



1 Freifach Programmieren

1.1 Numerische Algorithmen

✂ **Aufgabe A1** (Finanzmathematik) Für die Altersrente zahlt Frau Huber jedes Jahr am 1. Januar den Betrag 6000.- auf ein Sparkonto ein. Dem Konto wird am 31. Dezember jeweils ein Zins von 1% gutgeschrieben. Wie gross ist der Kontostand nach 35 Jahren?

Schreiben Sie ein Programm, das als erstes die drei Variablen `betrag=6000`, `zins=0.01` und `laufzeit=35` definiert und dann in einer `while`- oder `for`-Schleife für jedes Jahr den auf zwei Nachkommastellen gerundeten Kontostand am 31. Dezember (nach der Zinsgutschrift) ausgibt.

Zur Kontrolle: Die letzte Ausgabe sollte wie folgt lauten:

```
Kontostand am Ende des Jahres 35: 252461.27
```

Hinweis: Die Ausgabe von Variablen innerhalb von Strings geschieht besonders einfach mit f-strings (= formatted strings). Beispiel:

```
x = 2
print(f'Die Wurzel von {x} ist {x**0.5}.')
```

Das Zeichen `f` vor dem Anführungszeichen steht für `formatted` und verwandelt den normalen String in einen f-String.

Wenn man die Wurzel auf 5 Nachkommastellen runden möchte, geht das wie folgt.

```
print(f'Die Wurzel von {x} ist {x**0.5:.5f}.')
```

Das Zeichen `f` in der letzten geschweiften Klammer steht für `float`.

✂ **Aufgabe A2** (Wurzel einer Zahl berechnen, ohne die Wurzelfunktion von Python zu verwenden)

Die Wurzel \sqrt{x} einer positiven Zahl x kann man näherungsweise berechnen, indem man zwei Variablen `links` und `rechts` so initialisiert, dass $\text{links} \leq \sqrt{x} \leq \text{rechts}$ gilt. Der gesuchte Wert \sqrt{x} ist also «zwischen `links` und `rechts` eingesperrt». Zum Beispiel kann man `links = 0` und `rechts = x` setzen.

Nun berechnet man den Mittelwert $\text{mittel} = \frac{\text{links} + \text{rechts}}{2}$ und vergleicht sein Quadrat mittel^2 mit x . Je nachdem, wie dieser Vergleich ausfällt, ersetzt man `links` oder `rechts` durch `mittel`, so dass nach der Ersetzung wieder $\text{links} \leq \sqrt{x} \leq \text{rechts}$ gilt.

Diesen Prozess wiederholt man solange, bis die Differenz `rechts - links` kleiner als ein gewünschter Fehler ist.

Ergänzen Sie das folgende Programm so, dass der oben beschriebene Algorithmus realisiert wird!

```
x = 40
fehler = 0.000001
links = 0
rechts = x

while rechts - links > fehler:
    #
    # Hier ist Code zu schreiben.
    #
    schaeztung = (links + rechts) / 2
    print(f'{schaeztung} ist ungefähr die Wurzel aus {x}.')
    print(f'{x**0.5} ist laut Python die Wurzel aus {x}.')
    print(f'Der Fehler ist etwa {abs(mittel-x**0.5):.10f}.')
```

Bemerkung: Das beschriebene Verfahren heisst **Intervallschachtelung**, da \sqrt{x} immer besser durch das kleiner werdende Intervall `[links, rechts]` eingeschachtelt wird.



✂ Aufgabe A3 Die Zahlenfolge

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

heisst **Fibonacci-Folge**. Die beiden Folgenglieder am Anfang sind $f_0 = 0$ und $f_1 = 1$, jedes nachfolgende Folgenglied ist die Summe seiner beiden Vorgänger, d. h. $f_n = f_{n-2} + f_{n-1}$ für alle $n \geq 2$.

Schreiben Sie ein Programm, das zeilenweise jeweils die Nummer n und dann das zugehörige Folgenglied f_n ausgibt. Eine am Anfang des Programms definierte Variable `maxn` gibt dabei an, bis zu welchem n die Folge ausgegeben werden soll. Für `maxn = 6` soll die Ausgabe wie folgt aussehen.

```
Nr Fibonaccizahl
0 0
1 1
2 1
3 2
4 3
5 5
6 8
```

Bemerkung: Die Zahlen in der Fibonacci-Folge heissen Fibonacci-Zahlen. Sie beschreiben zum Beispiel das Wachstum einer idealisierten Kaninchenpopulation, vgl. [Wikipedia: Fibonacci-Folge, Antike und Mittelalter in Europa](#).

✂ Aufgabe A4 Fibonacci-Folge als Liste: Schreibe ein Programm, das mit Hilfe einer `while`- oder `for`-Schleife die Liste aller Fibonacci-Zahlen bis zur Zahl mit dem Index `maxn` erzeugt und diese ausgibt.

Hinweis: Definiere anfangs die Liste der Fibonacci-Zahlen per

```
fibonaccifolge = [0, 1]
```

und hänge die weiteren Folgenglieder an diese Liste an. Der folgenden Befehl hängt die Zahl 42 ans Ende einer Liste `liste`:

```
liste.append(42)
```

✂ Aufgabe A5 Fibonacci-Folge mit expliziter Formel: Die explizite Formel für das n -te Glied f_n der Fibonacci-Folge lautet (Formel von Moivre-Binet):

$$f_n = \frac{1}{\sqrt{5}} (\varphi^n - \psi^n) \quad \text{wobei } \varphi = \frac{1 + \sqrt{5}}{2} \text{ und } \psi = \frac{1 - \sqrt{5}}{2}.$$

Schreibe ein Programm, das die beiden reellen Zahlen φ und ψ in Variablen `phi` und `psi` speichert und dann mit der Formel von Moivre-Binet und «list comprehension» (siehe unten) die Liste aller Fibonacci-Zahlen bis zur Zahl mit dem Index `maxn` erzeugt und diese ausgibt.

Hinweise: Mit «list comprehension» kann man elegant Listen, deren Elemente von einer Laufvariablen abhängen, erzeugen. Beispiel:

```
liste = [2**n for n in range(10)]
print(liste)
```

Bemerkung: Die reelle Zahl φ ist der goldene Schnitt; φ und ψ sind die beiden Lösungen der quadratischen Gleichung $x^2 - x - 1 = 0$.



✂ **Aufgabe A6** (Kreiszahl π durch ein Zufallsexperiment näherungsweise bestimmen)

Der folgende Python-Code erzeugt Zufallszahlen zwischen 0 und 1:

<pre>from random import random n = 6 while n>0: z = random() print(z) n = n-1</pre>	<pre>0.3531273692097092 0.7544354595223802 0.5152512277249354 0.801798917741774 0.8009730208078284 0.10048357648286532</pre>
--	--

In dieser Aufgabe sollen Sie die Kreiszahl π mit einem Zufallsexperiment näherungsweise bestimmen. Schreiben Sie dafür ein Python-Programm wie folgt:

- Bestimmen Sie zwei Zufallszahlen x und y zwischen 0 und 1.
- Dann ist (x, y) ein Punkt Einheitsquadrat (hier gemeint: das Quadrat der Seitenlänge 1 mit den Eckpunkten $(0, 0), (1, 0), (1, 1), 0, 1)$ im Koordinatensystem).
Stellen Sie fest, ob dieser Punkt auch im Einheitskreis (= Kreis mit Radius 1 um den Ursprung des Koordinatensystems) liegt.
- Wiederholen Sie obiges Experiment 1000 oder 1'000'000 mal und zählen Sie die Anzahl der Punkte im Kreis.
- Berechnen Sie den Anteil der Punkte, die im Einheitskreis liegen.
- Wie gross müsste dieser Anteil theoretisch sein? Wie lässt sich somit π näherungsweise berechnen? Wie genau ist das Resultat?

Hinweis: Die Kreiszahl π (oder genauer gesagt die Annäherung an π , mit der Python rechnet) kann man wie folgt ausgeben.

```
from math import pi
print(pi)
```

✂ **Aufgabe A7** Kleines Einmaleins:

Schreibe ein Programm, das die folgende Multiplikationstabelle ausgibt.

Hinweise:

- Verwende zwei ineinander verschachtelte Schleifen.
In der inneren Schleife wird jeweils eine Zeile der Ausgabe aufgebaut.
- Der f-String (= formatted string) `f'{i:4}` erzeugt einen String der Länge 4, in dem die ganze Zahl i rechtsbündig steht und davor Leerzeichen (falls genügend Platz vorhanden ist).

*	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100



✂ **Aufgabe A8** Das folgende Programm erzeugt zwei Zufallszahlen zwischen 2 und 10.

```
import random
x = random.randrange(2, 11)
y = random.randrange(2, 11)
print(x)
print(y)
```

Erweitere dieses Programm so, dass der Computer den Benutzer nach dem Produkt der Variablen x und y fragt und ihm mitteilt, ob er richtig gerechnet hat. Der Dialog mit dem Computer sollte sinngemäss so aussehen:

```
Was ist das Produkt von 5 und 7? 24
Dies ist leider falsch. Richtig wäre 35 gewesen.
```

Bonus: Es gibt viele Erweiterungsmöglichkeiten. Etwa könnten dem Benutzer 10 Multiplikationsaufgaben gestellt werden und es wird ihm danach mitgeteilt, wie viel Prozent der Aufgaben er richtig gelöst hat. Es könnten auch zufällig Divisionsaufgaben (ohne Rest) eingestreut werden.



1.2 Listen

1.2.1. Basics zu Listen: https://www.w3schools.com/python/python_lists.asp

✂ **Aufgabe A9** Schreibe ein Python-Programm, das vom Benutzer so lange Zahlen einliest und diese in einer Liste speichert, bis dieser 'q' (wie «quit») eingibt. In diesem Fall soll die gesamte Liste ausgegeben werden.

Hinweis: Starte mit der leeren Liste `l=[]`. Mit der Methode `append` kann man ein Element an das Ende einer Liste anhängen, siehe etwa https://www.w3schools.com/python/python_lists_add.asp.

✂ Aufgabe A10

- (a) Das folgender Programm erzeugt eine Liste von `n` Zufallszahlen und gibt einige Werte aus. Ausserdem illustriert es die «built-in Python functions» `sum`, `min`, `max`, `sorted`.

```
from random import *
n = 10
zahlenliste = [randrange(1, 7) for _ in range(n)]
print(zahlenliste)
print(f'Summe: {sum(zahlenliste)}; Minimum: {min(zahlenliste)}; Maximum:
    ↪ {max(zahlenliste)};')
print(f'sortiert: {sorted(zahlenliste)}.')
print(zahlenliste)
```

Schreibe ein Programm, dass die folgenden Werte ausgibt, **ohne** diese built-in-Funktionen zu verwenden:

- den Durchschnitt aller Zahlen der Liste;
 - die kleinste und die grösste Zahl der Liste;
 - ✂ den Median der Liste (also «das» Element, das grösser bzw. kleiner ist als die «Hälfte» der Listenelemente). Um zu testen, dass dein Programm korrekt ist: Gib aus, wie viele Elemente grösser als bzw. kleiner als bzw. gleich gross wie der von dir ermittelte Median sind;
 - ✂ eine sortierte Version der Liste erzeugt (und die Ausgangsliste `zahlenliste` unverändert lässt).
- Bemerkung: Später werden diverse Sortieralgorithmen behandelt werden.

