Philipp Gressly Freimann | Martin Guggisberg

Programmieren lernen

Aufgaben für den Informatikunterricht

orell füssli

Philipp Gressly Freimann | Martin Guggisberg

Programmieren lernen

Philipp Gressly Freimann | Martin Guggisberg

Programmieren lernen

Aufgaben für den Informatikunterricht Sekundarstufe II

orell füssli Verlag AG

Lektorat: Silvia Bartholdi Schucan
Umschlag: Nora Weyermann
Grafik: Philipp Gressly Freimann
Gesamtherstellung: Kösel, Krugzell

ISBN 978-3-280-04066-9

———

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Vorwort

Die meisten Schülerinnen und Schüler nutzen den Computer bereits als Kommunikations-, Informations- und Unterhaltungsplattform. Das konkrete Erstellen eines eigenen Programms, das Umsetzen eines einfachen Algorithmus, ist für sie jedoch Neuland. Zahlreiche internationale Untersuchungen zeigen, dass der Einstieg in die Programmierung eine außerordentlich hohe Herausforderung ist. Der Weg bis zu ersten eigenen Erfolgen in der Programmierung kann sehr steil und steinig verlaufen.
Das vorliegende Buch beschreibt über 150 einfache, im Unterricht erprobte Programmieraufgaben. Die Philosophie dieses Buches ist das Lernen an konkreten Aufgaben und Beispielen. Die Schülerinnen und Schüler sollen sich auf das *Programmieren im Kleinen* fokussieren.

Von der Instruktion zur Konstruktion

Bereits Comenius[1] war es ein großes Anliegen, Lernvorgänge mit Handlung und Aktivität der Lernenden zu initiieren: *Lernen durch eigenes Tun*. In der aktuellen Lernpsychologie wird das Lernen als ein konstruktiver und aktiver Prozess beschrieben, der dem Subjekt durch Zuweisung von Bedeutungen die Interpretation und die Konstruktion von Wissen ermöglicht. Nach dieser Definition kann eine Lehrperson das Wissen nicht vermitteln, sondern nur angemessene Lernsituationen bereitstellen. Es erfordert von den Lernenden eine Selbstkompetenz, im Prozess des Lernens aktiv zu werden.
Der Wissenserwerb im Fach Informatik, im Speziellen der Programmierung, unterliegt einem eigenen aktiven Prozess, dem Konstruieren eines eigenen Algorithmus und der entsprechenden Formulierung in einer Programmiersprache.

1 Johann Amos Comenius (1592)

Im Unterschied zu anderen Fächern, wo der Wissenserwerb nach kognitions- und lernpsychologischen Erkenntnissen häufig als Erwerb von Informationen resp. deklarativem Wissen stattfindet.
Anstelle eines theoretischen Einstiegs in die Programmierung sollen die Lernenden von Beginn an eigene Programme planen und konstruieren. Beim systematischen Erlernen der Programmierung müssen sie aktiv prozedurales Wissen erwerben, Handlungsabläufe üben, mit eigenen Ideen kleine Algorithmen entwerfen und mit diesen experimentieren. Die Schülerinnen und Schüler stehen im Mittelpunkt des Lernprozesses und übernehmen eine aktive Rolle bei der Konstruktion resp. der Entwicklung ihrer Programme. Nach der Kreation eines eigenen Programms muss dieses kritisch analysiert werden. Beim vorgeschlagenen exemplarischen Zugang lernen die Schülerinnen und Schüler von Anfang an den Umgang mit Fehlern.
Auf der Internetseite www.programmieraufgaben.ch befinden sich zu jeder Aufgabe verschiedene Lösungsvarianten meist in verschiedenen Programmiersprachen. Die Seite ist als kooperatives Portal zu Programmieraufgaben gedacht. Wir möchten sowohl die Lernenden wie auch die Lehrpersonen motivieren, eigene Lösungen oder neue Aufgabenstellungen auf der Plattform zu publizieren.
Für das *Programmieren im Kleinen* spielt die Wahl der Programmiersprache eine untergeordnete Rolle. Alle Aufgaben können meist auf sehr ähnliche Weise in verschiedenen Programmiersprachen gelöst werden. Es ist den Autoren ein Anliegen, dass die Wahl der Programmiersprache bei den Lehrpersonen liegt und idealerweise schulhausintern geregelt werden kann.

Dank

Ein spezieller Dank geht an unsere Revisoren und Mitautoren Igo Schaller, Michael Weiss und Valéry Gareiss. Die Webseite zum Buch wurde von Stefan Dürrenberger, umgesetzt: www.programmieraufgaben.ch.
Mit freundlicher Unterstützung der Universität Basel (Departement Informatik), der Universität Zürich (Institut für Informatik) und der Hasler Stiftung. Ein besonderer Dank geht an Beate Kuhnt. Ohne ihre Projektleitung wäre das Buch nicht zustande gekommen.
Nicht vergessen möchte ich die vielen Anregungen der Helfer und Korrekturleser: R. Schweikert, E. Freimann, M. Sismanovič, A. Bandi, M. Schweizer und M. Ković.

Winterthur und Basel, im September 2010
Philipp Gressly Freimann
Martin Guggisberg

Aufbau und Verwendung des Buches

Das Erlernen des Programmierens fordert eine außerordentlich hohe kognitive Leistung. Einsteiger sind durch Sprachelemente wie auch algorithmische Konzepte sehr stark gefordert.
In diesem Buch soll anhand von zahlreichen Beispielen das Programmieren erlernt und von den Lernenden selbst umgesetzt werden. Mit Hilfe von authentischen Aufgabenstellungen sollen die Lernenden animiert werden, einfache Probleme des Programmierens eigenständig zu lösen.[1] Sie können verschiedene Elemente der Programmierung verknüpfen und daraus ein eigenes lauffähiges Programm erschaffen, das die gestellte Aufgabe löst.
Die Kapitel sind fachlich aufeinander aufbauend angeordnet. Somit bietet es sich an, die Aufgaben der Reihe nach zu lösen. Die Fülle der Aufgaben erlaubt es, einige Aufgaben zu überspringen. Pro Kapitel sollten jedoch mindestens 3 bis 5 Aufgaben gelöst werden, damit keine Wissenslücken für nachfolgende Kapitel auftreten.
Das Buch eignet sich zum Selbststudium wie auch als Ergänzung im Informatikunterricht, indem einzelne Aufgaben zu speziellen Themen herausgepickt werden.
Die Aufgabensammlung beginnt mit den Grundkonzepten der Programmierung:

- Ausdrücke und Datentypen
- Anweisungen und Abfolgen
- Selektionen
- Schleifen

[1] Nach der Taxonomie von Bloom bewegen sie sich dabei auf dem kognitiven fünften Niveau, der Synthese.

Mit diesen Konzepten sind die meisten Problemstellungen der Informatik lösbar. Es folgen weiterführende Konzepte der modernen Programmiersprachen: Unterprogramme, Felder, Zeichenketten, Rekursion usw.
Ob die Aufgaben korrekt gelöst sind, prüft man an einfachen eigenen Testdaten. Im Lösungsteil (S. 150 ff.) sind, wo dies nötig, Daten zur Verifikation des Programms angegeben.
Am Anfang der einzelnen Kapitel wird die Theorie knapp diskutiert, jedoch lediglich, um die Begriffe abzugrenzen. Unklare Begriffe können von den Lernenden im Stichwortverzeichnis nachgeschlagen und im entsprechenden Theorieteil nachgelesen werden.
Erfahrungen aus dem Informatikunterricht zeigen, dass in den meisten Klassen eine große Heterogenität in Bezug auf das Erlernen der Programmierung vorhanden ist. Die zahlreichen Aufgaben sollen den Schülerinnen und Schülern ein individuelles selbstständiges Erwerben der Programmierfähigkeit ermöglichen. Die ausgewählten Aufgaben lassen sich im zeitlichen Rahmen einer Lektion mit verschiedenen Schwierigkeitsstufen lösen.
Bei vielen Aufgaben sind **Zusatzaufgaben** angegeben. Sie dienen

- als Zusatz für Fortgeschrittene,
- zur Vertiefung eines Themas,
- zur Prüfungsvorbereitung.

Differenzierung

Die Differenzierung der Aufgaben ist wie folgt gekennzeichnet:

einfach 
mittel 
schwierig 

Web-Code

Einige Aufgaben benötigen Ergänzungen, welche von der Webseite zum Buch www.programmieraufgaben.ch heruntergeladen werden können. Bei jeder Aufgabe ist ein entsprechender **Web-Code** angegeben. Mit diesem Code können von der Webseite Daten oder Musterlösungen heruntergeladen werden.

Aufgaben mit vorgegebenen Programmteilen

Dieses Buch enthält fast ausschließlich Aufgaben, die **unabhängig von einer Programmiersprache** formuliert sind. Aufgaben, die einen Teil der Lösung bereits zur Verfügung stellen, fallen nicht in diese Kategorie. Dennoch haben wir einige spannende JAVA™-Aufgaben unter www.programmieraufgaben.ch ins Netz gestellt.

info@programmieraufgaben.ch

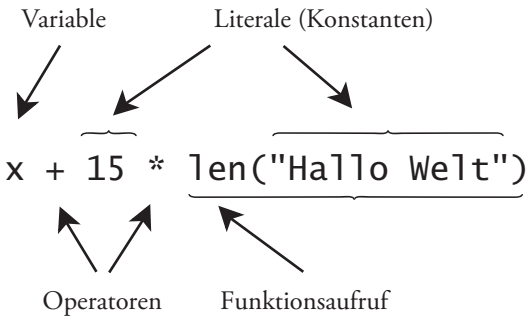
Rückmeldungen zum Inhalt, im Speziellen zu den Aufgaben, nehmen wir jederzeit gerne entgegen. Sind Aufgaben zu schwierig oder liegt ihre Bearbeitungszeit nicht im angegebenen Rahmen, sind Aufgaben mehrdeutig formuliert, hat es Fehler im Theorieteil, fehlen wichtige Stichworte usw., so bitten wir um eine Nachricht an die Autoren (info@programmieraufgaben.ch), damit eine Neuauflage des Buches entsprechend verbessert bzw. korrigiert werden kann. Inhalte auf dem Internet (www.programmieraufgaben.ch) lassen sich selbstverständlich laufend aktualisieren.

Inhaltsverzeichnis

1. Ausdrücke und Datentypen

1.1 Operatoren und Ausdrücke

Ein grundlegender Begriff der Computerprogrammierung ist der **Ausdruck** (Term). Der folgende zusammengesetzte Ausdruck veranschaulicht die wichtigsten Komponenten:



Operatoren verketten

- Literale = unveränderbare Werte eines Datentyps (z. B. Zahlzeichen, Zeichen, Wörter oder Wahrheitswerte),
- Variable,
- Resultate von Funktionen oder
- zusammengesetzte Ausdrücke

zu neuen **Ausdrücken**. Die zu verkettenden Teilausdrücke werden dabei **Operanden** genannt.
Die wichtigsten Operatoren der meisten Programmiersprachen sind

- Addition (+)
- Subtraktion (-)
- Vorzeichen (-)
- Multiplikation (*)
- Division (/)
- Restbildung (modulo[1])

[1] Der Modulo-Operator liefert den Divisionsrest bei einer ganzzahligen Division. Einige Programmiersprachen verwenden das Symbol `%`, andere das Schlüsselwort `MOD` für diesen Operator.

Neben der Klammerregel gilt eine erweiterte Form der **Punkt-vor-Strich-Regel**:

```
a + b * c = a + (b * c)
a + b * c ≠ (a + b) * c
```

Datentypen und ihre Literale (Konstanten)

Ausdrücke sind in den meisten Programmiersprachen an einen Datentypen (Wertebereich) gebunden oder werden im Kontext als solcher interpretiert. Die wichtigsten Typen, meist Standardtypen genannt, sind:

Datentyp	Beschreibung	Beispiele zugehöriger Literale (Konstanten)
`boolean`	wahr oder falsch	`true`, `false`
`integer`	ganze Zahl	`55`, `-83`
`real`	Dezimalbruch	`3.882`, `2.8e3`
`char`	Zeichen / Buchstabe	`r`, `#`, `-`, `@`, `7`
`string`	Zeichenkette, Wort	`Hallo Welt`

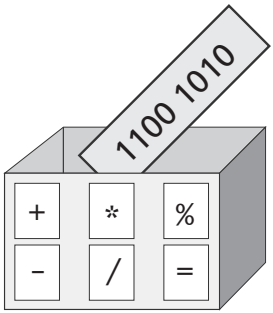
Die Standardtypen und deren Bezeichnungen (`boolean`[1], `integer`[2], `real`, `char`, `string`[3]) variieren je nach Programmiersprache.
Zur Darstellung von Werten unterscheiden Computer lediglich zwei Zustände (**Bits**), die als 0 und 1 (bzw. `false` und `true`) aufgefasst werden. Um komplexere Werte darzustellen, werden mehrere Bits in sogenannte **Bitmuster** zusammengefasst.
Die folgende Grafik zeigt, wie dasselbe Bitmuster (hier 1100 1010) je nach Typ eine andere Bedeutung erhalten kann. 1100 1010 als ganze Zahl hat den Wert 202, als Buchstabe in der Unicode-Kodierung bezeichnet es den Buchstaben Ê:

---

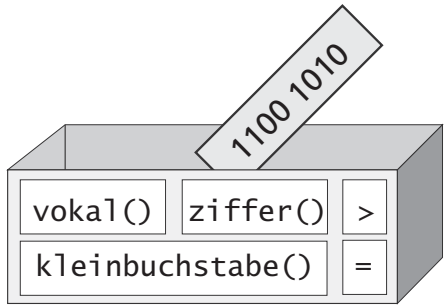
1 Nach George Boole (1815 – 1864).

2 Auch unter Namen wie `byte`, `int`, `short`, `long`, `WORD`, `CARD`, … bekannt. Hier unterscheiden sich die Datentypen lediglich in der Anzahl Bits und in der Möglichkeit, negative Zahlen darzustellen.

3 (siehe Kapitel 7)



integer (=202)



char (='Ê')

Der Datentyp «Byte» ist in maschinennahen Programmiersprachen bekannt und bezeichnet die kleinste adressierbare Größe. Auf heutigen Rechneranlagen besteht diese fast ausschließlich aus 8 Bits.

1.2 Funktionsresultate

Funktionen können wir als Operatoren auffassen:

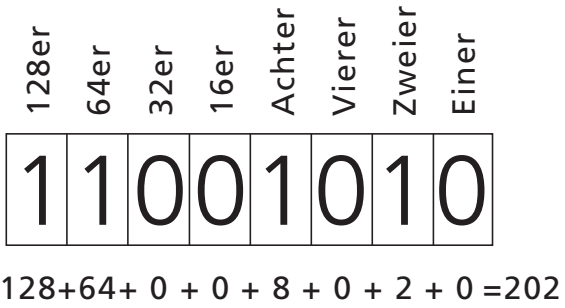
```
einargumentig:
  neg(a) = (-a)
  sin(x) = Sinus von x
zweiargumentig:
  mul(a, b) = (a * b)
dreiargumentig:
  sum(a, b, c) = (a + b + c)
```

Somit können Funktionsresultate auf natürliche Weise auch als Ausdrücke verwendet werden:

```
y := sin(x) + 2 * cos(ln(x) + phi * min(a, b))
```

1.3 Binärsystem

Zahlen werden üblicherweise im sogenannten Binär- oder Dualsystem (Zweiersystem) abgespeichert. Dabei stehen, wie im Zehnersystem, ganz rechts die Einer. Links davon stehen aber nicht etwa die Zehner, sondern die Zweier, gefolgt von den Vierern, Achtern, 16ern usw. Somit bedeutet:
«1100 1010» = 202 (128 + 64 + 8 + 2)



1.4 Kodierungen

Unter einer **Kodierung** versteht man eine Zuordnung der Buchstaben eines Alphabets zu den möglichen Nummern eines Ganzzahl-Datentyps.
Zeichen (Buchstaben, Ziffern, Sonderzeichen) werden ebenfalls binär abgespeichert. Hierzu bedient man sich Tabellen, die jedem Zeichen eine eindeutige Zahl zuordnen. Diese Tabellen werden Kodierungen genannt.
Einen Auszug aus der heute am meisten verwendeten Tabelle (dem ASCII) finden Sie im Anhang (S. 175).

1.5 Aufgaben

Die folgenden Aufgaben zum Thema «Ausdrücke und Datentypen» sind noch ohne Computer lösbar. Wer sofort mit dem Programmieren loslegen will, startet mit dem nächsten Kapitel und kommt zu diesen Übungen zurück, sobald die ersten Programme geschrieben sind und Lücken in der Theorie auftreten.

Aufgabe 1.1 Operationen in meiner Programmiersprache

Zeit:	0.5 h
Web-Code:	a7he 6pki

Erstellen Sie eine Liste aller Operatoren in Ihrer Programmiersprache. Falls es die Programmiersprache erlaubt, geben Sie zudem an, auf welche Datentypen die Operatoren wirken, und beschreiben Sie in einem Stichwort die ausgeführte Operation.
Beispiel:

Operation	Datentyp(en)	Beschreibung
`+`	`integer`, `real`	Addition
…	…	…

Aufgabe 1.2 Berechnung von Ausdrücken (1)

Zeit:	0.25 h
Web-Code:	8kqe uout

Berechnen Sie die untenstehenden Ausdrücke.
Zusätzlich zu den Ihnen bekannten Symbolen sollen die folgenden Operatoren benutzt werden:

1. `a % b` liefert den Rest der Ganzzahldivision `a / b`. Zum Beispiel ist `30 % 4 = 2`.
2. Für das mathematische Multiplikationszeichen · wird in Programmiersprachen meist der Stern * verwendet.

- `7 + 3 * 5`
- `16 % 3`
- `4 * 3 + (3 + 4) * 2`
- `1 % 15`

Aufgabe 1.3 **Berechnung von Ausdrücken (2)**

Zeit:	0.5 h
Web-Code:	mupd pv9h

Berechnen Sie die folgenden Ausdrücke. Die Variable `a` hat den Wert 5 und die Variable `x` den Wert (–7).

- `(-x) + (2 - 1 - 1) * 6`
- `a + x * a`
- `a % (x + 9)`
- `x + 12 = a`

Aufgabe 1.4 **Bits**

Zeit:	0.5 h
Web-Code:	xjyi extf

Wie viele Bits benötige ich, um 365 (bzw. 3, 9, 2000, 5000, 100 000) mögliche Werte zu speichern?
Beispiel: Um 200 mögliche Werte zu speichern, benötigen wir 8 Bits, denn mit 7 Bits sind lediglich 128 Zustände möglich. Deshalb erhöht sich die Größenordnung auf 8 Bits, womit sich dann maximal 256 Zustände abspeichern lassen.

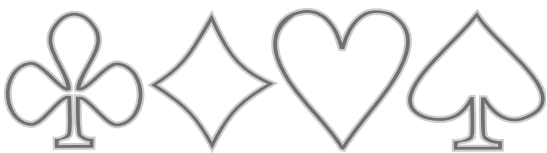
Aufgabe 1.5 Binäre Darstellung

Zeit:	0.5 h
Web-Code:	i55j 5v5u

a) Stellen Sie im Binärsystem dar: 14, 33, 54, 100, 1000, 3737
b) Stellen Sie die folgenden Binärzahlen im Dezimalsystem dar: 1001, 1110, 100 1011, 1 0110 1110, 11 0011 0111 1101
c) Berechnen Sie binär, indem Sie die folgenden Dezimalzahlen zuerst in das Binärsystem verwandeln, danach (binär) addieren und das Resultat wieder in das Dezimalsystem zurückverwandeln.
   - 5 + 8
   - 19 + 22
   - 7 + 15
   - 192 + 258

Aufgabe 1.6 Spielkarten

Zeit:	0.5 h
Web-Code:	8mso qivy



Ordnen Sie alle 52 Spielkarten eines Poker-Sets (Spielkarten von Rommé, Bridge, Canasta) einem Bitmuster zu. Eine solche Zuordnung von Bitmustern auf vorgegebene Werte nennt man eine Kodierung. Wie viele Bits benötigen Sie? Ist ihre Zuordnung eindeutig?
Zusatzaufgabe: Wie viele Bits benötigen Sie zusätzlich, wenn Sie die drei Joker-Karten auch festlegen?

Aufgabe 1.7 Go-Spiel

Zeit:	0.25 h
Web-Code:	9c7m tjxk

Im Go-Spiel kreuzen sich 19 horizontale mit 19 vertikalen Linien. Auf die 361 Schnittpunkte können schwarze bzw. weiße Steine gesetzt werden. Jeder dieser Kreuzpunkte hat nach jedem Zug einen der drei Zustände *schwarz*, *weiß* oder *leer* (noch unbesetzt). Eine Spielposition besteht demnach aus 361 Punkten, wobei jeder Punkt entweder schwarz, weiß oder leer ist.
Modellieren Sie eine Go-Spielposition und versuchen Sie, mit möglichst wenig Speicher, die gesamte Position festzuhalten. Wer schafft es unter 80 Byte?
Tipp: Zahlensysteme.

Aufgabe 1.8 Binäre Verschiebung

Zeit:	0.5 h
Web-Code:	8uxd zron

Zeigen Sie, dass eine Verschiebung einer Binärzahl um zwei Stellen nach links eine Zahl vervierfacht. Was geschieht, wenn Sie eine Binärzahl
a) um sechs Stellen nach links schieben?
b) um eine Stelle nach rechts schieben?

Aufgabe 1.9 Zweierkomplement

Zeit:	1 h
Web-Code:	kjb8 rjjm

Im **Zweierkomplement** (= Binärkomplement) werden die positiven Zahlen (inkl. der Null) einfach im Binärsystem dargestellt (0 = 0000 0000, 1 = 0000 0001, 2 = 0000 0010, 3 = 0000 0011, …). Die negativen Zahlen jedoch werden wie

folgt dargestellt: Die Zahl –1 wird durch lauter binäre Einerziffern dargestellt (–1 = 1111 1111), jede um Eins kleinere Zahl wird auch durch eine um Eins kleinere Binärzahl dargestellt (–2 = 1111 1110; –3 = 1111 1101, –4 = 1111 1100, –5 = 1111 1011, …).

a) Stellen Sie die folgenden (Zweierkomplement-)Binärzahlen dezimal dar:
   - 1110 1011
   - 1110 1110
   - 1101 1100

b) Schreiben Sie im Zweierkomplement: –6, –8, –17, –97, +5

c) Berechnen Sie die folgenden Subtraktionen (bzw. Additionen) im Zweiersystem. Verwenden Sie ausschließlich binäre Darstellungen. Schreiben Sie die negativen Zahlen zunächst im Zweierkomplement, damit Sie zur Lösung ausschließlich binäre Additionen verwenden.
   - 0 – 1
   - –1 – 1
   - 1 – 2
   - 2 – 3
   - 18 – 5
   - 5 – 13
   - –32 – 16
   - –25 + 9

   Beispiel:
   –13 – 5 = (–13) + (–5) = 1111 0011 + 1111 1011 = 1 1110 1110 = –18

Aufgabe 1.10 **UTF-8**

Zeit:	0.25 h
Web-Code:	v9hy tsk4

Um alle Zeichen lebender Sprachen in ein und derselben Kodierung unterzubringen, wurde **Unicode** (`www.unicode.org`) geschaffen. Die meisten Programme und Betriebssysteme sind heute in der Lage, Unicode-Zeichen zu verarbeiten und korrekt darzustellen. Der ursprüngliche Unicode verwendet 16 Bits pro Zeichen; damit sind 2^{16} = 65 536 Symbole darstellbar. Der aktuelle Unicode

(Version 6.0) verwendet sogar 20 Bit, doch sehr viele Plätze sind noch nicht vergeben. Die meisten Texte (Deutsch, Englisch, Französisch) kommen jedoch mit viel weniger Zeichen aus. Daher entstand die Idee, diese häufigsten Zeichen auch nur mit 8 Bits (= ein Byte) darzustellen.
Die UTF-8 Kodierung – die obige Komprimierung umsetzt – ist wie folgt definiert:

- Zeichen mit einem (Unicode) Wert < 128 werden in einem Byte gespeichert (dabei bleibt das erste Bit immer auf 0 (Null): 0xxx xxxx.
- Zeichen mit einem Wert von 128 bis 2047 werden in zwei Byte gespeichert: 110x xxxx, 10xx xxxx. Dabei beginnt das erste Byte mit zwei Einsen gefolgt von einer Null und fünf Bits für die «Nutzdaten». Das zweite Byte startet mit «10» und sechs Nutzdaten-Bits. Somit sind elf Nutzdaten-Bits vorhanden. Damit können 2^{11} Zeichen dargestellt werden.
- Analog verhält es sich mit Zeichen im Bereich 2048 bis 65 535. Diese werden in drei Byte mit sechzehn Nutzdaten-Bits wie folgt gespeichert: 1110 xxxx, 10xx xxxx, 10xx xxxx.

Diese Aufgabe ist ohne Programmieren lösbar: Welche Zeichen verbergen sich hinter den folgenden Zahlen? Schreiben Sie die Zahlen zunächst binär (mit je 8 Binärziffern) auf, suchen Sie anschließend die «Nutz-Bits» und errechnen Sie daraus die Zahl des kodierten Zeichens, das Sie abschließend unter `www.unicode.org` nachschlagen.

- 64
- 80
- 194, 190
- 195, 139
- 239, 172, 129
- 226, 137, 136

2. Anweisungen und Abfolgen

Ein einfaches Computerprogramm besteht aus einer Abfolge (= Sequenz oder Aneinanderreihung) von Anweisungen (Befehlen). Nach dem Starten des Programms wird die Sequenz in Leserichtung (also der Reihe nach von oben nach unten) verarbeitet.

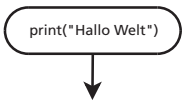
2.1 Anweisungen

Die wichtigsten Anweisungen imperativer[1] Programmiersprachen sind:

- Aufrufen von vorgegebenen Befehlen wie:
  - Ausgabe von Text und Zahlen auf die Konsole
  - Einlesen von Text und Zahlen von der Tastatur
  - Mathematische Funktionen (Wurzelziehen, …)
  - Hilfsfunktionen (Kalender, Ausgabeformat, …)
  - Öffnen, Lesen, Beschreiben und Schließen von Dateien
- Zuweisen von Werten an Variable

Daneben gibt es umfassende Bibliotheken mit Hilfsfunktionen. Zum Beispiel existieren für die Benutzerschnittstellen Hilfsfunktionen zum Erzeugen, Darstellen und Verändern von Fenstern, Schaltflächen, Linien, Farben usw.

Eine einzelne Anweisung (hier `print("Hallo Welt")`) kann durch die folgende abgerundete Form mit ausgehendem Pfeil dargestellt werden[2]:



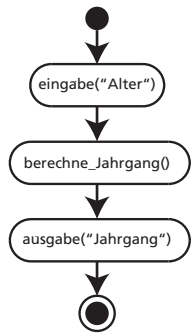
[1] imperativ = befehlsorientiert

[2] Es gibt viele weitere Darstellungstechniken. Dieses Buch verwendet die Notation der UML = Unified Modelling Language.

2.2 Abfolgen

Eine **Abfolge** (= Sequenz) bestimmt, in welcher Reihenfolge die Anweisungen auszuführen sind.
Mit einer Anweisung wird erst begonnen, wenn die vorangehende Anweisung komplett abgeschlossen ist.
Mehrere Anweisungen in einer Abfolge werden durch ein Trenn- oder Abschlusssymbol in ihre Reihenfolge gebracht. Dieses Trennsymbol ist je nach Programmiersprache ein Strichpunkt, ein Komma oder ein Zeilenumbruch. Beispiel[1]:

```
eingabe("Alter")
berechne_Jahrgang()
ausgabe("Jahrgang")
```



Variable

Variable sind Platzhalter für Werte. Sie können Zahlen, Zeichen oder Wörter, aber auch größere Speicherbereiche über längere Zeit festhalten. Diese Werte können später wieder abgefragt werden. Sie werden durch Wörter oder Einzelbuchstaben gekennzeichnet. Hat beispielsweise die Variable x den Wert 2010, so hat der Ausdruck 2x + 3 den Wert 4023.
In typisierten Sprachen hat jede Variable einen ihr zugehörigen Datentypen. Dies wird in einer Deklarationsanweisung angegeben. Beispiel:

```
eingabeKorrekt: boolean
anzahlZellen  : integer
mmAbstand     : real
```

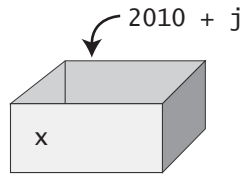
[1] Die beiden schwarzen Kreisflächen im Diagramm bezeichnen den Programmanfang bzw. das Programmende.

2.3 Zuweisung

Die wohl am häufigsten eingesetzte Anweisung ist die Zuweisung. Variable erhalten ihre Werte durch Zuweisung eines Ausdrucks.

variable:=<ausdruck>

Beispiel: `x := 2010 + j`



Die Zuweisung eines Wertes an eine Variable (im Symbol «:=» bzw. «←») wird in vielen Programmiersprachen durch ein einfaches Gleichheitszeichen dargestellt. In diesem Fall bedeutet

```
y = 1 + 2 * y
```

nicht etwa, dass die Gleichung nach der Variablen `y` zu lösen ist.
Sondern `y = 1 + 2 * y` bedeutet, dass der Wert des Ausdrucks auf der rechten Seite der Variable `y` zugewiesen wird. Im Beispiel wird zunächst der Wert des Ausdrucks `(1 + 2 * y)` berechnet; dieser neue Wert überschreibt danach den aktuellen Wert der Variable `y`.

2.4 Aufgaben

Aufgabe 2.1	**Hello World**
Zeit:	0.25 h
Web-Code:	hr5f yysf

Schreiben Sie ein Programm, das den Text «Hello World» ausgibt.

Aufgabe 2.2	**Anweisungen und Ausdrücke**
Zeit:	0.25 h
Web-Code:	qw25 yppb

Wie viele **Anweisungen** (a) und **Ausdrücke** (b) sind in folgendem Programmlisting? Zählen Sie eine Zuweisung (`x := 4`) nicht als Ausdruck, obschon das in einigen Programmiersprachen ein erlaubter Ausdruck wäre!

```
a := input()

x := 44 + a
y := 44 + 3 * x
z := a * x + 4 * y

print(z)
```

Beispiel:

```
s := 5 + 6
print (s)
```

In obigem Beispiel haben wir in jeder Zeile eine Anweisung und in der ersten Zeile die Ausdrücke 5, 6 und 5 + 6 und in der zweiten Zeile den Ausdruck s.

