässt sich auf bequeme Weise Programmtext notieren, der leicht durchschaubar und gut verständlich ist. Syntax und Semantik einer höheren Programmiersprache sind auf die Bedürfnisse von Menschen zugeschnitten und nicht auf die technischen Spezifika der Maschine, die das Programm ausführen soll.

Damit ein Programmtext – man spricht auch von Quelltext (*source code*) – vom Computer »verstanden« wird und abgearbeitet werden kann, muss er in ein ausführbares Programm übersetzt werden.

Dazu gibt es zwei unterschiedliche Methoden.

Ein *Compiler* übersetzt einen kompletten Programmtext und erzeugt ein direkt ausführbares (*executable*) Programm, das vom Betriebssystem geladen und gestartet werden kann. Bei der Übersetzung müssen natürlich die Besonderheiten des Rechners, auf dem das Programm laufen soll, berücksichtigt werden. Es gibt dann z.B. unterschiedliche Fassungen für MS-Windows- und Unix-Systeme. Programmiersprachen, bei denen kompiliert wird, sind z.B. Pascal, C, C++.

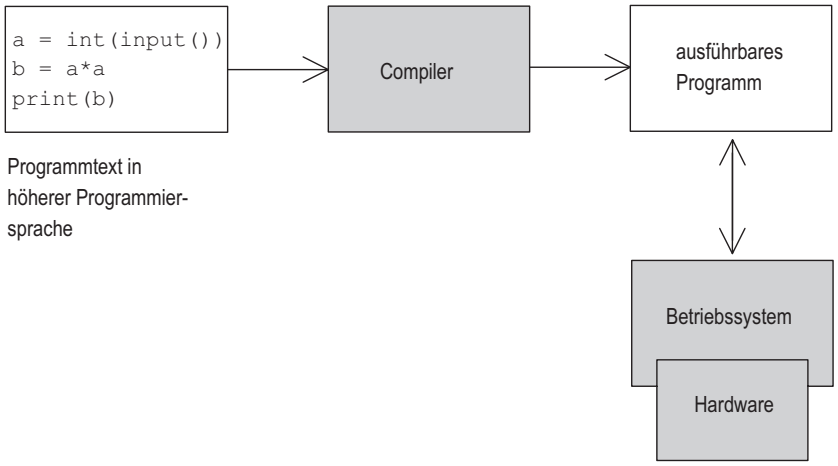

Abb. 1.3: Arbeitsweise eines Compilers

Ein *Interpreter* liest einen Programmtext Zeile für Zeile und führt (über das Betriebssystem) jede Anweisung direkt aus. Wenn ein Programm gestartet werden soll, muss zuerst der Interpreter aufgerufen werden. Für jedes Betriebssystem gibt es zu der Programmiersprache einen eigenen Interpreter. Wer ein Programm in einer interpretativen Sprache verwenden möchte, benötigt also zusätzlich zu dem Anwendungsprogramm noch einen Interpreter.

Python ist eine interpretative Programmiersprache. Dies hat den Vorteil, dass ein und dasselbe Programm auf allen Rechnerplattformen läuft. Als nachteilig könnte man aus Entwicklersicht empfinden, dass der Quelltext einer Software, die man verkaufen möchte, immer offen gelegt ist (*open source*). Damit besteht das Risiko, dass jemand illegalerweise den Programmtext leicht verändert und ihn unter seinem Namen weiterverkauft. Das geistige Eigentum des Programmentwicklers ist also schlecht geschützt. Auf der anderen Seite gibt es einen gewissen Trend, nur solche Software einzusetzen, deren Quelltext bekannt ist. Denn nur dann ist es möglich, etwaige Fehler, die erst im Lauf des Betriebes sichtbar werden, zu finden und zu beseitigen. Wer Software verwendet, deren Quelltext geheim gehalten ist, macht sich vom Software-Hersteller abhängig, und ist im Störungsfall »auf Gedeih und Verderb« auf ihn angewiesen.