- (b) Fertige eine Kopie deines Programms an und schreibe für jede der obigen Teilaufgaben eine Funktion, die eine Liste als Argument entgegennimmt und den jeweiligen Wert als Rückgabewert hat; du kannst diese Funktionen etwa `durchschnitt(l)`, `maximum(l)`, `minimum(l)`, `median(l)`, `sortiert(l)` nennen.

✂ **Aufgabe A11** Schreibe in Python ein Quiz-Programm, das dem Benutzer nacheinander die Fragen in der unten angegebenen Liste `liste_fragen` stellt und ihm anhand der Liste `liste_korrekte_antworten` mitteilt, ob die Antwort korrekt war oder nicht. Am Ende soll ausgegeben werden, wie viel Prozent der Antworten korrekt waren.

Das Programm soll auch funktionieren, wenn du die Liste der Fragen und Antworten veränderst oder vergrösserst.

```
liste_fragen = ["Wie hoch ist der Säntis? (in Metern) ",
    "Bei welcher Ortschaft entspringt die Sitter? ",
    "Wie tief ist der Bodensee an seiner tiefsten Stelle? (in Metern) "]
liste_korrekte_antworten = ["2502", "Weissbad", "251"]
```

1.3 Dictionaries = Datenstruktur aus «durch Strings indizierten Werten»

1.3.1. Basics zu «dictionaries»: https://www.w3schools.com/python/python_dictionaries.asp

✂ **Aufgabe A12** In Aufgabe [A11](#) wirkt es etwas umständlich, dass zwei Listen verwendet werden: Eine Liste für die Fragen und eine Liste für die Antworten.

Besser wäre eine einzige Liste von «Aufgaben» (= Frage-Antwort-Paaren), so dass bei jeder Aufgabe sowohl Frage als auch Antwort gespeichert sind.



Eine «Aufgabe» könnte man als dictionary wie folgt definieren.

```
aufgabe_1 = {
    "Frage"    : "Wie hoch ist der Säntis? (in Metern) ",
    "Antwort"  : "2502",
}
```

Definiere die anderen beiden «Aufgaben» in ähnlicher Weise als dictionary. Erstelle dann eine Liste von Fragen per

```
liste_aufgaben = [aufgabe_1, aufgabe_2, aufgabe_3]
```

Verwende diese Liste, um das Quiz zu programmieren.

✂ Aufgabe A13

- (a) Wenn man für alle Schüler einer Klasse die Noten im Fach Mathematik abspeichern möchte: Welche Datenstruktur verwendet man sinnvollerweise, damit die üblicherweise auftretenden Situationen (Schüler bekommt neue Note; welche Note hat ein Schüler in einer Prüfung? neuer Schüler kommt in Klasse; ...) leicht zu verarbeiten sind?

Definiere in Python eine Variable namens `notenliste` dieser Datenstruktur und fülle sie mit Beispieldaten (etwa drei Schüler mit Namen deiner Wahl, je zwei Noten pro Schüler).

Zum neuen Semester kommt ein neuer Schüler namens Pinocchio in die Klasse und schreibt in der Klausur (= der vierten im Schuljahr) eine 6. Welche Befehle passen deine Datenstruktur der neuen Situation an?

- (b) Wie kann man deine Datenstruktur sinnvoll erweitern, wenn man die Noten der Klasse in mehreren Fächern abspeichern möchte? Wenn man die Noten mehrerer Klassen abspeichern möchte?

1.4 Listen von Listen

1.4.1. Tabellen bzw. «zweidimensional angeordnete Daten» speichert man oft sinnvoll in Listen von Listen.

✂ Aufgabe A14 Die Variable `spielfeld` im folgenden Programm zeigt, wie man eine Stellung eines Tic-Tac-Toe-Spiels speichern kann und wie man auf Steine auf den einzelnen Feldern zugreift.

```
# Per Unicode Code Points definierte Strings.
blau = '\U0001f535'
gelb = '\U0001f7e1'
leer = '\u2b1c'
spielfeld = [[blau, leer, gelb], [leer, gelb, leer], [blau, leer, leer]]
print(spielfeld)
print(spielfeld[0])
print(spielfeld[0][1])
```

Verwende diese, um zwei Menschen am Computer Tic-Tac-Toe gegeneinander spielen zu lassen. Gehe dabei anfangs davon aus, dass nur korrekte Züge eingegeben werden. Du kannst dich an dem folgenden Programmgerüst orientieren. [Click here to save Python code](#)

```
# Spielfeld initialisieren.
an_der_reihe = gelb
while True:
# Start der Game loop (Endlosschleife).
    # Gib aktuellste Stellung des Spiels aus.
    # Gib aus, welcher Spieler an der Reihe ist.
    # Erfrage den Spielzug (etwa Felder von 1 bis 9 durchnummerieren).
    # Ändere die Spielfeldvariable entsprechend.
    # Ermittle, ob Endstellung erreicht.
    # Wenn ja:
        # Gib aus, wer gewonnen hat.
        break # verlässt die Game loop.
    # Nun kommt der andere Spieler an die Reihe.
```



1.5 Einführung in die Grafik- und Spielprogrammierung mit Pygame

✂ Aufgabe A15

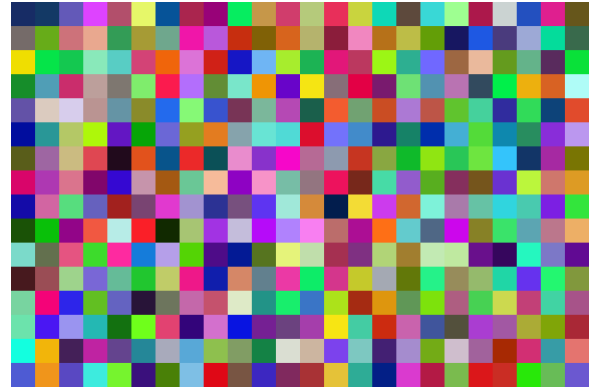
- (a) Bouncing Ball: Speichere das folgende Python-Programm auf deinem Computer.
[bouncing-ball.py \(bitte anklicken\)](#) (Falls der Pin nicht anklickbar ist: Lade das Programm von der Freifach-Dokuwiki-Seite herunter.)
Ändere das Programm so, dass der Ball an allen Rändern des Zeichenfensters und an dem grünen Balken korrekt reflektiert wird (Einfallswinkel = Ausfallswinkel).
Die Reflektion soll auch dann funktionieren, wenn du die Variable `ballgeschwindigkeit` änderst.
- (b) Paddle steuern: Speichere das folgende Python-Programm auf deinem Computer.
[paddle-steuern.py \(bitte anklicken\)](#) (Falls der Pin nicht anklickbar ist: Lade das Programm von der Freifach-Dokuwiki-Seite herunter.)
Ändere das Programm so, dass der Spieler das Paddle durch die Pfeiltasten Links und Rechts steuern kann. Das Paddle soll am unteren Rand des Zeichenfensters entlanggleiten. Ausserdem soll das Paddle stoppen, wenn es am Rand des Zeichenfensters anstösst.
- (c) Fusioniere deine beiden Programme zu «Pong-Solo»:
- Für jede Reflektion des Balls am oberen Bildschirmrand gibt es einen Punkt.
 - Unten wird der Ball nur reflektiert, wenn sich das Paddle direkt unter dem Ball befindet. Sonst verliert der Spieler ein Leben und der Ball startet an einer zufälligen Position mit einem zufälligen Geschwindigkeitsvektor.
 - Verwende die Funktion `schreibe(s, x, y)`, um Punkte und Anzahl der Leben anzuzeigen.
- (d) Sei kreativ: Erweitere das Spiel oder denke dir selbst ein Spiel aus.
Einige Ideen:
- Geschwindigkeit des Balls wächst mit der Zeit; gewisse Anzahl an Punkten liefert Extraleben;
 - Winkel des Balls kann verändert werden, wenn der Ball am Rand des Paddle reflektiert wird;
 - Gravitation;
 - das Treffen gewisser zufällig erzeugter Objekte liefert Bonusleben oder Bonuspunkte;
 - Zusatzstrukturen, an denen der Ball reflektiert wird;
 - ein zweiter Spieler steuert anderes Paddle (Paddle links und rechts, wie bei Pong, <https://en.wikipedia.org/wiki/Pong>);
 - mehrere Bälle (die auch aneinanderstossen können);
 - Sound;
 - Breakout;
 - dreidimensional;
 - etc.



✂ **Aufgabe A16** Moderne Kunst: Schreibe ein Programm, das mit Pygame ein Zufallsbild aus Quadraten erzeugt.

Am Anfang des Programms definierte Variablen/Konstanten `breite`, `hoehe`, `laenge` geben an, aus wie vielen Quadraten das Bild in x - und y -Richtung bestehen soll und wie lang eine Quadratseite in Pixel ist.

Jedes Quadrat soll durch eine Funktion `zeichne_quadrat(x, y, farbe)` gezeichnet werden, wobei die Parameter x und y die Position des Quadrats in einem «normalen» Koordinatensystem mit x -Achse nach rechts und y -Achse nach oben festlegen. Dabei ist x eine natürliche Zahl zwischen 0 und `breite-1` und y ist eine natürliche Zahl zwischen 0 und `hoehe-1`.



Der Farbparameter `farbe` ist eine in Pygame erlaubte Farbkodierung, etwa ein Tripel (`rot`, `gruen`, `blau`) aus RGB-Werten zwischen 0 und $255 = 2^8 - 1$.

Du kannst dich an dem folgenden Programmgerüst orientieren (die mit `#` markierten Kommentare musst du nicht abschreiben).

```
import pygame
import random

# Anzahl Quadrate in x-Richtung
breite = 24
# Anzahl Quadrat in y-Richtung
hoehe = 16
# Seitenlänge eines Quadrats in Pixel
laenge = 60

def zeichne_quadrat(x, y, farbe):

    # Die Definition der Funktion ist zu ergänzen.

# Initialisiere Pygame.

# Zeichne mit Hilfe der Funktion zeichne_quadrat(x, y, farbe)
# und zweier ineinander verschachtelter for-Schleifen
# alle Quadrate in Zufallsfarben.
# Eine Zufallsfarbe kannst du wie folgt erzeugen:
rot = random.randrange(256)
gruen = random.randrange(256)
blau = random.randrange(256)
# Das Tupel (rot, gruen, blau) ist dann
# eine in Pygame erlaubte Farbe.

while True:
    for ereignis in pygame.event.get():
        if ereignis.type == pygame.QUIT:
            exit()
    pygame.display.flip()
    clockobject.tick(10)
```

Bonus: Steuere ein «Spieler» alias einen Kreis mit den Pfeiltasten durch das Bild (jeweils Verschiebung um ein Quadrat). Der Spieler fräst auf diese Weise einen schwarzen Gang durch das Bild.

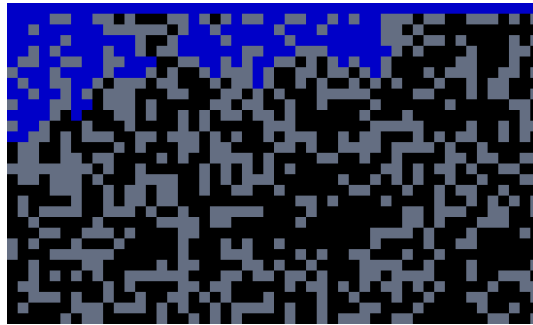
eventuell ins Skript integrieren: in Unterverzeichnis `spielwiese` habe Vorlage-Datei und dann diverse Ideen, die die Schüler programmieren sollen.

Simulationen

✂ **Aufgabe A17** Nächstes Mal: Bessere Vorlage: In der Variablen `zustand` stehen schlicht Strings, also `'fels'`, `'wasser'`, `'luft'`. Per Dictionary `farbe` übersetzt man diese Strings dann in konkrete RGB-Farben.

