

2

Programmierumgebung

Algorithmen & Datenstrukturen · Sommersemester 2026

Prof. Dr. Sebastian Wild

2 Programmierumgebung

- 2.1 Standardbibliotheken
- 2.2 Algs4
- 2.3 Malen nach Zahlen
- 2.4 Mehr zu entdecken

Unser Spagat

- ▶ ADS ist halb Theorie, halb Praxis
 - ▶ einerseits lernen Sie asymptotische Analysen und abstrakte Modelle
 - ▶ ohne dass wir ein Mathematik-Modul daraus machen wollen
 - ▶ andererseits werden Sie die Toolbox der Algorithmik auch *einsetzen*
 - ▶ aber Syntax und Details von Java sollen nicht im Vordergrund stehen
- ↪ an manchen Stellen erlauben wir uns Abkürzungen (an anderen absichtlich *nicht*)

2.1 Standardbibliotheken

Vokabular einer Programmiersprache

Programmiersprachen als Fremdsprachen

- ▶ eine Programmiersprache definiert die “Grammatikregeln” einer Sprache
- ▶ eine interessante Unterhaltungen braucht aber auch reichhaltiges Vokabular und idiomatische Redewendungen!

Programmierbibliotheken

- ▶ In Programmiersprachen liefern das Bibliotheken
- ▶ in der Regel (hauptsächlich) in der Sprache selbst geschrieben
- ▶ kapselt Interaktion mit Betriebssystem

Standardbibliothek

- ▶ Teil des “Lieferumfangs”, immer verfügbar
- ▶ Beispiele: Java Runtime Library, C++ Standard Template Library, Python Standard Library

Java Runtime Library

- ▶ alle Klassen in `java.lang` und `java.util` (und weitere packages)
- ▶ z.B. auch `System.out.println()` etc.
- ▶ insgesamt über 4400 Klassen und Interfaces

(Java 21, <https://docs.oracle.com/en/java/javase/21/docs/api/allclasses-index.html>)

- ▶ kann etwas überwältigend sein . . .
 - ▶ *niemand* kennt alle diese Klassen auswendig!
 - ▶ aber: gute Programmierer:innen bekommen ein Gespür dafür, für welche Funktionalität es eine Library-Lösung geben *müsste*
- ↪ um diese dann zu finden und korrekt zu verwenden gibt es Werkzeuge

Hier müssen Sie von Fall zu Fall entscheiden:

- | | |
|---|---|
| ▶ von KI machen lassen | ▶ lernen, wie die Bibliothek funktioniert |
| ↪ beim nächsten Mal wieder Hilfe nötig | ▶ beim nächsten Mal schneller |
| ↪ OK für Aufgaben, die nur einmal kommen und einfach sind | ↪ Vokabular erweitert |

2.2 Algs4

Algs4

“Algs4” = Sedgewick & Wayne: Algorithms 4th edition

“algs4” = Java-Bibliothek mit Hilfsklassen

<https://github.com/kevin-wayne/algs4>

Bequemlichkeit

- ▶ Standardbibliothek per definitionem *one-size-fits-all*

↪ keine Rücksicht auf Anfänger:innen ...

- ▶ außerdem: backwards compatibility oft zwingend

↪ Designfehler schwer zu beheben

Beispiele:

- ▶ Date ist mutable

<https://docs.oracle.com/javase/tutorial/datetime/iso/legacy.html>

- ▶ Methoden in Vector, Hashtable, Stack sind synchronized

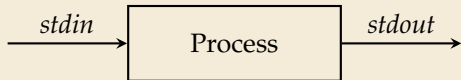
↪ Für einige nützliche Funktionen verwenden wir die vereinfachten Wrapper von Sedgewick & Wayne

- ▶ verfügbar unter <https://algs4.cs.princeton.edu/code/> sowie auf GitHub
- ▶ kompakter und lesbarer als direkt via Runtime Library
- ▶ grundsätzlich für Programmieraufgaben erlaubt*

UNIX input/output stream model

Abstraktion von Ein- und Ausgabe: Streams

- ▶ i.W. unverändert in Benutzung seit 1970er Jahre in UNIX
- ▶ flexibel und einfach
- ▶ in Java Runtime Library
System.in und System.out
≠ command-line arguments



Design „fehler“

- ▶ System.in hat keine bequemen Methoden um primitive Daten einzulesen

↪ `new java.util.Scanner(System.in)`

- ▶ System.out ist langsam für große Ausgaben (automatisches Flush bei `'\n'`)

↪ `new java.io.PrintStream(new java.io.BufferedOutputStream(System.out))`

- ▶ beide sind byte-basiert, nicht Zeichen-basiert

Insbesondere System.in ist unbequem ...