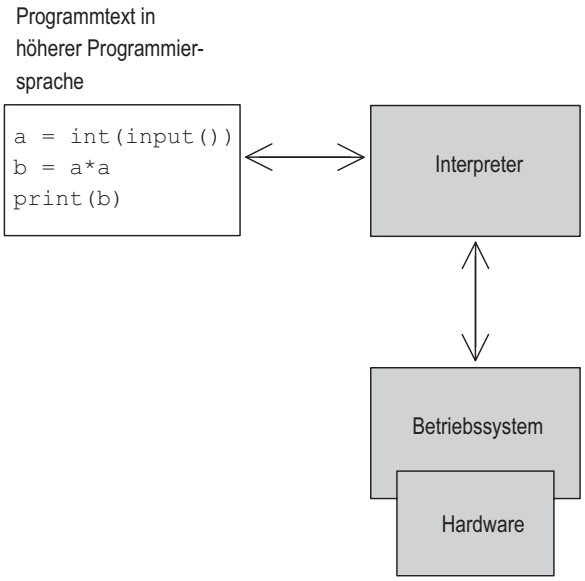

Abb. 1.4: Arbeitsweise eines Interpreters

1.6 Programmierparadigmen

Ein Paradigma ist allgemein ein Denk- oder Handlungsmuster, an dem man sich z.B. bei der Formulierung einer Problemlösung orientiert. Wenn man ein Programm als Algorithmus betrachtet, also als System von Befehlen, folgt man dem *imperativen* Programmierparadigma (*imperare*: lat. befehlen).

Zur Abgrenzung sei kurz darauf hingewiesen, dass es auch andere Programmierparadigmen gibt. *Prolog* z.B. ist eine *deklarative* Programmiersprache. Ein deklaratives Programm beschreibt Eigenschaften der Lösung des Problems. Der Programmierer legt sein Augenmerk auf die Frage, *was* berechnet werden soll, und nicht, *wie* man es berechnet. Dagegen stellt ein imperatives Programm eine Anleitung dar. Sie beschreibt, *wie* – Schritt für Schritt – die Aufgabe gelöst werden soll.

Das folgende kleine Experiment veranschaulicht den Unterschied. Wenn Sie es selbst durchspielen wollen, benötigen Sie sieben Streichhölzer. Die beiden folgenden Texte beschreiben auf deklarative und auf imperative Weise, wie die Streichhölzer angeordnet werden sollen. Probieren Sie aus, mit welchem Paradigma Sie besser zurechtkommen.

Deklaratives Paradigma:

- Insgesamt gibt es sieben Streichhölzer.
- Genau ein Streichholz berührt an beiden Enden jeweils zwei weitere Streichhölzer.
- Wenigstens ein Streichholz bildet mit zwei benachbarten Streichhölzern jeweils einen rechten Winkel.

- Drei Streichhölzer liegen zueinander parallel, berühren sich aber nicht.
- Es gibt kein Streichholz, das nicht an jedem Ende wenigstens ein anderes Streichholz berührt.

Imperatives Paradigma:

- Legen Sie zwei Streichhölzer (A und B) in einer geraden Linie nebeneinander auf den Tisch, so dass sie sich an einer Stelle berühren.
- Legen Sie ein Streichholz C mit einem Ende an der Stelle an, wo sich A und B berühren. Das Streichholz soll einen rechten Winkel zu A und B bilden.
- Legen Sie an die äußeren Enden von A und B jeweils ein weiteres Streichholz mit einem Ende an (D und E), so dass diese neuen Streichhölzer jeweils einen rechten Winkel zu A und B bilden und in die gleiche Richtung gehen wie das mittlere Streichholz.
- Verbinden Sie die noch freien Enden von C, D und E mit den verbleibenden zwei Streichhölzern.

Eine Abbildung der korrekten Anordnung finden Sie am Ende des Kapitels. Vermutlich haben Sie die zweite Aufgabe schneller lösen können. Tatsächlich benötigen auch in der Computertechnik imperative Programme weniger Rechenzeit als deklarative.

Zum Schluss sei noch das Paradigma der *funktionalen* Programmierung erwähnt. Mit funktionalen Programmiersprachen wie z.B. *Haskell* oder *Scheme* kann man ein Programm als (mathematische) Funktion definieren. Einfache vorgegebene Funktionen werden zu einer komplexen Funktion verknüpft, die das Gewünschte leistet. Mathematisch geschulten Menschen fällt diese Art der Programmentwicklung bei bestimmten Problemen leichter.