Versickerung von Wasser in porösem Gestein

- Startkonfiguration:
 - Alle Felder in der obersten Zeile bestehen aus Wasser (blau).
 - Alle anderen Felder sind mit einer gewissen Wahrscheinlichkeit Fels (grau) oder Luft (schwarz).
- In jedem Simulationsschritt werden alle Luftfelder, die direkt zu einem Wasserfeld benachbart sind, zu Wasser.
- Simuliere dies, bis sich nichts mehr ändert, das Wasser also alle erreichbaren Hohlräume ausfüllt.



Vorlage (oder versuche es selbst, eventuell aufbauend auf der Lösung von [A16](#)):

[ex-versickerung-vorlage.py](#) (bitte anklicken)

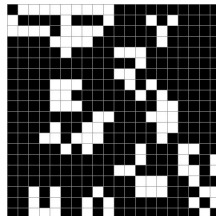
(Falls der Pin nicht anklickbar ist: Lade das Programm von der Freifach-Dokuwiki-Seite herunter.)

In der Vorlage führt das Drücken bzw. Gedrückt-Lassen der Leertaste zu einem bzw. zu mehreren Simulationsschritten.

✂ **Aufgabe A18** Programmiere das Game of Life.

- Startkonfiguration: lebende und tote Felder (zufällig oder vorgegeben).
- Simulationsschritt:
 - Jedes lebende Feld mit genau zwei oder drei Nachbarn überlebt, sonst stirbt es an Über- bzw. Unterbevölkerung. (Jedes Feld hat 8 Nachbarn.)
 - Jedes unbesiedelte Feld mit genau drei lebenden Nachbarn wird lebendig.

https://en.wikipedia.org/wiki/Conways_Game_of_Life.



✂ **Aufgabe A19** Simulation von fallenden Gestein

Starte mit einer zufälligen Felskonfiguration (nur Fels und Luft, eventuell Felsen in unterschiedlichen Grautönen). Lass die Felsen in jedem Simulationsschritt ein Feld nach unten fallen, falls dort Platz ist.

Bemerkung: Falls wir später das Spiel 2048 programmieren, so fallen die Steine in dieser Art in Richtung der Spielfeldränder.

✂ Aufgabe A20 Simulation von Korallenwachstum

Beginne mit dem leeren Spielfeld.

Links oben startet ein Partikel, das sich mit einer gewissen Wahrscheinlichkeit in jedem Simulationsschritt entweder nach unten oder nach rechts bewegt. Stösst es auf den unteren oder rechten Rand, bleibt es dort haften.

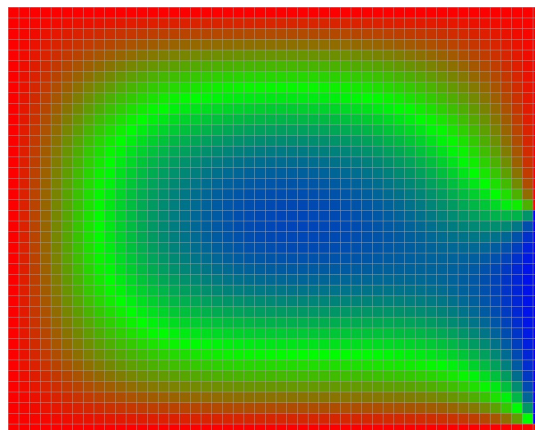
Dann startet das nächste Partikel genauso, bleibt aber auch an vorherigen Partikeln haften, usw.

Die Simulation endet, wenn die Koralle bis zum Feld links oben gewachsen ist.



✂ Aufgabe A21 Wärmeleitungsgleichung

In einem anfangs unbeheizten Raum mit offenem Fenster breitet sich die Wärme von den Heizkörpern an den Wänden aus. In dem Bild sind die Rechtecksseiten die Wände des Raumes; sie haben konstante Temperatur, etwa 20°C; die rechte untere Hälfte der rechten Rechtecksseite ist ein Fenster; dort herrscht die Aussentemperatur 0°C.



Starte mit einer beliebigen Temperaturverteilung.

Die neue Temperatur $T_{\text{neu}}(x, y)$ an einer Position (x, y) berechnet sich in jedem Zeitschritt wie folgt aus den alten Temperaturen $T_{\text{alt}}(x, y)$; dabei ist γ eine Konstante (siehe Vorlage), die für numerische Stabilität sorgt.

$$T_{\text{neu}}(x, y) = T_{\text{alt}}(x, y) + \gamma \cdot \left((T_{\text{alt}}(x-1, y) - 2T_{\text{alt}}(x, y) + T_{\text{alt}}(x+1, y)) + (T_{\text{alt}}(x, y-1) - 2T_{\text{alt}}(x, y) + T_{\text{alt}}(x, y+1)) \right)$$

Vorlage:

[ex-waermeleitungsgleichung-vorlage.py](#) (bitte anklicken)

(Falls der Pin nicht anklickbar ist: Lade das Programm von der Freifach-Dokuwiki-Seite h

(Dies ist die Diskretisierung der Wärmeleitungsgleichung $\frac{\partial u}{\partial t} = \Delta u$, vgl.

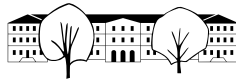
<https://de.wikipedia.org/wiki/W%C3%A4rmeleitungsgleichung>

1.5.1. Sei kreativ und überlege dir selbst eine Simulation (etwa Ausbreitung von Corona, bereits Infizierte werden nicht mehr krank, bewegen sich zufällig über das Spielfeld etc.)

✂ Aufgabe A22 Programmiere Snake. [https://en.wikipedia.org/wiki/Snake_\(video_game_genre\)](https://en.wikipedia.org/wiki/Snake_(video_game_genre))

(Vorlage auf Freifach-Webpage.)

Speichere die Positionen, an denen sich die Snake befindet, in einer Liste.



Wenn sich die Snake beispielsweise an den Positionen (4, 5) und (4, 6) und (4, 7) befindet mit Kopf bei (4, 7), so würde ich dies wie folgt speichern:

```
schlange = [[4, 7], [4, 6], [4, 5]]
```

Wenn man die Schlange um ein Feld nach links bewegen will, muss man das letzte Element der Liste (= Position des Schlangenenendes) löschen und die neue Position des Kopfes, also (4, 8), vorne einfügen. Dies geht wie folgt:

```
schlange.pop()           # Löscht das letzte Element der Liste.
schlange.insert(0, [4, 8]) # Fügt an Position 0 das Element [4, 8] ein.-
```

- Steuerung der Snake mit Pfeiltasten.
- Für jeden gefressenen Apfel gibt es einen Punkt, die Schlange verlängert sich um ein Feld und die Frame rate wächst um 1.
- Du hast 5 Leben.

Vorlage:

[ex-snake-vorlage.py](#) (bitte anklicken)

(Falls der Pin nicht anklickbar ist: Lade das Programm von der Freifach-Dokuwiki-Seite herunter.)

✂ **Aufgabe A23** Programmiere Nibbles.

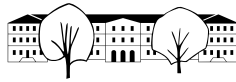
So ähnlich wie Snake; als Vorlage kann man dieselbe wie für Snakes verwenden.

[https://en.wikipedia.org/wiki/Nibbles_\(video_game\)](https://en.wikipedia.org/wiki/Nibbles_(video_game))

✂ **Aufgabe A24** Programmiere Tetris.

✂ **Aufgabe A25** Ziel: 2048 programmieren, vgl. [https://en.wikipedia.org/wiki/2048_\(video_game\)](https://en.wikipedia.org/wiki/2048_(video_game)).

- 1-dimensional, ohne Graphik: Verwende die Vorlage `ex-2048-eindimensional-ohne-graphik-vorlage.py` auf der Freifach-Homepage.
- 1-dimensional, mit Graphik:
 - Programmiere es selbst oder
 - integriere die gerade definierten Funktionen in die Vorlage `ex-2048-eindimensional-vorlage.py` auf der Freifach-Homepage oder
 - verwende gewisse Teile der Vorlage, etwa die Farben oder die Zeichenfunktionen, und programmiere den Rest selbst.
- Programmiere 2048, nun wie üblich 2-dimensional (eventuell wird noch eine Vorlage zur Verfügung gestellt).
- Bonus: Niemand soll daran gehindert werden, 2048 dreidimensional zu programmieren.



1.6 Etwas Algorithmik: Sortieren von Daten

✂ Aufgabe A26

- (a) Finde ein Verfahren (= einen Algorithmus), mit dem man einen gut durchmischten Stapel Spielkarten sortieren kann. Dabei solltest du dir vorstellen, dass du die unsortierten Karten in einer Reihe nebeneinanderlegst und dann wiederholt Karten an gewünschten Positionen miteinander vergleichen und umordnen darfst.
- Führe das von dir gefundene Verfahren mit Spielkarten durch.
 - Bist du sicher, dass dein Verfahren seinen Zweck erfüllt? Kannst du das schlüssig begründen? Hinweis: Was verbessert sich in jedem Schritt?
 - Schreibe eine Beschreibung deines Verfahrens möglichst genau auf. Es sollte eine Art Anleitung sein, die ohne weitere Hinweise von jemand anderem verstanden werden kann.
 - Schreibe dein Verfahren in Pseudo-Code auf. <https://de.wikipedia.org/wiki/Pseudocode>
- (b) Finde ein weiteres Verfahren, das «schlechter» ist als das von dir gefundene Verfahren (es darf auch «extra dumm» sein). In wiefern ist es schlechter?
- (c) Finde ein weiteres Verfahren, das «besser» ist als die von dir gefundenen Verfahren. In welcher Hinsicht ist es besser?
- (d) Finde weitere Verfahren.
- (e) Implementiere die von dir gefundenen Sortierverfahren in Python.¹ Du kannst entweder für jedes Verfahren ein eigenes Programm schreiben oder für jedes Verfahren eine eigene Funktion. Gib die Liste am Anfang und nach jedem Sortierschritt aus. Versuche, nur die elementarsten Listenbefehle zu verwenden:
- Zugriff auf ein Element der Liste (`a[7]`);
 - Einer Listenposition einen Wert zuweisen (`a[3] = 42`);
 - Funktionen bzw. Methoden wie `append`, `insert`, `pop`, `remove`, `sort`, `count`, `sorted`, ... dürfen nicht verwendet werden.

Kontrollstrukturen (Schleifen, Verzweigungen etc.) und weitere Variablen dürfen selbstverständlich verwendet werden.

Vorlage:

```
import random

# Länge der zu sortierenden Liste
n = 10
# Listenelemente haben Werte von 1 bis
max = 50

a = [random.randrange(1, max+1) for i in range(n)]

# Absteigende Liste, zum Testen.
# a = [n-i for i in range(n)]

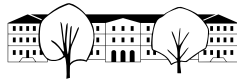
print(a)
```

- (f) Analysiere die von dir implementierten Suchverfahren:
- Wie lange brauchen die Verfahren, um eine Liste von 1'0000'000 Einträgen zu sortieren?
 - Wie viel zusätzlichen Speicherbedarf benötigen die Verfahren? Hängt dies von der Länge der Liste ab oder nicht?
- Wenn dieser nicht von der Länge der Liste abhängt und somit konstant und meist gering ist, spricht man von einem «in-place»-Verfahren oder «in situ»-Verfahren.