StdIn

`edu.princeton.cs.algs4.StdIn` aus `algs4.jar` ist ein bequemerer Wrapper für `System.in`

```
public class StdIn
methods for reading individual tokens from standard input
    boolean isEmpty()           is standard input empty (or only whitespace)?
        int readInt()          read a token, convert it to an int, and return it
        double readDouble()    read a token, convert it to a double, and return it
    boolean readBoolean()      read a token, convert it to a boolean, and return it
    String readString()        read a token and return it as a String
methods for reading characters from standard input
    boolean hasNextChar()      does standard input have any remaining characters?
    char readChar()            read a character from standard input and return it
methods for reading lines from standard input
    boolean hasNextLine()      does standard input have a next line?
    String readLine()          read the rest of the line and return it as a String
methods for reading the rest of standard input
    int[] readAllInts()         read all remaining tokens and return them as an int array
    double[] readAllDoubles()   read all remaining tokens and return them as a double array
    boolean[] readAllBooleans() read all remaining tokens and return them as a boolean array
    String[] readAllStrings()   read all remaining tokens and return them as a String array
    String[] readAllLines()     read all remaining lines and return them as a String array
    String readAll()            read the rest of the input and return it as a String
```

- ▶ `StdIn` besteht aus static methods
- ▶ `edu.princeton.cs.algs4.In` ist eine Objekt-orientierte Variante
 - ▶ gleiche Methoden
 - ▶ Konstrukturen für Dateien, URLs, ...

<https://introc.cs.princeton.edu/java/15inout/>

StdOut

- ▶ `edu.princeton.cs.algs4.StdOut`
 - ▶ ähnliche Methoden wie `System.out`
spart ein paar Zeichne beim Tippen 😊
- ▶ `edu.princeton.cs.algs4.Out`
 - ▶ wie In Objekt-orientierte Version
 - ▶ Konstrukturen für Dateien, Sockets, Streams

Beispiel

```
1 import edu.princeton.cs.algs4.*; // algs4 Klassen
2
3 class Average {
4     public static void main(String[] args) {
5         double sum = 0.0;
6         int n = 0;
7         while (!StdIn.isEmpty()) {
8             double x = StdIn.readDouble();
9             sum += x;
10            n++;
11        }
12        StdOut.println(sum/n);
13    }
14 }
```

algs4.jar einbinden

Kompilieren: `javac -cp algs4.jar Average.java`

Ausführen: `java -cp algs4.jar:. Average`

2.3 Malen nach Zahlen

StdDraw

Nur Text ist etwas dröge . . . viele spannende Dinge sind ebenso einfach!

Zum Beispiel einfache **Graphiken**

```
public class StdDraw (basic drawing commands)  
    void line(double x0, double y0, double x1, double y1)  
    void point(double x, double y)
```

viele weitere Methoden \rightsquigarrow <https://introcs.cs.princeton.edu/java/15inout/>

```
public class StdDraw (basic control commands)  
    void setCanvasSize(int w, int h) create canvas in screen window of  
width w and height h (in pixels)  
    void setXscale(double x0, double x1) reset x-scale to (x0, x1)  
    void setYscale(double y0, double y1) reset y-scale to (y0, y1)  
    void setPenRadius(double radius) set pen radius to radius
```

Methoden um Canvas zu konfigurieren

Malen nach Zufallszahlen

Ein einfacher Zufallsprozess:

1. Starte mit 3 Punkten
 $p_0 = (0, 0)$, $p_1 = (1, 0)$ und
 $p_2 = (\frac{1}{2}, \sqrt{3}/4)$
(gleichseitiges Dreieck)
2. Starte mit $p = p_0$
3. Wähle zufälligem Punkt
 $q \in \{p_0, p_1, p_2\}$
4. Setze p auf den Mittelpunkt
zwischen p und q
5. Male Punkt p , zurück zu 3.

```
1 import edu.princeton.cs.algs4.*;
2
3 class Chaos {
4     public static void main(String[] args) {
5         int trials = Integer.parseInt(args[0]);
6         double c = Math.sqrt(3.0) / 2.0;
7         double[] cx = {0.0, 1.0, 0.5 };
8         double[] cy = {0.0, 0.0, c };
9         StdDraw.setPenRadius(0.01);
10        double x = 0, y = 0; // p = (x,y)
11        for (int t = 0; t < trials; ++t) {
12            int r = (int) (Math.random() * 3);
13            x = (x + cx[r]) / 2;
14            y = (y + cy[r]) / 2;
15            StdDraw.point(x, y); // Punkt malen
16        }
17    }
18 }
```

edu.princeton.cs.algs4.Math ein weiterer Wrapper,
z.B. einfachere $\text{Math.random() } \stackrel{D}{=} \text{Uniform}(0, 1)$

Allgemeines Iterated Function System (IFS)

Das *Sierpinski-Dreieck* ist ein Beispiel für eine ganze Familie von solchen Prozessen.