Aufgabe 2.3 Variablenwert nach einer Sequenz

Zeit:	0.5 h
Web-Code:	pi2t 72oe

Was wird durch folgende Sequenz ausgegeben? Gehen Sie davon aus, dass die Anweisung `print(x)` einen Wert auf der Konsole ausgibt und
`y := eingabe("Geben Sie y ein")`
den Benutzer bzw. die Benutzerin um eine Zahleneingabe bittet und den eingegebenen Wert in die Variable y speichert.

```
x := 4
y := eingabe("Geben Sie y ein")
x := y + x
x := x + y
x := 3 * y + x
x := x - 5 * y
print(x)
```

Aufgabe 2.4 Ohm'sche Gesetze

Zeit:	0.5 h
Web-Code:	po9n fjn6

Berechnen Sie elektrische Widerstände für a) Serienschaltung und b) Parallelschaltung von zwei gegebenen Widerständen R_1 und R_2.

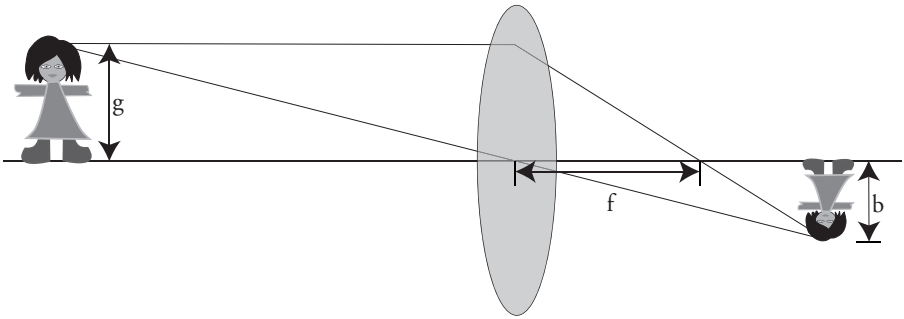
a) $R = R_1 + R_2$

b) $R = \frac{1}{\frac{1}{R_1}+\frac{1}{R_2}}$

Schreiben Sie dazu ein Programm, das R_1 und R_2 entgegennimmt und obige Resultate ausgibt.

Aufgabe 2.5 Linsengleichung (Optik)

Zeit:	0.25 h
Web-Code:	2v7c bobb



Berechnen Sie den Brechungsindex f einer Linse, wenn die zwei Größen Bildweite b und Gegenstandsweite g gegeben sind, nach folgender Formel:
1/f := 1/b + 1/g
Schreiben Sie dazu ein Programm, das b und g erfragt und den Brechungsindex f ausgibt.

Aufgabe 2.6 Runden (1)

Zeit:	0.5 h
Web-Code:	gs4u sre7

Schreiben Sie ein Programm, das eine Zahl auf das nächstgelegene Vielfache von 100 auf- bzw. abrundet. Zum Beispiel wird 250 auf 300 aufgerundet, 15 348 jedoch auf 15 300 abgerundet.
Option: Schreiben Sie ein Programm, das einen Preis auf 5 Rappen rundet und in Franken und Rappen ausgibt.

Zusatzaufgabe: Verallgemeinern Sie das Programm, so dass auf beliebige Zahlen (z. B. nächste Zahl der Siebnerreihe, auf Drittel usw.) gerundet werden kann.
Tipp: Falls Ihre Programmiersprache nicht runden kann, so verwandeln Sie $(x + 0.5)$ in eine ganze Zahl, um auf ganze Zahlen auf- bzw. abzurunden.

Aufgabe 2.7 Runden (2)

Zeit:	2 h
Web-Code:	z296 j92g

Schreiben Sie eine Methode, die eine gebrochene Zahl (`real`) auf vier signifikante Ziffern rundet.
Beispiele:

Eingabe (real)	Ausgabe (real)
0.777777777777…	0.7778
23.456789123…	23.46
0.00223322332233…	0.002233 (= 2.233e-3)
12 372 889	12 370 000 (= 1.237e6)

Aufgabe 2.8 Stunden, Minuten, Sekunden

Zeit:	1 h
Web-Code:	deeb acqv

Schreiben Sie ein Programm, das eine Größe, die in Stunden, Minuten und Sekunden angegeben ist, ins metrische System (Zehnersystem) mit der Maßeinheit «Stunden» umrechnet.
Beispiel: 1 h 15’ wird in 1.25 h verwandelt.
Zusatzaufgabe: Schreiben Sie auch die Umkehrung. Beispielsweise soll 1.23 h in 1 h 13' 48" verwandelt werden.
Tipp: 0.01 h = 36 Sekunden und 1 Sekunde = 0.0002777… h.

Aufgabe 2.9 Ganzzahlarithmetik (Kommutativgesetz)

Zeit:	0.25 h
Web-Code:	rdhf 4kvg

Erklären Sie, warum der nachfolgende Programmcode je nach Programmiersprache für die ganzzahligen(!) Variablen `r1` und `r2` ein anderes Resultat liefern kann (verifiziert mit Java™). Mit anderen Worten: Warum gilt das **Kommutativgesetz** in der Computerprogrammierung nicht immer?

```
i, j, k: integer
r1, r2:  integer

i  := 500
j  := 27
k  := 5
r1 := i / j * k
r2 := i * k / j

print("r1: " + r1)
print("r2: " + r2)
```

Aufgabe 2.10 Gleitkommaarithmetik (Assoziativgesetz)

Zeit:	0.25 h
Web-Code:	kwhz einv

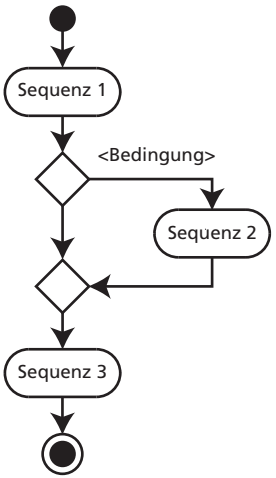
Erklären Sie, warum der nachfolgende Programmcode für die Variablen `d1` und `d2` in bestimmten Programmiersprachen ein anderes Resultat liefern kann (verifiziert mit Java™). Mit anderen Worten: Warum gilt das **Assoziativgesetz** in der Computerprogrammierung nicht immer?

```
a, b, c: real
d1, d2:  real

a  := 1e17 // 100…000 (17 Nullen)
b  := -2
c  := 1 - a
d1 := a + b + c + 1
d2 := a + (b + c + 1)

print("d1 " + d1)
print("d2 " + d2)
```

3. Selektion (Verzweigung)

Neben der sequenziellen Ausführung von Anweisungen ist die **Selektion**[1] ein elementares Konzept der Programmierung. Eine Selektion steuert den Verlauf eines Programms. Bestimmte Teilsequenzen werden nur unter einer gegebenen Bedingung ausgeführt.
Die Selektion wird durch das Schlüsselwort `if` (wenn) eingeleitet. So wird in folgendem Beispiel die 2. Sequenz (`Sequenz 2`) **nur** ausgeführt, wenn die Bedingung (`<Bedingung>`) wahr ist.

```
Sequenz 1
if(<Bedingung>)
{
  Sequenz 2
}
Sequenz 3
```



Teilsequenzen (Blöcke) werden durch spezielle Schlüsselwörter (`begin`, `end`), durch Klammern (viele Programmiersprachen verwenden geschweifte Klammern) oder durch Einrücken eingeleitet und beendet.

3.1 Boole'sche Ausdrücke

Die Bedingungen werden durch Wahrheitsausdrücke (Boole'sche Ausdrücke) angegeben. Solche Ausdrücke haben als Resultat den Wert `true` oder `false`.
Ausdrücke werden durch Vergleichs- oder durch logische Operatoren zusammengesetzt.

[1] Selektion wird auch Verzweigung, Entscheidung oder Auswahl genannt.

Vergleichsoperatoren

Vergleichsoperatoren vergleichen numerische Werte miteinander.

Operator	in Programmiersprachen	Bedeutung
$<$	`<`, `.lt.`	ist kleiner als
$\leq$	`<=`, `.le.`	kleiner oder gleich
$>$	`>`, `.gt.`	ist größer als
$\geq$	`>=`, `.ge.`	größer oder gleich
$=$	`=`, `==`, `===`, `.eq.`	identisch, gleich
$\neq$	`!=`, `^=`, `<>`, `¬=`, `~=`, `.neq.`, `!==`	ungleich

Logische Operatoren

Die wichtigsten logischen Operatoren sind:

Operator	C/Java™-Symbol	Beschreibung
AND	&	mathematisches Und
OR	|	mathematisches Oder
NOT	!	Verneinung, Umkehrung, Negation
XOR	^	exklusives Oder

Beispiele

- `if(isOK) {...}`
- `if((NOT eingabeKorrekt) OR eingabeLeer) {...}`
- `if((alter < 19) AND inAusbildung) {...}`

In obigem Beispiel sind `isOK`, `eingabeKorrekt`, `eingabeLeer` und `inAusbildung` Variable vom Datentyp `boolean`. Die Variable `alter` hingegen ist eine Zahl vom Datentyp Integer.

3.2 Aufgaben

Aufgabe 3.1 **IF-Formulierungen**

Zeit:	1 h
Web-Code:	gxi7 afay

Formulieren Sie die folgenden Aussagen als `if(...)`-Bedingung:

a) `y` ist positiv.
b) `a` liegt im Intervall –2 bis 5.3.
c) Der Preis ist kleiner als 5 % von 20.– CHF.
d) `x` ist kleiner als 5 und ungerade.
e) `t` liegt im Intervall [–3, 6] oder im Intervall (2, 8). Die Intervallgrenzen werden üblicherweise mit eckigen ‹[]› oder runden ‹()› Klammern angegeben. Bei eckigen Klammern gehören die Intervallgrenzen dazu (inklusive). Bei runden Klammern gehören die Grenzen nicht dazu (exklusive). In der Aufgabe gehört also z. B. die Zahl –3 dazu, wohingegen die Zahl 8 nicht Teil des Intervalls ist.

Aufgabe 3.2 **Boole'sche Ausdrücke**

Zeit:	0.5 h
Web-Code:	tby9 5gzh


Die Symbole «|» bzw. «&» stehen hier für die logische ODER- bzw. die logische UND-Verknüpfung. «&» wird prioritär vor «|» ausgeführt. Das Symbol «!» bezeichne die logische Umkehrung (Negation).

a) Berechnen Sie damit (`true | false`) & (`false | !true`).
b) Berechnen Sie die folgenden Ausdrücke mit der Bedingung, dass a den Wert 5 und x den Wert (–7) hat.

- $17 < 19$
- $-3 < -8$
- $(4 > 3)$ & $(3 > 4)$
- $(8 \,\%\, 3 = 0)$ | $(8 \,\%\, 2 = 0)$
- $(-13 < -8)$ | $(-1 < 0)$ & $(2 > 3)$
- `true | false` = $(5 < 2 * 3)$ & (`false & true`)
- $a < x$
- $-x < -a - 2$

c) Gegeben `a = true`, `b = false`, `c = true`. Berechnen Sie:

- `a & (b | c)`
- `(b & a) | c`
- `b & (a | c)`
- `(a & b) | (a & c)`
- `(( ! a) | (! b)) & ((! a) | (! c))`

d) Versuchen Sie die folgenden logischen Ausdrücke nur mit den Operatoren AND und NOT darzustellen. Ein ODER (|) darf also nicht mehr vorkommen.

- `! (a | b)`
- `a | (b & c)`

Aufgabe 3.3 **Was wird ausgegeben?**

Zeit:	0.25 h
Web-Code:	wp6s 39bx

Betrachten Sie folgendes Programm (lösbar ohne Computer):

```
x := 18
if(x * 2 > 30)
{
  x := x + 4
}
x := x / 2
x := x + 1
if(x - 11 < 2 * x - 23)
{
  x := x - 1
}
print(x)
```

Was wird bei `print(x)` ausgegeben?

Aufgabe 3.4 **Selektionsinvariante**

Zeit:	0.25 h
Web-Code:	3t94 v5nz

Entfernen Sie die Invarianten aus folgender Selektion. Mit anderen Worten: Welche Anweisungen könnten auch vor oder nach der Selektion stehen?

```
if(x < y)
{
      print("kleiner")
      a := einlesen()
}
else
{
      print("größer")
      a := einlesen()
}
```

Aufgabe 3.5 **Bedingte Zuweisung**

Zeit:	0.5 h
Web-Code:	2ib4 q6ax

Schreiben Sie eine Selektionsanweisung, die der Variablen `output` den Wert der Variablen `input` nur dann zuweist, wenn `input` gerade ist. Wenn `input` ungerade ist, soll die Variable `output` nicht verändert werden.

Aufgabe 3.6 **AHV**

Zeit:	0.5 h
Web-Code:	yo4o 3stj

Die Beitragspflicht für die AHV (Alters- und Hinterlassenenversicherung) beginnt ab dem 1. Januar nach Vollenden des 17. Altersjahres und endet (bis zur Umsetzung des Artikels 8 der Schweizer Bundesverfassung vom 18. Dez. 1998) unterschiedlich; nämlich im Alter von 64 Jahren bei Frauen bzw. 65 Jahren bei Männern.
Zur Vereinfachung sollen alle 18- bis und mit 64- (bzw. 65-) Jährigen beitragspflichtig sein.
Schreiben Sie ein Programm, bei dem die Anwenderin Alter und Geschlecht eingeben kann. Die Ausgabe des Programms soll entweder «noch nicht AHV-beitragspflichtig», «AHV-beitragspflichtig» oder «bereits AHV-Empfänger» sein.

Aufgabe 3.7 Body-Mass-Index

Zeit:	0.5 h
Web-Code:	4du5 ufpz

Der BMI (Body-Mass-Index) bewertet das Körpergewicht eines Menschen im Verhältnis zum Quadrat seiner Größe.
BMI = kg/m^2
Folgende Indizien werden vorgeschlagen:

w	m	Klassifikation
< 19	< 20	Untergewicht
19 – 24	20 – 25	Normalgewicht
> 24	> 25	Übergewicht

Schreiben Sie ein Programm, bei dem der Anwender sein Geschlecht (w, m), seine Masse (in kg) und seine Körpergröße (in Metern) eingeben kann. Die Ausgabe soll dann neben dem BMI auch die Klassifikation (Untergewicht, Normalgewicht bzw. Übergewicht) enthalten.
Bemerkung: Der BMI berücksichtigt nicht den Körperfettanteil und kann daher nur begrenzt Aussagen zur Gesundheit machen. Ein moderneres Indiz zu Über- und Untergewicht liefert das Taille-zu-Höhe-Verhältnis *(Waist to Height Ratio).*

Aufgabe 3.8 Hundert Binärzahlen

Zeit:	1 h
Web-Code:	an5r qqzp

Schreiben Sie ein Programm, das eine Zahl zwischen 0 und 100 entgegennimmt und diese Zahl im Binärsystem ausgibt.

Aufgabe 3.9 Kilobyte

Zeit:	2 h
Web-Code:	nk4t g9hc

Ein Kilogramm ist bekanntlich 1000 Gramm. Die ISO-Abkürzung für Kilo (1000) ist k. Somit ist ein Kilobyte gleich 1000 Byte (kB). In der Informatik werden aber (für RAM Bausteine) häufiger 1024 Byte als *Kilobyte* verwendet. Das kommt daher, dass ganze Zahlen im Zweiersystem repräsentiert werden und 2 hoch 10 eben 1024 (und nicht 1000) ergibt. Die korrekte ISO-Abkürzung für 1024 Byte lautet jedoch *Kibibyte,* abgekürzt KiB. Analog ist ein *Mebibyte* (MiB) = 1024 * 1024, ein Megabyte ist hingegen *nur* 1 000 000 Byte. G steht für Giga, also 1000 Millionen, usw.
Schreiben Sie ein Programm, das mindestens die folgenden Größen umrechnen kann: Bit, Byte (8 Bits), kB (Kilobyte), KiB (Kibibyte), MB (Megabyte), MiB (Mebibyte), GB (Gigabyte) und GiB (Gibibyte).
Bei Ihrem Programm wird der Anwender nach der Ausgangsgröße (Bit, Byte, kB, KiB, MB, …), der Zielgröße (Bit, …) und dem umzurechnenden Wert gefragt.
Beispiel:

```
"Folgende Größen können umgerechnet werden:
Bit, Byte, kB, KiB, MB, MiB, GB, GiB"

Geben Sie die Ausgangsgröße "VON" an: KiB
Geben Sie die Zielgröße "NACH" an   : Bit
Geben Sie den umzurechnenden Wert an: 2
Resultat: 2 KiB = 16384 Bits
```

Aufgabe 3.10 Elektrische Spannung

Zeit:	1 h
Web-Code:	43b5 76us

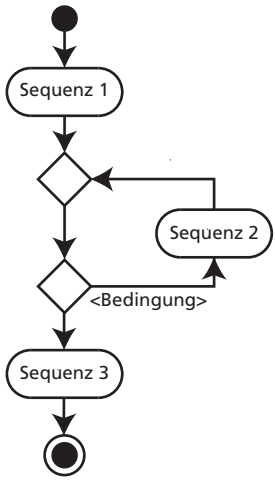
Nach der bekannten Formel U = R * I werden Spannung (U [Volt]), Widerstände (R [Ohm, Ω]) und Stromstärken (I [Ampère]) umgerechnet. Schreiben Sie ein Programm, bei dem der Anwender wahlweise zwei der drei Zahlen aus der Formel eingeben kann. Für die gesuchte Größe soll der Anwender ein Fragezeichen (?) eingeben. Das Programm berechnet dann die dritte Zahl; diejenige, bei der der Anwender das Fragezeichen eingegeben hat.
Beispiel:

```
U := 240
R := ?
I := 2.5
Resultat R = 96 Ohm
```

4. Schleifen

Neben der Selektion (Auswahl) ist die **Schleife** (auch Wiederholung oder Iteration genannt) elementar für den sogenannten Kontrollfluss. Unter «Kontrollfluss» versteht man die Abarbeitungsreihenfolge der Befehle.
Für eine Wiederholung steht in den meisten Programmiersprachen das Schlüsselwort `while`. In folgendem Beispiel wird die 2. Sequenz (`Sequenz 2`) **so lange** ausgeführt, wie die Bedingung (`<Bedingung>`) wahr ist.

```
Sequenz 1
while(<Bedingung>)
{
  Sequenz 2
}
Sequenz 3
```



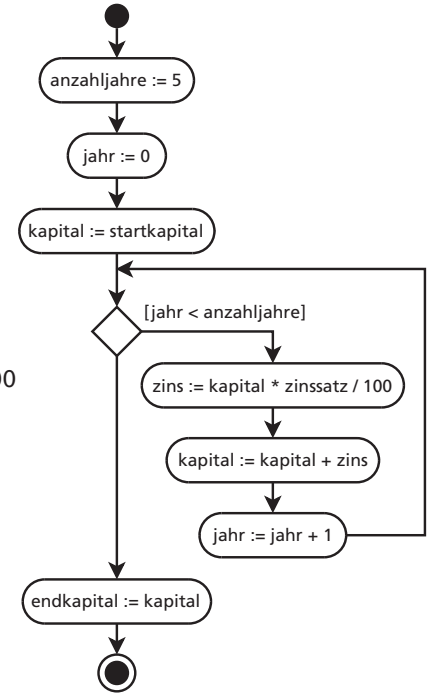
Einführungsbeispiel Zinseszins: Um ein Kapital nach 5 Jahren bei einem gegebenen Zinssatz zu berechnen, kann wie folgt vorgegangen werden:

```
nach1 := kapital + kapital * zinssatz / 100
nach2 := nach1   + nach1   * zinssatz / 100
nach3 := nach2   + nach2   * zinssatz / 100
nach4 := nach3   + nach3   * zinssatz / 100
nach5 := nach4   + nach4   * zinssatz / 100
```

Flexibler, kürzer und weniger fehleranfällig ist das Programm mit einer Schleife[1]:

[1] Falls eine (mathematisch) geschlossene Formel existiert, wäre diese der Schleife natürlich vorzuziehen. Hier: `endkapital := kapital * (1 + zinssatz/100)`^{5}.

```
anzahljahre:= 5

jahr       := 0

kapital    := startkapital

while(jahr < anzahljahre)
{

  zins     := kapital * zinssatz / 100

  kapital  := kapital + zins

  jahr     := jahr    + 1
}
endkapital := kapital
```



Um das Kapital nach 10 oder 20 Jahren zu berechnen, ist bei Programmierung mittels Iteration lediglich eine einzige Zahl in diesem Programm zu ändern.

4.1 Aufgaben

Aufgabe 4.1	**Iterationen**
Zeit:	0.25 h
Web-Code:	vrft w7zr

Schreiben Sie ein Programm, das 10-mal «Hopp Schwiiz» schreibt. Schreiben Sie das Programm

- zunächst als Sequenz (10 Befehle hintereinander),
- danach als Iteration mit einer Variablen `(integer) zaehler := 1` bis `10` und
- zuletzt mit einer Variablen `i`, beginnend bei `i := (-5)` bis (?) na, was wohl?

Aufgabe 4.2 Hundert Quadratzahlen

Zeit:	0.5 h
Web-Code:	rnuy razx

Schreiben Sie ein Programm (mit einer `while()`-Schleife), das die ersten hundert Quadratzahlen ausgibt.
Zusatzaufgabe: Erstellen Sie die Schleife auch so, dass keine Multiplikation vorkommt. Verwenden Sie ausschließlich die viel schnellere Addition. Tipp: Betrachten Sie die Differenzen je zweier aufeinanderfolgender Quadratzahlen oder schauen Sie sich die Summe der ungeraden Zahlen von 1 bis n an.

Aufgabe 4.3 Kleines Einmaleins

Zeit:	1 h
Web-Code:	86my ifu5

Geben Sie die Multiplikationstafel des kleinen Einmaleins (bis 10 x 10) als Standard-Ausgabe aus. Option: Der Anwender bestimmt das Maximum (n x n).
Zusatzaufgabe: Fügen Sie bei der Ausgabe jeweils vor jeder Zahl so viele Leerzeichen ein, dass die Zahlen in einem schönen quadratischen Raster angeordnet werden.

Aufgabe 4.4 Fahrenheit

Zeit:	0.5 h
Web-Code:	kcyy 2mt8

Schreiben Sie ein Programm, das von –20 bis 100 Grad die Celsiuswerte in 5er-Schritten in Fahrenheit umrechnet und als Tabelle ausgibt.
Die Formel dazu lautet: fahrenheit = (9.0/5.0) ∗ celsius + 32.
Zusatzaufgabe: Schreiben Sie auch die Umkehrung.

Aufgabe 4.5 Domino

Zeit:	0.5 h
Web-Code:	3ika 8b3t

Schreiben Sie ein Programm, das alle möglichen Dominosteine (0|0), (0|1), …, (6|6) erzeugt. Dabei dürfen keine Duplikate auftreten: Wenn (3|5) schon erschienen ist, darf also (5|3) nicht mehr auftreten.

Aufgabe 4.6 Wertetabelle

Zeit:	1 h
Web-Code:	smw6 u9sy

Geben Sie zu einer quadratischen Funktion $y = ax^2 + bx + c$ eine Wertetabelle aus. Die Anwenderin kann a, b und c eingeben. Ebenso kann sie den Startwert, den Endwert und das Intervall festlegen.
Beispiel: Die Anwenderin will eine Wertetabelle der quadratischen Funktion $y = 5x^2 - 3x + 2$ im Intervall -5 bis 5 mit Schrittweite 0.5.

```
a         :=  5
b         := -3
c         :=  2
start     := -5
end       :=  5
intervall :=  0.5
```

Es wird eine Wertetabelle mit 21 Werten ausgegeben.
Zusatzaufgabe: Erweitern Sie Ihre Lösung für kubische Funktionen $y = ax^3 + bx^2 + cx + d$.

Aufgabe 4.7 Schleifenabbruch

Zeit:	0.5 h
Web-Code:	m7iq 4iy5

Lösen Sie die Aufgaben a) bis d) ohne Computer.
Welche der folgenden Schleifen enden nach endlicher Anzahl von Durchläufen?

a)
```
a := 0
while(a < 7)
{
  b := 5
  b := b + b
}
```

b)
```
a := 0
while(a < 7)
{
  b := a
  a := a + b
}
```

c)
```
a := 1
while(a < 7)
{
  b := a
  a := a + b
}
```

d)
```
x := 4
y := 3
while(x > y)
{
  t := x
  x := y
  y := t
}
```

Aufgabe 4.8 Selektion vermeiden

Zeit:	0.5 h
Web-Code:	c75n hpo8

Schreiben Sie folgendes Programm, ohne eine Verzweigung auszuführen. Verwenden Sie lediglich die Iteration (`while()`). Fügen Sie bei Bedarf beliebige Boole'sche Variable ein. Natürlich darf kein «break» oder analoges Konstrukt verwendet werden!

```
a := input("A")
b := input("B")
c := input("C")
if(a < b)
{
  print(a)
}
else
{
  print(c)
}
```

Diese Aufgabe aus der theoretischen Informatik deutet an, dass jedes Programm keine weitere Kontrollstruktur als `while()` benötigt.

Aufgabe 4.9 Schleifeninvariante

Zeit:	0.25 h
Web-Code:	erdg qc8i

Bei einem Testlauf einer Software hat sich ergeben, dass die Schleife zu viel Rechenzeit in Anspruch nimmt. Entfernen Sie daher die Invarianten (d. h. die Teile, welche nicht innerhalb der Schleife stehen müssen) aus der Iteration.

```
while(a < 7)
{
  b := a + 4
  print (b)
  y := a * 2
  a := a + 2
}
```

Aufgabe 4.10 Zahlensumme (1)

Zeit:	1 h
Web-Code:	3uy9 4fbo

Schreiben Sie ein Programm, das alle Zahlen von 1 bis 2011 zusammenzählt.
Zusatzaufgaben:

- Ändern Sie das Programm so ab, dass auch andere Startwerte möglich sind.
- Finden Sie eine mathematische Formel, so dass Ihr Programm ganz ohne Schleife auskommt.

Aufgabe 4.11 Multiplikation

Zeit:	0.5 h
Web-Code:	a82k ofbb

Schreiben Sie ein Programm, das zwei Zahlen miteinander multipliziert, indem nur die Addition verwendet wird.
Zusatzaufgabe: Lassen Sie in einer erweiterten Lösung auch zu, dass jeder der beiden Faktoren negativ sein darf.

Aufgabe 4.12 Fakultäten

Zeit:	0.5 h
Web-Code:	neh4 hqjt

Programmieren Sie die Fakultätsfunktion mittels Schleife. Die Fakultät gibt an, auf wie viele Arten sich n Objekte auf n Plätze verteilen können. Zum Beispiel können sich zwei Personen auf zwei Arten auf zwei Stühle setzen. Sind jedoch drei Personen und drei Stühle gegeben, so sind es bereits sechs Möglichkeiten ($3 * 2$). Bei vier Personen erhalten wir zwölf Möglichkeiten ($4 * 3 * 2$).
Testen Sie Ihr Programm auch mit großen Eingabewerten.

Aufgabe 4.13 Zahlensumme (2)

Zeit:	0.5 h
Web-Code:	8thj ygxd

Berechnen Sie die folgenden Summen. Die Anwenderin soll die Zahlen m und n selbst eingeben können.
$S1 = 1^2 + 2^2 + 3^2 + 4^2 + \ldots + (n-1)^2 + n^2$
$S2 = 1^3 + 2^3 + 3^3 + 4^3 + \ldots + (n-1)^3 + n^3$
$S3 = m + (m+1) + (m+2) + \ldots + (n-2) + (n-1) + n$
Versuchen Sie, mit Hilfe des Internets, für jede Summe eine Formel zu finden.

Aufgabe 4.14 Sinusfunktion

Zeit:	1 h
Web-Code:	g5cx nzxg

Schreiben Sie ein Programm, das die Sinusfunktion im Bogenmaß annähert.

Verwenden Sie für sin(x) die Näherungsformel
$\sin(x) \approx x - x^3/3! + x^5/5! - x^7/7! + \ldots - \ldots$
Dabei bezeichnet n! die Fakultätsfunktion $n! = 1 * 2 * 3 * \ldots * n$.
Wenn Sie die jeweiligen Potenzen und Fakultäten aus denjenigen der vorangehenden Summanden berechnen, sparen Sie ein Vielfaches an Rechenleistung.

Aufgabe 4.15 Zahlenspiel

Zeit:	1 h
Web-Code:	vg42 rghx

In einem Spiel werden die Zahlen von 1 bis n durchgezählt. Man kann nun folgende Feststellung machen: Die 12. Zahl, die mit den Ziffern 253 beginnt, ist 25300; denn 253 ist die 1., 2530 die 2., 2531 die 3., 2539 die 11., und 25300 die 12. Zahl.
Schreiben Sie eine Programm, das die 1000. Zahl ermittelt, die mit den Ziffern 9876 beginnt.
Zusatzaufgabe: Verallgemeinern Sie die Aufgabe (die n-te Zahl, die mit vorgegebenen Ziffern beginnt).