¹In Python gibt es «built-in» Sortierverfahren: Ist `a` eine Liste,

- so wird diese durch den Aufruf der Methode `a.sort()` sortiert (dies verändert die Liste `a`);
- so liefert die Funktion `sorted(a)` eine neue sortierte Liste, die aus den Elementen von `a` besteht (wobei `a` nicht verändert wird).

Diese Verfahren solltest du natürlich nicht verwenden.



- Wenn dasselbe Element mehrfach vorkommt: Bleibt die Reihenfolge derselben Einträge erhalten (stabiles Sortierverfahren) oder nicht.
Dies ist relevant, wenn man Beispielsweise eine Liste hat, deren Einträge Vor- und Nachnamen sind. Ist die Liste bereits nach den Vornamen sortiert, so will man eher nicht, dass die relative Reihenfolge verändert wird, wenn man die Liste nach den Nachnamen sortiert.
 - Wie viele Zuweisungen bzw. Vergleiche finden statt?
 - Genauere Laufzeitanalyse:
 - Erstelle zum Beispiel einen Plot (Bibliothek `matplotlib`), der die Laufzeit in Abhängigkeit von der Länge der zu sortierenden Liste darstellt.
Wächst der Aufwand quadratisch?
 - Mathematische Analyse: Finde eine Funktion $t(n)$, die die Laufzeit (oder die Anzahl der Vergleiche, ...) in Abhängigkeit von der Listenlänge angibt.
- (g) Im nachfolgenden Text sind die bekanntesten elementaren Sortierverfahren beschrieben. Implementiere diejenigen in Python, die du nicht selbst gefunden hast.

Bemerkungen: Es gibt sehr viele Sortierverfahren. Auf der Wikipedia-Seite «Sortierverfahren» ist eine Liste von gut 20 Sortierverfahren. Diese haben sehr unterschiedliche Laufzeiten, vgl. etwa https://en.wikipedia.org/wiki/Sorting_algorithm#Table_of_running_times_of_popular_algorithms.



Elementare Suchverfahren

(Von meiner Dokuwiki-Seite übernommen. In der c't war kürzlich eine Serie übers Sortieren, die ich (vermutlich) abfotografiert bzw. gescannt habe.)

Random Sort: Sortieren mit zufälliger Komponente

1.6.1. Kurzbeschreibung: Wähle zufällig zwei Positionen und vertausche die zugehörigen Einträge, wenn die Reihenfolge nicht stimmt. Wiederhole dies, bis die Liste sortiert ist.

1.6.2. Genauer:

Wiederhole, bis sortiert:

- (i) Wähle zwei zufällige Indizes (= Positionen in der Liste).
- (ii) Wenn die beiden zugehörigen Einträge in der falschen Reihenfolge stehen, vertausche sie.

Bubble Sort: Sortieren durch Aufsteigen

1.6.3. Sortieren durch Aufsteigen (man denkt an Blasen = bubbles, die in einer Flüssigkeit aufsteigen, siehe Animation).

Kurzbeschreibung: Man vergleicht nacheinander jedes Element der Liste mit seinem Nachfolger und vertauscht die beiden Elemente, wenn sie in der falschen Reihenfolge stehen. Dies wiederholt man so lange, bis die gesamte Liste sortiert ist.

1.6.4. Genauer:

Sei n die Länge der zu sortierenden Liste.

Wiederhole n Mal (oder so lange, bis keine Vertauschung mehr stattgefunden hat:

- (i) Gehe alle Listenpositionen von Position 0 bis Position $n - 1$ durch:
 - Wenn das Element an der aktuell betrachteten Listenposition echt grösser ist als das nachfolgende Listenelement, vertausche die beiden Elemente.

Selection Sort: Sortieren durch Auswählen

1.6.5. Kurzbeschreibung: Man wählt nacheinander das jeweils kleinste Element aus den noch nicht ausgewählten Elementen und verschiebt es vorne an die richtige Position.

1.6.6. Genauer:

Sei n die Länge der zu sortierenden Liste.

Suche das kleinste Element der Liste und tausche es an Position 0.

Suche ab Position 1 das kleinste Element der Liste und tausche es an Position 1.

Suche ab Position 2 das kleinste Element der Liste und tausche es an Position 2.

...

Suche ab Position $n - 2$ das kleinste Element der Liste und tausche es an Position $n - 2$.

(Unnötig:) Suche ab Position $n - 1$ das kleinste Element der Liste und tausche es an Position $n - 1$.

Insertion Sort: Sortieren durch Einfügen

1.6.7. Kurzbeschreibung: Man fügt nacheinander alle Elemente an der richtigen Position im bisher sortierten Teil ein.

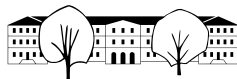
1.6.8. Genauer:

Man stelle sich vor, dass die zu sortierende Liste aus (Spiel-)Karten besteht, die verdeckt in einer Reihe liegen.

Man deckt die erste Karte auf. Dann ist die Liste der aufgedeckten Karten offensichtlich sortiert.

Man deckt dann die nächste Karte auf. Man lässt sie durch Vertauschungen so lange jeweils eine Position nach vorne rücken, bis sie an der richtigen Position liegt, so dass die Liste der aufgedeckten Karten sortiert ist.

Man wiederholt den vorherigen Schritt so lange, bis keine Karte mehr verdeckt daliegt.



1.7 Teile und herrsche – divide and conquer – divide et impera

1.7.1. Teilen-und-Herrschen ist eine Problemlösestrategie, die ständig angewandt wird, ob bewusst oder unbewusst.

Sie besteht darin, dass man ein gegebenes Problem in Teilprobleme zerlegt, diese löst und danach die Lösungen der Teilprobleme zur Lösung des Ausgangsproblems kombiniert.

1.8 Rekursion

1.8.1. Zwei der bekanntesten und schnellsten Sortierverfahren, Mergesort und Quicksort, verwenden Rekursion. Unser Ziel ist, diese Verfahren zu implementieren.

Dazu lernen wir zuerst anhand einfacher Beispiele, was Rekursion ist.

Eine gute Quelle mit vielen netten Beispielen ist das Buch «The recursive Book of Recursion» von Al Sweigart (online frei verfügbar unter <https://inventwithpython.com/recursion/>).

Definition 1.8.2 rekursive Funktion

Eine Funktion heisst **rekursiv**, wenn sie sich selbst aufruft, dies aber nicht unendlich oft geschieht. Dies wird durch geeignete Abbruchbedingungen sichergestellt.

Worterklärung: *rekursiv*, etwa «auf bekannte Werte zurückgehend», von lateinisch *recurere* zurücklaufen)

1.8.3. Die Idee ist, dass man ein Problem in einfachere Teilprobleme **derselben Art** zerlegt, diese dann (rekursiv) löst und die Teillösungen zur Gesamtlösung zusammensetzt. Problemlösen durch Rekursion ist also ein Spezialfall der allgemeinen Teile-und-herrsche-Strategie: Sie kann immer dann verwendet werden, wenn das Ausgangsproblem und die Teilprobleme dieselbe Struktur haben (aber die Teilprobleme weniger kompliziert sind).

Dabei ist darauf zu achten, dass die Zerlegung in Teilaufgaben nicht unendlich oft geschieht: Man muss durch geeignete Abbruchbedingungen sicherstellen, dass man (nach endlich vielen Schritten) bei gewissen Basisfällen ankommt, die direkt lösbar sind.

Beispiel 1.8.4 (Fibonacci-Folge). Die Fibonacci-Folge

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \dots$$

ist dadurch definiert, dass sie mit den beiden Folgengliedern 0 und 1 startet und danach jedes Folgenglied die Summe seiner beiden Vorgänger ist. Mathematisch drückt man dies so aus:

$$\text{Die Fibonacci-Folge } (f_n) \text{ ist definiert durch } \begin{cases} f_0 = 0 & \text{Startwert} \\ f_1 = 1 & \text{weiterer Startwert} \\ f_n = f_{n-1} + f_{n-2} & \text{für alle } n \geq 2 \end{cases}$$

Die Fibonacci-Folge kann man durch das folgende Wachstumsmodell einer Kaninchenpopulation motivieren.

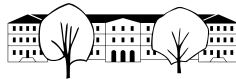
- Im Monat $n = 0$ startet man mit einem Jungtier-Kaninchenpaar.
- Jedes Kaninchenpaar lebt ewig, wird im Alter von 1 Monat erwachsen und wirft ab dem Alter von 2 Monaten (also bereits im Alter von 2 Monaten) jeden Monat ein neues Jungtier-Kaninchenpaar.

Wenn man die die Anzahl der erwachsenen Kaninchenpaare im Monat n mit f_n bezeichnet, so erhält man die Fibonacc-Folge.

Monat n (Monatsende)	Anzahl der Jungtier-Paare	f_n = Anzahl erwachsener Paare
0	1	0
1	0	1
2	1	1
3	1	2
4	2	3
5	3	5
6	5	8

Hier ist ein rekursives Programm zur Berechnung der n -ten Fibonacci-Zahl f_n :

```
def fib(n):
    # Abbruchbedingungen für n=0 und n=1:
    if n == 0:
        return 0
    elif n == 1:
        return 1
```



```

else:
    # Rekursiver Aufruf der Funktion (hier zweifach)
    return fib(n-1)+fib(n-2)
print(fib(10))
print(fib(38))
    
```

Hier ist ein iteratives Programm zur Berechnung von f_n (die Laufvariable i wird nicht verwendet; es geht nur darum, dass die Schleife $n - 1$ mal ausgeführt wird):

```

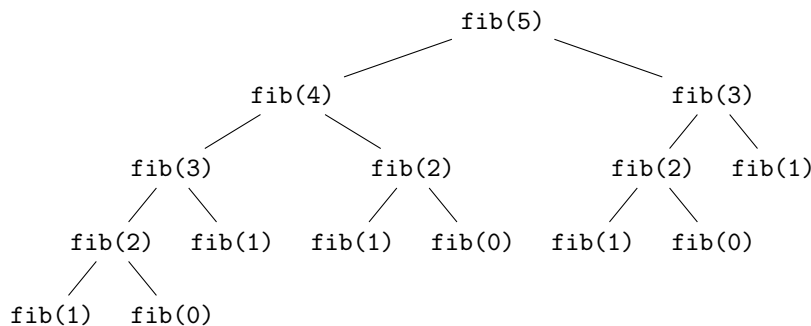
def f(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a+b
    return a
print(f(1))
print(f(10))
print(f(38))
    
```

Ein weiteres iteratives Programm zur Berechnung von f_n ; genauer erzeugen wir hier die Liste aller Fibonacci-Zahlen von f_0 bis f_n :

```

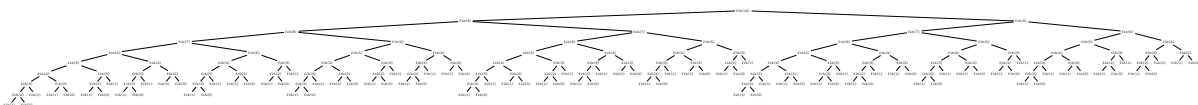
def fi(n):
    fib = [0, 1]
    for i in range(n-1):
        fib.append(fib[-1]+fib[-2])
    return fib
print(fi(10))
    
```

Warnung 1.8.5. Die Berechnung von f_n mit Hilfe der oben angegebenen rekursiven Funktion ist zwar ein schönes erstes Beispiel für eine rekursive Funktion, jedoch ist in diesem Fall das iterative Vorgehen deutlich schneller. Der Grund ist, dass in der rekursiven Funktion dieselben Funktionswerte mehrfach berechnet werden. Dies verdeutlicht der folgende Baum, der die Funktionsaufrufe zur Berechnung von `fib(5)` darstellt.