Man kann mit Fug und Recht sagen, dass unter diesen drei Paradigmen der imperative Ansatz am verbreitetesten ist. Funktionale und deklarative Sprachen spielen heute in der Praxis der Software-Entwicklung eher eine untergeordnete Rolle. Auch die *objektorientierte Programmierung* (OOP) wird als eigenes Programmierparadigma beschrieben. Das hört sich so an, als wäre die objektorientierte Programmierung etwas ganz anderes als das imperative oder funktionale Paradigma. Aber ganz so ist es eigentlich nicht. Vielmehr betrifft das Paradigma der Objektorientierung einen Aspekt der Programmentwicklung, den ich bisher noch nicht erwähnt habe. Es geht um die Beherrschung von Komplexität.

1.7 Objektorientierte Programmierung

1.7.1 Strukturelle Zerlegung

Die ersten Computerprogramme waren einfach und dienten der Lösung eines relativ kleinen, gut umgrenzten Problems. Die Situation wird ganz anders, wenn man umfangreiche Software erstellen möchte, etwa ein Textverarbeitungsprogramm oder ein Verwaltungsprogramm für eine Bibliothek. Solche großen Systeme lassen sich nur beherrschen, wenn man sie zunächst in kleinere überschaubare Teile aufbricht. Abbildung 1.5 soll diesen Gedanken veranschaulichen.

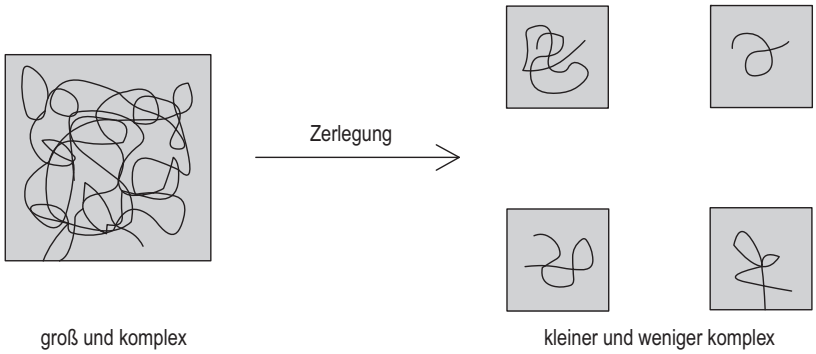

Abb. 1.5: Zerlegung eines komplexen Systems

Die Vorteile liegen auf der Hand:

Die kleineren Teile des Ganzen lassen sich einfacher programmieren. Die Wahrscheinlichkeit, dass sie Fehler enthalten, ist geringer. Mehrere Personen können zeitgleich und unabhängig voneinander die Einzelteile erstellen. Das spart Zeit. Und es kann sein, dass man später einen Baustein, den man früher einmal programmiert hat, wieder verwenden kann. Das spart Kosten.

Das objektorientierte Paradigma bietet ein Verfahren, nach dem große Systeme in kleinere Teile zerlegt werden können.

1.7.2 Die Welt als System von Objekten

In der objektorientierten Sichtweise stellt man sich die Welt als System von Objekten vor, die untereinander Botschaften austauschen. Zur Veranschaulichung betrachten wir ein Beispiel aus dem Alltag, das in Abbildung 1.6 illustriert wird.

Leonie in Bonn möchte ihrer Freundin Elena in Berlin einen Blumenstrauß schicken. Sie geht deshalb zu Mark, einem Blumenhändler, und erteilt ihm einen entsprechenden Auftrag. Betrachten wir Mark als Objekt. In der Sprache der objektorientierten Programmierung sagt man: Leonie sendet an das Objekt Mark eine Botschaft, nämlich: »Sende sieben gelbe Rosen an Elena, Markgrafenstr. 10 in Berlin.«. Damit hat sie getan, was sie tun konnte. Es liegt nun in Marks Verantwortung, den Auftrag zu bearbeiten. Mark versteht die Botschaft und weiß, was zu tun ist. Das heißt, er kennt einen Algorithmus für das Verschicken von Blumen. Der erste Schritt ist, einen Blumenhändler in Berlin zu finden, der die Rosen an Elena liefern kann. In seinem Adressverzeichnis findet er den Floristen Sascha. Ihm sendet er eine leicht veränderte Botschaft, die nun zusätzlich noch den Absender enthält. Damit ist Mark fertig und hat die Verantwortung für den Prozess weitergegeben. Auch Sascha hat einen zur Botschaft passenden Algorithmus parat. Er stellt den gewünschten Blumenstrauß zusammen und beauftragt seinen Boten Daniel, die Rosen auszuliefern. Daniel muss nun den Weg zur Zieladresse finden und befragt seine Straßenkarte. Sie antwortet ihm mit einer Wegbeschreibung. Nachdem Daniel den Weg zu Elenas Wohnung gefunden hat, überreicht er die Blumen und teilt ihr in einer Botschaft mit, von wem sie stammen. Damit ist der gesamte Vorgang, den Leonie angestoßen hat und an dem mehrere Objekte beteiligt waren, beendet.