- ▶ In jedem Schritt aktualisieren wir den aktuellen Punkt $p = (x, y)$ anhand einer linearen Funktion $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2: (x, y) \mapsto (f_x(x, y), f_y(x, y))$
- ▶ f in jedem Schritt zufällig aus einer Liste von Funktionen gewählt.

Funktion	Wahrscheinlichkeit	$f_x(x, y)$	$f_y(x, y)$
$f_0(p_0)$	$\frac{1}{3}$	$\frac{1}{2}x$	$\frac{1}{2}y$
$f_1(p_1)$	$\frac{1}{3}$	$\frac{1}{2}x + \frac{1}{2}$	$\frac{1}{2}y$
$f_2(p_2)$	$\frac{1}{3}$	$\frac{1}{2}x + \frac{1}{4}$	$\frac{1}{2}y + \frac{1}{2}\sqrt{3}/4 \leftarrow \approx 0.433$

- ▶ Allgemein:

$$x \mapsto (e_R)^T \cdot A \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad y \mapsto (e_R)^T \cdot B \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad e_R \text{ zufälliger Einheitsvektor}$$

$$\text{Sierpinski: } R \stackrel{\text{D}}{=} \text{Uniform}\{1, 2, 3\} \quad A = \begin{pmatrix} 0.5 & 0 & 0 \\ 0.5 & 0 & 0.5 \\ 0.5 & 0 & 0.25 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 0.5 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0.5 & 0.433 \end{pmatrix}$$

IFS Implementierung

```
1 import edu.princeton.cs.algs4.*;
2
3 public class IFS {
4     public static void main(String[] args) {
5         // number of iterations
6         int trials = Integer.parseInt(args[0]);
7         // prob. distribution for functions
8         double[] dist = StdArrayIO.readDouble1D();
9         // update matrices
10        double[][] cx = StdArrayIO.readDouble2D();
11        double[][] cy = StdArrayIO.readDouble2D();
12
13        // current value of (x, y)
14        double x = 0.0, y = 0.0;
15        // for faster drawing, delay update to draw()
16        StdDraw.enableDoubleBuffering();
```

```
17
18        for (int t = 0; t < trials; t++) {
19            // random index according to prob. dist.
20            int r = StdRandom.discrete(dist);
21            // compute new coordinates
22            double x0, y0;
23            x0 = cx[r][0]*x + cx[r][1]*y + cx[r][2];
24            y0 = cy[r][0]*x + cy[r][1]*y + cy[r][2];
25            x = x0;
26            y = y0;
27            // draw the resulting point
28            StdDraw.point(x, y);
29        }
30        // show final picture
31        StdDraw.show();
32    }
```

- ▶ verwendet weitere Bequemlichkeits-Wrapper:
 - ▶ `edu.princeton.cs.algs4.StdArrayIO` liest Vektoren und Matrizen ein
 - ▶ Format: Anzahl (Zeilen und) Spalten gefolgt von Einträgen
 - ▶ `edu.princeton.cs.algs4.StdRandom.discrete` zieht diskrete Zufallsvariable gegeben die Wahrscheinlichkeitsgewichte

sierpinsky.txt

```
1 3
2     0.333     0.333     0.334
3 3 3
4     0.5     0     0
5     0.5     0     0.5
6     0.5     0     0.25
7 3 3
8     0     0.5     0
9     0     0.5     0
10    0     0.5     0.433
```

Iterated function systems

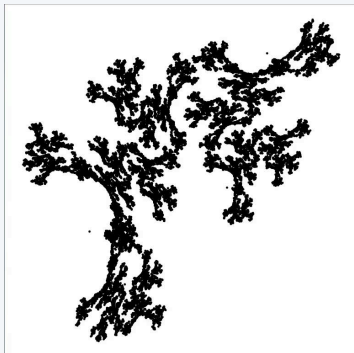
What happens when we change the rules?