Aufgabe 4.16 Mondlandung

Zeit:	0.5 h
Web-Code:	y4qj nv2g

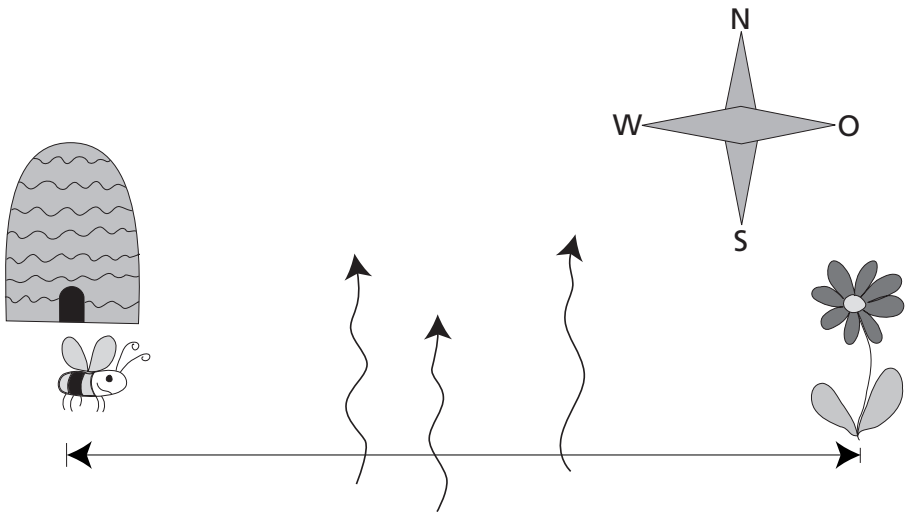
Für ein Computerspiel (Mondlandung) wird die Höhe einer fallenden Rakete nach folgender vereinfachter Formel (Annäherung mit endlich vielen Zeitschritten) berechnet:
Gegeben sind die Geschwindigkeit v [in Pixeln pro Zeiteinheit] und eine Höhe h über der Mondoberfläche [in Pixeln]. Nach jeder Zeiteinheit verändern sich v und h nach den folgenden Formeln:

- v := v + 1.5
- h := h – v

Starten Sie das Programm mit der Starthöhe h_0 = 800 [Pixel] und der Startgeschwindigkeit v_0 = 0. Wie viele Zeiteinheiten dauert die Landung? Geben Sie für jede Zeiteinheit v und h aus. Wie groß ist v beim Aufprall?

Aufgabe 4.17 **Bienenflug**

Zeit:	2 h
Web-Code:	jgjj c8uv



Eine Biene fliegt von ihrem Stock aus auf eine 50 m entfernte Blume zu, wobei sie eine Geschwindigkeit von 1.5 m/s hat. Die Blume befindet sich im Osten des Bienenstocks. Nach Norden weht der Südwind mit 60 cm/s. Die Biene reagiert darauf, indem sie ihren Kurs immer wieder neu auf die Blume ausrichtet.

a) Bestimmen Sie ihre Flugbahn (x- und y-Koordinaten), indem Sie die Position der Biene für jede Sekunde wieder neu berechnen.
b) Variieren Sie die Zeit zwischen zwei Neuberechnungen der Bienenposition.
c) Stellen Sie die Flugbahn grafisch dar.

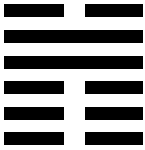
Aufgabe 4.18 Das Buch der Weissagungen

Zeit:	1 h
Web-Code:	95ew 4rif

Die Chinesische Weissagung I-Ging (I-Ching) bedient sich zweier Muster

- ——— = Yang und
- – – = Yin,

die an 6 Plätzen auftreten können. Dadurch ergeben sich 64 sog. Hexagramme, von denen jedem eine bestimmte Bedeutung zukommt. So bedeutet zum Beispiel das folgende Symbol «Die Sammlung»:



Schreiben Sie ein Programm, das alle 64 möglichen I-Ging-Hexagramme ausgibt. Beispiel in Textform:

```
– –   – –
– – – – –
– – – – –
– –   – –
– –   – –
– –   – –
```

Aufgabe 4.19 Parallelschaltung Ohm'scher Widerstände

Zeit:	1 h
Web-Code:	eisv 86s7

Werden Ohm'sche (elektrische) Widerstände (R_1, R_2, R_3, R_4, …) parallel geschaltet, wird der Gesamtwiderstand R durch die folgende Formel berechnet:

$$R = \frac{1}{\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}+\ldots}$$

oder

$R = 1/R_{inv}$; $R_{inv} = 1/R_1 + 1/R_2 + 1/R_3 + 1/R_4 + \ldots$

Zunächst soll die Anwenderin eingeben können, wie viele Widerstände insgesamt zu erfassen sind. Danach werden alle Widerstände sukzessive eingelesen und die Teilsumme $R_{inv} := 1/R_1$, $R_{inv} := R_{inv} + 1/R_2$, $R_{inv} := R_{inv} + 1/R_3$, … jedes Mal um den neuen Kehrwert ergänzt. Zum Schluss wird der Anwenderin $R := 1/R_{inv}$ (der Gesamtwiderstand) ausgegeben.
Prüfen Sie Ihre Lösung mit den Zahlen im Verifikationsteil.

Aufgabe 4.20 Anzahl Ziffern

Zeit:	1 h
Web-Code:	qgf8 hnro

Schreiben Sie ein Programm, das die Anzahl Ziffern einer Zahl ermittelt. In diesem Kontext hat die Zahl 0 eine Ziffer. Die Zahl 2023 hat 4 Ziffern. Die Zahl 00343 hat nur drei Ziffern, denn die beiden (führenden) Nullen werden nicht dazugezählt.
Erweitern Sie das Programm derart, dass die Ziffernanzahl aller Zahlen von 1 bis 1024 ausgegeben wird.

Aufgabe 4.21 Ziffernfolge umdrehen

Zeit:	1 h
Web-Code:	6s3s xoyx

Schreiben Sie ein Programm, das von einer gegebenen ganzen Zahl (`integer`) ihre Umkehrung ausgibt. Aus 2454 wird also 4542. Aus 20010 wird 1002 (= 01002 die neue führende Null wird abgeschnitten).

Aufgabe 4.22 **FizzBuzz (1)**

Zeit:	0.5 h
Web-Code:	pf6d wqtr

Schreiben Sie ein Programm, das die Zahlen von 1 bis 100 ausgibt. Bei jeder Zahl, die durch 5 teilbar ist, soll «fizz» ausgegeben werden und bei jeder Zahl, die Zahl durch 7 teilbar ist, soll «buzz» ausgegeben werden. Wenn die Zahl sowohl durch 7 als auch durch 5 teilbar ist, soll «fizzbuzz» ausgegeben werden. Der Modulo-Operator ermittelt den Rest bei Division. Somit ist eine Teilbarkeit einfach dann erreicht, wenn die Modulo-Operation (`%`, `MOD`) den Rest 0 liefert.

Aufgabe 4.23 **FizzBuzz (2)**

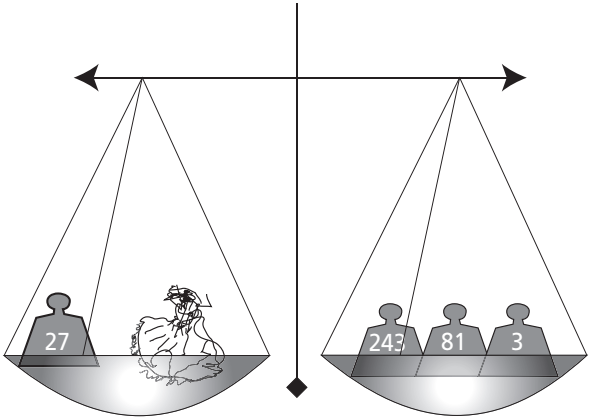
Zeit:	0.5 h
Web-Code:	8sax at2x

Diese Version von FizzBuzz soll wieder alle Zahlen von 1 bis 100 ausgeben. Bei jeder Zahl, die eine Ziffer 7 **enthält**, soll «fizz» ausgegeben werden. Wenn die Zahl durch 7 teilbar ist, soll «buzz» ausgegeben werden. Wenn die Zahl sowohl durch 7 teilbar ist als auch eine Ziffer 7 enthält, soll «fizzbuzz» ausgegeben werden.

Aufgabe 4.24 **Waage**

Zeit:	1 h
Web-Code:	d57z se4g

Ein Kaufmann besitzt eine Waage mit zwei Waagschalen und sieben Gewichtssteine: 1 kg, 2 kg, 4 kg, 8 kg, 16 kg, 32 kg und 64 kg. Auf die eine Schale legt der Kaufmann jeweils die Ware, während er auf die andere Schale die Gewichtssteine hinstellt. Er ist mit seinen sieben Gewichtssteinen in der Lage, alle Gewichte von einem Kilo bis zu 127 kg abzuwägen.



Schreiben Sie ein Programm, das ein Gewicht (0 kg … 127 kg) entgegennimmt und auf der Konsole ausgibt, welche Gewichtssteine zu verwenden sind.
Tipp: Binärsystem.
Zusatzaufgabe: Der Kaufmann hat seine sieben Gewichtssteine durch die folgenden sechs Gewichte eingetauscht: 1 kg, 3 kg, 9 kg, 27 kg, 81 kg und 243 kg. Er hat vor, Gewichtssteine auf beide Seiten der Waage zu stellen, und hat herausgefunden, dass er nun sogar alle Gewichte von einem Kilo bis zu 364 kg abwägen kann. Die Ware legt er jeweils auf die linke Seite. Ihr Programm soll nun für jedes Gewicht (0 kg … 364 kg) angeben, welche Steine rechts und welche links aufzustellen sind.
Beispiel zur Zusatzaufgabe: Die Ware wiegt 300 kg. Er muss rechts die Steine 243 kg, 81 kg und 3 kg hinstellen, während er links das Gewicht mit dem 27-kg-Stein auf 327 kg erhöht.
Tipp zur Zusatzaufgabe: Viele Strategien führen zum Ziel.
Beispiel:
Zunächst sollte Ihr Warengewicht durch 3 kg teilbar sein. Ist dies nicht der Fall, so korrigieren Sie das Gewicht mit dem 1-kg-Stein (war z. B. das Warengewicht 13 kg, so kommt der 1-kg-Stein nach rechts, und das Warengewicht wird damit auf 12 kg korrigiert). Mit dem 3-kg-Stein korrigieren Sie das Warengewicht weiter, sodass es durch 9 kg teilbar wird, usw.

5. Unterprogramme

Ein **Unterprogramm** (engl. *Function, Subroutine, Procedure* oder Methode) ist eine mit einem Namen versehene Sequenz. Diese Sequenz kann (mittels ihres Namens) als Anweisung an jeder beliebigen Stelle auftauchen. Programme lassen sich so in einfachere Teilprogramme mit klarer Funktionalität unterteilen. Die Vorgehensweise, ein Programm in immer kleinere Teile aufzuteilen, ist auch unter den Begriffen *Top-Down,* «Schrittweise Verfeinerung» oder «funktionale Dekomposition» bekannt.
Bereits im Kapitel über Anweisungen (s. 2.1, S. 25) sind uns Unterprogramme begegnet. Wir hatten dort vorgegebene Systemfunktionen aufgerufen (`sqrt()`, `print()`).
Unterprogramme lösen zwar keine neuen Problemstellungen, da man nämlich an jeder Stelle des Unterprogramm-Aufrufs die Unterprogramm-Anweisungen stellen könnte. Aber Unterprogramme vereinfachen die Programmierung wesentlich.

5.1 Parameter und Rückgabewerte

Dem Unterprogramm können in Form von **Parametern** Werte (aber auch Variable[1]) mitgegeben werden. Ein Unterprogramm kann dem aufrufenden Programm zudem berechnete Werte zurückgeben. Gibt es einen Wert zurück, so sprechen wir auch von **Funktion**. In diesem Fall stehen die Aufrufe von Subroutinen nicht anstelle einer Anweisung, sondern anstelle eines Ausdrucks[2].

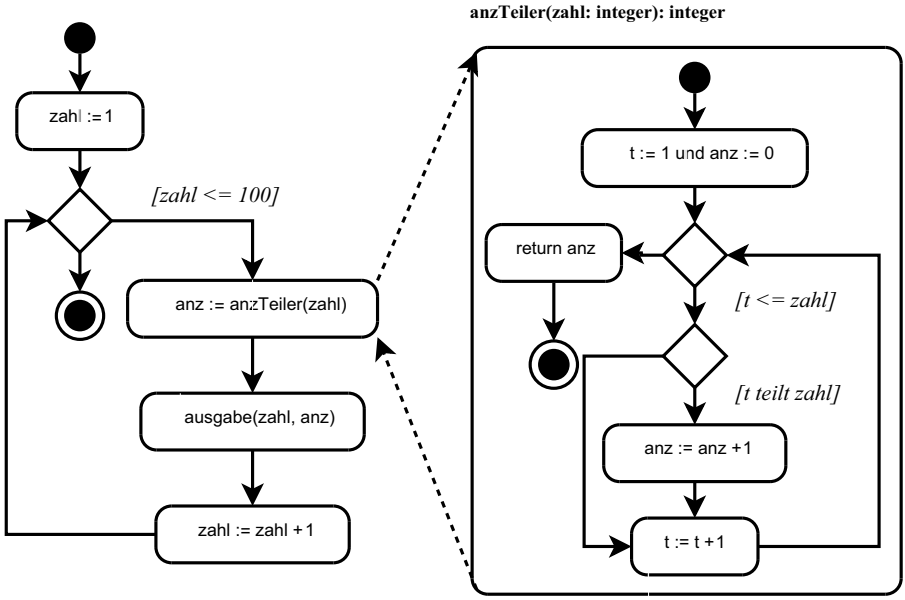
5.2 Beispiel

Beispiel: Das Unterprogramm `anzTeiler` (rechter Teil der Grafik) zählt die Anzahl Teiler einer gegebenen Zahl. Das Hauptprogramm links gibt von den ersten hundert Zahlen die Anzahl ihrer Teiler aus.

---

1 Bei Wertübergabe spricht man von «call by value», bei Variablenübergabe spricht man von «call by reference».

2 Im Unterschied zu Funktionen in der Mathematik muss der Rückgabewert nicht unbedingt eine Zahl sein. Ein möglicher Rückgabewert kann ein beliebiger Datentyp sein.



Aufrufendes Programm:

```
zahl := 1
while(zahl <= 100) {
  anz  := anzTeiler(zahl)
  ausgabe(zahl, anz)
  zahl := zahl + 1
}
```

Unterprogramm:

```
anzTeiler(zahl: integer): integer {
  anz := 0
  t   := 1
  while(t <= zahl){
      if(0 == zahl MOD t) {
        anz := anz + 1
      }
      t := t + 1
  }
  return anz
}
```

5.3 Lokale und globale Variable

Innerhalb von Unterprogrammen existiert i. d. R. ein eigenständiger Speicherbereich (Stapel bzw. *stack*). Damit ist es möglich, Variable zu deklarieren, die nur während der Ausführung des Unterprogramms Gültigkeit haben. Diese Variable werden **lokale Variable** genannt. Sie können dieselben Namen tragen wie Variable im aufrufenden Programm, ohne mit diesen in Konflikt zu geraten. Sobald zum aufrufenden Programm zurückgekehrt wird, verlieren lokale Variable ihre Werte, d. h., die lokalen Variable können nur innerhalb der entsprechenden Unterprogramme verwendet werden. Damit ein Unterprogramm auch Werte im Hauptprogramm verändern kann, verwendet man entweder **globale Variable**, die aus allen Programmteilen sichtbar sind, oder man übergibt den Unterprogrammen Referenzen (Speicheradressen) auf Variable, die im aufrufenden (Haupt-)Programm deklariert sind (Referenzvariable oder «variable Parameter» genannt).

5.4 Aufgaben

Aufgabe 5.1	**Ungerade Zahlen**
Zeit:	0.5 h
Web-Code:	jsju e6xd

Programmieren Sie die Funktion `istUngerade()`.
Eingabe: Eine ganze positive Zahl.
Ausgabe: `true`, falls die Zahl ungerade ist, und `false`, falls die Zahl gerade ist.
Schreiben Sie zudem ein Hauptprogramm, das die Benutzerin zuerst nach einer positiven ganzen Zahl fragt.

Aufgabe 5.2 **RGB (1)**

Zeit:	0.5 h
Web-Code:	829e g8uz

Schreiben Sie eine Funktion, die drei Zahlen (`real`) zwischen 0.0 und 1.0 entgegennimmt und eine RGB-kodierte Farbe (RGB = **R**ot, **G**rün, **B**lau) in 24 Bits in einem `integer` zurückgibt. Hier der geforderte Prototyp (auch Funktionsprototyp, Signatur, Funktionsdeklaration oder Funktionskopf genannt):

```
rgb(rot: real, gruen: real, blau: real): integer
```

Schreiben Sie dazu zunächst eine Funktion, die einen Wert zwischen 0.0 und 1.0 in eine `integer`-Zahl von 0 bis 255 verwandelt:

```
integer zahl := (integer) (wert * 256)
```

Beachten Sie, dass der Wert **zwischen** 0.0 und 1.0 liegt, dass speziell die Zahl Eins (1.0) nicht vorkommt. Dank dem Abrunden ist somit im Resultat tatsächlich 255 die größtmögliche Zahl. Nun ist es nötig, die drei Resultate in demselben `integer` abzulegen. Das erreichen wir, indem wir den Rotanteil mit 256^2 und den Grünanteil mit 256 multiplizieren; den Blauanteil können wir zum Schluss einfach addieren.

Aufgabe 5.3 **RGB (2)**

Zeit:	0.5 h
Web-Code:	4zn2 dxzd

Schreiben Sie die Umkehrung der obigen Funktion RGB (1):
Gegeben ist eine RGB-kodierte Ganzzahl (`integer` à 32 Bits):

```
0000'0000'rrrr'rrrr'gggg'gggg'bbbb'bbbb
```

Den Blauanteil erhalten Sie einfach, indem Sie das Bitmuster mit der Hexadezimalzahl «000000FF» maskieren, d. h. bitweise mit dem UND-Operator ver-

knüpfen. Um den Grün- bzw. den Rotwert zu erhalten, muss eine Division oder eine Bit-Verschiebung vorgenommen werden. Am Schluss sind die drei Werte lediglich durch 256 zu dividieren, um eine Zahl zwischen 0.0 und 1.0 zu erhalten.
Hier die geforderten Funktionsprototypen:

```
r(rgb: integer): real
g(rgb: integer): real
b(rgb: integer): real
```

Bemerkung: Im Gegensatz zur vorangehenden Aufgabe kann der Wert 0.0 nun effektiv als Resultat auftreten. Tatsächlich wären in der Aufgabe RGB (1) Eingabewerte im Intervall [0.0 … 1.0) möglich.

Aufgabe 5.4	**Quersumme**
Zeit:	1 h
Web-Code:	wvfg ndmx

Schreiben Sie eine Funktion, die von einer vorgegebenen natürlichen Zahl die Quersumme bestimmt. Natürliche Zahlen sind positive ganze Zahlen (1, 2, 3, 4, 5, …).

Aufgabe 5.5	**Abstand**
Zeit:	0.25 h
Web-Code:	tybn j6zi

Um die Länge der Luftlinie zwischen zwei Punkten auf der Landkarte zu berechnen, wird ein Programm benötigt. Schreiben Sie eine Subroutine, um den Abstand zweier Punkte in der Ebene zu berechnen.
P1 = (x1, y1), P2 = (x2, y2).

Berechnen Sie damit unter anderem den (Kilometer-)Abstand der Orte Zell (704.4, 256.2) und Neuburg (693.3, 261.4), die hier in Schweizerischen Landeskoordinaten (CH1903) gegeben sind.

Aufgabe 5.6 **Dreiecksflächen**

Zeit:	0.5 h
Web-Code:	4hpi tuo2

Schreiben Sie die folgenden Funktionen zur Berechnung der Dreiecksfläche und verwenden Sie diese als Methoden in einem Hauptprogramm:

- `flaeche(s: real, hs: real)`
  Aus Grundseite s und der zugehörigen Höhe h_s:
  Fläche = $\frac{s \times h_s}{2}$
- `flaeche(a: real, b: real, c: real)`
  Aus drei Seiten nach der Formel von Heron:
  Fläche $= \sqrt{s(s-a)(s-b)(s-c)}$
  Dabei bezeichnet s den halben Umfang. Schreiben Sie eine Hilfsmethode `umfang()`, die zunächst den Umfang des Dreiecks berechnet.
- `flaeche(ax: real, ay: real, bx: real, by: real, cx: real, cy: real)`
  Aus drei Eckpunkten. Dabei können die Seitenlängen einfach mit dem Satz des Pythagoras berechnet werden. Die Seite a z. B. wird durch
  $a = \sqrt{(c_x - b_x)^2 + (c_y - b_y)^2}$
  ermittelt. Verwenden Sie nach der Ermittlung der drei Seiten die bereits geschriebene Funktion `flaeche(a, b, c)`.

Aufgabe 5.7 Geometrie

Zeit:	0.5 h
Web-Code:	ma5m oj6k

Programmieren Sie die folgenden Formeln der Geometrie als Subroutinen. Versuchen Sie, bereits geschriebene Subroutinen so oft wie möglich wieder zu verwenden.

- **Kreisumfang** U = 2 r π. Dabei ist der Kreisradius r gegeben und π die Kreiszahl 3.1416…
- Berechnen Sie die **Kreisfläche** A bei gegebenem Radius r. ($A = r^2\pi$)
- Berechnen Sie die **Oberfläche einer Blechdose** bei gegebenem Radius r und gegebener Höhe h. Oberfläche:

```
oberflaeche(r, h) := deckel(r) + boden(r) + mantel(r, h)
```

Aufgabe 5.8 ggT nach Euklid

Zeit:	1 h
Web-Code:	7k49 32v7

Berechnen Sie den größten gemeinsamen Teiler (ggT) zweier Zahlen nach Euklid[1] mittels Unterprogramm. Der *ggT* ist die größte Zahl, durch die sowohl A als auch B ohne Rest teilbar sind. Das Verfahren ist relativ einfach. Es wird wie folgt vorgegangen:

1. R wird zum Divisionsrest aus A / B. (Falls B > A, so wird R einfach zu A.)
2. Ist R = 0, so ist B der ggT, und das Unterprogramm wird beendet.
3. Der Variable A wird der Wert von B zugewiesen, und der Variable B wird R zugewiesen.
4. Zurück zu 1.

[1] Euklid von Alexandria (um 360 v. Chr. – 280 v. Chr.)

Aufgabe 5.9 **Exponenten**

Zeit:	0.5 h
Web-Code:	6i5m xtvw

Schreiben Sie eine Funktion, die eine Zahl a potenziert (a^b), ohne dabei die Mathematik-Methoden Ihrer Programmiersprache zu verwenden (insbesondere keine Exponenten- und Potenzfunktionen). Dabei ist a eine reelle Zahl (`real`) und b eine ganze positive Zahl (`integer` mit `b >= 0`).

Aufgabe 5.10 **Bremsweg**

Zeit:	1 h
Web-Code:	dh44 7qtd

Berechnen Sie die **Anhaltestrecke** eines Autos bei gegebener Geschwindigkeit [km/h] für trockene bzw. nasse Straße. In eckigen Klammern stehen die Maßeinheiten, die jedoch für die Programmierung nicht von Bedeutung sind. Für die Anhaltestrecke gelten folgende Gleichungen:

- Die Geschwindigkeit [km/h] kann vom Anwender eingegeben werden.
- Geschwindigkeit [m/s] = Geschwindigkeit [km/h] / 3.6
  (z. B. 72 [km/h] = 20 [m/s])
- Die Bremsbeschleunigung (auch Bremsverzögerung genannt) wird in Metern pro Quadratsekunden gemessen. Trockene Straße: Bremsbeschleunigung = 7 [m/s^2]. Nasse Straße: Bremsbeschleunigung = 4 [m/s^2]. Wer das Programm benutzt, kann dabei «t» für die trockene bzw. «n» für die nasse Straße eingeben.
- Reaktionszeit [s] = 1.44
- Reaktionsweg [m] = Geschwindigkeit [m/s] * Reaktionszeit [s]
- Bremsweg [m] = $Geschwindigkeit^2$ [$(m/s)^2$] / (2 * Bremsbeschleunigung [m/s^2])
- Anhaltestrecke [m] = Reaktionsweg [m] + Bremsweg [m]

Programmieren Sie möglichst viele der benötigten Größen als Funktionen mit sinnvollen Eingabeparametern. Beispiele:

- `reaktionsweg()`
- `bremsweg()`
- `anhaltestrecke()`
- `geschwindigkeitInMeterProSekunde()`

Zusatzaufgabe (Auffahrunfall): Wie viel Abstand (in Metern und in Sekunden) zum vor ihm fahrenden Wagen muss ein Fahrer einhalten, um einen Auffahrunfall zu vermeiden, falls der vor ihm Fahrende augenblicklich stillsteht?
Dazu ist zusätzlich die Bremszeit zu berechnen:

- Bremszeit [s] = Geschwindigkeit [m/s] / Bremsbeschleunigung [m/s^2]

Aufgabe 5.11 Schaltjahre

Zeit:	0.25 h
Web-Code:	4xw9 mhjm

Schreiben Sie ein Unterprogramm, das von einer gegebenen Jahrzahl ermittelt, ob es sich um ein Schaltjahr handelt. Dabei sind alle durch 4 teilbaren Jahre sog. Schaltjahre mit einem im Februar angefügten Schalttag.
Seit der Einführung des Gregorianischen Kalenders im Jahr 1582 gilt zusätzlich folgende Regel: In allen durch 100 teilbaren Jahren wird kein Schalttag eingefügt, es sei denn, das Jahr ist durch 400 teilbar.

Aufgabe 5.12 Plausibilität eines Kalenderdatums

Zeit:	2 h
Web-Code:	fa38 e9da

Prüfen Sie, ob eine Eingabe zu einem Kalenderdatum sinnvoll ist. Dazu werden Jahr, Monat und Tag eingegeben. Prüfen Sie, ob das Datum existiert. Geben Sie

als Resultat einen Boole'schen Wert (`true` / `false`) aus. Verwenden Sie für Daten ≤ Donnerstag, 4. Oktober 1582, den Julianischen Kalender. Verwenden Sie für Daten ≥ Freitag, 15. Oktober 1582, den Gregorianischen Kalender. Auf den 4. Oktober 1582 folgte nach Papst Gregors Kalender direkt der 15. Oktober 1582.
Verwenden Sie das Unterprogramm aus Aufgabe 5.11.

Aufgabe 5.13 Ewiger Kalender

Zeit:	2 h
Web-Code:	m2g9 636g

Schreiben Sie ein Programm, das von einem gegebenen Datum im 21. Jahrhundert (vom 1. 1. 2001 bis 31. 12. 2100) den Wochentag bestimmt. Dabei muss Folgendes beachtet werden:

- Der 1. 1. 2001 war ein Montag.
- Jedes durch 4 teilbare Jahr (in diesem Bereich) ist ein Schaltjahr mit 366 (statt 365) Tagen. Ausnahme: Das Jahr 2100 ist kein Schaltjahr!

Einfach zum Ziel kommen Sie, wenn Sie zunächst alle Jahre mit 365 Tagen, dann die Schalttage, weiter die Monate und zuletzt die Tage im Monat behandeln. Verwenden Sie die Subroutine aus Aufgabe 5.11.
Zusatzaufgabe: Erweitern Sie das Programm so, dass es für alle Daten im Gregorianischen Kalender (ab 15. 10. 1582) funktioniert.

Aufgabe 5.14 Zahlenmuster

Zeit:	1 h
Web-Code:	n8f6 73nj

Programmieren Sie ein Programm, welches das folgende Zahlenmuster ausgibt: 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, … Die Zahl **x** kommt also **x** Mal vor.

Zusatzaufgabe: Schreiben Sie eine Methode `muster(integer)`, bei welcher der Index (= Position) eingegeben werden kann und dabei die Zahl **x** berechnet wird. Beispiele: `muster(1)` → 1, `muster(6)` → 3, `muster(7)` → 4. Berechnen Sie damit `muster(1'000)`, `muster(10'000)` bzw. `muster(1'000'000'000)`. Wer schafft die größten Zahlen, wer die schnellste Berechnung?
Tipp zur Zusatzaufgabe: Führen Sie im ursprünglichen Muster neben der Zahl auch den Index wie folgt mit: (1.:) 1, (2.:) 2, (3.:) 2, (4.:) 3, (5.:) 3, (6.:) 3, (7.:) 4, …

Aufgabe 5.15 Frankengewinnspiel

Zeit:	2–4 h
Web-Code:	qnfv rirb

Das Spielfeld des Frankengewinnspiels besteht aus beliebig vielen, ab Eins (1) nummerierten Feldern. Auf jedem Feld gibt es einen Gewinn, welcher entsprechend der Feldnummer ist. Auf dem Feld 8 liegen 8.– Fr. usw. Der Betrag von jedem Feld, das man besucht, darf als Gewinn mitgenommen werden.
Die Spielvorschrift besteht aus den folgenden Regeln:

1. Wählen Sie ein Feld zwischen 1 und 99
2. Nehmen Sie das Geld und gehen Sie nach folgender Vorschrift zum nächsten Feld:
   - `Feldnummer` ungerade? Feld ← Feld * 3 + 1
   - `Feldnummer` gerade? Feld ← Feld / 2

Wiederholen Sie Schritt 2, bis Sie auf dem Feld 1 landen.

a) Schreiben Sie ein Unterprogramm, welches zu einem gewählten Startwert den Gewinn ausgibt.
b) Schreiben Sie ein Programm, welches die Gewinne für alle 99 Startmöglichkeiten ausgibt.
c) Was ist der maximale Gewinn dieses Spiels?
d) Wie heißt die höchste erreichte Feldnummer, wenn Sie den Startwert frei wählen? Ergänzen Sie dazu das Programm aus Aufgabe c).

Hinweis: Die Spielvorschrift dieser Aufgabe ist als Collatz-Folge[1] oder Ulam-Sequenz[2] bekannt.