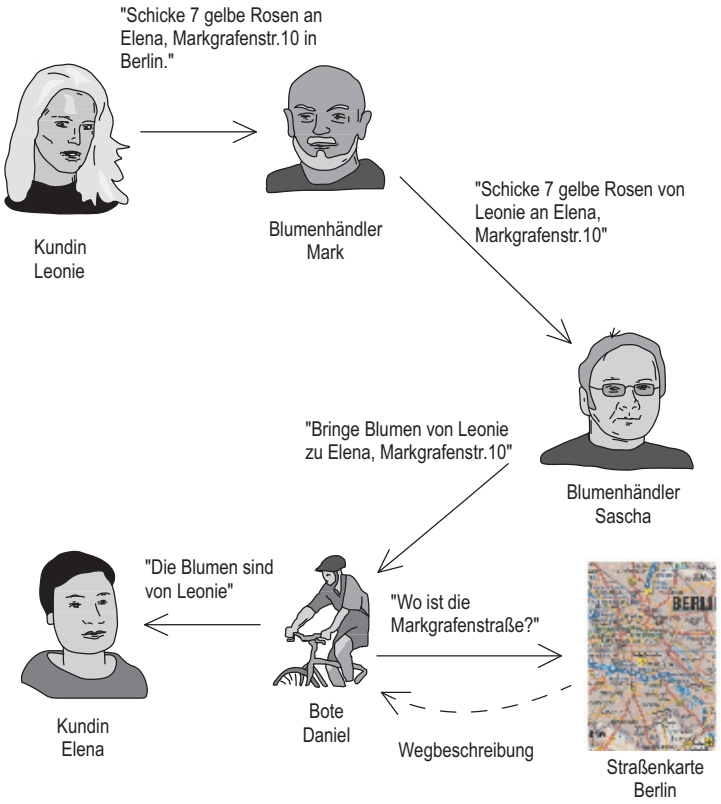

Abb. 1.6: Objektorientiertes Modell eines Blumenversandsystems

1.7.3 Objekte besitzen Attribute und beherrschen Methoden

Jedes Objekt besitzt Eigenschaften oder *Attribute*. Ein Attribut eines Blumenhändlers ist z.B. die Stadt, in der er sein Geschäft hat. Dieses Attribut ist auch für die Umwelt wichtig. So musste Mark einen Blumenhändler mit dem Attribut »wohnhaft in Berlin« suchen. Weitere typische Attribute von Blumenhändlern sind Name, Telefonnummer, Warenbestand oder Öffnungszeiten.

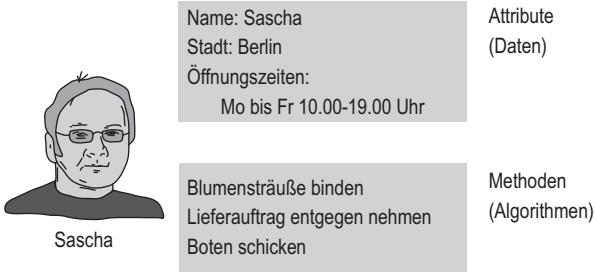

Abb. 1.7: Objekte besitzen Attribute und beherrschen Methoden.