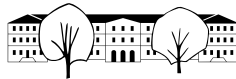


Beispielsweise wird f_3 zweimal und f_2 dreimal berechnet. (Dass f_0 und f_1 mehrfach vorkommen, ist eher kein Problem, denn dies sind gespeicherte Startwerte).

Zur grösseren Abschreckung hier noch der Baum der Funktionsaufrufe für `fib(10)`. (Ohne es genau geprüft zu haben: Die Anzahl der Aufrufe der Funktion `fib` mit einem festen Argument, also etwa `fib(2)`, ist wohl wieder eine Fibonacci-Zahl.)



Der Leser ist eingeladen, die Fibonacci-Zahl f_{100} mit der iterativen Funktion `f(100)` und der rekursiven Funktion `fib(100)` zu berechnen – die rekursive Berechnung mag etwa eine Million Jahre dauern.



✂ Aufgabe A27 Die Folge

0, 1, 2, 4, 11, 25, 59, 142, 335, 796, 1892, 4489, ...

ist wie folgt definiert.

- Die Anfangsglieder sind 0, 1, 2.
- Jedes nachfolgende Folgenglied ist die Summe aus
 - dem vorhergehenden Folgenglied und
 - dem Doppelten des vorvorhergehenden Folgenglieds und
 - dem Dreifachen des vorvorvorhergehenden Folgenglieds.

Beispielsweise gilt $25 = 11 + 2 \cdot 4 + 3 \cdot 2$.

Schreibe zwei Funktionen, die das n -te Folgenglied berechnen: einmal rekursiv, einmal iterativ. Orientiere dich dabei an den obigen Beispielen zur Berechnung der Fibonacci-Folge.

Beispiel 1.8.6. Das folgende Programm berechnet die Summe der Zahlen von 1 bis n iterativ, also $1 + 2 + 3 + \dots + 10$.

```
def s(n):
    summe = 0
    for i in range(1, n+1):
        # Die folgende Zeile erhöht summe um i.
        # Ausführlich würde man schreiben
        # summe = summe + i
        summe += i
    return summe
print(s(10))
print(s(20))
```

Das folgende Programm berechnet die Summe der Zahlen von 1 bis n rekursiv.

```
def summe(n):
    if n == 0:
        return 0
    else:
        return n+summe(n-1)

print(summe(10))
print(summe(20))
```

Dasselbe geht natürlich auch direkt mit der Python-Funktion `sum`:

```
def su(n):
    a = list(range(n+1))
    return sum(a)

print(su(10))
print(su(20))
```

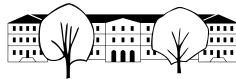
✂ Aufgabe A28 Die Fakultät $n!$ einer natürlichen Zahl n ist definiert als das Produkt aller natürlichen Zahlen von 1 bis n , d. h.

$$n! := 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

(Per Konvention bzw. als leeres Produkt gilt $0! = 1$.)

Äquivalent kann man $n!$ rekursiv definieren:

$$n! := \begin{cases} 1 & \text{falls } n = 0, \\ n \cdot (n-1)! & \text{falls } n > 0. \end{cases}$$



Schreibe zwei Funktionen, die $n!$ berechnen: einmal `rekursiv`, einmal `iterativ`. Orientiere dich dabei an dem obigen Beispiel zur Berechnung der Summe der ersten n Zahlen.

(Mit geeigneten Python-Bibliotheken kann man $n!$ auch direkt berechnen. Wie? Auf Englisch spricht man von « n factorial».)

Beispiel 1.8.7. Das folgende Programm zeigt, wie man das grösste Element einer (nicht leeren) Liste auf drei Arten berechnen kann:

- rekursiv (was ist die Idee des Programms?);
- iterativ;
- mit der Python-Funktion `max`.

```
from random import randrange
n = 10
# Werte von 1 bis
max_wert = 50

# rekursive Funktion
def maximum(a):
    if len(a) == 1:
        return a[0]
    else:
        t = len(a) // 2
        linke_haelfte = a[:t]
        rechte_haelfte = a[t:]
        # Es folgen zwei rekursive Aufrufe.
        max_links = maximum(linke_haelfte)
        max_rechts = maximum(rechte_haelfte)
        if max_links >= max_rechts:
            return max_links
        else:
            return max_rechts

# iterative Funktion
def m(a):
    k = a[0]
    for x in a:
        if x > k:
            k = x
    return k

z = [randrange(max_wert+1) for i in range(n)]
print(z)
print(maximum(z))
print(m(z))
# per built-in-function max
print(max(z))
```

✂ **Aufgabe A29** Schreibe zwei Funktionen, die die Summe aller Elemente einer (zufällig erzeugten) Liste berechnen: einmal `iterativ`, einmal `rekursiv`, einmal rekursiv durch «Zerteilung der Liste in zwei Hälften» wie in dem obigen Beispiel zur rekursiven Berechnung des Maximums einer Liste von Zahlen.

Beispiel 1.8.8. Das folgende rekursive Programm zeigt, wie man feststellt, ob ein String ein **Palindrom** ist (= eine Buchstabenfolge, die von hinten gelesen dieselbe Buchstabenfolge liefert wie von vorne).

Bevor du das Programm liest: Halte kurz inne und überlege, wie du dieses Problem rekursiv lösen kannst.

```
def ist_palindrom(s):
    if len(s) < 1:
```



```
        return True
    else:
        start = s[0]
        ende = s[-1]
        mitte = s[1:-1]
        return start == ende and ist_palindrom(mitte)

text = 'level'
print(ist_palindrom(text))

text = 'racecar'
print(ist_palindrom(text))

text = 'tacocat'
print(ist_palindrom(text))

# A man, a plan, a canal: Panama
text = 'amanaplanacanalpanama'
print(ist_palindrom(text))

text = 'reliefpfeiler'
print(ist_palindrom(text))

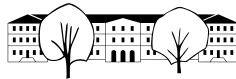
text = 'ananas'
print(ist_palindrom(text))
```

✂ **Aufgabe A30** Schreibe eine rekursive Funktion `umgekehrt(s)`, die einen String umkehrt (also von hinten gelesen zurückliefert).

Beispielsweise soll `lager` zu `regal` werden.

✂ **Aufgabe A31** Schreibe eine rekursive Funktion, die alle Permutationen (= Umordnungen) eines Strings (oder einer Liste von Elementen) liefert (etwa direkte Ausgabe oder durch Rückgabe einer Liste, deren Elemente alle Permutationen sind). Dabei nehmen wir an, dass kein Buchstabe in dem String doppelt vorkommt.

Beispiel: Die Permutationen von `abc` sind: `abc`, `acb`, `bac`, `bca`, `cab`, `cba`

**Algorithmus 1.8.9** Mergesort (John von Neumann 1945)

Mergesort ist ein (oft rekursiv implementierter) Algorithmus zum Sortieren einer Liste.

Gegeben: Eine Liste a zu sortierender Objekte.

In Pseudo-Code funktioniert Mergesort wie folgt:

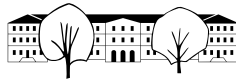
```
mergesort(a):
    Wenn a nur aus einem Element besteht:
        Liefere a zurück.
    Sonst:
        Teile a in zwei Teillisten links und rechts (beide etwa halb so
        ↪ gross).
        l_sortiert = mergesort(links)
        r_sortiert = mergesort(rechts)

        Verschmelze die beiden Listen l_sortiert und r_sortiert zu einer
        ↪ sortierten Liste b wie folgt:
            Kreiere eine leere Liste b.
            Vergleiche jeweils die 'vordersten' Elemente von l_sortiert
            ↪ und r_sortiert und
                hänge das kleinere der beiden Elemente ans Ende der Liste b.

        Gib b als Funktionswert zurück.
```

✂ Aufgabe A32

- (a) Implementiere den Algorithmus Mergesort 1.8.9 als rekursive Funktion in Python (und teste dein Programm mit einigen zufällig erzeugten Listen).
Hinweis: Das Beispiel 1.8.7 dürfte beim Teilen der Liste in zwei Teillisten helfen.
- (b) Wie lange Listen kann das Mergesort in 10 Sekunden sortieren?
Wie lange Listen kann Bubble sort bzw. Insertion sort bzw. Selection sort in 10 Sekunden sortieren?
Warum ist Mergesort so viel besser?

**Algorithmus 1.8.10** Quicksort (C. Antony R. Hoare, ca. 1960)

Quicksort ist ein (oft rekursiv implementierter) Algorithmus zum Sortieren einer Liste.

Gegeben: Eine Liste a zu sortierender Objekte.

In Pseudo-Code funktioniert Quicksort wie folgt:

```
quicksort(a):
    Wenn a nur aus einem Element besteht:
        Liefere a zurück.
    Sonst:
        Wähle ein beliebiges Element der Liste a und nenne es p.
        (p wird Pivot-Element genannt, von französisch pivot, Dreh-,
         ↪ Angelpunkt).

        Kreiere zwei Teillisten kleiner und grösser mit:
            kleiner enthält alle Elemente von a, die kleiner-gleich p sind
            ↪ ;
            grösser enthält alle Elemente von a, die grösser als p sind.

        k_sortiert = quicksort(kleiner)
        g_sortiert = quicksort(groesser)

        Fusioniere k_sortiert und g_sortiert in offensichtlicher Weise zu
        ↪ einer sortierten Liste.
        Gib diese fusionierte Liste als Funktionswert zurück.
```

✂ Aufgabe A33

- Implementiere den Algorithmus Quicksort 1.8.10 als rekursive Funktion in Python (und teste dein Programm mit einigen zufällig erzeugten Listen).
- Wie lange Listen kann das Mergesort in 10 Sekunden sortieren?
Wie lange Listen kann Bubble sort bzw. Insertion sort bzw. Selection sort in 10 Sekunden sortieren?
Warum ist Mergesort so viel besser?



Quicksort in der funktionalen Programmiersprache Haskell

1.8.11. Haskell ist eine rein funktionale Programmiersprache.

Hier siehst du eine Quicksort-Implementation in Haskell.

```
main = do
  let a = [10, 9..0]
  print a
  print(quicksort a)
  let a = [4, 2, 7, 10, 3, 0, 9, 1, 6, 5, 8]
  print a
  print(quicksort a)

quicksort :: Ord a => [a] -> [a]
quicksort []      = []
quicksort (p:xs) = quicksort kleiner ++ [p] ++ quicksort groesser
  where
    kleiner = filter (<p) xs
    groesser = filter (>=p) xs
```

Man kann auch die letzten drei Zeilen in die darüberliegende Zeile packen und Quicksort so im Wesentlichen in einer Zeile programmieren:

```
main = do
  let a = [4, 2, 7, 10, 3, 0, 9, 1, 6, 5, 8]
  print(quicksort a)

quicksort :: Ord a => [a] -> [a]
quicksort []      = []
quicksort (p:xs) = quicksort(filter (<p) xs)++[p]++quicksort(filter (>=p) xs)
```

Oft sind funktionale Programme deutlich kompakter als iterative und ich habe das Gefühl, dass das Abstraktionsniveau oft höher ist als bei imperativen Programmen.

Hier sind zwei Links, falls jemand mehr Lust auf Haskell hat:

- HaskellWiki <https://wiki.haskell.org/Haskell>,
- Installation: <https://www.haskell.org/get-started/>
- Online-Kurs (der auf den ersten Blick nett aussieht): <https://web.cs.dal.ca/~nzeh/Teaching/3137/haskell/>.