Aufgabe 5.16 Parameter vs. globale Variable

Zeit:	0.5 h
Web-Code:	58wm 7x5h

Schreiben Sie die folgenden Methoden `top()`, `f()` und `g()` so um, dass die Variable `x` über einen Parameter mit- und via `return` zurückgegeben wird. `x` soll nicht mehr als globale Variable (resp. Klassenvariable oder Datenfeld) auftreten!

```
x: integer
top()
{
  f()
  g()
}

f()
{
  x := 7
}

g()
{
  print("x = " + x)
}
```

[1] Lothar Collatz (1910 – 1990)

[2] Stanislaw Ulam (1909 – 1984)

Aufgabe 5.17 Codeverbesserungen (Refactoring)

Zeit:	1 h
Web-Code:	f32t c578

Diese Aufgabe hat ihren Ursprung in der Programmierung von grafischen Benutzerschnittstellen (GUI = Graphical User Interface). Doch selbst ohne die hier verwendeten Funktionen und Variablen zu kennen(!), sind Sie in der Lage, den Code zu verbessern (engl. *refactoring*): Fügen Sie eine weitere Subroutine ein, die Sie vier Mal aufrufen.
(Diese Aufgabe soll ohne Computer gelöst werden.)

```
init()
{
  butt1 := makeButton("OK")
  setCommand(butt1, "OK")
  add(butt1, panel)
  addListener(butt1, myListener)
  butt2 := makeButton("Abbruch")
  setCommand(butt2, "CANCEL")
  add(butt2, panel)
  addListener(butt2, myListener)
  butt3 := makeButton("Zurück")
  setCommand(butt3, "BACK")
  add(butt3, panel)
  addListener(butt3, myListener)
  butt4 := makeButton("Weiter")
  setCommand(butt4, "FORWARD")
  add(butt4, panel)
  addListener(butt4, myListener)
}
```

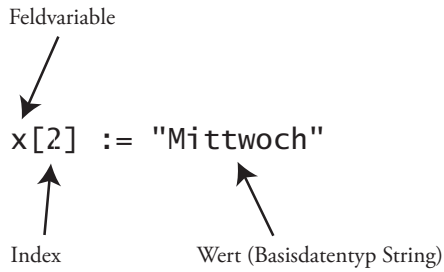
6. Felder

In der Praxis hat man es oft mit sehr vielen Variablen desselben Datentyps zu tun (81 Sudokufelder, 52 Pokerkarten, 12 Monate, 8×8 Schachfelder, 7 Wochentagsnamen, 19×19 Punkte im Go-Spiel, …). Um solche Werte zu speichern, bietet sich das Konzept zusammenhängender Speicherstellen an: **Felder** (bzw. engl. *arrays*).

In diesem Kapitel verwenden wir die Notation `x : Typ[]`, um anzugeben, dass es sich bei `x` um eine Feldvariable zum Basistypen `Typ` handelt. Auf die Elemente des Feldes wird mit einem Index[1] zugegriffen, indem der Index in Klammern[2] hinter die Variable gestellt wird.

Index	**x[0]**	**x[1]**	**x[2]**	**x[3]**	**x[4]**	**x[5]**	**x[6]**
Wert	Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag	Sonntag

```
x: string[]

x[0] := "Montag"
x[1] := "Dienstag"
```



[1] Der erste Index ist in den gebräuchlichen Programmiersprachen die Zahl Null [0].

[2] Viele Programmiersprachen verwenden eckige Klammern [ ], aber auch runde Klammern oder andere Symbole sind gebräuchlich.

6.1 Schleifen

Felder kommen häufig in Zusammenhang mit Schleifen vor – sei dies, um alle Felder zu füllen oder um jedes Element des Feldes zu behandeln. Beispiele:

Daten in Feld einlesen	Daten aus Feld behandeln

`pos : integer`
`pos := 1`
`while(NOT dateiEnde())`
`{`
`  array[pos] := einlesen()`
`  pos := pos + 1`
`}`
	`array: Typ[]`
	`array := ...`
	`for(element: Typ in array)`
	`{`
	`  behandle(element)`
	`}`

6.2 Tabellen

Zweidimensionale Felder werden auch **Tabellen** genannt[1]. Tabellen werden mit zwei Indizes (Zeile, Spalte) angesprochen. Beispiel: Umsatztabelle nach Mitarbeiter und Quartal.

	Quartal 1	Quartal 2	Quartal 3	Quartal 4
Meier	48 000	36 000	52 000	68 000
Freimann	12 000	11 000	8 000	24 000
Huber	50 000	51 000	53 000	46 000
Müller	5 500	5 500	3 800	2 200
Guggisberg	38 000	36 000	48 000	54 000

Eine mögliche Zuweisung in obiger Tabelle sieht wie folgt aus:

```
umsaetze["q2"]["Huber"] := 51000
```

[1] In Zusammenhang mit Zahlentabellen, auf welchen mathematische Berechnungen durchgeführt werden, spricht man auch von **Matrizen** (Einzahl: Matrix) anstelle von Tabellen.

6.3 Aufgaben

Aufgabe 6.1 Feldelemente aufsummieren

Zeit:	0.25 h
Web-Code:	pgez dr6b

Füllen Sie ein Feld (Array) mit den Zahlen:
2, 17, 10, 9, 16, 3, 9, 16, 5, 1, 17, 14.
Schreiben Sie nun zwei Funktionen, welche die Summe und das Produkt aller Zahlen aus diesem Array ermittelt. Testen Sie Ihr Programm auch mit anderen Zahlen.

```
summe  (feld: integer[]): integer
produkt(feld: integer[]): integer
```

Aufgabe 6.2 Feld elementweise bearbeiten

Zeit:	0.25 h
Web-Code:	vkei ya8q

Füllen Sie ein Feld mit den Zahlen:
2, 17, 10, 9, 16, 3, 9, 16, 5, 1, 17, 14.
Schreiben Sie nun ein Programm, das aus einem (im Testfall obigem) Feld ein neues Feld erzeugt, bei dem jeder Wert um 10 erhöht ist.

```
plusZehn(original: integer[]): integer[]
```

Aufgabe 6.3 Feld filtern (1)

Zeit:	1 h
Web-Code:	zxku 89hw

Füllen Sie ein Feld mit den Zahlen:
2, 17, 10, 9, 16, 3, 9, 16, 5, 1, 17, 14.
Schreiben Sie nun ein Programm, das aus dem obigem Feld ein neues Feld erzeugt, das nur noch die Zahlen enthält, die **größer als 10** sind. Der Prototyp der Funktion soll wie folgt aussehen:

```
filter(original: integer[]) : integer[]
```

Aufgabe 6.4 Feld filtern (2)

Zeit:	1 h
Web-Code:	3hiq vtfi

Füllen Sie ein Feld mit den Zahlen:
2, 17, 10, 9, 16, 3, 9, 16, 5, 1, 17, 14.
Schreiben Sie nun ein Programm, das aus dem obigem Feld ein neues Feld erzeugt, das **das größte und das kleinste Element eliminiert.** Der Prototyp der Funktion soll wie folgt aussehen:

```
elim(original: integer[]): integer[]
```

Zusatzaufgabe: Sollte das Maximum (bzw. Minimum) mehrfach auftreten, so sind alle Exemplare zu löschen. Im gegebenen Zahlenbeispiel würden die Zahl Eins (1) und beide Siebzehner (17) gelöscht.

Aufgabe 6.5 Feld filtern (3)

Zeit:	1 h
Web-Code:	gtmt n6b2

Füllen Sie ein Feld mit den Zahlen:
2, 17, 10, 9, 16, 3, 9, 16, 5, 1, 17, 14.
Schreiben Sie nun ein Programm, das aus dem obigem Feld ein neues Feld erzeugt und das **alle Duplikate eliminiert.** Der Prototyp der Funktion soll wie folgt aussehen:

```
unikate(original: integer[]): integer[]
```

Aufgabe 6.6 Felder umdrehen

Zeit:	0.5 h
Web-Code:	jxsv r4dw

Füllen Sie ein Feld mit den Zahlen:
2, 17, 10, 9, 16, 3, 9, 16, 5, 1, 17, 14.
Schreiben Sie nun ein Programm, das die Reihenfolge im Feld umkehrt. Schreiben Sie die Methode so, dass sie auch für eine ungerade Anzahl von Werten funktioniert. Prototyp:

```
umkehren(feld: integer[])
```

Falls Ihre Programmiersprache keine Variablen-Parameter (Referenzen) erlaubt, sieht der Prototyp wie folgt aus:

```
umkehren(original: integer[]) : integer[]
```

Aufgabe 6.7 Transponieren einer Matrix

Zeit:	1 h
Web-Code:	wmjo fn97

Schreiben Sie ein Programm, das die Einträge eines zweidimensonalen Arrays (Tabelle) an der Hauptdiagonalen (von links oben nach rechts unten) spiegelt. Im mathematischen Kontext nennt sich das Resultat die *transponierte* Matrix.

Aufgabe 6.8 Auswahlsortierung

Zeit:	1 h
Web-Code:	y45k 6qgz

Lesen Sie eine Reihe von ganzen Zahlen in ein Array ein.

a) Setzen Sie eine Indexvariable auf den ersten Index im Array.
b) Sortieren Sie wie folgt: Für jeden Index suchen Sie ab dort das Minimum im verbleibenden Teil ($\geq$ Index). Vertauschen Sie den gefunden Wert mit dem aktuellen Wert.
c) Erhöhen Sie den Index um 1.
d) Wenn Sie durch obiges Verfahren den zweitletzten Index erreicht haben, ist Ihr Array sortiert. Ansonsten zurück zu b).
e) Ausgabe des Resultates auf der Konsole.

Aufgabe 6.9 Schriftsetzer

Zeit:	1 h
Web-Code:	p8vk 9x7k

Ein Schriftsetzer musste bei einem Buch die Seitenzahlen mit *Lettern* drucken. Lettern sind spiegelverkehrte Schriftzeichen, die heute nur noch selten für den Buchdruck verwendet werden. Dafür musste unser Schriftsetzer wissen, wie viele

solcher Lettern er für die Seitennummerierung benötigt. Seitennummerierungen bestehen ausschließlich aus den Ziffern 0 bis 9. In dieser Aufgabe werden Lettern und Ziffern synonym verwendet. Ab Seite 5 waren die Seiten nummeriert. Die Seite 10 benötigt als Erste zwei Ziffern (nämlich die Ziffer «1» und die Ziffer «0»). Die Lettern konnten für verschiedene Seiten nicht rezykliert werden, da alle Seiten gleichzeitig für den Druck bereitstehen mussten. Schreiben Sie ein Programm, das für eine gegebene Anzahl Seiten die benötigten Anzahlen für jede Ziffer ausgibt. Da die Nummerierung erst bei 5 beginnt, hatte ein Buch mit 4 Seiten noch keine Ziffern. Ein Buch mit 11 Seiten benötigte bereits die Ziffern 5 (1x), 6 (1x), 7 (1x), 8 (1x), 9 (1x), 1 (3x) und 0 (1x).
Zur Programmierung verwenden Sie ein Array von [0] bis [9] (also mit 10 Einträgen), worin je die Anzahl der benötigten Ziffern steht. Die Anwenderin kann die letzte Seitennummer beliebig wählen. Zählen Sie in einer Schleife ab 5 bis zur letzten Seitennummer die benötigten Ziffern zu den Arraywerten dazu und geben Sie am Ende alle zehn Werte aus.

Aufgabe 6.10 n-ter Tag im Jahr

Zeit:	2 h
Web-Code:	gqsc 8u29

Initialisieren Sie ein Feld mit den folgenden zwölf Werten: 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31. Kurz, jeder Wert bezeichnet die Anzahl Tage eines Monats. Schreiben Sie eine Methode `tagNr(Tag, Monat): integer`, die von einem Tag ausgibt, um den wievielten Tag im Jahr es sich handelt. Gehen Sie vorerst nicht auf Schaltjahre ein. Schreiben Sie eine weitere Methode `tagNr(Tag, Monat, Jahr): integer`, die berücksichtigt, ob es sich um ein Schaltjahr handelt. Der erste Tag im Jahr ist der 1. Januar. Verwenden Sie wiederum die Funktion aus Aufgabe 5.11.

Aufgabe 6.11 Freitag, der 13.

Zeit:	2 – 4 h
Web-Code:	bomo xjx6

Zeigen Sie, dass der 13. eines Monats im Gregorianischen – also unserem – Kalender häufiger auf einen Freitag als auf jeden anderen Wochentag fällt.
Zeigen Sie zunächst, dass die Anzahl Tage innerhalb von 400 Jahren genau durch 7 teilbar ist. Somit reicht es, die Behauptung für 400 Jahre zu verifizieren. Erstellen Sie einen Array mit 7 Stellen (Montag bis Sonntag), wobei für jeden Tag anfänglich die Zahl 0 gespeichert wird. Dieser Array soll zeigen, wie häufig ein Wochentag am 13. eines Monats innerhalb von 400 Jahren vorkommt. Iterieren Sie nun vom 1. 1. 1600 bis zum 31. 12. 1999. Sie können entweder tageweise iterieren oder die Wochentagsfunktion aus Aufgabe 5.13 verwenden.
Tipp: Verwenden Sie auch wieder die Subroutine aus Aufgabe 5.11.
Geben Sie die 7 Anzahlen (Montag bis Sonntag) auf dem Schirm aus.

Aufgabe 6.12 Waschautomat

Zeit:	2 h
Web-Code:	vyyu vu3u

Für eine öffentliche Waschmaschine soll eine automatische Kasse programmiert werden. Dazu wird die folgende Methode benötigt:

```
wechselgeldInCent(
  vorhandeneMuenzenInCent:   integer[],
  waschPreisInCent:          integer ,
  eingeworfeneMuenzenInCent: integer[] ) : integer[]
{...}
```

Dabei sind die `vorhandenenMuenzenInCent` einfach diejenigen Münzen, die bereits vor der Wäsche im Apparat waren, um Wechselgeld herausgeben zu kön-

nen. Beachten Sie auch, dass nach der Kundeneingabe (`eingeworfeneMuenzenInCent`) neue Münzen im Automaten sind, die gleich wieder als `vorhandeneMuenzenInCent` verwendet werden können. Beispiele:

```
vorhandeneMuenzenInCent   := {100, 100, 200, 200, 500, 500, 500}
waschPreisInCent          := 400
eingeworfeneMuenzenInCent := {100, 100, 100, 500}

mögliche Ausgabe:           "{200, 200}"
```

Die geschweiften Klammern deklarieren hier Arrays und füllen diese sogleich mit Werten.

Aufgabe 6.13 Geldbetrag

Zeit:	1 h
Web-Code:	6juh 7y3z

Geben Sie von einem Geldbetrag (in Euro und Cent) die nötigen Noten und Münzen aus. In Euro existieren die folgenden Beträge:

- Münzen (in Cent): 1, 2, 5, 10, 20, 50
- Münzen und Noten (in Euro): 1, 2, 5, 10, 20, 50, 100, 200, 500

Beispiele:

```
betraege(53.26)
-> 50.00, 2.00, 1.00, 0.20, 0.05, 0.01
betraege(99.95)
 -> 50.00, 20.00, 20.00, 5.00, 2.00, 2.00, 0.50, 0.20, 0.20, 0.05
```

Bemerkung zu Rundungsfehlern: Vergleichen Sie gebrochene Zahlen (`real`) nicht auf Gleichheit «=» (bzw. ≥ und ≤). Anstelle von `if(x = y)` schreiben Sie für eine vorgegebene Rechengenauigkeit (z. B. `eps := 0.0001`)
`if(abs(x - y) < eps)`.

Aufgabe 6.14 Maximale Teilsumme

Zeit:	0.5 h
Web-Code:	sypi dsoh

Bei der Mustererkennung in Bildern musste Grenander[1] ein möglichst «helles» Rechteck ausschneiden (Anwendung z. B. Gesichtserkennung). Dabei ist er auf das folgende Problem gestoßen:
Ein **Segment** in einem Zahlenarray sei eine lückenlose Reihe benachbarter Elemente. Unter der **Teilsumme** einer Zahlenreihe versteht man die Summe aller Zahlen aus einem Segment dieser Reihe. Bei einer «maximalen Teilsumme» ist ein Segment zu suchen, dessen Teilsumme von keinem anderen Segment überstiegen werden kann.
Bestimmen Sie mittels Programm die maximale Teilsumme aus der folgenden Zahlenreihe:
–1, 2, –6, –14, 15, 12, –5, 0, 6, –6, 11, –4, –7, 1, –5, 1

Aufgabe 6.15 Tic-Tac-Toe

Zeit:	1 h
Web-Code:	imhb 6pb8

Schreiben Sie eine Subroutine, die eine 3 x 3-Matrix (Spielfeld) entgegennimmt und prüft, ob es sich dabei um eine Siegesposition im Tic-Tac-Toe-Spiel handelt. Die Matrix besteht aus neun ganzen Zahlen (`integer`), und die Werte sind Null (0), sofern an der entsprechenden Stelle noch kein Zug getätigt wurde. Steht jedoch an einer Stelle eine Eins (1) oder eine Zwei (2), so entspricht dies der Spielernummer. Die Funktion soll genau dann `true` zurückgeben, wenn ein Spieler in einer horizontalen, vertikalen oder diagonalen Reihe alle drei Felder mit seiner Nummer besetzt hat. Prototyp:

```
siegesPosition(ticTacToe: integer[][]) : boolean
```

[1] Ulf Grenander, Schwedischer Statistiker (*1923)

Aufgabe 6.16 Springer auf dem Schachbrett

Zeit:	2 h
Web-Code:	o9zn pwdy

Der Anwender wird vom Programm nach einer Position auf dem Schachbrett gefragt (z. B. «c1»). Ab dieser Startposition berechnet nun das Programm für jedes andere Feld (= Zielposition) die Anzahl Züge, die der Springer minimal benötigt, um diese Zielposition zu erreichen. Geben Sie das Resultat als Tabelle aus.
Bemerkung: An der Startposition des Springers steht bei der Ausgabe natürlich eine Null.

Aufgabe 6.17 Exponententabelle

Zeit:	1 h
Web-Code:	n2gz 2e3u

Schreiben Sie ein Programm, das 10 000 000 zufällige Potenzierungen mit Basis und Exponent von je 1 bis 100 durchführt (z. B. 15^{14} oder 41^{89}). Messen Sie die Zeit, die das Programm dafür benötigt.
Schreiben Sie ein zweites Programm, das zunächst eine quadratische Matrix mit allen Potenzresultaten von 1 bis 100 (als Basis und Exponent) füllt und danach 10 000 000 zufällige Potenzierungen mittels Nachschlagen in dieser Tabelle «löst».
Messen Sie die Zeit für das Füllen der Tabelle und das Nachschlagen und diskutieren Sie das Resultat.
Zusatzaufgabe: Ersetzen Sie schließlich die Potenzfunktion durch eine einfache Division und diskutieren Sie auch dieses Resultat.

Aufgabe 6.18 Gleichungssystem

Zeit:	2 h
Web-Code:	k43i 8vjt

Für 3D-Grafiken, Computerspiele und zur Lösung linearer Gleichungssysteme spielt die Matrixmultiplikation eine wesentliche Rolle. Lineare Gleichungssysteme haben die folgende Form:

$5a + 7b = -10$

$3a - 4b = -88$

Die obige Gleichung wird mit Matrizen wie folgt geschrieben:

$$\begin{pmatrix} 5 & 7 \\ 3 & -4 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} -10 \\ -88 \end{pmatrix}$$

Die Zahlen 5, 7, 3 und –4 werden in ein doppelt indiziertes Array geschrieben:

```
m[0][0] := 5; m[0][1] := 7; m[1][0] := 3; m[1][1] := -4;
```

Mit dieser Definition können die rechten Seiten (–10, –88) für verschiedene Zahlen a und b leicht nach folgendem Schema (s. Seite 86) errechnet werden:

```
mult(m: real[][]; ab: real[]): real[]
{
  xy: real[]
  xy := new real[2]
  row: integer
  row := 0
  while(row <= 1)
  {
    col: integer
    col := 0
    while(col <= 1)
    {
      xy[row] := xy[row] + m[row][col] * ab[col]
      col     := col + 1
    }
  row := row + 1
  }
  return xy
}
```

Rufen Sie obige Methode mit (a, b) = (–16, 10) auf.
Die rechte Seite sollte (x, y) = (–10, –88) lauten. Machen Sie von Hand zwei bis drei andere Vorausberechnungen, die Sie dann mit Ihrem Programm verifizieren.

7. Zeichenketten

Ein spezieller Typ eines Feldes sind die Zeichenketten *(Strings)*. Zeichenketten sind nichts anderes als Felder zum Basistypen `char`[1] oder Byte. Strings treten in den meisten Programmiersprachen selbst als Literale auf.
Beispiele:

```
vorname := "Hubert"
formel  := "a + f(x)"
```

Bemerkung: Die folgende Zuweisung (`bemerkung`) verwendet das Begrenzungszeichen für Zeichenketten (in diesem Fall ") innerhalb einer Zeichenkette. Um ein Begrenzungszeichen innerhalb einer Zeichenkette zu verwenden, muss es auf spezielle Art (in diesem Fall \") eingegeben werden:
`bemerkung := "Tippen Sie 'name := \"Bach\"'"`
Dadurch wird die Zeichenkette `Tippen Sie 'name := "Bach"'` zugewiesen.

7.1 Verarbeitung von Zeichenketten

Bevor Sie die Aufgaben lösen, sollten Sie wissen, welche Befehle in Ihrer Programmiersprache zur Verfügung stehen, um mit Zeichenketten zu arbeiten.

- Literal: Wie wird eine Zeichenkette in der Programmiersprache festgelegt? Wie sieht das Begrenzungszeichen aus und wie werden Sonderzeichen eingegeben?
- Variable: Wie wird eine Zeichenkette einer Variablen zugewiesen (Datentyp)?
- Ausgabe: Wie wird das Literal (bzw. die Variable) auf der Konsole ausgegeben?
- Einlesen: Wie werden Zeichenketten von der Konsole eingelesen (vom Anwender erfragt)?
- Länge: Wie viele Zeichen hat mein String? Was ist die Maximallänge für Zeichenketten?
- Zeichen ermitteln: Wie wird ein einzelnes Zeichen an einer vorgegebenen Position ermittelt?

[1] (s. 1.1, S. 15)

- Zusammenfügen: Wie werden zwei Strings zu einem neuen String verkettet?
- Kopieren: Wie kann ich eine Kopie (Klon) meiner Zeichenkette anfertigen?
- Suchen: Wie kann ich einen Teilstring in einem String suchen?
- Prüfen: Wie kann ich prüfen, ob mein String auf ein vorgegebenes Muster passt? (Beispiele: Endet mein String auf `.txt`? Enthält mein String nur Ziffern?)
- Extrahieren: Wie lässt sich ein Teil (Auszug) des Strings herauskopieren?
- Code des Zeichens: Wie kann ich ein Zeichen in die entsprechende Zahl der Kodierung umwandeln[1]?
- Zeichen des Codes: Wie kann ich eine Zahl in das entsprechende Zeichen der Kodierung umwandeln?
- Ziffern und Zahlen: Wie kann ich Zifferzeichen oder Zeichenketten bestehend aus Ziffern in die entsprechende Zahl verwandeln?
- Wie kann ich Zahlen (`integer`, `real`) in Strings verwandeln?

7.2 Zeichenketten aus Dateien

In der Praxis kommen Zeichenketten oft in Dateien (Files) vor. Für die Verarbeitung in Programmen werden Texte von Dateien gelesen oder in solche geschrieben. Für dieses Kapitel ist es von Vorteil, sich vorab mit dem Lesen und Schreiben von Zeichenketten in Dateien zu befassen – sei dies Zeichenweise oder Zeilenweise. Das Öffnen und Schließen von Dateien wie auch das Anfügen von Texten oder das zeilenweise Auslesen variiert von Programmiersprache zu Programmiersprache. Ab diesem Kapitel wird bei einigen Aufgaben vorausgesetzt, dass solche grundlegenden Operationen auf Dateien bekannt sind. Informieren Sie sich daher neben den Zeichenmanipulationen auch über die folgenden Datei-Befehle in Ihrer Programmiersprache:

[1] Die C-Programmierer z. B. unterscheiden meist nicht zwischen *Feld von Byte* und *String*.

- Auffinden von Dateien im Verzeichnisbaum
- Öffnen und Schließen von Dateien
- Zeilenweise Lesen aus einer Datei
- Zeilenweise Schreiben in eine Datei
- Zeichenweise Lesen aus einer Datei
- Zeichenweise Schreiben in eine Datei
- Zeilen oder Zeichen ans Ende einer bestehenden Datei anfügen

7.3 Aufgaben

Aufgabe 7.1 **Befehle für Zeichenkettenmanipulation**

Zeit:	2 h
Web-Code:	8zx2 pfkp

Schreiben Sie zu jedem Befehl für Zeichenketten aus dem Theorieteil (s. 7.1) ein Beispiel.

Aufgabe 7.2 **Buchstabenmanipulationen**

Zeit:	2 – 4 h
Web-Code:	2ogd g4gf

Schreiben Sie die folgenden kleinen Programme. Alle erwarten einen String, bestehend aus einigen Sätzen. Scheiben Sie zunächst also ein Hauptprogramm, das einen längeren Text mit Leerschlägen und Sonderzeichen als String oder `char[]`-array enthält oder vom Anwender erfragt (`scan`).

- `auspressen()`: Schreiben Sie eine Methode, die aus dem Text alle Buchstaben «e» entfernt. Dabei werden die «e» nicht etwa durch einen Leerschlag ersetzt, sondern die Zeichenkette wird bei jedem «e» um ein Zeichen kürzer.
- `nurBuchstaben()`: Schreiben Sie eine Methode, die alle Zeichen, die weder Buchstaben, Ziffern noch Leerschläge sind, eliminiert. Der ausgegebene Text (Rückgabewert) enthält also insbesondere keine Sonderzeichen mehr. Aus «Hallo-oh! Welt!» wird «Hallooh Welt».
- `miniKompression()`: Schreiben Sie eine Methode, die im Text alle Vorkommen der Zeichenkette «en» durch «x» ersetzt und umgekehrt. Daneben sollen auch alle Vorkommen von «er» mit «q» vertauscht werden («en» → «x» und «x» → «en»). Zu guter Letzt soll noch jedes «y» durch ein «ch» ersetzt werden, dafür bei jedem «ch» nur noch ein «y» stehen. Haben Sie bemerkt worauf es hinausläuft? Wir haben soeben selbst eine kleine Kompressionsroutine geschrieben. Suchen Sie weitere reversible Ersetzungen und implementieren Sie diese.

Aufgabe 7.3 Wortweise umkehren

Zeit:	1 h
Web-Code:	q3bk 4ekq

In einem Text soll jedes Wort einzeln umgekehrt werden. Es wird also nicht der ganze Text umgedreht, sondern lediglich jedes Wort für sich. Beispiel:

```
"Lamatier: reite!"
```

wird zu

```
"reitamaL: etier!"
```

Zusatzaufgabe: Schütteln Sie die Wörter nach dem folgenden Verfahren:
Bei Kurzwörtern mit weniger als vier Buchstaben geschieht nichts, diese werden unverändert ausgegeben. Bei längeren Wörtern (ab vier Buchstaben) werden die mittleren Buchstaben gemischt. Der erste und der letzte Buchstabe bleiben jedoch am Ort. Überprüfen Sie die Lesbarkeit der neuen Texte.

Aufgabe 7.4 Teilstring ersetzen

Zeit:	0.5 h
Web-Code:	tuuj 552f

Schreiben Sie ein Programm, das in einem String einen Teilstring sucht und diesen durch einen anderen Text ersetzt. Prototyp:

```
ersetze(original: string, teil: string, ersetzung: string): string
```

Beispiele:

```
ersetze("hello universe","uni"  ,"multi" ) -> "hello multiverse"
ersetze("jonathan"      ,"jonat","kernig") -> "kernighan"
ersetze("bin auf neinva","nein" ,"ja" )    -> "bin auf java"
```

Bemerkung: Diese Aufgabe ist so elementar, dass die meisten Programmiersprachen bereits Lösungen dazu anbieten.

Aufgabe 7.5 Schriftlich addieren

Zeit:	2 h
Web-Code:	oh4f nrsz

Schreiben Sie eine Funktion, die zwei ganzzahlige Arrays (`integer`) entgegennimmt und diese schriftlich addiert. Dabei dürfen die beiden Arrays an jeder Stelle nur die Werte von 0 bis 9 enthalten. Das Resultat ist wieder ein Array mit Werten im Bereich 0 bis 9. Das neue Array ist möglicherweise um einen Eintrag größer als das größere der beiden Eingabefelder (Übertrag). Dokumentieren Sie, ob die «Einer» durch den höchsten bzw. den niedrigsten Index dargestellt werden.
Zusatzaufgabe: Die Eingabe wie auch die Ausgabe soll ein Array mit Basistyp `char` (z. B. ASCII) sein. Dabei ist zu beachten, dass die Null nicht durch das Byte 0, sondern durch das ASCII-Zeichen Null («0») dargestellt wird. Trick:

```
wert[i] := arr[i] - '0'
```

Aufgabe 7.6 Geheimschrift MIX

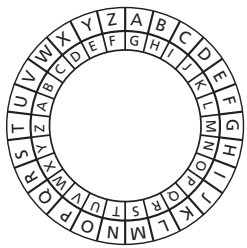
Zeit:	1 h
Web-Code:	5z5i vp58

Ein Text (Zeichenkette) soll zunächst in zwei gleich lange Teile aufgeteilt werden, und danach werden diese Teile zeichenweise wieder zusammengefügt. Besteht der Text aus einer ungeraden Anzahl Zeichen, wird beim Aufteilen der erste Teil um ein Zeichen länger gemacht als der zweite. Beim Zusammenfügen wird mit dem ersten Buchstaben des ersten Teils begonnen, dann mit dem ersten Buchstaben des zweiten Teils weitergemacht; nun kommt der zweite Buchstabe des ersten Teils, und weiter geht es mit dem zweiten Buchstaben des zweiten Teils usw. Programmieren Sie zur Überprüfung Ihres Programms die «Entschlüsselungsmethode», sprich die Umkehrung dieser einfachen Geheimschrift.

Aufgabe 7.7 Cäsar

Zeit:	2 h
Web-Code:	kb99 fe5k

Schreiben Sie ein Programm, das einen Text nach dem Cäsar-Algorithmus[1] verschlüsselt. Dabei wird zu jedem Buchstaben im Alphabet eine fixe Zahl (= Code) dazugezählt. Nach «Z» soll wieder «A» folgen. Mit der negativen Zahl als Code kann der Text wieder entschlüsselt werden.
Knacken Sie damit die Botschaft: «Hslh qhjah lza»



[1] Die ursprüngliche Cäsar-Verschlüsselung nach Gajus Julius Cäsar (100 v. Chr. – 44 v. Chr.) verwendete den Schlüssel 3.

Aufgabe 7.8 Sekunden-Umwandlung

Zeit:	1 h
Web-Code:	sxmw sks3

Schreiben Sie ein Programm, das eine Anzahl Sekunden in die Form «h:mm:ss» (h = Stunden, mm = Minuten und ss = Sekunden) bringt.
Zum Beispiel wird die Zahl 3674 in die Zeichenkette «1:01:14» umgewandelt.
Zusatzaufgabe: Schreiben Sie auch die Umkehrung.