Objekte sind in der Lage, bestimmte Operationen auszuführen, die man *Methoden* nennt. Ein Blumenhändler z.B. kann einen Lieferauftrag für Blumen entgegennehmen, Sträuße binden, einen Boten schicken, beim Großhandel neue Blumen einkaufen usw. Wenn ein Objekt eine geeignete Botschaft empfängt, wird eine zur Botschaft passende Operation gestartet. Man sagt: Die Methode wird aufgerufen. Der Umwelt, das heißt den anderen Objekten, ist bekannt, welche Methoden ein Objekt beherrscht. Die Umwelt weiß von den Methoden nur,

- was sie bewirken
- welche Daten sie als Eingabe benötigen

Die Umwelt weiß aber nicht, *wie* das Objekt funktioniert, das heißt, nach welchen Algorithmen die Botschaften verarbeitet werden. Dieses bleibt ein privates Geheimnis des Objektes.

Leonie hat keine Ahnung, wie Mark den Blumentransport bewerkstelligt. Es interessiert sie auch gar nicht. Ihre Aufgabe bestand allein darin, für ihr Problem ein geeignetes Objekt zu finden und ihm eine geeignete Botschaft zu senden. Ein ungeeignetes Objekt wäre zum Beispiel Tom, der Zahnarzt, oder Katrin, die Leiterin des Wasserwerks gewesen. Diese Objekte hätten Leonis Nachricht gar nicht verstanden und zurückgewiesen. Außerdem ist für Leonie wichtig, wie sie die Botschaft an Mark formuliert. Sie muss ihm ihren Namen mitteilen (damit der Empfänger weiß, von wem die Blumen sind), die Adresse des Empfängers sowie Anzahl und Sorte der Blumen, die gesendet werden sollen.

Eine Methode ist die Implementierung (technische Realisierung) eines Algorithmus. Bei der Programmierung einer Methode mit Python (oder einer anderen objektorientierten Sprache) wird also wieder das imperative Paradigma wichtig.

1.7.4 Objekte sind Instanzen von Klassen

Die Objekte des Beispiels kann man in Gruppen einteilen. Sascha und Mark sind beide Blumenhändler. Sie beherrschen beide dieselben Methoden und besitzen dieselben Attribute (z.B. die Stadt), allerdings mit unterschiedlichen Werten. Man sagt: Sascha und Mark sind *Instanzen* der Klasse »Blumenhändler«. In der objektorientierten Programmierung ist eine Klasse die Definition eines bestimmten Typs von Objekten. Sie ist so etwas wie ein Bauplan, in dem die Methoden und Attribute beschrieben werden. Nach diesem Schema können Objekte (Instanzen) einer Klasse erzeugt werden. Ein Objekt ist eine Konkretisierung, eine Inkarnation einer Klasse. Alle Instanzen einer Klasse sind von der Struktur her gleich. Sie unterscheiden sich allein in der Belegung ihrer Attribute mit Werten. Die Objekte Sascha und Mark besitzen dasselbe Attribut »Stadt«, aber bei Sascha trägt es den Wert »Berlin« und bei Mark »Bonn«.

1.8 Hintergrund: Geschichte der objektorientierten Programmierung

Die Grundideen der Objektorientierung (wie z.B. die Begriffe Klasse und Objekt) tauchen zum ersten Mal in der Simulationssprache SIMULA auf. Sie wurde von Ole-Johan Dahl and Kristen Nygaard am Norwegian Computing Centre (NCC) in Oslo zwischen 1962 und 1967 entwickelt und diente zur Simulation komplexer Systeme der realen Welt. Die erste universell verwendbare objektorientierte Programmiersprache wurde in den Jahren 1970 bis 1980

am Palo Alto Research Center der Firma Xerox von Alan Key und seinem Team entwickelt und unter dem Namen SmallTalk-80 in die Öffentlichkeit gebracht. Wenig später entstand in den Bell Laboratories (AT&T, USA) unter der Leitung von Bjarne Stroustrup die Sprache C++ als objektorientierte Erweiterung von C. Sie wurde zu Beginn der Neunzigerjahre zur dominierenden objektorientierten Sprache. Mitte der Neunzigerjahre etablierte sich Java (Sun Microsystems Inc.) auf dem Markt. Die Entwicklung von Python wurde 1989 von Guido van Rossum am Centrum voor Wiskunde en Informatica (CWI) in Amsterdam begonnen und wird nun durch die nichtkommerzielle Organisation Python Software Foundation (PSF) koordiniert. Gegenwärtig gibt es eine rasch wachsende Communitiy von Python-Programmierern.