1.9 Fraktale zeichnen (rekursiv mit der turtle-Bibliothek)

1.9.1. Beim Zeichnen mit «Turtle-Graphik» steuert man eine imaginäre Schildkröte über den Bildschirm, indem man ihr beispielsweise sagt, wie weit sie gehen soll oder um welchen Winkel sie sich drehen soll. Die Schildkröte zeichnet mit einem Stift ihren Weg auf, sofern der Zeichenstift nicht angehoben ist.

Man experimentiere mit dem folgenden Programm, um die grundlegenden Turtle-Befehle zu lernen.

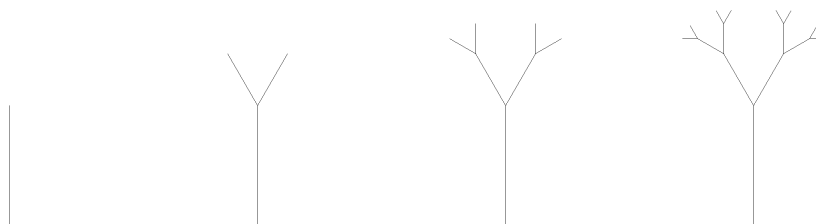
```

from turtle import *
# Grösse des Zeichenfensters festlegen.
Screen().setup(800, 800)
# Für schnelleres Zeichnen die beiden folgenden Zeilen
# und die Zeile mit dem update()-Befehl auskommentieren.
# tracer(0)
# hideturtle()
forward(100)
left(90)
forward(200)
right(90)
penup()
forward(100)
pendown()
forward(100)
# Position per Koordinaten setzen
setposition(-100, 100)
# Blickwinkel setzen (in Grad, von x-Achse in mathematisch positivem Drehsinn gemessen)
setheading(90)
forward(100)
# update()
exitonclick()

```

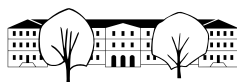
Wer mehr Befehle lernen möchte, sei auf <https://docs.python.org/3/library/turtle.html> verwiesen.

✂ **Aufgabe A34** Wachstumssimulation eines Baums. Die folgenden Bilder entstehen wie folgt. Vom ersten zum zweiten Bild wird die Ausgangsstrecke durch die zwei gezeigten «Äste» ergänzt (jeder Ast hat die halbe Länge der Ausgangsstrecke). Um dann jeweils das nächste Bild zu erhalten, wird jeder zuvor neu gezeichnete Ast um zwei neue Äste der halben Länge ergänzt.



Schreibe eine Funktion `baum(tiefe, laenge)`, so dass die folgenden Funktionsaufrufe den angegebenen Effekt haben.

- `baum(0, laenge)` zeichnet das erste Bild: Die Turtle geht um `laenge` Pixel vorwärts und dann wieder genauso weit rückwärts, so dass sie wieder in der Ausgangsposition ist.
- `baum(1, laenge)` zeichnet das zweite Bild: Die Turtle geht um `laenge` Pixel vorwärts. Dann dreht sie sich um einen fixierten Winkel α nach links, ruft (rekursiv) `baum(0, laenge/2)` auf (was den linken Ast zeichnet), dreht sich um 2α nach rechts, ruft wieder `baum(0, laenge/2)` auf (was den rechten Ast zeichnet), dreht sich um α nach links und geht dann wieder `laenge` Pixel rückwärts (so dass sie wieder in der Ausgangsposition ist).
- `baum(tiefe, laenge)`: Die Turtle geht um `laenge` Pixel vorwärts. Dann dreht sie sich um einen fixierten Winkel α nach links, ruft `baum(tiefe-1, laenge/2)` auf, dreht sich um 2α nach rechts, ruft wieder



`baum(tiefe-1, laenge/2)` auf, dreht sich um α nach links und geht dann wieder `laenge` Pixel rückwärts (so dass sie wieder in der Ausgangsposition ist).

Bemerkung: Du darfst gerne etwas mit den Bewegungsdaten spielen: Etwa drei Äste statt zweier Äste zeichnen; unterschiedliche Winkel in den Verzweigungen wählen; die Astlänge mit anderen Faktoren als $\frac{1}{2}$ multiplizieren (man könnte bei zwei Ästen zum Beispiel die Astlänge beim linken Ast mit $\frac{2}{3}$ multiplizieren und die Astlänge beim rechten Ast mit $\frac{1}{3}$).

✂ **Aufgabe A35** Die folgenden Bilder entstehen wie folgt. Vom ersten zum zweiten Bild wird die Ausgangsstrecke durch den gezeigten Streckenzug aus 5 Strecken (der gedrittelten Länge) ersetzt. Um dann jeweils das nächste Bild zu erhalten, wird jede Strecke durch denselben, geeignet verkleinerten Streckenzug ersetzt.

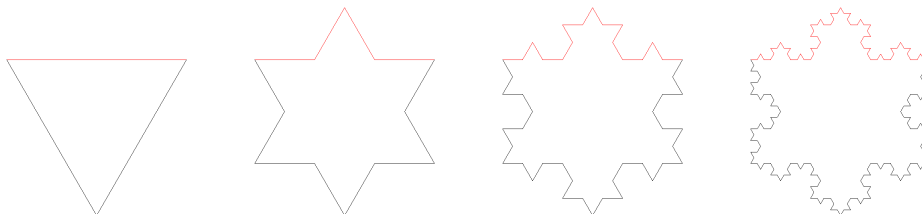


Schreibe eine Funktion `hutkurve(tiefe, laenge)`, so dass die folgenden Funktionsaufrufe den angegebenen Effekt haben.

- `hutkurve(0, laenge)` zeichnet das erste Bild: Die Turtle geht um `laenge` Pixel vorwärts.
- `hutkurve(1, laenge)` zeichnet das zweite Bild: Die Turtle ruft fünfmal `hutkurve(0, laenge/3)` auf und dreht sich dazwischen um geeignete Winkel. (Die Turtle geht nicht zurück, sondern schaut am Ende der Funktion in dieselbe Richtung wie am Anfang, ist jedoch um `laenge` Pixel nach rechts gewandert.)
- `hutkurve(n, laenge)`: Die Turtle ruft fünfmal `hutkurve(tiefe-1, laenge/3)` auf und dreht sich dazwischen um geeignete Winkel.

Bemerkung: Laut https://en.wikipedia.org/wiki/L-system#Example_4:_Koch_curve heisst diese Kurve wohl auch Koch-Kurve.

✂ **Aufgabe A36** (Kochkurve und kochsche Schneeflocke) Der in Rot gezeichnete Teil der folgenden Bilder entsteht wie folgt. Vom ersten zum zweiten Bild wird die Ausgangsstrecke durch den gezeigten Streckenzug aus 3 Strecken (der gedrittelten Länge) ersetzt, alle Abbiegewinkel sind Vielfache von 60° . Um dann jeweils das nächste Bild zu erhalten, wird jede Strecke durch denselben, geeignet verkleinerten Streckenzug ersetzt.

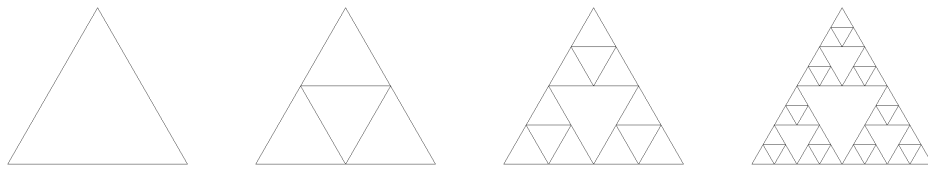


Schreibe eine rekursive Funktion `kochkurve(tiefe, laenge)`, die für `tiefe= 0` die rote Kurve in der linken Zeichnung zeichnet, für `tiefe= 1` die rote Kurve in der zweiten Zeichnung von links etc.

Nutze diese Funktion, um die sogenannte Kochsche Schneeflocke zu zeichnen, die aus drei «roten Kurven» gebildet wird.

✂ **Aufgabe A37** (Sierpinski-Dreieck)

- Vom ersten zum zweiten Bild wird das Ausgangsdreieck durch drei Dreiecke der halben Seitenlänge ersetzt, und dies geht dann genau weiter.



Schreibe eine rekursive Funktion `sierpinksi(tiefe, laenge)`, die für geeignete Argumente die obigen Zeichnungen anfertigt.

(b) (Achtung, diese Teilaufgabe ist etwas schwieriger, als man auf den ersten Blick denkt.)



Schreibe eine rekursive Funktion `sierpinski_variante` mit geeigneten Parametern, die für geeignete Argumente die obigen Zeichnungen anfertigt.

(Für die zweite Variante, vgl. https://en.wikipedia.org/wiki/L-system#Example_5:_Sierpinski_triangle oder https://en.wikipedia.org/wiki/Sierpinski_curve#Arrowhead_curve.)

✂ **Aufgabe A38** (Hilbert-Kurve; raumfüllende Kurve) Schreibe eine Funktion, die die unten dargestellten Hilbert-Kurven zeichnet.

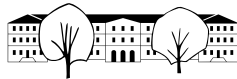
Hinweis: Vermutlich ist https://en.wikipedia.org/wiki/Hilbert_curve#Representation_as_Lindenmayer_system nützlich.



✂ **Aufgabe A39** Suche im Internet den Begriff «Pythagoras-Baum» und erstelle eine entsprechende Zeichnung mit Hilfe einer rekursiven Funktion.

1.9.2. Wohl auch interessant:

- <https://en.wikipedia.org/wiki/L-system>
- <https://en.wikipedia.org/wiki/Rewriting>



1.10 Lösungen

Hinweise zu den Symbolen:

✂ Diese Aufgaben könnten (mit kleinen Anpassungen) an einer Prüfung vorkommen. Für die Prüfungsvorbereitung gilt: "If you want to nail it, you'll need it".

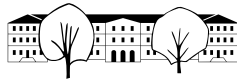
✪ Diese Aufgaben sind wichtig, um das Verständnis des Prüfungsstoffs zu vertiefen. Die Aufgaben sind in der Form aber eher nicht geeignet für eine Prüfung (zu grosser Umfang, nötige «Tricks», zu offene Aufgabenstellung, etc.). **Teile solcher Aufgaben können aber durchaus in einer Prüfung vorkommen!**

✂ Diese Aufgaben sind dazu da, über den Tellerrand hinaus zu schauen und/oder die Theorie in einen grösseren Kontext zu stellen.