Aufgabe 7.9 IP-Adressen (1)

Zeit:	1 h
Web-Code:	8zmd r65m

Schreiben Sie eine Funktion, die eine vierstellige IP-Adresse als Zeichenkette entgegennimmt (z. B. 192.122.45.45). Als Resultat soll eine 32-Bit-Zahl zurückgegeben werden, die in je einem Byte (8-Bit) eine der vier IP-Dezimalstellen enthält. Schreiben Sie zunächst den Funktionsprototypen

```
ipTransformation(dec: string): integer
```

Achtung: Ihre Programmiersprache muss dazu einen vorzeichenlosen `integer`-Datentypen mit mindestens 32 Bit erlauben (z. B. `long` in Java).

Aufgabe 7.10 IP-Adressen (2)

Zeit:	2 h
Web-Code:	ndma 4uff

Kehren Sie die Aufgabe 7.9 um: Gegeben ist eine IP-Adresse (als vorzeichenlose ganze Zahl mit mind. 32 Bits). Gesucht ist die vierteilige Dezimalnotation

(z. B. «192.168.0.203» als String). Der Prototyp sieht wie folgt aus:

```
ipTransformation(bin: integer): string
```

Aufgabe 7.11 Dreiersystem (1)

Zeit:	1 h
Web-Code:	kwsx k6s4

Programmieren Sie eine Methode `dreier(dreier: string): integer`, die eine Zeichenkette von Ziffern in eine Zahl verwandelt. Die Zeichenkette `dreier` soll im Dreiersystem angegeben werden. Anstelle von Einern, Zehnern, Hundertern und Tausendern gibt es im Dreiersystem Einer, Dreier, Neuner, Siebenundzwanziger, … Im Dreiersystem kommen lediglich die Ziffern 0, 1 und 2 vor. Die Zahl «120» bedeutet also 1 Neuner, 2 Dreier und 0 Einer (= Fünfzehn). Der Wert einer Ziffer (als `character`) wird beispielsweise in Java mit dem einfachen Trick berechnet: `wert := ziffer - '0'`
Zusatzaufgabe: Implementieren Sie die Funktion auch für andere Zahlensysteme:

```
system(ziffernfolge: string, basis: integer): integer.
```

Tipp: Überlegen Sie, ob die gegebene Ziffernfolge besser von links oder von rechts her zu behandeln ist.

Aufgabe 7.12 Dreiersystem (2)

Zeit:	1 h
Web-Code:	8ghq ywrr

Programmieren Sie die Umkehrung der Aufgabe 7.11. Eine Zahl ist in einer `integer`-Variablen gespeichert. Das Resultat ist ein String, bestehend aus den Ziffern 0, 1 und 2, der die Zahl im Dreiersystem repräsentiert.
Zusatzaufgabe: Implementieren Sie auch andere Zahlensysteme.

Aufgabe 7.13 Nachkommastellen im Binärsystem

Zeit:	2 h
Web-Code:	ozvv txgt

a) Was bedeuten die Binärzahlen 1.1; 0.101; 0.111…; 0.010101…?
b) Wie schreiben wir die folgenden Zahlen im Binärsystem: 9.5; 2.75; 0.2; 0.3333…?
c) Implementieren Sie Teilaufgabe a).
d) Implementieren Sie Teilaufgabe b).

Aufgabe 7.14 Deutsche Zahlnamen

Zeit:	2 h
Web-Code:	s8y6 rqfn

Schreiben Sie ein Programm, das die Zahlen von 1 bis 99 in Worte fasst. Die Zahl «97» soll also als «siebenundneunzig» ausgegeben werden. Programmieren Sie zunächst den allgemeinen Fall:

- 57 → siebenundfünfzig
- 96 → sechsundneunzig

Programmieren Sie nun die Spezialfälle:

- 10 → zehn (und nicht nullundeinszig)
- 12 → zwölf (und nicht zweiundzehn)
- 66 → sechsundsechzig (und nicht sechsundsech**s**zig)

Zusatzaufgabe: Erweitern Sie das Programm von 0 bis 1000.
Bemerkung: Ob Sie in der Ausgabe «Hunderteins» oder «Hundertundeins» schreiben, ist irrelevant.

Aufgabe 7.15 Taschenrechner

Zeit:	2 h
Web-Code:	6nmz eiot

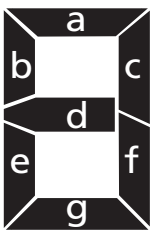
Schreiben Sie ein Programm, das einfache Berechnungen, die als Zeichenketten (Strings) gegeben sind, durchführen kann. Die Eingabe besteht aus einer ersten positiven ganzen Zahl, gefolgt von einem Operator (+, –, ∗ oder /) und einer zweiten positiven ganzen Zahl. Beispiele:

```
"5  + 77"
"66 - 121"
"33 * 46"
"84 / 32"
```

Aufgabe 7.16 7-Segment-Displays

Zeit:	1 h
Web-Code:	m8xi kzq3

Taschenrechner, Aufzüge, Kochherde, … sehr viele elektronische Anzeigen verwenden zur Darstellung von Ziffern sogenannte *7-Segment-Displays* (Display = Anzeige).



Diese bestehen aus sieben kurzen Liniensegmenten, die einzeln ein- bzw. ausgeschaltet werden können.

Hier sind diese Elemente mit a, b, c, d, e, f und g bezeichnet. Schreiben Sie nun je ein Unterprogramm, das für eine gegebene Ziffer (`ziffer`)

1. angibt, ob ein gegebenes Segment (char) vorkommt:
   `kommtVor(ziffer: integer; segment: char) : boolean`
   z. B. `kommtVor(2, 'd')` → `true`
2. ausgibt, welche Segmente vorkommen müssen:
   `welcheKommenVor(ziffer: integer) : string`
   z. B. `welcheKommenVor(7)` → «a c f»

Aufgabe 7.17 ISBN/EAN

Zeit:	1 h
Web-Code:	gg7a xaou

Schreiben Sie eine Methode, welche die Prüfziffern von ISBN (Internationale Standardbuchnummer) bzw. EAN (European Article Number) prüfen kann. Dabei wird ein EAN ohne Prüfziffer (als String) eingegeben, von welcher die Prüfziffer als Rückgabewert ausgegeben wird. Prototyp:

```
berechnePruefziffer(isbnOhnePruefziffer: string): integer
```

Daraus leiten Sie eine weitere Methode ab, die von einer Zeichenkette, diesmal inklusive Prüfziffer, ausgibt, ob die Prüfziffer korrekt ist (`true`/`false`):

```
pruefePruefziffer(isbnMitPruefziffer: string): boolean
```

Der Algorithmus zur ISBN-10-Prüfziffer (= EAN-10) ist wie folgt:

1. Alle Trennzeichen (Leerschläge und Bindestriche) sind zu entfernen. Es bleiben neun Ziffern (z1, z2, z3, …, z9).
2. s := z1 + 2∗z2 + 3∗z3 + 4∗z4 + … + 9∗z9

3. p := s MODULO 11
4. Falls 10 = p: Prüfziffer := «X», ansonsten: Prüfziffer := p

Zusatzaufgabe: Informationen zur Berechnung der Prüfziffer zur ISBN-13 (und EAN-13) finden Sie im Internet. Programmieren Sie zusätzlich eine Prüfung zum ISBN-13.

Aufgabe 7.18 File-Statistik

Zeit:	1 h
Web-Code:	gcx8 7ud7

Schreiben Sie ein Programm, das von einer Datei (Textdatei oder Programm-Quellcode) die Anzahl Zeilen, die Anzahl Wörter und die Anzahl Buchstaben (ohne Leerschläge und Sonderzeichen) ausgibt.
Informieren Sie sich zunächst, wie eine Datei geöffnet wird und wie daraus die Zeilen gelesen werden können. Schreiben Sie eine Boole'sche Funktion `istBuchstabe()` oder verwenden Sie die gegebene Funktion Ihrer Programmiersprache.

Aufgabe 7.19 Datei analysieren

Zeit:	2 h
Web-Code:	ppxv jgs4

Hier sehen Sie den Anfang einer Datenreihe:

```
104.55 CHF
124.40 CHF
 80.35 CHF
 58.80 CHF
 62.65 CHF
 68.50 CHF
 ...
```

Laden Sie die Zahlenreihe mit Web-Code herunter: www.programmieraufgaben.ch. Lösen Sie danach die folgenden Aufgaben:

a) Schreiben Sie eine Subroutine, die eine eingelesene Zeile (String) in eine ganze Zahl (`integer`) verwandelt. Der Wert soll die Rappen (Cent) darstellen. Aus «104.55 CHF» wird also die Zahl 10 455. Es ist wichtig, ganze Zahlen zu verwenden, denn gebrochene Zahlen (`real`) enthalten oft Rundungsfehler.
b) Geben Sie die Summe aller Zahlen aus. Einmal als Summe der Rappen, das andere Mal im gegebenen Format («xxxxx.xx CHF»). Tipp: Speichern Sie die Daten vor den Berechnungen in einem Array.
c) Zählen Sie diesmal nur die Zahlen zusammen, deren Betrag größer als 100.00 CHF ist.
d) Teilen Sie bei dieser Aufgabe die Zahlen in vier Gruppen:

   Gruppe A: < 80.00 CHF,
   Gruppe B: ≥ 80.00 CHF und < 100.00 CHF,
   Gruppe C: ≥ 100.00 CHF und < 120.00 CHF,
   Gruppe D: ≥ 120.00 CHF.
   Geben Sie pro Gruppe die Summe und die Anzahl Werte aus.

Aufgabe 7.20 Wortliste filtern

Zeit:	0.5 h
Web-Code:	pomd oige

Für die Erstellung eines Lernprogramms zum Maschinenschreiben werden geeignete Wörter aus einer Wortliste herausgepickt. Dafür wird die folgende Funktion benötigt:

```
passt(wort: string, darf: string, muss: string): boolean
```

Dabei dürfen im String `wort` nur Buchstaben vorkommen, die auch im String `darf` enthalten sind. Jedoch muss der String `wort` alle Buchstaben enthalten, welche im String `muss` vorkommen. Beispiele:

```
passt("apfelsaft", "asdfghjklö", "as") // liefert false: p nicht ok
passt("sdd",       "asdfghjklö", "as") // liefert false: a fehlt
passt("saft",      "asdfghjklö", "as") // liefert false: t nicht ok
passt("lasag",     "asdfghjklö", "as") // liefert true
```

Überlegen Sie, inwiefern die Hilfsfunktion

```
muss(wort: string, mussMenge: string): boolean
```

von Nutzen sein kann. Hierbei werden Wörter nur akzeptiert, wenn jeder Buchstabe aus der `mussMenge` auch wirklich vorkommt.
Zusatzaufgabe: Laden Sie die Wortliste mit Web-Code herunter: www.programmieraufgaben.ch. Suchen Sie nun alle Wörter, bei welchen die Buchstaben «g» und «o» vorkommen und zusätzlich alle Buchstaben aus «tsrneia» enthalten sein können.

Aufgabe 7.21 Wörter raten

Zeit:	2 – 4 h
Web-Code:	o2wh jaz5

Schreiben Sie ein Programm, bei dem eine Spielerin ein Wort eingeben kann. Eine zweite Spielerin soll dieses Wort erraten. Dabei sieht die zweite Spielerin zu Beginn nur die Anzahl Buchstaben als Platzhalter. Das sieht dann z. B. wie folgt aus:

```
*******
```

Wenn die zweite Spielerin nun Buchstaben ausprobiert, gibt es jeweils zwei Möglichkeiten. Wenn der Buchstabe nicht vorkommt, wird eine entsprechende Meldung angezeigt. Wenn hingegen der Buchstabe vorkommt, wird das Zeichen an der entsprechenden Stelle eingefügt:

```
*n****n
```

Ist das Wort gefunden, endet das Programm mit einer Meldung. Optional können der zweiten Spielerin auch bereits versuchte Zeichen gemeldet werden.
Zusatzaufgabe: Anstelle einer ersten Spielerin können die Wörter auch zufällig ab einer Wortliste geladen werden.

Aufgabe 7.22 Palindrome

Zeit:	2 h
Web-Code:	brfr qbyf

Schreiben Sie ein Programm, das eine Eingabe als Zeichenkette (`string` oder `char[]`) erwartet und danach ausgibt, ob es sich beim eingegebenen Text um ein sog. *Palindrom* handelt (`true`/`false`). Ein Palindrom ist ein Satz, der von beiden Seiten gelesen denselben Text ergibt. Einige der folgenden Textzeilen sind Palindrome:

```
Linux, super, hier Gnu.
Reihung, oho, Gnu hier!
Reit amal a Lamatier.
Sugus - AHA! Sugus.
Reihungen, nenne Gnu hier!
Leben Sie mit im Eisnebel!
Reihung lief stets feil. Gnu hier.
Ein Lama mal' nie.
```

Entfernen Sie zunächst alle unnötigen Satzzeichen und Leerschläge. Verwandeln Sie danach alles in Kleinbuchstaben. Laden Sie obige Palindrome mit Web-Code herunter: www.programmieraufgaben.ch.

8. Datenstrukturen und Sammelobjekte

Eigene Datentypen können aus den vorgegebenen Standardtypen (s. 1.1, S. 15) zusammengesetzt werden.

Ein Clubmitglied weist z. B. die folgenden Attribute auf:

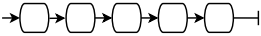
- Name, Vorname, Adresse, Ort: `string`
- Postleitzahl, Eintrittsjahr, Geburtsjahr: `integer`
- Ehrenmitglied: `boolean`

Clubmitglied	
name	: string
vorname	: string
adresse	: string
ort	: string
postleitzahl	: integer
eintrittsjahr	: integer
geburtsjahr	: integer
ehrenmitglied	: boolean

Diese Attribute werden auch *Properties* oder Member-Variable genannt. Ist ein solcher Datentyp einmal definiert, können jederzeit Objekte dieses neuen Datentyps generiert werden. Dies geschieht mit dem Anfordern von Hauptspeicher beim Betriebssystem (in Programmiersprachen beispielsweise mittels Konstruktoren, `new`, *memory alloc*, …).
Datenstrukturen sind auch unter anderen Namen (mit evtl. leicht modifizierten Konzepten) bekannt: Struct, Record, Verbunds-Datentyp und Klasse. Konkrete Exemplare werden danach Instanzen genannt.

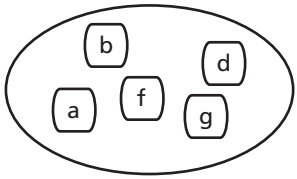
8.1 Sammelobjekte

Die lineare Struktur, wie sie von Arrays bekannt ist, ist nicht immer die optimale Speicherungsform. Alle modernen Programmiersprachen bieten diverse Sammel-Behälter (Collections) an, die Daten für einen bestimmten Zweck besser speichern:

- **List:** Die Elemente sind linear geordnet, d. h. in der Reihenfolge, wie sie eingefügt wurden. 
- Linked**List:** Geordnete Liste, bei der am Anfang und am Ende hinzugefügt, abgeholt und gelöscht werden kann. Somit kann eine LinkedList auch als

Queue (First-in/First-out-Warteschlange) oder Stack (First-in/Last-out-Kellerspeicher) verwendet werden. Das Einfügen in der Mitte geht sehr rasch, da nur zwei Referenzvariable (Pointer) gesetzt werden müssen. Zugriff via Index und das Auffinden einzelner Elemente ist jedoch zeitaufwendig.

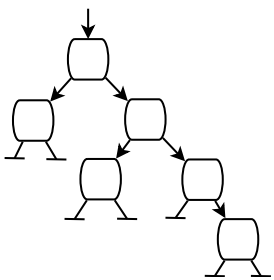
- **Set** = Menge: Elemente können nur einmal angefügt werden.



- (Hash)**Map** (Assoziative Arrays): Speichern der Objekte und Auffinden mit einem Schlüssel (Key).
  Beispiel Bücherliste:

978-3-280-04066-9	Programmieren lernen
3-411-04011-4	Die deutsche Rechtschreibung
0-201-89683-4	The Art of Computer Programming
3-8266-0883-6	Shell-Skript-Programmierung

- **Tree:** Baumstrukturen erlauben ein rasches, sortiertes Einfügen und Suchen. Zugriff via Index ist ähnlich aufwendig wie bei Listen.



8.2 Aufgaben

Aufgabe 8.1 Adäquate Datentypen

Zeit:	0.5 h
Web-Code:	rw4r 9u46

Suchen Sie für die folgenden Größen adäquate (d. h. sinnvolle) Datentypen: `boolean`, `integer`, `real`, Array oder Datenstruktur (Klasse):

- Geldbeträge
- Bevölkerungszahlen
- Bevölkerungszuwachs
- Datum des 21. Jahrhunderts
- Kennzeichen, ob ein Jahr ein Schaltjahr ist
- Adresse (Straße, PLZ, Ort)
- Schachfigur
- Vorzeichen (positiv, negativ, ohne)

Aufgabe 8.2 Datenpaare (Datensequenz)

Zeit:	2 h
Web-Code:	wc4q st5h

Schreiben Sie ein Programm, das von 100 Personen die Familiennamen und Vornamen aufnehmen kann. Laden Sie die Namensliste mit Web-Code herunter: www.programmieraufgaben.ch. Dabei sollen alle Angaben in einem einzigen Array (`data`) wie folgt abgelegt werden:

```
data[0] "Hans"   // erster  Vorname
data[1] "Meier"  // -"-     Familienname
data[2] "Vreni"  // zweiter Vorname
data[3] "Sutter" // -"-     Familienname
```

```
data[4] "Johanna" // dritter Vorname
data[5] "Keller"  // -"-     Familienname
data[6] "Peter"   // vierter Vorname
data[7] ... usw.
```

Schreiben Sie danach die folgenden Subroutinen:

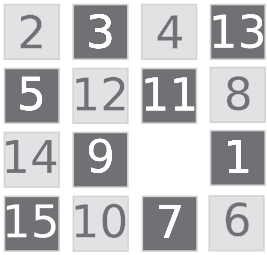
```
speicherePerson (data : string[], nummer : integer,
                 vorname: string, familienname: string)

leseFamilienname(data : string[], nummer : integer) : string
leseVorname     (data : string[], nummer : integer) : string
```

In den Subroutinen bezeichnet der Parameter `nummer` nicht den Index, sondern die Personennummer: 1 = «Hans Meier», 2 = «Vreni Sutter», 3 = «Johanna Keller».

Aufgabe 8.3 **Fünfzehner-Spiel**

Zeit:	2–4 h
Web-Code:	4ksp tn54

Schreiben Sie eine Datenstruktur `Fuenfzehner`, die die Zustände des berühmten 15-er-Spiels angibt. Das Spiel (auch bekannt unter den Namen Schiebepuzzle oder «Sliding Puzzle») besteht aus 15 verschiebbaren Plättchen, die auf 16 Plätzen im Quadrat umhergeschoben werden können. Dabei ist (nach Adam Ries) immer eine Position frei.



Schreiben Sie dazu in einer separaten (Haupt-)Klasse die folgenden Methoden:

```
initialisiere   (f: Fuenfzehner)
schiebeVonRechts(f: Fuenfzehner)
schiebeVonLinks (f: Fuenfzehner)
schiebeVonOben  (f: Fuenfzehner)
schiebeVonUnten (f: Fuenfzehner)
print           (f: Fuenfzehner)
```

Dabei wird beim Initialisieren einfach in Leserichtung eingefüllt (erste Zeile: 1, 2, 3, 4, zweite Zeile: 5, 6, 7, 8, …). Das letzte Feld bleibt beim Initialisieren leer. Die `print()`-Methode gibt den aktuellen Zustand auf der Konsole zur Fehlersuche aus. Alternativ darf die Datenstruktur auch grafisch ausgegeben werden. Die Methoden `schiebeVonXYZ()` schieben aus der Sicht des leeren Feldes von oben, unten, links oder rechts nach. Sollte die Lücke am Rand stehen, können einzelne Aufrufe auch sinnlos sein und sollten deshalb ignoriert werden.
Zusatzaufgabe: Schreiben Sie eine Methode `mischen(f: Fuenfzehner)`, die eine gegebene Position wieder gut[1] mischt.

Aufgabe 8.4 Personen sortieren

Zeit:	2 h
Web-Code:	bkqq ryq7

Schreiben Sie ein Programm, das in einer Liste Personen (Namen, Vornamen und Postleitzahl) speichert. Die Benutzerin gibt so lange Personennamen ein, bis sie mit «ENDE DER LISTE» die Eingabe beendet. Jede eingegebene Person wird an die Liste angefügt. Am Schluss wird die Liste sortiert; dazu verwenden Sie eine Sortiermethode Ihrer Programmiersprache. Geben Sie die Liste je sortiert nach Name, Vorname, aber auch nach Postleitzahl wieder aus.

[1] «Gut mischen» steht hier im Sinne eines Mischalgorithmus, der als Resultat jede Permutation (also auch jede sortierte Position) mit gleicher Wahrscheinlichkeit zulässt.

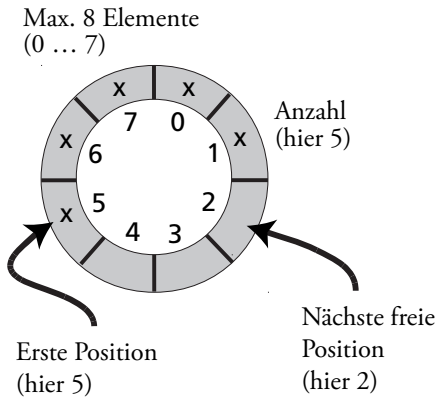
Aufgabe 8.5 Ringpuffer mit Arrays

Zeit:	2 h
Web-Code:	yui8 z5du

Ein **Ringpuffer** (engl. *Buffer*) ist eine Warteschlange einer vorgegebenen Größe. Er kann Daten auf einer Seite anfügen und auf der anderen Seite wieder auslesen. Die folgenden Methoden müssen Sie zur Verfügung stellen:

- `init(maxCount: integer)`: initialisiert den Puffer mit einer maximalen Elementzahl.
- `add(o: Object)`: fügt ein Objekt am Ende des Puffers ein.
- `get(): Object`: liest das Element aus, das schon am längsten im Puffer ist.

Implementieren Sie diese Schnittstelle mit einem Array. Daneben gibt es zwei (versteckte) Attribute: `erster: integer` und `anzahl: integer`.
Die Variable `erster` zeigt auf das Element, das zuerst eingefügt wurde, also dasjenige, das nun als nächstes ausgelesen werden soll (First in/First out = FiFo). Die Variable `anzahl` gibt an, wie viele Elemente zurzeit gültig sind und somit auch, wo das nächste Element in den Array eingefüllt werden muss.
Das nächste einzufügende Element kann (falls `erster > 0`) die Array-Grenze sprengen, obschon wieder freier Platz auf den ersten Feldern vorhanden ist (s. Grafik). Die nächste freie Position ist somit `(erster + anzahl) MODULO maxCount`. MOD `maxCount` ist insofern wichtig, denn der nächste freie Platz kann auch vor dem `ersten` zu liegen kommen. Daher der Name **Ring**puffer (s. Grafik).



Wenn wir nun diesem Datentypen Ringpuffer jedes erdenkliche Objekt (String, `integer`, `boolean`, Verbund-Variable, …) hinzufügen können, sprechen wir von einem **abstrakten Datentypen.**

9. Algorithmen

Ein **Algorithmus** ist eine präzise, schrittweise Vorschrift, die angibt, wie ein Problem gelöst werden soll.
In diesem Kapitel finden wir Aufgaben zu den folgenden Klassen von Algorithmen:

- Sortieren
  Beispiele: Bubblesort, Mergesort, binäre Bäume
- Suchen: linear, binär
- Alte und neue Algorithmen zu Zahlen und Kodierungen
- Primzahlen
- Entwickeln eines eigenen Algorithmus für spezielle Probleme

9.1 Korrektheit und Berechenbarkeit

Beim Lösen einer Aufgabe mit Hilfe eines Computers stellt sich die Frage nach der **Korrektheit** des Algorithmus. Korrekte Algorithmen lösen die gestellte Aufgabe fehlerfrei. Die Korrektheit soll von Ihnen durch einige aussagekräftige Tests verifiziert werden.
Eine weitere Herausforderung von Algorithmen ist die **Berechenbarkeit.** Darunter versteht man die Frage nach der **Laufzeit** des Programms, also diejenige Zeit, die ein Programm braucht, um die entsprechende Aufgabe zu lösen. Ein Programm mit quadratischer **Ordnung** benötigt bei einer Vergrößerung der Problemgröße (Datengröße) um einen Faktor hundert bereits ca. 10 000-fache Laufzeit. Oft führen bessere Algorithmen zu massiv kürzerer Laufzeit.
Auf das weite Feld der Berechenbarkeit wird in den Aufgaben nicht explizit eingegangen. Bei einigen unserer Standardaufgaben können verbesserte Algorithmen jedoch einiges an Geschwindigkeit herausholen. Ein spannendes Buch zu diesem Thema sind die «Sieben Wunder der Informatik»[1].

[1] Jurai Hromkovič: Sieben Wunder der Informatik. 2. Auflage, 2009. Vieweg + Teubner. ISBN: 978-3-8351-0172-2

9.2 Aufgaben

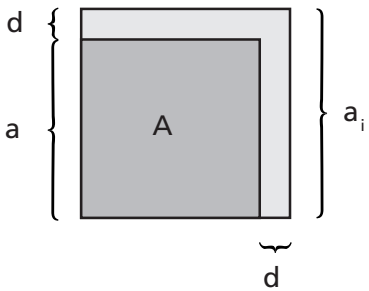
Aufgabe 9.1 **Heron**

Zeit:	1 h
Web-Code:	kugc tg53

Berechnen Sie die Quadratwurzel a einer Zahl A, mit dem Verfahren von Heron[1]. Dabei wird die Hälfte von A als erste Näherung a_1 verwendet. Jede weitere Näherung a_{i+1} wird berechnet durch die Formel $a_{i+1} := (A + a_i^2) / 2a_i$. Als Code:

```
a := (A + a * a) / (2 * a)
```

Begründung:



$$d = a_i - a \quad \text{und}$$

$$d \cdot 2a_i \approx (a_i^2 - A)$$

$$\Rightarrow (a_i - a) \cdot 2a_i \approx (a_i^2 - A)$$

$$\Rightarrow \quad 2a_i a \approx A + a_i^2$$

$$\Rightarrow \quad a_{i+1} := (A + a_i^2) / 2a_i \approx a$$

Zusatzaufgabe: Berechnen Sie die Kubikwurzel (3. Wurzel) einer Zahl nach dem folgenden Verfahren: $a_{i+1} := (A + 2a_i^3) / 3a_i^2$

[1] Heron von Alexandria (ca. 1. Jh. v. Chr.)

Aufgabe 9.2 Gray Code

Zeit:	2 h
Web-Code:	gtwa psr3

Ein Gray[1]-Code ist eine **binäre** Kodierung der Zahlen (0, 1, 2, 3, …) derart, dass sich von Zahl zu Zahl nur gerade ein Bit ändert. Der klassische Gray-Code sieht wie folgt aus:

```
Zahl  Dualsystem    Gray Code
0     ...00000      ...00000
1     ...00001      ...00001
2     ...00010      ...00011
3     ...00011      ...00010
4     ...00100      ...00110
5     ...00101      ...00111
6     ...00110      ...00101
7     ...00111      ...00100
8     ...01000      ...01100
9     ...01001      ...01101
...   ...           ...
```

Um von der Darstellung im Dualsystem auf obige Darstellung als Gray-Code zu gelangen, kann einfach wie folgt vorgegangen werden: Betrachten Sie die duale Darstellung von links nach rechts Ziffer für Ziffer. Jedes Mal, wenn sich die Ziffer ändert (1 nach 0 oder 0 nach 1), schreiben Sie eine «1». Wenn sich die Ziffer nicht ändert (0 nach 0 oder 1 nach 1), so schreiben Sie eine «0». Beispiel:

```
Zahl: 6
Dual: ... 0 0 0 1 1 0
Gray: ...0 0 0 1 0 1
```

Schreiben Sie ein Programm, das eine ganze Zahl in obigen Gray-Code verwandelt. Geben Sie das Resultat binär aus. Beispiel: «6» wird zu «… 000101».

[1] Frank Gray (amerik. Physiker 20. Jh.)

Aufgabe 9.3 Primzahlen

Zeit:	1 h
Web-Code:	6yqz 4y5k

Schreiben Sie ein Programm, das eine gegebene Zahl daraufhin prüft, ob es sich um eine Primzahl (2, 3, 5, 7, 11, …) handelt.

Aufgabe 9.4 Primzahlsieb des Eratosthenes

Zeit:	1 h
Web-Code:	tmx9 wf4j

Schreiben Sie ein Programm, das mittels des Primzahlsiebs des Eratosthenes[1] alle Primzahlen bis zu einer vorgegebenen Zahl (sagen wir 100) ermittelt.
Dazu wird zunächst ein Array mit Zahlen von 2 bis und mit 100 aufgefüllt. In diesem Fall ist der Array natürlich 99 Elemente groß.

```
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ..., 100.
```

In einem zweiten Schritt wird die nächste noch vorhandene Zahl gesucht (beginnend bei der Zahl 2), und alle Vielfachen davon werden gelöscht, indem sie auf 0 (Null) gesetzt werden. Der Array sieht also nach dem ersten Durchlauf (mit Zahl 2) wie folgt aus:

```
2, 3, 0, 5, 0, 7, 0, 9, 0, 11, 0, 13, 0, 15, 0, ...
```

Nach demselben Verfahren wird die nächste noch vorhandene Zahl (hier 3) gesucht und alle Vielfachen davon gelöscht (also auf Null gesetzt):

```
2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, ...
```

[1] Eratosthenes (um 276 v. Chr. – 195 v. Chr.)

Das geht so weiter, bis die Quadratwurzel der höchsten Zahl überschritten wurde. Im 3. Schritt werden die verbleibenden Zahlen auf dem Bildschirm ausgegeben. Das sind dann gerade alle Primzahlen bis zur vorgegebenen Zahl. Um den Algorithmus zu vereinfachen, schreiben Sie eine Hilfsfunktion, die die Vielfachen löscht:

```
vielfacheLoeschen(primzahl: integer)
```

Aufgabe 9.5 Römische Zahlen

Zeit:	1 h
Web-Code:	sv9k jktq

Schreiben Sie ein Programm, welches Zahlen (von 1 bis 3000) in das römische Zahlensystem umwandeln kann. Die römischen Zahlzeichen haben die folgenden Bedeutungen:

- I = 1
- V = 5
- X = 10
- L = 50
- C = 100
- D = 500
- M = 1000

Beachten Sie, dass es unüblich ist, vier Einer, Zehner oder Hunderter hintereinander zu schreiben. Stattdessen wird der nächste Fünfer, Zehner, … benutzt und links davon eine Einheit subtrahiert. So ist XC = 90 und XCIX = 99 (selten auch IC).