<i>probability</i>	<i>new x</i>	<i>new y</i>
40%	$.31x - .53y + .89$	$-.46x - .29y + 1.10$
15%	$.31x - .08y + .22$	$.15x - .45y + .34$
45%	$.55y + .01$	$.69x - .20y + .38$

IFS. java (Program 2.2.3) is a *data-driven* program that takes the coefficients from *standard input*.

```
% more coral.txt
3
  0.40  0.15  0.45
3 3
  0.307692 -0.531469  0.8863493
  0.307692 -0.076923  0.2166292
  0.000000  0.545455  0.0106363
3 3
-0.461538 -0.293706  1.0962865
  0.153846 -0.447552  0.3383760
  0.692308 -0.195804  0.3808254
```

```
% java IFS 10000 < coral.txt
```



Iterated function systems

Another example of changing the rules

<i>probability</i>	<i>new x</i>	<i>new y</i>
2%	0.5	.27y
15%	$-.14x + .26y + .57$	$.25x + .22y - .04$
13%	$.17x - .21y + .41$	$.22x + .18y + .09$
70%	$.78x + .03y + .11$	$-.03x + .74y + .27$

```
% more barnsley.txt
4
.02 .15 .13 .70
4 3
.000 .000 .500
-.139 .263 .570
.170 -.215 .408
.781 .034 .1075
4 3
.000 .270 .000
.246 .224 -.036
.222 .176 .0893
-.032 .739 .270
```

```
% java IFS 10000 < barnsley.txt
```



Iterated function systems

Simple iterative computations yield patterns that are remarkably similar to those found in the natural world.

Q. What does computation tell us about nature?

Q. What does nature tell us about computation?

20th century sciences. Formulas.

21st century sciences. Algorithms?

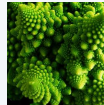
Note. You have seen many practical applications of integrated function systems, in movies and games.



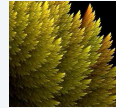
an IFS fern



a real fern



a real plant



an IFS plant



2.4 Mehr zu entdecken

StdAudio

edu.princeton.cs.algs4.StdAudio erlaubt einfache Soundausgabe

```
public class StdAudio
```

```
    int SAMPLE_RATE           44,100 Hz
```

```
    void play(String filename) play the given .wav file
```

```
    void playInBackground(String filename) play the given .wav file in a background thread
```

```
    void play(double[] samples) play the given samples
```

```
    void play(double sample) play sample
```

```
    void save(String filename, double[] a) save to a .wav file
```

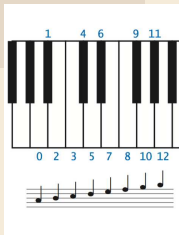
```
    double[] read(String filename) read from a .wav file
```

API for our library of static methods for standard audio

Mini-Synthesizer

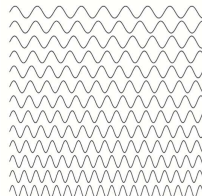
```
1 import edu.princeton.cs.algs4.*;
2
3 public class PlayThatTune {
4     public static void main(String[] args) {
5         int SAMPLING_RATE = 44100;
6         while (!StdIn.isEmpty()) {
7             // Read and play one note.
8             int pitch = StdIn.readInt();
9             double duration = StdIn.readDouble();
10            double hz = 440 * Math.pow(2, pitch / 12.0);
11            int n = (int) (SAMPLING_RATE * duration);
12            double[] a = new double[n+1];
13            for (int i = 0; i <= n; i++)
14                a[i] = Math.sin(2*Math.PI * i * hz
15                    / SAMPLING_RATE);
16            StdAudio.play(a);
17        }
18    }
19 }
```

- ▶ Spannender Physik-Exkurs:
Was sind Geräusche und Töne?
- ▶ Sinus-Schwingung für reine Töne
440 Hz \rightsquigarrow Kammerton
(eingestrichenes A)
- ▶ `StdAudio.play(double[])` spielt
gegebene Werte als Ton
- ▶ Soundkarte erwartet gewisse
Sample-Rate (Werte pro Sekunde)



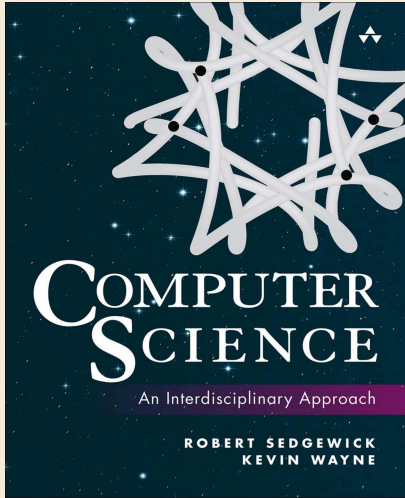
note	<i>i</i>	frequency
A	0	440.00
A \sharp or B \flat	1	466.16
B	2	493.88
C	3	523.25
C \sharp or D \flat	4	554.37
D	5	587.33
D \sharp or E \flat	6	622.25
E	7	659.26
F	8	698.46
F \sharp or G \flat	9	739.99
G	10	783.99
G \sharp or A \flat	11	830.61
A	12	880.00

$440 \times 2^{i/12}$



Programming with a Purpose

Viele weitere kreative Beispiele und Anwendungen bei Sedgewick & Wayne



Für Algorithmen und Datenstrukturen nur tangential relevant, aber macht Spaß 😊