✂ Lösung zu **A1** ex-renten-sparen

```
betrag = 6000
zins = 0.01
laufzeit = 35
n = 1
kontostand = 0
while n <= laufzeit:
    kontostand = kontostand+betrag
    kontostand = kontostand*(1+zins)
    print(f"Kontostand am Ende des Jahres {n}: {kontostand:.2f}")
    n = n+1
```

```
Kontostand am Ende des Jahres 1: 6060.00
Kontostand am Ende des Jahres 2: 12180.60
Kontostand am Ende des Jahres 3: 18362.41
Kontostand am Ende des Jahres 4: 24606.03
Kontostand am Ende des Jahres 5: 30912.09
Kontostand am Ende des Jahres 6: 37281.21
Kontostand am Ende des Jahres 7: 43714.02
Kontostand am Ende des Jahres 8: 50211.16
Kontostand am Ende des Jahres 9: 56773.28
Kontostand am Ende des Jahres 10: 63401.01
Kontostand am Ende des Jahres 11: 70095.02
Kontostand am Ende des Jahres 12: 76855.97
Kontostand am Ende des Jahres 13: 83684.53
Kontostand am Ende des Jahres 14: 90581.37
Kontostand am Ende des Jahres 15: 97547.19
Kontostand am Ende des Jahres 16: 104582.66
Kontostand am Ende des Jahres 17: 111688.49
Kontostand am Ende des Jahres 18: 118865.37
Kontostand am Ende des Jahres 19: 126114.02
Kontostand am Ende des Jahres 20: 133435.16
Kontostand am Ende des Jahres 21: 140829.52
Kontostand am Ende des Jahres 22: 148297.81
Kontostand am Ende des Jahres 23: 155840.79
Kontostand am Ende des Jahres 24: 163459.20
Kontostand am Ende des Jahres 25: 171153.79
Kontostand am Ende des Jahres 26: 178925.33
Kontostand am Ende des Jahres 27: 186774.58
Kontostand am Ende des Jahres 28: 194702.33
Kontostand am Ende des Jahres 29: 202709.35
Kontostand am Ende des Jahres 30: 210796.44
Kontostand am Ende des Jahres 31: 218964.41
Kontostand am Ende des Jahres 32: 227214.05
```



```
Kontostand am Ende des Jahres 33: 235546.19
Kontostand am Ende des Jahres 34: 243961.65
Kontostand am Ende des Jahres 35: 252461.27
```

✂ Lösung zu A2 ex-wurzel-annaehern

```
x = 40
fehler = 0.000001
links = 0
rechts = x

while rechts - links > fehler:
    mittel = (links + rechts) / 2
    if mittel**2 < x:
        links = mittel
    else:
        rechts = mittel

schaetzung = (links + rechts) / 2
print(f'{schaetzung} ist ungefähr die Wurzel aus {x}.')
print(f'{x**0.5} ist laut Python die Wurzel aus {x}.')
print(f'Der Fehler ist etwa {abs(mittel-x**0.5):.10f}.')
```

```
6.324555575847626 ist ungefähr die Wurzel aus 40.
6.324555320336759 ist laut Python die Wurzel aus 40.
Der Fehler ist etwa 0.0000000425.
```

✂ Lösung zu A3 ex-fibonacci-zahlen

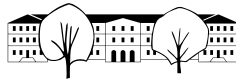
```
maxn = 6
print('Nr Fibonaccizahl ')
x = 0
y = 1
nummer = 0
while nummer <= maxn:
    print(nummer, x)
    summe = x + y
    x = y
    y = summe
    nummer = nummer + 1
```

Für Nerds, etwas kürzer und rekursiv (was man bei Fibonacci eigentlich nicht machen sollte) mit sogenannter Lambdanotation (zur raschen Definition einer Funktion):

```
maxn = 6
fib = lambda n: n if n < 2 else fib(n-1) + fib(n-2)
print([fib(x) for x in range(maxn + 1)])
```

✂ Lösung zu A4 ex-fibonacci-zahlen-als-liste

```
maxn = 6
fibonaccifolge = [0, 1]
n = 2
```



```
while n <= maxn:
    fibonaccifolge.append(fibonaccifolge[n-1]+fibonaccifolge[n-2])
    n = n+1
print(fibonaccifolge)
```

Mit einer for-Schleife statt einer while-Schleife und mit «negativen» Indizes -1 und -2 (sie beziehen sich auf das letzte und vorletzte Element der Liste). Unwichtige Laufvariablen werden in Python bisweilen «Unterstrich» genannt.

```
maxn = 6
fibonaccifolge = [0, 1]
for _ in range(2, maxn+1):
    fibonaccifolge.append(fibonaccifolge[-1]+fibonaccifolge[-2])
print(fibonaccifolge)
```

✂ Lösung zu A5 ex-fibonacci-zahlen-list-comprehension-explizite-formel

```
maxn = 6
phi = (1+5**0.5)/2
psi = (1-5**0.5)/2
fibonaccifolge = [1/5**0.5*(phi**n-psi**n) for n in range(maxn+1)]
print(fibonaccifolge)

# Alternative: Wir runden alle Ergebnisse mit round(...)
# zu ganzen Zahlen (integers), was Rundungsfehler beseitigt.

fibonaccifolge = [round(1/5**0.5*(phi**n-psi**n)) for n in range(maxn+1)]
print(fibonaccifolge)
```

✂ Lösung zu A6 ex-pi-wuerfeln

```
from random import random

n = 1000      # Anzahl Punkte = Experimente
i = 0        # Laufvariable für Wiederholung
drinnen = 0  # Zählvariable für Anzahl der Punkte im Kreis

while i < n:
    i += 1      # Kurzform für i = i + 1
    x = random()
    y = random()
    # Pythagoras lässt grüssen
    r = (x*x+y*y)**0.5
    if (r < 1):
        drinnen += 1 # Kurzform für drinnen = drinnen + 1

print("Drinne", drinnen, "von", n)
anteil = drinnen/n;
print("Anteil", anteil)
# Viertelkreisfläche ist pi/4, Quadratfläche ist 1
print("Pi ist ungefähr", anteil*4)
```

```
Drinne 773 von 1000
Anteil 0.773
Pi ist ungefähr 3.092
```



✂ Lösung zu [A7](#) ex-multiplikationstabelle

✂ Lösung zu [A8](#) ex-basics-if-kopfrechentrainer

✂ Lösung zu [A9](#) ex-zahlen-in-liste-einlesen

✂ Lösung zu [A10](#) ex-basics-listen

✂ Lösung zu [A11](#) ex-quiz

✂ Lösung zu [A12](#) ex-basics-dictionaries

✂ Lösung zu [A13](#) ex-notenliste-fuer-klasse

✂ Lösung zu [A14](#) ex-tic-tac-toe-mensch-gegen-mensch

✂ Lösung zu [A15](#) ex-bouncing-ball-and-paddle

✂ Lösung zu [A16](#) ex-raster-aus-quadraten

Moderne Kunst

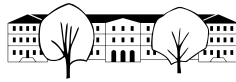
```
import pygame
import random

# Anzahl Quadrate in x-Richtung
breite = 24
# Anzahl Quadrat in y-Richtung
hoehe = 16
# Seitenlänge eines Quadrats in Pixel
laenge = 60

def zeichne_quadrat(x, y, farbe):
    xx = x*laenge
    yy = fensterhoehe-y*laenge
    rechteck = pygame.Rect(xx, yy-laenge, laenge, laenge)
    pygame.draw.rect(surface=fenster, color=farbe, rect=rechteck)

# Abmessungen des Spielfelds
fensterbreite = breite*laenge
fensterhoehe = hoehe*laenge

# Initialisierungen
pygame.init()
pygame.display.set_caption("Moderne Kunst")
clockobject = pygame.time.Clock()
fensterdimensionen = (fensterbreite, fensterhoehe)
fenster = pygame.display.set_mode(fensterdimensionen)
```



```
for x in range(breite):
    for y in range(hoehe):
        rot = random.randrange(256)
        gruen = random.randrange(256)
        blau = random.randrange(256)
        zeichne_quadrat(x, y, (rot, gruen, blau))

while True:
    for ereignis in pygame.event.get():
        if ereignis.type == pygame.QUIT:
            exit()
    pygame.display.flip()
    clockobject.tick(10)
```

✂ Lösung zu A17 ex-versickerung

```
import pygame
import sys
from random import *

BREITE = 40
HOEHE = 20

# EINHEIT gibt die Seitenlänge eines Quadrats in Pixel an.
EINHEIT = 30

# Wahrscheinlichkeit für Fels:
p_fels = 0.45

def zeichne_quadrat(x, y, farbe):
    rechteck = pygame.Rect(x*EINHEIT, HOEHE*EINHEIT-y*EINHEIT-EINHEIT, EINHEIT, EINHEIT)
    pygame.draw.rect(surface=fenster, color=farbe, rect=rechteck)

pygame.init()
pygame.display.set_caption("Simulation: Versickerung")
uhr = pygame.time.Clock()

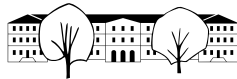
fenster = pygame.display.set_mode((BREITE*EINHEIT, HOEHE*EINHEIT))

# Dictionary: Speichert die farben für die drei Zustände eines Feldes:
farbe = {
    'luft' : [0, 0, 0],
    'fels' : [100, 110, 130],
    'wasser' : [0, 0, 200]
}

# Erstmal alle Felder auf Luft setzen.
zustand = [['luft' for y in range(HOEHE)] for x in range(BREITE)]

# Wasser in oberster Zeile.
for x in range(BREITE):
    zustand[x][HOEHE-1] = 'wasser'

# Zufällig Felsen verteilen.
for x in range(BREITE):
```



```

    for y in range(HOEHE-1):
        if random() < p_fels:
            zustand[x][y] = 'fels'

print("""
Simulation einer Versickerung
Nächster Schritt per Leertaste. Leertaste gedrückt lassen für mehrere Schritte.
Fenster schliessen per Maus oder per Taste 'q'
""")

# Sorgt dafür, dass das Gedrücktlassen der Leertaste nach 400 Millisekunden
# weitere Leertaste-gedrückt-Ereignisse liefert,
# im Abstand von 100 Millisekunden.
pygame.key.set_repeat(400, 100)

leertaste = False # wird kurz True, wenn der Benutzer die Leertaste drückt.
veraenderung = True # wird False, sobald der Endzustand erreicht ist.
programmende = False # wird True, sobald der Benutzer das Fenster schliesst oder die T

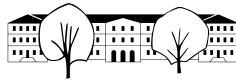
while True:
    for ereignis in pygame.event.get():
        if ereignis.type == pygame.QUIT:
            print('Fenster schliessen.')
            programmende = True
        elif ereignis.type == pygame.KEYDOWN:
            if ereignis.key == pygame.K_SPACE:
                print('Leertaste gedrückt')
                if veraenderung == False:
                    print('Endzustand bereits erreicht.')
                    leertaste = True
            elif ereignis.key == pygame.K_q:
                print("Taste 'q' gedrückt")
                programmende = True
    if programmende:
        pygame.quit()
        sys.exit()
    if leertaste and veraenderung:
        # Kopie das aktuellen Zustands.
        neu = [[zustand[x][y] for y in range(HOEHE)] for x in range(BREITE)]
        veraenderung = False
        for x in range(BREITE):
            for y in range(HOEHE):
                if neu[x][y] == 'luft':
                    if (x-1 >= 0 and zustand[x-1][y] == 'wasser') or (x+1<=BREITE-1 an
                        neu[x][y] = 'wasser'
                        veraenderung = True
        if veraenderung == False:
            print('Endzustand erreicht.')

        zustand = [[neu[x][y] for y in range(HOEHE)] for x in range(BREITE)]
        leertaste = False

    for x in range(BREITE):
        for y in range(HOEHE):
            zeichne_quadrat(x, y, farbe[zustand[x][y]])

pygame.display.flip()
uhr.tick(100)