Zusatzaufgabe: Schreiben Sie auch die Umkehrung, also die Umwandlung römischer Zahlen in unser Stellenwertsystem (10er-System).

Aufgabe 9.6 Wörter suchen

Zeit:	2 h
Web-Code:	9nz3 nf8o

Laden Sie die gepackte Wortliste mit Web-Code herunter: www.programmieraufgaben.ch. (Es handelt sich um eine leicht modifizierte Rechtschreibkorrektur des *Browsers* «Firefox».)
Überlegen Sie die Vor- und Nachteile einer linearen Suche im Gegensatz zu einer binären Suche. Lösen Sie damit die folgenden Aufgaben:

a) Schreiben Sie ein Programm, das alle Wörter der Datei ausgibt, bei welchen der folgende **Wortanfang** gegeben ist: «Distanz», «Erdbi», «Finanzind».
b) Schreiben Sie ein Programm, das ermittelt, welche der folgenden Wörter in der Liste **vorkommen:** «Darmlymphgefäß», «Fahrplanabschnitt», «Formamid».
c) Schreiben Sie ein Programm, das alle Wörter anzeigt, welche die folgende **Wortendung** aufweisen: «gulierung», «dcomputer», «chsdor».

Aufgabe 9.7 Lexikografisch sortieren

Zeit:	2 – 4 h
Web-Code:	t42p 7sny

Um eine lexikografische Ordnung von Wörtern zu erreichen, kann nicht einfach die triviale aufsteigende Sortierung einer Zeichentabelle durchgeführt werden.
Zum einen müssen große und kleine Buchstaben gleich behandelt werden, und zum anderen sollen Umlaute und Ligaturen (ß) nach ihren zugrunde liegenden Buchstaben einsortiert werden:

```
 A < a < AA < aa < AAb < Aachen < ...
...< aha < Ähre < Akne < ...
...< haschen < Häschen < hat < ...
...< Masse < Maße < Maßeinheit < ...
...< ZZ < zz
```

Entwickeln Sie nach obigem Beispiel eine Funktion, die zwei gegebene Wörter miteinander vergleicht.
Tipp: Ein «ä» wird wie ein «a» sortiert, es sei denn, es gibt auch eine ansonsten identische Wortvariante mit einem «a» an derselben Stelle. In einem solchen Fall wird das Wort mit Umlaut nach letzterem einsortiert. Analog verhält es sich mit den Groß- und Kleinbuchstaben. Mit Vorteil werden die Wörter zunächst in ihre lexikalische Grundform gebracht: nur Kleinbuchstaben, keine Umlaute [a statt ä], keine Ligaturen [ss statt ß], keine Satzzeichen [*Leerschlag* statt «.»]. Stimmen die Grundformen nicht überein, so wird nach eben dieser Grundform sortiert.

10. Simulationen

Mit dem Computer sind Probleme lösbar geworden, die zuvor aufgrund ihrer Komplexität unlösbar waren. Beispiel: **Der Vier-Farben-Satz**
Der Vier-Farben-Satz (Vierfarbenproblem) ist ein Klassiker der mathematischen Beweisführung. Dabei wurde 1976 von K. Appel und W. Haken bewiesen, dass jede Landkarte mit vier Farben so eingefärbt werden kann, dass nie zwei Länder mit derselben Farbe aneinandergrenzen. Um dies zu beweisen, musste obige Behauptung für alle Arten von Landkarten bewiesen werden. Man kann die Karten in sogenannte Äquivalenzklassen gliedern, damit der Satz nur für je eine Karte aus einer solchen Äquivalenzklasse bewiesen werden muss. Da es davon immer noch hunderte gibt, war der Beweis von Hand beinahe unmöglich. Ein Computerprogramm kann aus jeder Klasse eine Karte einfärben und das Problem so in sehr kurzer Zeit bejahend beweisen[1].
Wir behandeln in diesem Kapitel Simulation im Umfeld der kombinatorischen angewandten Mathematik, im Speziellen die folgenden zwei Kategorien[2]:

- Brute Force, d. h. Austesten aller Möglichkeiten,
- Monte Carlo Methoden.

Simulationen im Bereich der Naturwissenschaften und der Architektur modellieren die zu simulierenden Prozesse mit Hilfe von Differenzialgleichungen, welche den Rahmen dieses Buches überschreiten.

10.1 Brute Force – Rechnen mit maximaler Leistung

Eine spezielle Kategorie von Algorithmen befasst sich mit dem Auszählen kombinatorischer Probleme. Beispielsweise sollen alle möglichen Augenzahlen angegeben werden, die fünf Spielwürfel zeigen können.
Da heutige Computer sehr schnell rechnen, brauchen wir uns vor Anzahlen im Bereich von 100 Millionen Möglichkeiten nicht beeindrucken zu lassen!

---

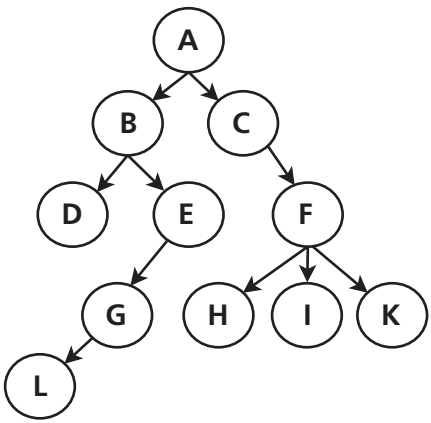
1 Der Satz wurde 1996 von N. Robertson, D. P. Sanders, P. Seymour und R. Thomas mit einem Computerprogramm von ca. 13 000 Codezeilen neu bewiesen.

2 Weitere Algorithmen, beruhend auf großen Datenmengen, sind «Genetische Algorithmen» und «Neuronale Netze».

Bei solchen Problemen können wir uns meistens die folgenden Fragen stellen:

- Gibt es eine Lösung?
- Zeige eine Lösung an.
- Gib die Anzahl der Lösungen aus.
- Zeige systematisch alle Lösungen an.
- Gib nur Lösungen aus, die im Wesentlichen verschieden von allen bereits ausgegebenen Lösungen sind. Dabei werden Symmetrien vernachlässigt.

Die Vorgehensweise zu «Brute Force»-Algorithmen unterscheiden sich in den Konzepten **Breitensuche** und **Tiefensuche**. Betrachten Sie zur Erklärung den folgenden Problemlösungsbaum, also die baumartige, hierarchische Darstellung, die ein komplexes Problem in einfachere Teilprobleme aufteilt. Knoten «A» bezeichnet das Gesamtproblem, und alle weiteren Knoten sind einfachere, meist gleichartige Teilprobleme.

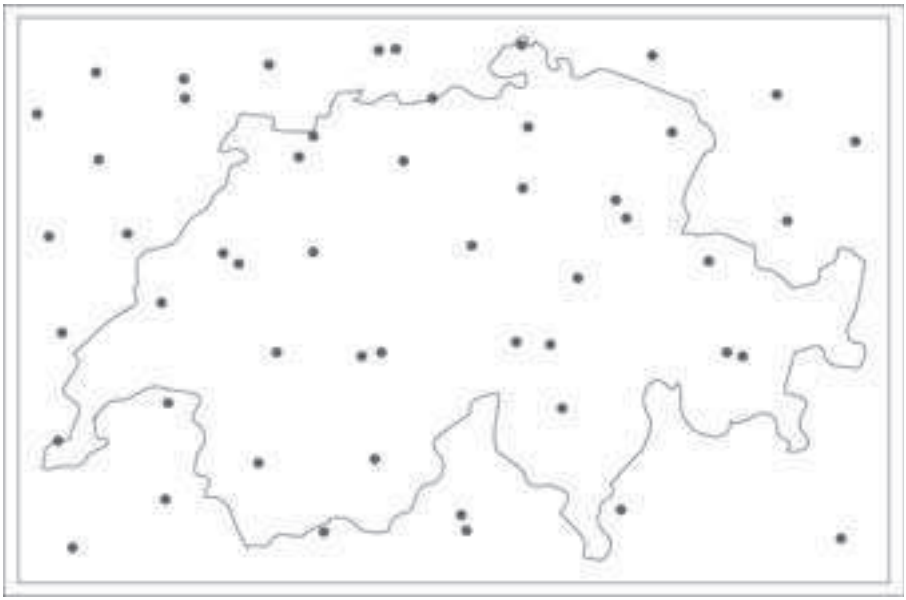


Bei der **Tiefensuche** werden die Teilbäume vollständig durchsucht, bevor der nächste Teilbaum derselben Ebene in Angriff genommen wird. Jedes Teilproblem wird also für sich vollständig gelöst, bevor ein anderes Teilproblem in Angriff genommen wird. In obigem Beispiel ergibt sich diese Reihenfolge:
A B D E G L C F H I K
Bei der **Breitensuche** werden zuerst alle Knoten derselben Hierarchiestufe (Ebene) untersucht. Alle Teilprobleme werden bis zu einer gewissen Tiefe gleich-

zeitig gelöst, bevor die nächste Tiefe oder Stufe berechnet wird. Bei der Strategie Breitensuche wird die Genauigkeit meistens von Stufe zu Stufe erhöht.
Hier ergibt sich die folgende Reihenfolge:
A B C D E F G H I K L
Je nach Problemstellung und Berechnungszeit ist die eine oder die andere Vorgehensweise vorzuziehen.
In der Regel verwendet die Tiefensuche eine höhere Laufzeit im Gegensatz zur Breitensuche, die dagegen mehr Speicher benötigt.

10.2 Monte-Carlo-Methoden

Monte-Carlo-Methoden untersuchen Probleme mit Tausenden von (Pseudo-) Zufallszahlen und kommen so auf eine gute Näherung der Lösung.
Das Standardbeispiel nähert einen gesuchten Flächeninhalt durch folgendes Verfahren an:



Die Fläche mit dem gesuchten Inhalt (hier das Territorium der Schweiz) ist Teil einer größeren Fläche mit bekanntem Ausmaß (hier ein Rechteck). In das Innere der Gesamtfläche (des umfassenden Rechtecks) werden zufällig Punkte geschossen. Bei genügender Anzahl Punkte kann aus dem Verhältnis der Anzahl Punkte in der gesuchten Fläche zur Gesamtzahl der geschossenen Punkte der gesuchte Flächeninhalt angenähert werden.

10.3 Aufgaben

Aufgabe 10.1 **Lineare Kongruenzmethode**

Zeit:	1 h
Web-Code:	qqrc bps2

Zum Testen von Software, insbesondere zur Testdatenerzeugung, zur Simulation von Ereignissen, zur Entscheidungsfindung und nicht zuletzt für die Verschlüsselung werden zuverlässige Zufallszahlen benötigt. Jedoch widerspricht sich das Konzept «Algorithmus» mit dem Konzept «Zufall».
In der Praxis hat sich dennoch ein Algorithmus zum Erstellen von sog. Pseudozufallszahlen durchgesetzt: Die **lineare Kongruenzmethode.**
Dazu werden drei Zahlen (a, b und m) fest vorgegeben. Eine Startzahl x_0 wird als sog. *seed* eingetragen. Nun werden «Zufallszahlen» x_1, x_2, x_3, … nach dem folgenden Algorithmus berechnet:
$x_{i+1} := (a * x_i + b)$ MODULO m
Schreiben Sie ein Programm, das vom Anwender a, b, m und x_0 erfragt und danach x_1 bis x_m auf der Konsole ausgibt.
Diskutieren Sie die folgenden Fälle:

- a = m
- m = 1
- ggT(a, b, m) > 1

Aufgabe 10.2 Ganze Zufallszahlen

Zeit:	0.25 h
Web-Code:	8v6n j82y

Die meisten Programmiersprachen kennen einen Zufallszahlengenerator, der eine (pseudo-)zufällige Zahl zwischen 0.0 (inklusive) und 1.0 (exklusive) erzeugt. Schreiben Sie eine Funktion, die eine solche gleichverteilte Zufallszahl in eine ganze Zufallszahl von `min` bis `max` umwandelt:

```
zufall(nullEinsVerteilt: real, min: integer, max: integer): integer
```

Aufgabe 10.3 Würfel (1)

Zeit:	1 h
Web-Code:	r2hn k3om

Suchen Sie in der API[1]-Dokumentation Ihrer Programmiersprache (z. B. Java-API) einen Weg, Pseudozufallszahlen zu generieren. Schreiben Sie ein Programm, das 100 Würfelresultate (Zahlen von 1 bis 6) simuliert. Prüfen Sie Ihr Resultat, indem Sie zählen, wie oft jede Zahl dabei vorgekommen ist.

Aufgabe 10.4 Casino (Würfel 2)

Zeit:	2 h
Web-Code:	5n2a jfjb

Gegeben sind drei ungewöhnlich nummerierte Spielwürfel mit den Farben *Blau*, *Rot* und *Gelb*. Die drei Würfel sind nicht wie üblich mit 1 bis 6 nummeriert sondern besitzen die folgende Nummerierung:

[1] API = Application Programmer Interface (Programmierschnittstelle an das System)

- Blauer Würfel mit den Zahlen 5, 7, 8, 9, 10, 18
- Roter Würfel mit den Zahlen 2, 3, 4, 15, 16, 17
- Gelber Würfel mit den Zahlen 1, 6, 11, 12, 13, 14

In einem Spiel wird jeweils mit zwei Würfeln gespielt. Es gewinnt der Wurf mit der höheren Augenzahl. Finden Sie den besten Würfel: Simulieren Sie dreimal 1 000 000 Spiele, indem Sie zunächst 1 000 000-mal BLAU gegen ROT, danach 1 000 000-mal BLAU gegen GELB und zuletzt noch ebenso oft ROT gegen GELB antreten lassen.
Folgende Funktion kann dabei von Nutzen sein.

```
anzahlSiege(a: string, b: string, anzahlSpiele: integer): integer
```

Dabei bezeichnen `a` und `b` einfach die Würfel, was auch mit einer Zahl (`integer`) zu bewältigen ist.

Aufgabe 10.5 **Web-Code**

Zeit:	0.5 h
Web-Code:	d2rh rur6

Für ein Buchprojekt wird ein Web-Code benötigt. Mit diesem Web-Code können Artikel direkt online abgefragt werden. Der Code soll aus acht Zeichen bestehen. Vorkommen dürfen Ziffern und Kleinbuchstaben. Um Verwechslungen zu vermeiden, kommen die Ziffer Eins (1) und der Kleinbuchstabe «ell» (l) nicht vor. Ebenso kommt die Null (0) nicht vor. Dass das große Oh (O) nicht vorkommen kann, ist klar, denn die Vorgabe erlaubt nur Kleinbuchstaben. Schreiben Sie ein Programm, das fünf zufällige Web-Codes erzeugt.

Aufgabe 10.6 **Farbstiftspitzen**

Zeit:	2 h
Web-Code:	i7ox fcxf

Voraussetzung: In einem Etui sind 20 Farbstifte. Alle sind stumpf und müssen gespitzt werden. Nach folgendem Verfahren werden die Farbstifte nun gespitzt:

1) Aufs Mal werden drei Stifte «blind» aus dem Etui genommen (beim ersten solchen «blinden Griff» sind natürlich alle drei stumpf).
2) Alle drei Stifte werden gespitzt (die bereits spitzen darunter natürlich nicht).
3) Die drei spitzen Stifte werden wieder zurückgelegt.

Die obigen drei Schritte werden so oft wiederholt, bis alle gespitzt sind. Dabei werden oft auch bereits gespitzte Farbstifte «gezogen».

a) Schreiben Sie zunächst die Funktion, die drei **verschiedene** Zufallszahlen im Bereich 0 bis 19 findet.
b) Simulieren Sie nun eine solche Spitzaktion und zählen Sie, wie oft herausgegriffen werden muss, bis alle gespitzt sind.
c) Simulieren Sie nun 10 000 «Spitzaktionen» mit 20 Stiften und drei Blindgriffen pro Mal und messen Sie dabei
   – die durchschnittliche Anzahl Griffe,
   – minimale und maximale Anzahl Griffe.
d) Verallgemeinern Sie die Aufgabe auf n Stifte ($n > 2$) und m Stifte pro Blindgriff ($1 \leq m < n$).

Aufgabe 10.7 **Nummern-Ratespiel**

Zeit:	1 h
Web-Code:	4i5d 58ws

Schreiben Sie ein Programm, bei dem die Anwenderin (Spielerin) eine ganze Zahl zwischen 1 und 100 erraten muss. Dabei gibt das Programm immer nur

aus, ob die gesuchte Zahl größer oder kleiner als der letzte Versuch ist. Schreiben Sie vorab ein Unterprogramm, das eine (Pseudo-)Zufallszahl zwischen 1 und 100 erzeugt (siehe Aufgabe 10.1). Diese Zufallszahl darf der Spielerin natürlich während des Ratens nicht angezeigt werden.
Zusatzaufgabe: Schreiben Sie auch die Umkehrung. Das Programm soll diejenige Zahl raten, die Sie sich als Spielerin oder Spieler vorab gemerkt hatten.

Aufgabe 10.8 Yahtzee

Zeit:	2 h
Web-Code:	iuv3 ttwv

Das Würfelspiel Yahtzee verwendet ähnliche Muster wie das Kartenspiel Poker, verzichtet aber auf die verschiedenen Farben. Bei Yahtzee werden fünf Spielwürfel (Augenzahlen 1 bis 6) geworfen. Verwenden Sie dazu Pseudozufallszahlen entweder aus Aufgabe 10.1 oder die Zufallszahlen Ihrer Programmiersprache. Beispiel:

```
integer wurf := random() * 6 + 1
```

Schreiben Sie ein Programm, das die fünf Zahlen in einem Array speichert und mittels einer Funktion auch in der Standardausgabe anzeigen kann. Entscheiden Sie dann, ob alle Zahlen gleich sind (5-Poker).

- `istPoker5(wuerfe: integer[])` `// alle sind gleich`

Zusatzaufgaben: Entscheiden Sie, ob es sich um ein anderes Poker-Muster (z. B. Full-House) handelt:

- `istPoker4(wuerfe: integer[])` `// vier gleiche`
- `istPaar(wuerfe: integer[])` `// zwei gleiche`
- `istTupel(wuerfe: integer[], n: integer)` `// 2 <= n <= 5`
- `istDoppelPaar(wuerfe: integer[])` `// zwei Paare`
- `istStrasse(wuerfe: integer[])` `// Reihe`
- `istFullHouse(wuerfe: integer[])` `// Paar + Tripel`

Aufgabe 10.9 Poker verteilen

Zeit:	1 h
Web-Code:	i4si 4ts5

Schreiben Sie ein Programm, das vier Spielerinnen je fünf Pokerkarten verteilt. Dabei ist ein Array zunächst mit den Zahlen 1 bis 55 zu füllen. Am einfachsten verwenden Sie dazu die Indizes 1 bis 55. 1 – 13 entspricht den Herz-Karten, 14 – 26 sind die Pik-Karten, danach folgen 13 Karo- und zuletzt die Kreuz-Karten. Die Nummern 53, 54 und 55 sind die drei Joker-Karten. Die erste bis und mit die zehnte Karte pro Farbe sind jeweils die Zahl-Karten, die elfte entspricht dem Jungen (J), die zwölfte der Queen (Q = Dame) und die dreizehnte ist der König (K).
Mischen Sie den Array nach folgendem Algorithmus (D. Knuth: The Art of Computer Programming Vol. 2; ISBN 0-201-89684-2; Addison-Wesley; S. 145 Shuffling) und verteilen Sie die ersten 20 Karten reihum an vier Spielende.
Misch-Algorithmus:

```
mischenAbPos := 1

while(mischenAbPos < 55)
{
  zufallsPos   := zufällige Position aus den Zahlen
                  [mischenAbPos bis 55]
  vertausche die Elemente "array[mischenAbPos]" mit
                  "array[zufallsPos]"
  mischenAbPos := mischenAbPos + 1
}
```

Aufgabe 10.10 Geburtstagszwillinge

Zeit:	2 h
Web-Code:	aadv mxmq

An einer Party nehmen 30 Gäste teil. Der Veranstalterin fällt auf, dass zwei Personen am selben Tag Geburtstag haben.
Die Gastgeberin fragt sich nun, wie viele Leute normalerweise nötig sind, damit zwei Gäste am selben Tag Geburtstag haben.
Theoretisch können dies bis zu 366 Personen sein; die 366. Person hat dann bestimmt einen bereits «aufgetretenen» Geburtstag (Schalttage vernachlässigt).
Simulieren Sie eine Party und zählen Sie dabei, wie viele Personen es braucht, bis sich zum ersten Mal ein Geburtstag wiederholt. Starten Sie danach 10 000 solcher Simulationen und bilden Sie den Mittelwert: Wie viele Personen braucht es im Durchschnitt an einer Party, damit zwei Gäste am selben Tag Geburtstag feiern?

Aufgabe 10.11 Einfache Kombinatorik

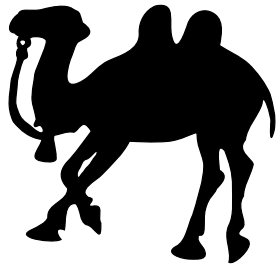
Zeit:	0.5 h
Web-Code:	4cnf kew2

Wer des Programmierens mächtig ist, kann viele kombinatorische Fragen auszählen lassen. Beispiel: Wie viele dreistellige Zahlen haben lauter verschiedene Ziffern?
Zusatzaufgabe: Wie viele Möglichkeiten gibt es, eine aufsteigende Folge von 6 aus 45 Zahlen zu bilden?

Aufgabe 10.12 Reisegruppe

Zeit:	2 h
Web-Code:	aoy8 chnm

Eine Reisegruppe will Ägypten bereisen und dort Sehenswürdigkeiten besuchen. Die Anzahl und die Reihenfolge der Sehenswürdigkeiten bestimmt, wie viele mögliche Touren es geben wird. Lösen Sie die folgenden vier Teilaufgaben:



a) Die Gruppe will vier Sehenswürdigkeiten besuchen. Nummerieren Sie diese und geben Sie alle möglichen Touren an: 1234, 1243, 1324, …, 4321
b) Wie Aufgabe a). Es sollen jedoch neun Sehenswürdigkeiten besucht werden. Geben Sie zudem die Anzahl der Touren aus.
c) Unter den neun Sehenswürdigkeiten der Aufgabe b) müssen die Pyramiden von Giseh (Nr. 1) und der Basar von Kairo (Nr. 2) mit dabei sein. Diese beiden Sehenswürdigkeiten müssen unbedingt unmittelbar nacheinander, wenn auch nicht unbedingt in dieser Reihenfolge besucht werden. Geben Sie die verbleibenden möglichen Touren und deren Anzahl an.
d) Der Karnak-Tempel bei Luxor (Nr. 3) ist unter den neun Sehenswürdigkeiten mit dabei. Zwischen den Pyramiden und dem Karnak-Tempel müssen mindestens drei andere Sehenswürdigkeiten liegen. Die Bedingung von Aufgabe c) muss aber weiterhin erfüllt sein. Geben Sie alle Touren und deren Anzahl an.

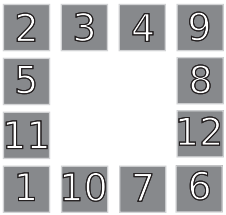
Tipp: Bei Aufgabe c) und d) kann eine `abstand(a, b)`-Funktion hilfreich sein. Diese Aufgabe zeigt, wie mathematisch komplizierte Problemstellungen durch simples Durchnummerieren aller Möglichkeiten gelöst werden können.

Zusatzaufgabe: Wie viele Sehenswürdigkeiten müssen total vorhanden sein, damit ein stures Durchnummerieren aller Möglichkeiten nicht mehr sinnvoll ist.

Aufgabe 10.13 25 im Quadrat

Zeit:	2 – 4 h
Web-Code:	ydd3 8ozc

Auf zwölf Karten sind die ersten natürlichen Zahlen gedruckt (1, 2, …, 12). Die zwölf Karten werden in einem Quadrat angeordnet, sodass die Mitte des Quadrates leer bleibt, aber auf den vier Kanten je vier Karten liegen:



Die Summe der Kartenwerte auf jeder Kante soll nun 25 ergeben. Gesucht ist also eine Anordnung, bei der für jede Kante alle Werte zusammengezählt den Wert 25 ergeben.

a) Geben Sie eine Lösung aus.
b) Geben Sie die Anzahl der Lösungen aus.
c) Geben Sie alle Lösungen aus. (Achtung: Es sind viele!)

Tipp: Auch wenn das Aufzählen aller 12! Möglichkeiten (ca. 480 Millionen Fälle) lange nicht der effizienteste Algorithmus ist, ist es für einen heutigen PC in wenigen Minuten zu schaffen.

Aufgabe 10.14 Magisches Quadrat

Zeit:	2 h
Web-Code:	cmi5 4vwv

Gegeben sind die folgenden neun Spielkarten eines Poker-Sets:

- Herz As (1), Herz 2 und Herz 3
- Pik As (1), Pik 2 und Pik 3
- Karo As (1), Karo 2 und Karo 3

Schreiben Sie ein Programm, das die Karten in einem Quadrat (drei Zeilen à je drei Spalten) so anordnet, dass in jeder Zeile und in jeder Spalte jede Farbe, aber auch jede Zahl einmal vorkommt.

a) Geben Sie eine Lösung aus.
b) Wie viele Lösungen sind es insgesamt?

Aufgabe 10.15 Socken

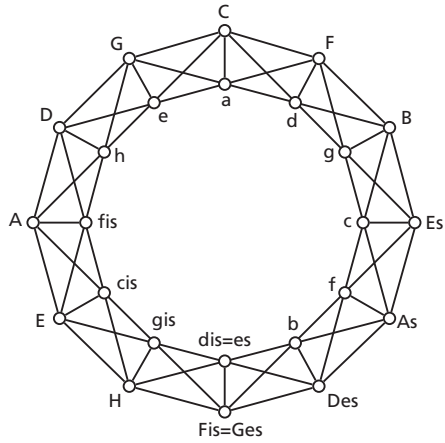
Zeit:	2 – 4 h
Web-Code:	rs4y t2k2

Herr Gressly besitzt fünf verschiedene Arten von Socken. Sie haben verschiedene Farben und Muster. Die Anzahlen der zusammengehörigen Socken sind 20, 6, 8, 16 bzw. 5(!). Bei einer Sockenart hat er demnach nicht mehr alle Paare vollständig. Jeden Morgen nimmt er im Dunkeln zufällig 4, 5 oder 6 Socken aus seiner Sockenschachtel und hofft, dass zwei derselben Sorte dabei sind. Ist ein zusammengehöriges Paar dabei, so werden die restlichen Socken zurückgelegt. Ist kein Paar dabei, wird dieser Tag als «Pechtag» verbucht, und Herr Gressly holt sich die Sockenschachtel ans Licht und entnimmt ihr irgendein passendes Paar. Nach 11 Tagen werden alle benutzten Socken gewaschen und die Schachtel wird wieder gefüllt.
Simulieren Sie ein Jahr (365 Tage) und geben Sie aus, an wie vielen Tagen Herr Gressly auf Anhieb ein zusammengehörendes Paar herausgefischt hat und wie oft er Pech hatte.

Aufgabe 10.16 Quintenzirkel

Zeit:	2 h
Web-Code:	v8mr dw3x

In der Grafik rechts ist der Quintenzirkel der Musiklehre abgebildet. Im äußeren Kreis sind die Dur-Tonarten, im inneren Kreis die Moll-Tonarten dargestellt. Diejenigen Akkorde, die miteinander nahe verwandt sind, sind durch Kanten miteinander verbunden: Zum Beispiel ist C-Dur (Tonika) mit G-Dur (Dominante), F-Dur (Subdominante), a-Moll (Tonikaparallele), e-Moll (Dominantenparallele) und mit d-Moll (Subdominantenparallele) verbunden. Simulieren Sie Musikstücke, die bei der Tonika C-Dur starten und sechs Mal die Tonart längs einer vorgegebenen Kante ändern. Messen Sie, bei welcher Tonart Sie am Schluss am häufigsten landen.



Tipp: Da jede Tonart genau fünf Parallelen aufweist, bietet sich eine Tabelle (zweidimensionales Array) an. Die erste Spalte bezeichnet die Tonika, die zweite z. B. die Dominante usw. Vergibt man den Tonarten Nummern, so können diese direkt als Indizes verwendet werden (die erste Spalte könnte in diesem Fall weggelassen werden). Eine weitere Möglichkeit ist das Verwenden von assoziativen Arrays, die als Indizes auch Zeichenketten («C», «cis», …) zulassen. Referenzvariable in Datenstrukturen einzubinden wäre eine dritte Möglichkeit, diesen Graphen abzubilden.

Aufgabe 10.17 Neuweltkamele

Zeit:	1 h
Web-Code:	jsqi 792x

In einer südamerikanischen Steppe wurden alle natürlichen Feinde der Neuweltkamele ausgerottet. Im Jahre 2040 zählte man einen Bestand von bereits 4500 Kamelen. Jedes Jahr nimmt der Bestand um geschätzte 4.8 % der Anzahl Weibchen zu. Es ist zwar die Anzahl aller Kamele bekannt, jedoch nicht, um wie viele Männchen bzw. Weibchen es sich handelt.
Voraussetzung: Füllen Sie in einer Programmschleife zwei Variable (`anzahlWeiblich`, `anzahlMaennlich`) so ab, dass erstens die Werte zufällig sind und zweitens die Summe der Variablen die erwähnten 4500 Individuen ergibt.
Die Jäger werden angehalten, jedes Jahr 200 Tiere zu schießen, um eine Überpopulation zu verhindern. Leider ist es den Jägern nicht möglich, vor dem Abschuss jeweils zu sehen, ob es sich um ein männliches oder um ein weibliches Kamel handelt. Schreiben Sie eine weitere Methode, die 200 Tiere «abschießt», mit zufälliger Wahl von Männchen und Weibchen.

a) Wie viele Kamele sind es voraussichtlich im Jahr 2041?
b) Wie lange dauert es nach Ihrem Programm, bis wieder eine natürliche Population von nicht mehr als 2000 Exemplaren vorhanden sein wird?