Etwa parallel zur Entwicklung von objektorientierten Programmiersprachen wurden Konzepte der objektorientierten Analyse (OOA) und des objektorientierten Entwurfs (OOD) veröffentlicht. Im Prozess einer objektorientierten Software-Entwicklung sind OOA und OOD der Implementierung in einer Programmiersprache vorgelagert. Im Gegensatz zur rein textuellen Notation der Programmiersprachen verwenden objektorientierte Analyse- und Entwurfsmethoden auch visuelle Darstellungen. Besonders zu erwähnen ist die Unified Modeling Language (UML), die in der Version 1.1 im September 1997 publiziert wurde und heute so etwas wie einen Industriestandard zur grafischen Beschreibung objektorientierter Software-Systeme darstellt.

1.9 Aufgaben

Aufgabe 1

Welche der folgenden Texte sind Algorithmen?

1. Liebesbrief
2. Formular zur Beantragung eines Personalausweises
3. Märchen
4. Musterlösung einer Mathematikaufgabe
5. Die christlichen Zehn Gebote

Aufgabe 2

Ordnen Sie den folgenden Beschreibungen einer Problemlösung passende Programmierparadigmen zu (imperativ, objektorientiert, deklarativ).

1. Um ein Zündholz zu entzünden, reiben Sie den Kopf des Zündholzes über die Reibfläche.
2. Um eine Menge von Blumenvasen der Größe nach zu sortieren, sorgen Sie davor, dass jede Blumenvase entweder am Anfang der Reihe steht oder größer als ihr linker Nachbar ist.
3. Der Betrieb in einem Restaurant funktioniert so: Es gibt einen Koch und einen Kellner. Der Kellner kümmert sich um die Gäste, säubert die Tische, bringt das Essen und kassiert. Der Koch bereitet das Essen zu, wenn er vom Kellner einen Auftragzettel mit den Nummern der bestellten Gerichte erhält.

1.10 Lösungen

Lösung 1

1. Liebesbriefe können natürlich sehr unterschiedlich aussehen, manche sind leidenschaftlich, andere poetisch und sensibel. Wenn auch nach Auffassung des Kommunikationstheoretikers Schulz von Thun jede sprachliche Botschaft (unter anderem) auch eine appellative Dimension hat, dürfte ein Liebesbrief insgesamt wohl kaum als Anweisung zur Lösung eines Problems zu sehen sein und ist damit kein Algorithmus.
2. Ein solches Formular besitzt die entscheidenden Merkmale eines Algorithmus. Es beschreibt (einigermaßen unmissverständlich) alle Aktionen, die ausgeführt werden müssen, um das Problem »Wie komme ich an einen Personalausweis?« zu lösen.
3. Märchen erzählen, was vor langer Zeit passiert ist. Sie sind keine Algorithmen.
4. Eine gut formulierte Musterlösung beschreibt in der Regel einen Lösungsweg, führt also die mathematischen Operationen (in der richtigen Reihenfolge) auf, die man ausführen muss, um die Aufgabe zu lösen. Sie kann somit als Algorithmus (mit Kommentaren zum besseren Verständnis) betrachtet werden.
5. Die Zehn Gebote sind zwar allgemeine Verhaltensvorschriften (soziale Normen), definieren aber kein konkretes Verhalten, das zu einer Problemlösung führt.

Lösung 2

1. Imperativ. Es handelt sich um eine Folge von Anweisungen.
2. Deklarativ. Es wird beschrieben, welche Eigenschaften die Lösung (nach Größe sortierte Blumenvasen) haben muss, aber nicht, *wie* man dieses Ziel erreicht.
3. Objektorientiert. Das Restaurant wird als System interagierender Objekte beschrieben.

Lösung des Experimentes »Programmierparadigmen« (Abschnitt 1.6)

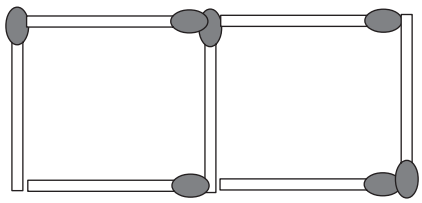

Abb. 1.8: Eine mögliche Lösung des Streichholz-Experiments

Stichwortverzeichnis

G

H

X

Y

Z