```



Geschickter (die aktuell relevanten Wasserfelder werden gespeichert; ausserdem werden nur die neu gezeichneten Quadrate upgedatet; in der Variablen `zustand` werden zusätzlich «Randfelder» gespeichert, so dass man keine “out of bounds”-Tests durchführen muss):

```
import pygame
import sys
from random import *

BREITE = 800
HOEHE = 400

# EINHEIT gibt die Seitenlänge eines Quadrats in Pixel an.
EINHEIT = 2

# Wahrscheinlichkeit für Fels:
p_fels = 0.41

def zeichne_quadrat(x, y, farbe):
    rechteck = pygame.Rect(x*EINHEIT, HOEHE*EINHEIT-y*EINHEIT-EINHEIT, EINHEIT, EINHEIT)
    return pygame.draw.rect(surface=fenster, color=farbe, rect=rechteck)

pygame.init()
pygame.display.set_caption("Simulation: Versickerung")

fenster = pygame.display.set_mode((BREITE*EINHEIT, HOEHE*EINHEIT))

# Dictionary: Speichert die farben für die drei Zustände eines Feldes:
farbe = {
    'luft' : [0, 0, 0],
    'fels' : [100, 110, 130],
    'wasser' : [0, 0, 200]
}

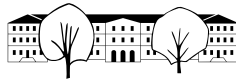
# Erstmal alle Felder auf Luft setzen.
zustand = [['luft' for y in range(HOEHE+2)] for x in range(BREITE+2)]

for x in range(BREITE):
    zustand[x][-1] = 'rand'
    zustand[x][HOEHE] = 'rand'

for y in range(HOEHE):
    zustand[-1][y] = 'rand'
    zustand[BREITE][y] = 'rand'

nass = []
# Wasser in oberster Zeile.
for x in range(BREITE):
    zustand[x][HOEHE-1] = 'wasser'
    nass.append([x, HOEHE-1])

# Zufällig Felsen verteilen.
for x in range(BREITE):
    for y in range(HOEHE-1):
        if random() < p_fels:
            zustand[x][y] = 'fels'
```



```

print("""
Simulation einer Versickerung
Fenster schliessen per Maus oder per Taste 'q'
""")

veraenderung = True # wird False, sobald der Endzustand erreicht ist.

pygame.event.set_allowed([pygame.QUIT])

for x in range(BREITE):
    for y in range(HOEHE):
        zeichne_quadrat(x, y, farbe[zustand[x][y]])

pygame.display.flip()

while True:
    for ereignis in pygame.event.get():
        if ereignis.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
    if veraenderung:
        veraenderung = False
        neu_nass = []
        for x, y in nass:
            nachbarn = [[x-1, y], [x+1, y], [x, y-1], [x, y+1]]
            for a, b in nachbarn:
                if zustand[a][b] == 'luft':
                    zustand[a][b] = 'wasser'
                    neu_nass.append([a, b])
        print(len(neu_nass))
        if len(neu_nass) > 0:
            veraenderung = True

        neu_gezeichnete_quadrate = []
        for x, y in neu_nass:
            quadrat = zeichne_quadrat(x, y, farbe[zustand[x][y]])
            neu_gezeichnete_quadrate.append(quadrat)
        pygame.display.update(neu_gezeichnete_quadrate)

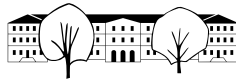
        nass = neu_nass
        if veraenderung == False:
            print('Endzustand erreicht.')
    else:
        # Ohne das Folgende kommt bisweilen "Window not responding."
        # https://stackoverflow.com/questions/20165492/pygame-window-not-responding-af
        pygame.event.pump()

```

✂ Lösung zu [A18](#) ex-game-of-life

✂ Lösung zu [A19](#) ex-fallendes-gestein

✂ Lösung zu [A20](#) ex-koralle

✂ Lösung zu A21 ex-waermeleitungsgleichung✂ Lösung zu A22 ex-snake

Mal eine erste Version. Weiss nicht, ob es so viele Äpfel geben sollte. Sie könnten auch wieder verschwinden nach einer gewissen Zeit...

```
import pygame
import sys
from random import *

BREITE = 60
HOEHE = 30

# EINHEIT gibt die Seitenlänge eines Quadrats in Pixel an.
EINHEIT = 25

# Dictionary: Speichert die Farben für die Zustände eines Feldes:
farbe = {
    'leer' : [0, 0, 0],
    'raster' : [150, 155, 165],
    'rand' : [0, 0, 255],
    'schlange' : [255, 0, 0],
    'apfel' : [0, 255, 0],
    'schrift' : [255, 255, 255],
}

def zeichne_quadrat_ohne_update(x, y, zeichenfarbe):
    rechteck = pygame.Rect(x*EINHEIT, HOEHE*EINHEIT-y*EINHEIT-EINHEIT, EINHEIT, EINHEIT)
    pygame.draw.rect(surface=fenster, color=farbe['raster'], rect=rechteck)
    rechteck = pygame.Rect(x*EINHEIT, HOEHE*EINHEIT-y*EINHEIT-EINHEIT, EINHEIT-1, EINHEIT)
    pygame.draw.rect(surface=fenster, color=zeichenfarbe, rect=rechteck)

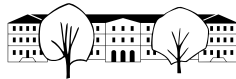
def zeichne_quadrat_mit_update(x, y, zeichenfarbe):
    rechteck = pygame.Rect(x*EINHEIT, HOEHE*EINHEIT-y*EINHEIT-EINHEIT, EINHEIT, EINHEIT)
    q1 = pygame.draw.rect(surface=fenster, color=farbe['raster'], rect=rechteck)
    rechteck = pygame.Rect(x*EINHEIT, HOEHE*EINHEIT-y*EINHEIT-EINHEIT, EINHEIT-1, EINHEIT)
    q2 = pygame.draw.rect(surface=fenster, color=zeichenfarbe, rect=rechteck)
    pygame.display.update([q1, q2])

def schreibe(x, y, text, groesse=1):
    # schrift = pygame.font.Font('freesansbold.ttf', round(groesse * faktor))

    # Mit pygame.font.SysFont kann man wohl jede der Fonts aus der Liste
    # pygame.font.get_fonts()
    # verwenden.

    schrift = pygame.font.SysFont('freemono', round(groesse * EINHEIT))
    formatierter_text = schrift.render(text, True, farbe['schrift'], farbe['leer'])
    # formatierter_text.set_alpha(127)
    rechteck = formatierter_text.get_rect()
    rechteck.center = [x*EINHEIT, HOEHE*EINHEIT-y*EINHEIT-EINHEIT]
    # rechteck.bottomleft =
    fenster.blit(formatierter_text, rechteck)
    pygame.display.update()

def feld(x, y):
```



```

    if x == 0 or x == BREITE-1 or y == 0 or y == HOEHE-1:
        return 'rand'
    if x==apfel[0] and y == apfel[1]:
        return 'apfel'
    for a, b in schlange:
        if x==a and y == b:
            return 'schlange'
    return 'leer'

def neue_schlange():
    while True:
        x = randrange(BREITE//3, 2*BREITE//3)
        y = randrange(HOEHE//3, 2*HOEHE//3)
        if feld(x, y) == 'leer':
            schlange = [[x, y] for _ in range(laenge)]
            break
    richtung = [[1, 0], [-1, 0], [0, 1], [0, -1]][randrange(4)]
    return schlange, richtung

def neuer_apfel():
    while True:
        x = randrange(1, BREITE-1)
        y = randrange(1, HOEHE-1)
        if feld(x, y) == 'leer':
            return [x, y]

pygame.init()
pygame.display.set_caption("Snake")
uhr = pygame.time.Clock()
fenster = pygame.display.set_mode((BREITE*EINHEIT, HOEHE*EINHEIT+3*EINHEIT))

print("""
Snake
Steuerung mit Pfeiltasten.
Für jeden gefressenen Apfel gibt es einen Punkt, die Schlange verlängert sich um ein F
Du hast 5 Leben.
Fenster schliessen per Maus oder per Taste 'q'
""")

# Initialisierung
for x in range(BREITE):
    for y in range(HOEHE):
        zeichne_quadrat_ohne_update(x, y, farbe['leer'])

for x in range(BREITE):
    zeichne_quadrat_ohne_update(x, 0, farbe['rand'])
    zeichne_quadrat_ohne_update(x, HOEHE-1, farbe['rand'])

for y in range(HOEHE):
    zeichne_quadrat_ohne_update(0, y, farbe['rand'])
    zeichne_quadrat_ohne_update(BREITE-1, y, farbe['rand'])

# Dummy-Definitionen
schlange = []
apfel = [0, 0]

laenge = 5
schlange, richtung = neue_schlange()

```



```
for x, y in schlange:
    zeichne_quadrat_ohne_update(x, y, farbe['schlange'])

apfel = neuer_apfel()
zeichne_quadrat_ohne_update(apfel[0], apfel[1], farbe['apfel'])

pygame.display.flip()

punkte = 0
leben = 5

frames_pro_sekunde = 25
spielende = False # wird True, sobald der Benutzer das Fenster schliesst oder die Tast

while True:
    for ereignis in pygame.event.get():
        if ereignis.type == pygame.QUIT:
            print('Fenster schliessen.')
            spielende = True
        elif ereignis.type == pygame.KEYDOWN:
            if ereignis.key == pygame.K_q:
                print('Taste "q" gedrückt.')
                spielende = True
            elif ereignis.key == pygame.K_UP:
                print('Pfeiltaste UP gedrückt')
                richtung = [0, 1]
            elif ereignis.key == pygame.K_DOWN:
                print('Pfeiltaste DOWN gedrückt')
                richtung = [0, -1]
            if ereignis.key == pygame.K_RIGHT:
                print('Pfeiltaste RIGHT gedrückt')
                richtung = [1, 0]
            if ereignis.key == pygame.K_LEFT:
                print('Pfeiltaste LEFT gedrückt')
                richtung = [-1, 0]

    x, y = schlange[0]
    xneu, yneu = x + richtung[0], y + richtung[1]
    felddtyp = feld(xneu, yneu)
    if felddtyp in {'rand', 'schlange'}:
        print(f"Spielende, Crash mit {felddtyp}")
        if leben == 0:
            spielende = True
        else:
            leben -= 1
            print(f'{punkte=}, {leben=}')
            for x, y in schlange:
                zeichne_quadrat_mit_update(x, y, farbe['leer'])
            schlange, richtung = neue_schlange()
    elif felddtyp in {'leer', 'apfel'}:
        if felddtyp == 'leer':
            zeichne_quadrat_mit_update(schlange[-1][0], schlange[-1][1], farbe['leer'])
            schlange.pop()
        elif felddtyp == 'apfel':
            apfel = neuer_apfel()
            zeichne_quadrat_mit_update(apfel[0], apfel[1], farbe['apfel'])
            punkte += 1
            laenge += 1
```



```
        frames_pro_sekunde += 1
        # print(f'{punkte=}, {leben=}, {laenge=}, {frames_pro_sekunde}')
        zeichne_quadrat_mit_update(xneu, yneu, farbe['schlange'])
        schlange.insert(0, [xneu, yneu])
        schreibe(BREITE//2, -2.5, f'{punkte=}, {leben=}, {laenge=}, {frames_pro_sekunde=}')
    if spielende:
        pygame.time.wait(1000)
        pygame.quit()
        sys.exit()

uhr.tick(frames_pro_sekunde)
```

✂ Lösung zu [A23](#) ex-nibbles

✂ Lösung zu [A24](#) ex-tetris

✂ Lösung zu [A25](#) ex-2048

✂ Lösung zu [A26](#) ex-sortierverfahren-selbst-finden

✂ Lösung zu [A27](#) ex-fibonacci-variante

✂ Lösung zu [A28](#) ex-fakultaet

✂ Lösung zu [A29](#) ex-summe-liste

✂ Lösung zu [A30](#) ex-string-umkehren

✂ Lösung zu [A31](#) ex-permutationen

✂ Lösung zu [A32](#) ex-mergesort

✂ Lösung zu [A33](#) ex-quicksort

✂ Lösung zu [A34](#) ex-turtle-baum

✂ Lösung zu [A35](#) ex-turtle-koch-kurve-quadratisch

✂ Lösung zu [A36](#) ex-turtle-koch-kurve



✂ Lösung zu **A37** ex-turtle-sierpinski-kurve

✂ Lösung zu **A38** ex-turtle-hilbert-kurve

✂ Lösung zu **A39** ex-turtle-pythagoras-baum