Aufgabe 10.18 Apfelweinproblem

Zeit:	2 – 4 h
Web-Code:	8twg hb3e

Ein Bauer kaufte mit seinem Freund ein 8-Liter-Fass, gefüllt mit bestem Apfelwein. Sie wollten den Wein gerecht aufteilen, besaßen aber nur einen 5-Liter-Krug und einen 3-Liter-Krug. Weder am Fass noch an den Krügen sind Markierungen angebracht. Dennoch gelang ihnen die Aufteilung.
Erstellen Sie ein Programm, welches verschiedene zufällige Reihenfolgen von Umschüttungen so lange durchspielt, bis die gerechte Aufteilung gefunden ist.

Ihr Programm startet mit dem vollen 8-Liter-Fass und den leeren Krügen. Ein Umschüttvorgang wählt nun zufällig zwei Gefäße (Fass oder Krug) und

a) leert das eine ganz oder
b) füllt das andere bis zum Rand.

Sobald sich in einem der Gefäße 4 Liter befinden, endet die Simulation. Geben Sie die Anzahl der Umschüttungen aus.
Bemerkung: In Aufgabe 10.19 wird ein Algorithmus zur Lösung dieses Problems angegeben. Dieser findet jedoch oft nicht die Variante mit den wenigsten Umschüttungen.

Aufgabe 10.19 Umschütten

Zeit:	1 h
Web-Code:	dob2 r5tp

Gegeben sind zwei Gefäße mit 15 bzw. 7 Liter Fassungsvermögen. Beide Gefäße sind anfänglich leer und weisen keinerlei Markierungen auf. Aus einem Brunnen kann beliebig oft Wasser nachgefüllt werden. Nun sollen 11 Liter mit diesen beiden Gefäßen abgemessen werden. Programmieren Sie nach folgendem Algorithmus eine Lösung, welche die Zwischenresultate und Zwischenschritte auch auf der Konsole ausgeben kann:

1. Beide Gefäße leeren (sofern nicht schon geschehen).
2. Solange keines der beiden Gefäße die gesuchte Menge (hier 11 Liter) aufweist, werden die folgenden drei Schritte ausgeführt. Wenn jedoch eines der beiden Gefäße (hier nur beim 15-Liter-Gefäß möglich) die gesuchte Menge enthält, wird das Programm mit einer Meldung «Heureka» und den Inhaltsangaben beider Gefäße abgebrochen.
   a) Ist das größere der beiden Gefäße **randvoll** (hier 15 Liter), so geben wir «Großes Gefäß leeren» aus, und wir geben den Inhalt der beiden Gefäße aus, nachdem der Inhalt des großen Gefäßes auf Null gesetzt wurde.

b) Ist das kleinere der beiden Gefäße (hier 7 Liter) **ganz leer**, so wird es gefüllt. Ausgabe: «Fülle kleines Gefäß», und wieder geben wir den Inhalt beider Gefäße nach der Operation aus.
c) Schütte den Inhalt des kleineren Gefäßes ins größere. Dabei wird nur so viel eingeschüttet, dass das größere Gefäß nicht überschwappt. Ausgabe auf der Konsole: «Schütte x Liter aus dem kleinen Gefäß ins große Gefäß.» Danach soll abermals der neue Inhalt beider Gefäße ausgegeben werden.

Die Gefäße sollen durch Variable symbolisiert werden. Das Anzeigen der beiden Inhalte kommt so oft vor, dass sich eine Subroutine anbietet.

Zusatzaufgabe 1: Ergänzen Sie die Aufgabe so, dass die Gefäßgrößen und die Zielmenge vom Anwender eingegeben werden können. Beachten Sie, dass Sie mit einem Gefäß von 9 Litern und einem Gefäß von 6 Litern niemals eine Menge von 5 Litern abmessen können. Sie können jedoch Vielfache des größten gemeinsamen Teilers der beiden Gefäßgrößen erzielen.

Zusatzaufgabe 2: Falls Sie Aufgabe 10.18 bereits gelöst haben, können Sie nun untersuchen, welches der beiden Verfahren zur besseren Lösung kommt.

Aufgabe 10.20 Monte-Carlo-Methode zur Bestimmung von Pi

Zeit:	1 h
Web-Code:	rvf2 f92q

Schätzen Sie Flächeninhalte mit der Monte-Carlo-Methode. Dabei wird der Inhalt einer Fläche angenähert, indem sehr viele zufällige Punkte in eine größere (bekannte oder leicht messbare) Fläche eingestreut werden, welche die zu messende Fläche als Teilfläche enthält. Die erste Teilaufgabe soll denn auch die Kreisfläche schätzen. Dazu wird einem Quadrat ein Kreis einbeschrieben. Nehmen Sie dazu das Quadrat mit Seitenlänge 2 und den Eckpunkten $(-1|-1)$, $(-1|1)$, $(1|1)$ und $(1|-1)$. Hier wird der Kreis mit Radius 1 um den Nullpunkt $(0|0)$ einbeschrieben.

Machen Sie dazu eine Zeichnung.
Die Funktion `random()` liefert eine Pseudozufallszahl zwischen 0.0 und 1.0. Zeigen Sie zunächst, dass der Punkt `(2 * random() - 1 | 2 * random() - 1)` im Inneren des Quadrats liegt. Weiter liegt ein Punkt (x|y) innerhalb des Kreises, falls sein Abstand zum Nullpunkt kleiner als 1 ist:

$$\sqrt{x^2 + y^2} < 1$$

Nun setzen wir zwei Zählervariable je auf 0: `imQuadrat := 0; imKreis := 0`
In einer Schleife erzeugen wir pro Durchlauf je einen Zufallspunkt im Inneren des Quadrats. Jetzt zählen wir bei der Variable `imQuadrat` eins dazu, und dies tun wir analog bei der Variable `imKreis`, aber nur, sofern der Punkt im Inneren des Kreises liegt. Nach 10 000 Iterationen bilden wir den Quotienten `imKreis / imQuadrat` und erhalten einen Viertel der Kreiszahl PI, denn der Flächeninhalt des Quadrats beträgt 4 und der Flächeninhalt des Kreises (πr^2) beträgt π.
Zusatzaufgabe: Berechnen Sie nach demselben Verfahren die Fläche unter der Parabelkurve $y = x^2$ im Intervall 0.0 bis 1.0. Mathematisch berechnet müsste das Resultat bei 1/3 liegen. Suchen Sie das Quadrat und entscheiden Sie, mit welcher Berechnung Sie herausfinden, ob ein Punkt unter oder über der Kurve liegt. Zählen Sie wieder 10 000 Punkte.

Aufgabe 10.21 Räuber-Beute-Verhalten (Lotka/Volterra)

Zeit:	2 h
Web-Code:	47gv shif

Eine deutsche Formulierung der bekannten Räuber-Beute-Beziehung finden wir auf `www.mathekiste.de`. Zitat:
In den meisten Regionen der Erde ist die Nahrungskette der verschiedenen Tierarten ein sehr komplexes Gefüge. Jeder Räuber frisst mehrere Beutearten, und jede Beuteart dient mehreren Räuberarten als Nahrung. Relativ einfach wird die Untersuchung nur bei artenarmen Ökosystemen. Ein solches Ökosystem besteht auf Neufundland mit nur 14 heimischen Säugetierarten. Dort stellt man bei der Anzahl der Luchse (Räuber) und der Schneeschuhhasen (Beute) periodische

Schwankungen fest. Die Anzahl der von der Hudson's Bay Company gefangenen Hasen und Luchse schwankte von 1850 bis 1900 periodisch im Zehnjahresrhythmus.
Die Anzahl der Luchse (`L` bzw. `Lalt` und `Lneu`) ändert sich in dem betrachteten Zeitintervall Δ`t` um Δ`L`. Der Zuwachs der Luchse ist proportional zur Anzahl der vorhandenen Luchse und, da ein höheres Nahrungsangebot eine höhere Geburtenanzahl mit sich bringt, auch proportional zur Anzahl der vorhandenen Hasen (`H` bzw. `Halt` und `Hneu`), also insgesamt proportional zum Produkt `L*H`.
Die Zuwachs- bzw. Sterberaten (Proportionalitätskonstanten) werden mit `fzu` bzw. `fab` bezeichnet. Für Luchse bzw. Hasen also `fzuHL`, `fabL` bzw. `fzuH` und `fabHL`.
Gleichzeitig wird im Zeitintervall Δ`t` (`alt` $\rightarrow$ `neu`) ein Teil der Luchse sterben (Feinde haben sie keine). Die Anzahl der sterbenden Luchse ist proportional zur Anzahl der vorhandenen Luchse. Damit ist die gesamte Änderung der Luchse
Δ`L = fzuHL * H * L - fabL * L`
und nach dem Zeitintervall Δ`t` ist die neue Anzahl der Luchse
`Lneu = Lalt +` Δ`L = Lalt + fzuHL * H * L - fabL * L`
während die Anzahl der Hasen sich wie folgt verändert:
`Hneu = Halt +` Δ`H = Halt + fzuH * H - fabHL * H * L`

Schreiben Sie ein Programm, welches obiges Räuber-Beute-Verhalten simuliert. Tipp: Es ist mit ganzen Zahlen eine grosse Herausforderung, eine stabile Lösung zu finden. Verwenden Sie daher sowohl für die Proportionalitätskonstanten, wie auch für die Anzahl Tiere gebrochene Zahlen (`real`).

Aufgabe 10.22 **Conways Game of Life**

Zeit:	2–4 h
Web-Code:	ch5c 67pm

Conways[1] *Game of Life* ist eine Populations-Simulation, die in einem Raster (beliebig großes Schachbrett) Zellen überleben, sterben oder neu bilden lässt.

[1] John H. Conway, englischer Mathematiker (*1937)

Das Überleben oder Neubilden von Zellen hängt allein von ihren umliegenden acht Quadraten ab.

Es gelten die folgenden Regeln:

- Wenn eine Zelle höchstens eine Nachbarzelle hat, stirbt sie an **Vereinsamung**. Das heißt, sie wird nicht in die nächste Generation übernommen.
- Wenn eine Zelle vier oder mehr Nachbarzellen hat, so stirbt sie an **Übervölkerung**. Das heißt, auch sie wird nicht in die nächste Generation übernommen.
- Hat eine Zelle zwei oder drei Nachbarzellen, so überlebt sie. Das bedeutet, sie wird in die nächste Generation übernommen.
- Wenn ein leeres Quadrat in den umliegenden acht Quadraten genau drei Nachbarzellen hat, dann entsteht dort durch **Vermehrung** eine neue Zelle in der nächsten Generation.

Schreiben Sie nun ein Programm, das ein 20 x 20-Raster Boole'scher Werte mit zufälligen Zellen füllt. (Eine Wahrscheinlichkeit von 25 %, dass ein Quadrat eine Zelle enthält, hat sich in Tests als spannend erwiesen.)
Programmieren Sie den «Überlebensschritt» nach obigen Regeln und geben Sie einige Generationen aus.

Aufgabe 10.23 Bowlingkugel

Zeit:	2 h
Web-Code:	sor4 5wf7

Die Aufgabe besteht darin, in einem 100-stöckigen Haus das Stockwerk zu bestimmen, aus welchem eine Bowlingkugel den freien Fall gerade noch übersteht. Sie haben für die Versuche zwei identische Kugeln zur Verfügung, welche beide in Bruch gehen dürfen. Nachdem die zweite Kugel defekt ist, müssen Sie das Stockwerk angeben können. Ziel ist es, möglichst wenige Fallversuche durchführen zu müssen. Je nach Material der Kugeln gehen diese bereits in einem unteren Stockwerk oder im extremen Fall gar nicht kaputt.

Schreiben Sie ein Programm, das mit möglichst wenigen Versuchen das gesuchte Stockwerk ermittelt. Legen Sie zu Beginn des Programms das Stockwerk, bei welchem die beiden Kugeln zerstört werden, per Zufallszahl zwischen 1 und 100 fest.

Aufgabe 10.24 Testdaten mit Reservoir (Sampling)

Zeit:	1 h
Web-Code:	x8kv r9ij

Beim Aussuchen von Testdaten aus einer Datenmenge kann man sehr einfach vorgehen. Wir suchen aus einer gegebenen Menge mit n Elementen t Testdaten ($t \leq n$) folgendermaßen: Wir mischen die gegebene Menge und wählen danach die ersten t Elemente aus der gemischten Menge aus.
Ist jedoch n (die gegebene Menge aller Datensätze) so **groß**, dass die Daten nicht im Hauptspeicher Platz finden, oder ist n überhaupt **nicht bekannt,** da die gegebene Datenmenge nur Datensatz um Datensatz eintrifft, so bietet sich folgende **Methode mit Reservoir** an:

1. Erstellen Sie ein Array mit t Plätzen. Das ist das Reservoir, das zunächst gefüllt werden muss; am Ende enthält das Reservoir die gesuchte Testdatenmenge.
2. Die ersten t Daten werden einfach ins Reservoir geschrieben (ist $n = t$, so sind wir fertig).
3. Für jeden weiteren k-ten Datensatz ($t < k \leq n$) wird eine Zufallsposition p zwischen 1 und k (bzw. zwischen 0 und $k - 1$, falls das Reservoir ab 0 indexiert ist) erzeugt. Ist $p \leq t$ (bzw. $\leq t - 1$), so wird der Datensatz an Stelle p ins Reservoir geschrieben. Ist p jedoch größer als t (bzw. $t - 1$), so wird der Datensatz ignoriert.

Erklärung: Ist $t = n$, so ist die Sache trivial. Jeder weitere Datensatz ($k = n + 1$) wird nur mit Wahrscheinlichkeit (t/k) ins Reservoir einsortiert. Der Beweis kann leicht über die vollständige Induktion geführt werden. Beweisen Sie zunächst (Induktionsanfang) den Fall $n = t + 1$.

11. Rekursion

Als Einstiegsbeispiel in die **Rekursion** soll das Sortierverfahren Merge-Sort[1] dienen. Dabei wird eine Menge

```
4, 9, 1, 8, 7, 3, 6, 8, 2, 3, 8
```

in zwei Teile aufgespalten:

```
4, 9, 1, 8, 7
3, 6, 8, 2, 3, 8
```

Jede dieser Teilmengen wird für sich sortiert:

```
1, 4, 7, 8, 9
2, 3, 3, 6, 8, 8
```

Die beiden Mengen werden nun zusammengemischt *(merge)*:

```
                    ...7, 8, 9
1, 2, 3, 3, 4,...
                    ...6, 8, 8
```

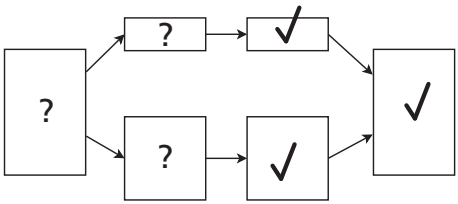
Viele Probleme lassen sich für große Datenmengen nicht auf einfache Art lösen, aber sie lassen sich auf eine einfacher zu lösende Problemstellung reduzieren. Wenn es gelingt, aus jeder Teilproblemstellung eine analoge Form mit einfacheren Werten zu finden, können wir induktiv denselben Algorithmus immer und immer wieder anwenden. Dieses Vorgehen wird **Rekursion** genannt.
Jedes rekursive Vorgehen braucht eine Abbruchbedingung. Diese ist jeweils gegeben, wenn das Problem trivialerweise lösbar bzw. unlösbar ist; oder dann, wenn das Problem bereits gelöst ist. Für den Merge-Sort ist diese Abbruchbedingung gegeben, sobald die zu sortierende Menge nur noch aus einem einzigen Element besteht.

11.1 Divide et Impera

Findet sich bei großen Datenmengen ein Algorithmus, der die Datenmenge oder die Problemstellung in kleinere Happen aufteilt, so sprechen wir oft vom

[1] engl. *to merge* = zusammenfügen, eins werden.

«Teile und herrsche»-Prinzip (*divide et impera* od. engl. *divide and conquer*)[1]. Meistens sind diese Probleme durch Rekursion leicht lösbar.



Beispiele:

- Merge-Sort
- Quicksort
- Kürzester Weg
- …

11.2 Aufgaben

Aufgabe 11.1 **Merge-Sort**

Zeit:	2 h
Web-Code:	puyx no2s

Schreiben Sie ein Programm, das die folgende Zahlenreihe sortiert. Verwenden Sie das in der Theorie beschriebene Merge-Sort-Verfahren:
4, 9, 1, 8, 7, 3, 6, 8, 2, 3, 8

[1] Sūntzǐ (um 500 v. Chr.) beschreibt in seinem Werk «Die Kunst des Krieges» eine Kriegskunst folgendermaßen: Teile Deine Armeen auf, um eine große Streitmacht zu führen.

Aufgabe 11.2 Fibonacci

Zeit:	2 h
Web-Code:	xqh4 mv36

Beim Versuch, eine Kaninchenpopulation vorauszusagen, benutzte Fibonacci[1] die nach ihm benannte Folge. Es stellte sich heraus, dass diese mathematische Folge eine sehr gute Näherung der Wirklichkeit war.
Die Fibonacci-Folge liest sich wie folgt: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, … und bezeichnet die Anzahl der Kaninchenpaare in jedem Monat. Dabei ist jeder Eintrag die Summe der beiden Vorgänger (bitte nachprüfen). Schreiben Sie eine rekursive Funktion zum Berechnen des i-ten Eintrags in der Folge. Für i = 1 und i = 2 liefert die Funktion einfach 1. Ist jedoch i > 2, so liefert sie einfach die Summe der beiden Vorgänger.
Option: Schreiben Sie die Funktion auch iterativ und vergleichen Sie die Laufzeiten mittels dem Auslesen der Systemzeit.

Aufgabe 11.3 ggT rekursiv

Zeit:	1 h
Web-Code:	ctnh j2cr

Berechnen Sie den größten gemeinsamen Teiler (ggT) zweier Zahlen. Gehen Sie diesmal rekursiv nach den folgenden Aussagen vor. Der ggT von zwei positiven ganzen Zahlen a und b ist:

- b, wenn bei der Division von a durch b kein Divisionsrest auftritt bzw.
- ggT(b, a % b) im anderen Fall.

Vergleichen Sie Ihre Lösung mit Aufgabe 5.8.

[1] Eigentlich Leonardo da Pisa (in Pisa um 1180 – 1250); bedeutender Mathematiker des Mittelalters

Aufgabe 11.4 Lotto

Zeit:	1 h
Web-Code:	epsk xisf

Um die Wahrscheinlichkeit zu berechnen, ob man bei der Lottoziehung 6 Richtige trifft, ist es nötig zu wissen, auf wie viele Arten überhaupt ein Lottoschein ausgefüllt werden kann. Nur eine einzige Möglichkeit entspricht dem Volltreffer. Dazu haben sich Mathematiker die Köpfe zerbrochen und sind mit folgender Formel an die Öffentlichkeit gelangt:
Die Funktion b(n | k) bezeichne die Anzahl Möglichkeiten, aus n Objekten k auszuwählen. Im Schweizer Zahlenlotto z. B. wäre also die Frage nach b(45 | 6). Es gelten die folgenden Definitionen:

- b(n | k) = Anzahl Möglichkeiten, k Objekte aus n auszuwählen
- b(n | 1) = n (Jedes Objekt kann einzeln ausgewählt werden. Dazu gibt es genau n Möglichkeiten.)
- b(a | a) = 1 (Es gibt nur eine Möglichkeit, alle Objekte zu wählen!)
- b(n | k) = b(n – 1 | k) + b(n – 1 | k – 1)

Mögliche einleuchtende Erklärung für die letzte Formel: Beim Ankreuzen von k Elementen auf n Plätzen gibt es zwei Arten von Möglichkeiten. Die erste Kategorie hat das erste Element angekreuzt. Dazu gibt es noch b(n – 1 | k – 1) Möglichkeiten, die anderen Felder anzukreuzen. Man bedenke, dass dabei ein Element weniger auf die verbleibenden Plätze verteilt werden kann. Die zweite Kategorie hat das erste Element nicht angekreuzt. Dazu gibt es b(n – 1 | k) Möglichkeiten, da noch alle k Elemente auf die anderen (n – 1) Plätze verteilt werden können.
Schreiben Sie die Funktion b(n, k), indem Sie zunächst die beiden trivialen (einfachen) Fälle berechnen. Sollte es sich aber nicht um einen einfachen Fall handeln, so rufen Sie die Funktion rekursiv nach der letzten angegebenen Formel auf. Berechnen Sie danach die Wahrscheinlichkeit, einen Lotto-Sechser zu erzielen b(45 | 6) oder die *Euro-Millions* zu gewinnen: b(50 | 5) ∗ b(9 | 2).
Option: Wie viel muss im Jackpot sein, damit sich ein Mitspielen lohnt – sofern natürlich niemand anders auch den Jackpot knackt?

Aufgabe 11.5 Unterverzeichnisse ausgeben

Zeit:	2 h
Web-Code:	udy5 egzz

Suchen Sie in Ihrer Programmiersprache eine Möglichkeit, das aktuelle Verzeichnis («Current working Directory», auch «Parent working Directory» genannt) auszulesen und alle darin enthaltenen Dateien und Verzeichnisse in Erfahrung zu bringen. Geben Sie diese Liste für einige Verzeichnisse aus.
Suchen Sie eine Möglichkeit zu entscheiden, ob es sich bei einem Eintrag um eine Datei oder um ein Verzeichnis handelt. Markieren Sie in der Ausgabe die Verzeichnisse mit einem «+» und die Dateien mit einem «–».
Geben Sie zuletzt die Verzeichnisse rekursiv (mit allen Unterverzeichnissen) in folgender Form aus:

```
- README.TXT
- Journal.doc
+ Aufgaben
  + Aufgabe_Muenzen
    - Muenzen.java
    - Muenzen.class
  + Aufgabe_Files
    - Files.py
  - README.TXT
- Screenshot.JPG
```

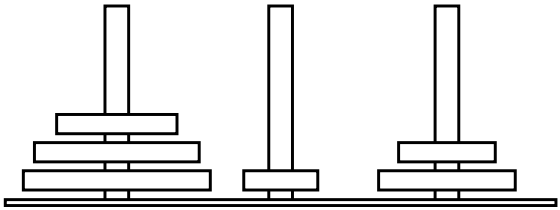
Aufgabe 11.6 Türme von Hanoi

Zeit:	1 h
Web-Code:	fk8y npoa

Dieses Spiel[1] besteht aus drei Stäben A, B und C, auf welche mehrere gelochte Scheiben gelegt werden. Alle Scheiben sind verschieden groß. Zu Beginn liegen

[1] Das Spiel der «Türme von Hanoi» wurde erstmals publiziert vom Mathematiker Édouard Lucas (1842 – 1891)

alle Scheiben der Größe nach auf Stab A (kleinste Scheibe oben). Ziel des Spiels ist es, den kompletten Stapel von A nach C zu versetzen.



Bei jedem Zug darf nur die oberste Scheibe eines beliebigen Stabes auf einen der beiden anderen Stäbe gelegt werden – vorausgesetzt, dort liegt nicht schon eine kleinere Scheibe. Folglich sind zu jedem Zeitpunkt des Spiels die Scheiben auf jedem Stab der Größe nach geordnet.
Erstellen Sie ein Programm, welches die einzelnen Schritte zum Verschieben eines aus fünf Scheiben bestehenden Turmes von A nach C beschreibt.

Aufgabe 11.7 Kamele beladen

Zeit:	4 – 8 h
Web-Code:	6gdd r4zm

Ein Kamel soll optimal beladen werden. Das Kamel kann maximal 270 kg tragen. Aktuell sind Waren mit den folgenden Gewichten zu transportieren: 5, 18, 32, 34, 45, 57, 63, 69, 94, 98 und 121 kg. Nicht alle Gewichte müssen verwendet werden; die 270 kg sollen aber möglichst gut, wenn nicht sogar ganz ohne Rest beladen werden. Die Funktion

```
beladeOptimal(kapazitaet: integer, vorrat: integer[]): integer[]
```

erhält die maximal tragbare Last (`kapazitaet`) und eine Menge von aufzuteilenden Gewichten (`vorrat`). Das Resultat (hier `integer[]`) ist die Auswahl aus dem Vorrat, die der Belastbarkeit möglichst nahe kommt. Gehen Sie wie folgt

rekursiv vor: Für jedes vorhandene Gewicht `g` aus dem Vorrat soll das Problem vereinfacht werden. Dazu wird dieses Gewicht probehalber aufgeladen:

```
tmpLadung:  integer[]
tmpLadung := beladeOptimal(kapazitaet - g, "vorrat ohne g")
```

Danach wird das beste Resultat `tmpLadung + g` gesucht und als Array zurückgegeben. Behandeln Sie in der Methode `beladeOptimal()` zunächst die Abbruchbedingungen:

- Vorrat leer
- alle vorhandenen Gewichte sind zu schwer
- nur noch ein Gewicht im Vorrat

Zusatzaufgabe: Informieren Sie sich zu iterativen Lösungen zum «Rucksackproblem». Implementieren Sie diese Lösung für obiges Problem und diskutieren Sie die Lösung und den Lösungsweg.

Aufgabe 11.8 Schnurlängen

Zeit:	1 h
Web-Code:	acgo 5ap7

Eine Schnur mit einer Gesamtlänge von 450 m soll in Teilstücke der Länge 17 m, 19 m und 21 m geteilt werden. Ist das ohne Rest möglich? Die Frage reduziert sich darauf, ob eine der drei Differenzen

- 450 – 17 = 433,
- 450 – 19 = 431 und
- 450 – 21 = 429

ohne Rest aufgeschnitten werden kann. Denn wenn dies für eine der genannten drei Differenzen möglich ist, so ist es sicher auch für 450 m möglich. Schreiben Sie eine Methode `zerlegbar(gesamt, laenge1, laenge2, laenge3)`, die im Wesentlichen prüft, ob eine der drei Bedingungen

- `zerlegbar(gesamt-laenge1, laenge1, laenge2, laenge3)`,
- `zerlegbar(gesamt-laenge2, laenge1, laenge2, laenge3)` oder
- `zerlegbar(gesamt-laenge3, laenge1, laenge2, laenge3)`

zutrifft.

Aufgabe 11.9 Dichtestes Punktpaar

Zeit:	4 – 8 h
Web-Code:	a93s 2si4

Gegeben sind 10 000 Punkte in derselben Ebene. Schreiben Sie ein Programm, das von diesen Punkten das Paar findet, dessen Punkte den geringsten Abstand voneinander haben.
Anstatt jeden Punkt mit jedem anderen zu vergleichen, ist ein rekursives Vorgehen angebracht. Dabei wird die Ebene zunächst unterteilt, danach suchen wir (rekursiv) in der ersten und in der zweiten Teilebene die nächsten Nachbarn. Nun brauchen wir noch zu überprüfen, ob sich nahe der Grenzlinie der beiden Ebenen nicht noch nähere Punkte befinden.
Ein Rekursionsabbruch kann bei einer Menge von drei oder weniger Punkten erfolgen.
Laden Sie die Punkteliste zum Testen mit Web-Code herunter: www.programmieraufgaben.ch.

Aufgabe 11.10 Malermeister Malcom

Zeit:	4 – 8 h
Web-Code:	durv de6t

Malermeister Malcom erhält den Auftrag, eine Wand mit runden Blumenblüten zu schmücken.
Die Wand misst 2000 cm x 1000 cm (Breite x Höhe). Vorgegeben sind 10 000 mögliche Mittelpunkte und zugehörige Kreisradien, in denen überhaupt Blumen gezeichnet werden dürfen.

Der Maler will keine Überschneidungen und wählt nun folgendes Verfahren: Er betrachtet je zwei Kreise. Schneiden sie sich nicht, so behält er beide. Schneiden sie sich, so verwirft er den größeren der beiden Kreise. Es ist ihm bewusst, dass er damit nicht das Optimum der Fläche mit Blumen abdeckt. Da es aber 10 000 Möglichkeiten gibt, bleiben ihm immer noch genügend Blüten zu zeichnen.
Bei genauerer Betrachtung bemerkt er, dass er nun ca. 50 000 000 Vergleiche anstellen muss (jeder mit jedem!), und halbiert diese Komplexität, indem er zuerst den linken Teil der Wand betrachtet (ca. 5000 Kreise) und danach den rechten Teil (auch ca. 5000 Kreise). Zuletzt betrachtet er nur noch die Kreise, welche die Mittellinie der Wand überlappen. Das Problem hat sich von 50 000 000 Vergleichen auf etwas mehr als 25 000 000 Vergleiche halbiert.
Er beschließt nun, die Wand in so viele Teilstücke zu zerlegen, bis in jede Teilfläche zehn oder weniger Kreise zu liegen kommen. Innerhalb dieser Teilflächen vergleicht er nun jeden Kreis mit jedem, was ihn maximal 45 Vergleiche kostet. Nun braucht er nur noch die Kante oben, unten, rechts und links auf Überschneidungen zu prüfen.
Lösen Sie Malermeister Malcoms Problem mittels Programm. Laden Sie die Liste der Punkte mit Web-Code herunter: www.programmieraufgaben.ch oder generieren Sie selbst 10 000 Kreise im Rechteck (0,0), (2000, 1000). Wenn Sie das Problem rekursiv lösen, machen Sie sich Gedanken über die Abbruchbedingung.

Lösungen und Verifikationen

1 Ausdrücke und Datentypen

Aufgabe 1.2 **Berechnung von Ausdrücken (1)**

Web-Code: 8kqe uout

- $7 + 3 \cdot 5 \rightarrow 22$
- $16 \,\%\, 3 \rightarrow 1$
- $4 \cdot 3 + (3 + 4) \cdot 2 \rightarrow 26$
- $1 \,\%\, 15 \rightarrow 1$

Aufgabe 1.3 **Berechnung von Ausdrücken (2)**

Web-Code: mupd pv9h

- $(-x) + (2 - 1 - 1) \cdot 6 \rightarrow 7$
- $a + x \cdot a \rightarrow -30$
- $a \,\%\, (x + 9) \rightarrow 1$
- $a = x + 12 = 5 \rightarrow$ `true`

Aufgabe 1.4 **Bits**

Web-Code: xjyi extf

Lösung: Aufgerundeter Zweierlogarithmus
Anzahl Bits = Aufrunden(Log_2 (Anzahl Zustände))

- $365 \rightarrow 9$
- $3 \rightarrow 2$
- $9 \rightarrow 4$
- $2000 \rightarrow 11$
- $5000 \rightarrow 13$
- $100\,000 \rightarrow 17$

Aufgabe 1.5 **Binäre Darstellung**

Web-Code: i55j 5v5u

a) 1110, 10 0001, 11 0110, 111 0100, 11 1110 1000, 1110 1001 1001

b) 9, 14, 75, 366, 13 181

Aufgabe 1.6 **Spielkarten**

Web-Code: 8mso qivy

Sechs Bits, ob mit oder ohne Joker. Mit 6 Bits sind 64 Zustände möglich.
1 – 13 (14 – 26, 27 – 39, 40 – 52) Farbkarten, 53 – 55 = Joker

Aufgabe 1.7 **Go-Spiel**

Web-Code: 9c7m tjxk

Eine Position kann als 361-stellige Zahl im 3er-System aufgefasst werden. Mit einer 361-stelligen Zahl im 3er-System können alle Zahlen von 0 bis $3^{361}-1$ dargestellt werden. Das sind $1.74 * 10^{172}$ Zahlen. Der Zweierlogarithmus davon liefert 572.2. Somit sind 573 Binärziffern (= 72 Byte) nötig, um jede Zahl von 0 bis $3^{361}-1$ darzustellen. Ihr Programm könnte dazu eine Umwandlung vom 3er- ins 2er-System und zurück verwenden.

Aufgabe 1.8 **Binäre Verschiebung**

Web-Code: 8uxd zron

Am Beispiel 7: 0000 0111 um zwei Stellen nach links
→ 0001 1100 (= 28)
Eine Verschiebung um 6 Stellen nach links multipliziert die Zahl um den Faktor 64. Eine Verschiebung um eine Stelle nach rechts halbiert die Zahl.

Aufgabe 1.9 **Zweierkomplement**

Web-Code: kjb8 rjjm

a) –21, –18, –36

b) 1111 1010, 1111 1000, 11 1110 1111, 1001 1111, 0000 0101

c) Verifikation trivial.

Aufgabe 1.10 **UTF-8**

Web-Code: v9hy tsk4

- 64 (@-Zeichen)
- 80 (großes P)
- 194, 190 (¾)
- 195, 139 (Ë)
- 239, 172, 129 (fi-Ligatur)
- 226, 137, 136 (Ungefähr-Zeichen: ≈)

2 Anweisungen und Abfolgen

Aufgabe 2.1 **Hello World**

Web-Code: hr5f yysf

Das Programm zeigt «Hello World» an.

Aufgabe 2.2 **Anweisungen und Ausdrücke**

Web-Code: qw25 yppb

5 Anweisungen.

14 Ausdrücke: input(), 44, a, 44+a, 3, x, 3∗x, 44+3∗x, 4, y, 4∗y, a∗x, a∗x+4∗y, z

Aufgabe 2.3 **Variablenwert nach einer Sequenz**

Web-Code: pi2t 72oe

Resultat = 4 für alle Eingaben y.

Aufgabe 2.4 **Ohm'sche Gesetze**

Web-Code: po9n fjn6

R1 = 40 Ohm und R2 = 50 Ohm

a) 90 Ohm

b) 22.22…Ohm

Aufgabe 2.5 **Linsengleichung (Optik)**
Web-Code: 2v7c bobb
(b, g) = (8, 12) → f = 4.8

Aufgabe 2.9 **Ganzzahlarithmetik (Kommutativgesetz)**
Web-Code: rdhf 4kvg
r1 = 90; r2 = 92 (Ganzzahlarithmetik)

Aufgabe 2.10 **Gleitkommaarithmetik (Assoziativgesetz)**
Web-Code: kwhz einv
d1 = 1.0; d2 = 0.0 (Genauigkeit der Fließkommazahlen)

3 Selektion (Verzweigung)

Aufgabe 3.1 **IF-Formulierungen**
Web-Code: gxi7 afay

a) `y > 0`
b) `-2 < a AND a < 5.3`
c) `preis < 0.05 * 20`
d) `x < 5 AND 0 ≠ x MOD 2`
e) `-3 ≤ x AND x < 8`

Aufgabe 3.2 **Boole'sche Ausdrücke**
Web-Code: tby9 5gzh

a) `false`
b) `Antworten:`
- $17 < 19 \rightarrow$ `true`
- $-3 < -8 \rightarrow$ `false`

- $(4 > 3)$ & $(3 > 4)$ → `false`
- $(8 \,\%\, 3 = 0)$ | $(8 \,\%\, 2 = 0)$ → `true`
- $(-13 < -8)$ | $(-1 < 0)$ & $(2 > 3)$ → `true`
- `true` | `false` = $(5 < 2 * 3)$ & (`false` & `true`) → `false`
- $a < x$ → `false`
- $-x < -a - 2$ → `false`

c) `true`, `true`, `false`, `true`, `false`

d)
- `(! a) & (! b)`
- `!(!(a) & !(b & c))`

Aufgabe 3.3 **Was wird ausgegeben?**

Web-Code: wp6s 39bx

x = 12 (2. Selektion liefert `false`, da 1 < 1 nicht wahr ist)

Aufgabe 3.4 **Selektionsinvariante**

Web-Code: 3t94 v5nz

Die Programmanweisung `a := einlesen()` kann nach der Selektion stehen (post-invariant).

Aufgabe 3.8 **Hundert Binärzahlen**

Web-Code: an5r qqzp

- 3 → 000 0011
- 5 → 000 0101
- 10 → 000 1010
- 80 → 101 0000

Aufgabe 3.9 **Kilobyte**

Web-Code: nk4t g9hc

2 kB = 0.001 907 349 MiB

Aufgabe 3.10 **Elektrische Spannung**

Web-Code: 43b5 76us

- 44 Volt, 33 Ampère → 1.333 Ohm
- 44 Volt, 50 Ohm → 0.8800 Ampère
- 36 Ampère, 50 Ohm → 1800 Volt

4 Schleifen

Aufgabe 4.4 **Fahrenheit**

Web-Code: kcyy 2mt8

- −20° Celsius = −4° Fahrenheit
- 0° Celsius = 32° Fahrenheit
- 100° Celsius = 212° Fahrenheit

Aufgabe 4.5 **Domino**

Web-Code: 3ika 8b3t

Das Programm liefert 28 verschiedene Teile.

Aufgabe 4.7 **Schleifenabbruch**

Web-Code: m7iq 4iy5

c) und d) brechen ab.

Aufgabe 4.9 **Schleifeninvariante**

Web-Code: erdg qc8i

In der Scheife stehen nach der Umformung lediglich zwei Anweisungen.

Aufgabe 4.10 Zahlensumme (1)

Web-Code: 3uy9 4fbo

Die Summe der ersten 2011 Zahlen ist 2 023 066.

Aufgabe 4.12 Fakultäten

Web-Code: neh4 hqjt

- $20! \approx 2.43 \times 10^{18}$
- $30! \approx 2.65 \times 10^{32}$

Aufgabe 4.14 Sinusfunktion

Web-Code: g5cx nzxg

sin(0.8) = 0.7173561…
sin(0.3) = 0.2955202…
Das Programm sollte bereits bei fünf Iterationen ca. sieben Nachkommastellen liefern.

Aufgabe 4.15 Zahlenspiel

Web-Code: vg42 rghx

Die 1000. Zahl, die mit den Ziffern 9876 beginnt, lautet 9 876 888.

Aufgabe 4.16 Mondlandung

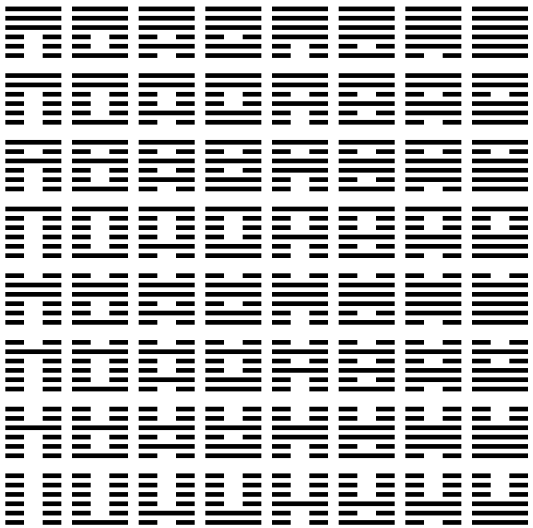
Web-Code: y4qj nv2g

Bei der Landung beträgt die «Geschwindigkeit» 49.5 Pixel pro Zeiteinheit.

Aufgabe 4.18 Das Buch der Weissagungen

Web-Code: 95ew 4rif

Alle 64 Hexagramme:



Aufgabe 4.19 Parallelschaltung Ohm'scher Widerstände

Web-Code: eisv 86s7

Parallelschaltung von

- 20, 30, 40 und 50 Ohm ergeben 7.7922… Ohm.
- 30, 30, 470, 1000, 2000 Ohm ergeben 14.2259… Ohm.

Suchmaschinen wie `www.wolframalpha.com` und `www.bing.com` erlauben direkt die Eingabe der Formel: 1/R = 1/20 + 1/30 + 1/40 + 1/50

Aufgabe 4.22 FizzBuzz (1)

Web-Code: pf6d wqtr

Bei der Zahl 35 erscheint zum ersten Mal der Text «fizzbuzz».

Aufgabe 4.23 **FizzBuzz (2)**

Web-Code: 8sax at2x

Bereits bei der Zahl sieben erscheint das erste Mal «fizzbuzz».

Aufgabe 4.24 **Waage**

Web-Code: d57z se4g

50 kg sind mit den Steinen 32 kg, 16 kg und 2 kg abzumessen.

5 Unterprogramme

Aufgabe 5.2 **RGB (1)**

Web-Code: 829e g8uz

Mittleres Grau: (r, g, b) = (0.5, 0.5, 0.5) liefert 8 421 504 (als `integer`)

Aufgabe 5.3 **RGB (2)**

Web-Code: 4zn2 dxzd

Verwenden Sie die Aufgabe RGB (1) zur Verifikation.

Aufgabe 5.4 **Quersumme**

Web-Code: wvfg ndmx

in 446 → out 14
in 12034 → out 10

Aufgabe 5.5 **Abstand**

Web-Code: tybn j6zi

P1 = (–4, 2); P2 = (–1, 6) → Abstand = 5
P1 = (–1, 6); P2 = (4, 18) → Abstand = 13
Zell und Neuburg liegen 12.26 Kilometer auseinander.

Aufgabe 5.10 **Bremsweg**

Web-Code: dh44 7qtd

50 km/h (trockene Straße) → 33.78 m
30 km/h (nasse Straße) → 20.68 m

Aufgabe 5.11 **Schaltjahre**

Web-Code: 4xw9 mhjm

1500 → true
1582 → false
1600 → true
1700 → false
1701 → false
2000 → true
2010 → false
2100 → false

Aufgabe 5.13 **Ewiger Kalender**

Web-Code: m2g9 636g

Datum	Wochentag
1. 1. 2001	Montag
31. 12. 2001	Montag
10. 12. 2037	Donnerstag
29. 2. 2088	Dienstag
1. 3. 2100	Montag

Aufgabe 5.14 Zahlenmuster

Web-Code: n8f6 73nj

muster(1) = 1
muster(2) = 2
muster(3) = 2
muster(1 000) = 45
muster(10 000) = 141
muster(1 000 000 000) = 44 721

Aufgabe 5.15 Frankengewinnspiel

Web-Code: qnfv rirb

a, b) In = 1: Out = 1; In = 2: Out = 3; In = 15: Out = 694
c) In = 63; Out = 105 098
d) In = 27; Maxfeld = 9232

Aufgabe 5.17 Codeverbesserungen (Refactoring)

Web-Code: f32t c578

Sie kommen neu mit total 8 statt 16 Anweisungen aus: 4 Aufrufe einer Subroutine mit 4 Anweisungen.

6 Felder

Aufgabe 6.7 Transponieren einer Matrix

Web-Code: wmjo fn97

Das Transponieren der Matrix

$$\begin{pmatrix} 3 & 5 \\ 4 & 2 \\ 7 & 8 \end{pmatrix} \quad \text{liefert:} \quad \begin{pmatrix} 3 & 4 & 7 \\ 5 & 2 & 8 \end{pmatrix}$$

Aufgabe 6.9 **Schriftsetzer**

Web-Code: p8vk 9x7k

Für ein Buch mit 250 Seiten werden folgende Lettern benötigt: 0: 45-mal. 1: 154-mal. 2: 105-mal. 3: 54-mal. 4: 54-mal. 5: 46-mal. 6: 45-mal. 7: 45-mal. 8: 45-mal. 9: 45-mal.

Aufgabe 6.10 **n-ter Tag im Jahr**

Web-Code: gqsc 8u29

`tagNr(1, 1)` → 1
`tagNr(1, 2)` → 32
`tagNr(31, 12, 2011)` → 365

Aufgabe 6.14 **Maximale Teilsumme**

Web-Code: sypi dsoh

Erste Verifikation: `-1, 2, -6, -14, 15, 12, -5, 0, 6, -6, 11, -4, -7, 1, -5, 1`
Die Teilsumme ergibt 33 (vom 5. bis zum 11. Element).
Zweite Verifikation: `31, -41, 59, 26, -53, 58, 97, -93, -23, 84`
Die Teilsumme ergibt 187.

Aufgabe 6.16 **Springer auf dem Schachbrett**

Web-Code: o9zn pwdy

Beispiel auf dem 4 x 4 Brett:

$$\begin{pmatrix} 3 & 0 & 3 & 2 \\ 2 & 3 & 2 & 1 \\ 1 & 2 & 1 & 4 \\ 2 & 3 & 2 & 3 \end{pmatrix}$$

Aufgabe 6.17 Exponententabelle

Web-Code: n2gz 2e3u

Das Resultat ist stark davon abhängig, wie die Potenzfunktion optimiert ist. Eine Tabelle kann bis zu 5-fache Geschwindigkeit liefern: «Doppelter Speicher = doppelte Geschwindigkeit».
Wird anstelle einer Potenz eine einfache Multiplikation oder Addition verwendet, kann meist nichts eingespart werden – im Gegenteil: Solche Tabellen verlangsamen das Programm, da das Nachschlagen in Tabellen selbst Multiplikationen und Additionen verwendet.

Aufgabe 6.18 Gleichungssystem

Web-Code: k43i 8vjt

Werden für m[0][0] = cos(α), m[0][1] = $-$sin(α), m[1][0] = sin(α) und m[1][1] = cos(α) eingesetzt, so beschreibt die Matrixmultiplikation gerade eine Drehung des Punktes (x, y) um den Winkel α.

7 Zeichenketten

Aufgabe 7.6 Geheimschrift MIX

Web-Code: 5z5i vp58

Streng geheim $\rightarrow$ Srn eemtegghi

Aufgabe 7.7 Cäsar

Web-Code: kb99 fe5k

Die Botschaft lautet: «Der Würfel ist geworfen.» Natürlich lateinisch! Da damals nur die wenigsten Leute lesen konnten, galt das Verfahren als sicher.

Aufgabe 7.9 IP-Adressen (1)

Web-Code: 8zmd r65m

110.120.130.140 als `integer` ist gleich 1 853 391 500
211.43.23.156 als `integer` ist gleich 3 542 816 668

Aufgabe 7.10 IP-Adressen (2)

Web-Code: ndma 4uff

Prüfen Sie die Resultate mit denjenigen aus Aufgabe IP-Adressen (1).

Aufgabe 7.11 Dreiersystem (1)

Web-Code: kwsx k6s4

120 (Dreiersystem) = 15 (Dezimalsystem)

1212 (Dreiersystem) = 27 + 18 + 3 + 2 = 50 (Dezimalsystem)

Aufgabe 7.12 Dreiersystem (2)

Web-Code: 8ghq ywrr

142 → 12021

33 → 1020

Aufgabe 7.17 ISBN/EAN

Web-Code: gg7a xaou

Hier einige Testdaten zu ISBN-10 (nach dem Pfeil steht die Prüfziffer):

- 0-13-110362 → 8
- 3-89864-117 → 9
- 3-446-13878 → 1

Aufgabe 7.20 Wortliste filtern

Web-Code: pomd oige

Verifikation zur Zusatzaufgabe: Es werden 664 Wörter gefunden:

aerogen Agarose Agentenorganisation Agentenstories Aggregation

…

Trigon Trigonitis Tritagonist trog TROGE TROGEN Trogon Trossingen

Aufgabe 7.22 Palindrome

Web-Code: brfr qbyf

Einzig die erste Zeile ist kein Palindrom.

8 Datenstrukturen und Sammelobjekte

Aufgabe 8.1 Adäquate Datentypen

Web-Code: rw4r 9u46

Größe	adäquater Datentyp
Geldbeträge	`real`
Bevölkerungszahlen	`integer`
Bevölkerungszuwachs	`real`
Datum des 21. Jahrhunderts	Datenstruktur, Klasse
Kennzeichen, ob ein Jahr ein Schaltjahr ist	`boolean`
Adresse (Straße, PLZ, Ort)	Datenstruktur, Klasse
Schachfigur	`integer`, String oder Enumerationstyp
Vorzeichen (positiv, negativ, ohne)	`integer`, String oder Enumerationstyp

Aufgabe 8.5 Ringpuffer mit Arrays

Web-Code: yui8 z5du

Beachten Sie, dass mit der Methode `add()` nicht mehr als `maxCount` Elemente in Serie eingefügt werden dürfen. Damit das nicht möglich ist, muss die Methode `add()` einen Fehlercode generieren können. Dies kann auf verschiedene Arten geschehen:

- Globale Fehlervariable
- Boole'scher Return-Wert
- Ausnahmebehandlung (*Exception*)

9 Algorithmen

Aufgabe 9.1 **Heron**

Web-Code: kugc tg53

5 → 2.236067977915804

Aufgabe 9.2 **Gray Code**

Web-Code: gtwa psr3

19 = 0001 0011 (binär) = 0001 1010 (Gray)

Aufgabe 9.3 **Primzahlen**

Web-Code: 6yqz 4y5k

1 → false, 2 → true, 3 → true, 4 → false, …

Aufgabe 9.6 **Wörter suchen**

Web-Code: 9nz3 nf8o

Diskussion: Um binär zu suchen, muss ein Index bekannt sein. Für die Wortendungen bleibt unter anderem die lineare Suche oder eine sortierte Tabelle nach den Spiegelbild-Wörtern (Umkehrung: «Hallo» → «ollaH»).

Aufgabe 9.7 **Lexikografisch sortieren**

Web-Code: t42p 7sny

Beispiele von Grundformen:

- Schweiz → schweiz
- für → fur
- Öl → ol
- Maße → masse

10 Simulationen

Aufgabe 10.1 Lineare Kongruenzmethode

Web-Code: qqrc bps2

Für a = m oder m = 1 ist eine Periode von 1 vorherbestimmt. Der `ggT(a, b, m)` sollte > 1 sein. Dies allein garantiert aber noch keine großen Periodenlängen. Die Periode kann nie länger als m werden.

Die folgenden Zahlen für (a, b, m und x_0 = seed) eingesetzt, liefern die angegebenen Periodenlängen:

a	b	m	x_0	Periodenlänge
9	10	11	12	5
2	3	11	4	10
11	11	1	11	1
3	7	31	6	30
5	6	31	7	3
101	102	103	1	102
4	5	103	6	51
2	2	1 000 003	2	1 000 002
10	11	1 000 003	12	166 667

Aufgabe 10.2 Ganze Zufallszahlen

Web-Code: 8v6n j82y

Rufen Sie 30 Mal die Funktion mit den Werten

`random(5, 7)`

auf. Die Zahlen 5, 6 und 7 sollten alle ca. 10 Mal auftauchen.

Aufgabe 10.4 Casino (Würfel 2)

Web-Code: 5n2a jfjb

Jeder der drei Würfel hat einen «Schwächeren», der (bei genügend vielen Würfen) geschlagen werden kann: BLAU schlägt ROT, ROT schlägt GELB und GELB schlägt BLAU (als Casino-Betreiber lassen Sie den Spieler den Würfel zuerst auswählen und nehmen daraufhin selbst den besseren).

Aufgabe 10.5 Web-Code

Web-Code: d2rh rur6

Vergleichen Sie Ihre Codes mit den Web-Codes in vorliegendem Buch.

Aufgabe 10.6 Farbstiftspitzen

Web-Code: i7ox fcxf

Bei 10 000 Spitzaktionen, 20 Stiften und 3 Stiften pro «blindem Griff» ergeben sich in etwa die folgenden Zahlen für die Anzahl Griffe:
Mittelwert: 23.05 ± 0.05
Minimum: 7 (mathematisch), meist 8 oder 9
Maximum: ca. 65 bis 100; mathematisch gibt es keine(!) Obergrenze.

Aufgabe 10.9 Poker verteilen

Web-Code: i4si 4ts5

Keine Karten dürfen doppelt vergeben werden.
Keine Muster dürfen wiederholt auftreten.

Aufgabe 10.10 Geburtstagszwillinge

Web-Code: aadv mxmq

Circa 24 bis 25 Personen sind nötig, damit mit Wahrscheinlichkeit > 50% zwei von ihnen am selben Tag Geburtstag feiern.

Aufgabe 10.11 Einfache Kombinatorik

Web-Code: 4cnf kew2

Es gibt 648 dreistellige Zahlen mit lauter verschiedenen Ziffern.

Aufgabe 10.12 Reisegruppe

Web-Code: aoy8 chnm

a) $4! = 24$
b) $9! = 362\,880$
c) $2! * 8! = 80\,640$
d) $36\,000$

Aufgabe 10.13 25 im Quadrat

Web-Code: ydd3 8ozc

Total gibt es 5248 Möglichkeiten (inklusive aller Drehungen, Spiegelungen und Vertauschungen von zwei Karten auf der Kante).
Nach dem «Wegdividieren» aller Ähnlichkeiten bleiben 41 grundverschiedene Lösungen.

Aufgabe 10.14 Magisches Quadrat

Web-Code: cmi5 4vwv

Es sind insgesamt 72 Möglichkeiten.

Aufgabe 10.15 Socken

Web-Code: rs4y t2k2

Pro Jahr entstehen ca. 19 Pechtage.

Aufgabe 10.16 Quintenzirkel

Web-Code: v8mr dw3x

Am häufigsten sollte Ihr Programm wieder bei der Tonika (C-Dur) oder der Tonikaparallelen (a-Moll) landen.

Aufgabe 10.17 Neuweltkamele

Web-Code: jsqi 792x

a) Im Jahr 2041 sind es voraussichtlich 4406 bis 4409 Kamele.
b) Es dauert etwa 22 Jahre, bis die natürliche Population wiederhergestellt ist. Dies wird somit etwa im Jahr 2061 oder 2062 der Fall sein.

Aufgabe 10.19 Umschütten

Web-Code: dob2 r5tp

Bei Gefäßen mit 11 und 21 Litern und einer Suchmenge von 15 Litern sind mit diesem Algorithmus 27 Schritte notwendig.

Aufgabe 10.22 Conways Game of Life

Web-Code: ch5c 67pm

Hier eine mögliche «Life»-Population. Nach einigen Generationen stellt man immer wieder dieselben Muster fest:

Aufgabe 10.24 Testdaten mit Reservoir (Sampling)

Web-Code: x8kv r9ij

Füllen Sie zum einfachen Test die drei Zahlen 1, 2 und 3 der Reihe nach in ein Reservoir der Größe 2. Testen Sie dies einige Male. Jedes der Paare (1; 2), (1; 3) und (3; 2) sollte mit Wahrscheinlichkeit 1/3 auftreten.

11 Rekursion

Aufgabe 11.1 **Merge-Sort**

Web-Code: puyx no2s

1, 2, 3, 3, 4, 6, 7, 8, 8, 8, 9

Aufgabe 11.4 **Lotto**

Web-Code: epsk xisf

b(45 | 6) = 8 145 060

Option: Obige Zahl muss einfach mit dem Geldbetrag pro Einsatz multipliziert werden.

Aufgabe 11.6 **Türme von Hanoi**

Web-Code: fk8y npoa

Die ersten drei Züge (falls n ungerade) sind:

a → c

a → b

c → b

Aufgabe 11.7 **Kamele beladen**

Web-Code: 6gdd r4zm

Kapazität 270 und Gewichte: 69, 63, 45, 34, 32, 18, 5 → Summe = 266

Für Maximalgewicht 1000 kg mit den Gewichten «181, 130, 128, 125, 124, 121, 104, 101, 98, 94, 69, 61, 13» erhält man genau die 1000 kg, wenn man folgende Auswahl trifft: «125, 101, 181, 128, 124, 104, 94, 69, 13, 61».

Achtung: Ab ca. 14 Gewichten beginnt die Aufgabe die Berechenbarkeit zu «knacken»! Die Standardlösung des sogenannten «Rucksack-Problems» weist eine bessere Laufzeit aus, der Lösungsweg ist hingegen nicht sofort einleuchtend.

Aufgabe 11.8 **Schnurlängen**

Web-Code: acgo 5ap7

450 (17, 19, 21) → Teilbar
100 (17, 19, 21) → Unteilbar
101 (17, 19, 21) → Teilbar

Aufgabe 11.9 **Dichtestes Punktpaar**

Web-Code: a93s 2si4

Bei den gegebenen 10 000 Punkten liegen A(– 13.221, 61.283) und B(– 13.212, 61.297) mit Abstand 0.0166433 am nächsten beieinander.

Aufgabe 11.10 **Malermeister Malcom**

Web-Code: durv de6t

Nach Elimination bleiben zwischen 400 und 500 Kreise stehen.

Anhang

ASCII-Tabelle

Nachfolgend ein Auszug aus der heute am häufigsten verwendeten Code-Tabelle, dem ASCII[1].
Die ASCII-Tabelle weist jedem 7-Bit-Muster ein Zeichen zu und kann demnach 128 Zeichen kodieren. Über die 8-Bit-Fortsetzung der Kodierung ab Nummer 128 sind sich die Hersteller der Betriebssysteme (Unix, MS-Windows, Mac-OS) nicht einig geworden. Dies führt oft zu Problemen mit Umlauten und Sonderzeichen beim Datenaustausch zwischen verschiedenen Betriebssystemen und Anwendungsprogrammen. Moderne Programme verwenden daher meist den sehr umfassenden Unicode zur Datenspeicherung.

[1] ASCII = American Standard Code for Information Interchange (Windows, Mac, Linux)

Wert	Zeichen	binär
0 – 31	nicht druckbar	
32	Leerschlag	010 0000
33	!	010 0001
34	”	010 0010
35	#	010 0011
36	$	010 0100
37	%	010 0101
38	&	010 0110
39	’	010 0111
40	(	010 1000
41	)	010 1001
42	*	010 1010
43	+	010 1011
44	,	010 1100
45	-	010 1101
46	.	010 1110
47	/	010 1111
48	0	011 0000
49	1	011 0001
…	…	… …
57	9	011 1001
58	:	011 1010
59	;	011 1011
60	<	011 1100
61	=	011 1101
62	>	011 1110

Wert	Zeichen	binär
63	?	011 1111
64	@	100 0000
65	A	100 0001
66	B	100 0010
67	C	100 0011
68	D	100 0100
…	…	… …
89	Y	101 1001
90	Z	101 1010
91	[	101 1011
92	\	101 1100
93	]	101 1101
94	^	101 1110
95	-	101 1111
96	‘	110 0000
97	a	110 0001
98	b	110 0010
99	c	110 0011
100	d	110 0100
…	…	… …
122	z	111 1010
123	{	111 1011
124	|	111 1100
125	}	111 1101
126	~	111 1110
127	(delete)	111 1111

Schlagwortverzeichnis

D

E

F

G

T

U

V

W

Y

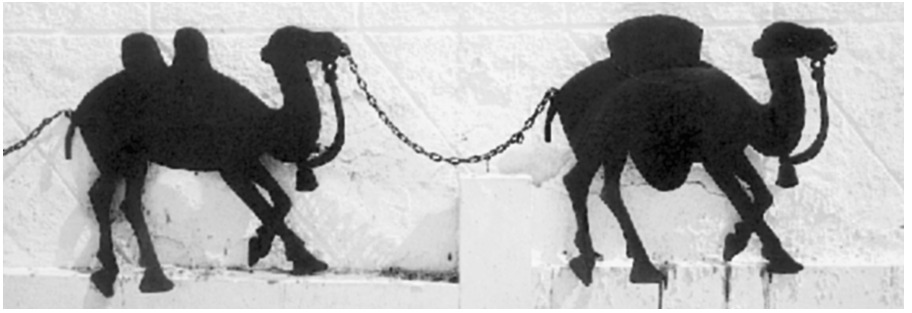
Z

Die Autoren

Philipp Gressly Freimann (∗1966), dipl. Math., unterrichtet bei der Santis Training AG Grundkonzepte aktueller Programmiersprachen.
Nach dem Abschluss des höheren Lehramts in Mathematik und Informatik an der Universität Zürich arbeitete er mehrere Jahre als Programmierer, bevor er zurück zum Unterricht fand.

Martin Guggisberg (∗1970) besitzt ein Diplom und einen Doktortitel für Experimentalphysik der Universität Basel. Er ist wissenschaftlicher Mitarbeiter am Departement Informatik der Universität Basel und Dozent für Didaktik der Informatik an der pädagogischen Hochschule der FHNW.
Seine Forschungsschwerpunkte liegen im Bereich e-Learning, Web-Technologien, Augmented Reality und 3D-Visualisierung. Er ist Mitglied der Studienleitung der Lehrerzusatzausbildung EFI Informatik.

Bildnachweis



«Werbeschild» für die Seidenstrasse in Baku, Aserbaidschan.
Foto: Georg Freivogel, Schaffhausen, www.tianshantours.ch

Das Kamel ist ein Sinnbild für ein ausdauerndes, ruhiges Wesen, das viele Lasten trägt. Die lebensfeindliche, steinige Wüste kann ihm nichts anhaben, da es sich Reservoire anlegt. Das Erlernen von Programmierfähigkeit verlangt Ausdauer und einen ruhigen, abgeklärten Umgang mit Fehlern.