1 Freifach Programmieren

1.1 while-Schleifen

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 für jedes Jahr den Kontostand am 31. Dezember ausgibt.

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 `links` $\leq \sqrt{x} \leq$ `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 `mittel` $= \frac{\texttt{links}+\texttt{rechts}}{2}$ 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 `links` $\leq \sqrt{x} \leq$ `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.
    #
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}.')
```

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

Aufgabe A3 Die Zahlenfolge

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

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, dass 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 (Kreiszahl π durch ein Zufallsexperiment näherungsweise bestimmen)

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

```
from random import random
n = 6
while n>0:
    z = random()
    print(z)
    n = n-1
```

```
0.5902146854591198
0.4716598536566161
0.13425499931386298
0.3035070706273332
0.7218711426031923
0.5757062110677602
```

In dieser Aufgabe sollen Sie die Kreiszahl π mit einem Zufallsexperiment annähern. 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 im Einheitsquadrat. Stellen Sie fest, ob dieser Punkt auch im Einheitskreis 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 Kreis liegen.
- Wie gross müsste dieser Anteil theoretisch sein? Wie lässt sich somit π näherungsweise berechen? 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)
```

1.2 Verzweigungen = if-Statements

Aufgabe A5 (Lösungen einer quadratischen Gleichung in allen Fällen angeben)

Vorbemerkung/Erinnerung: Der «Entscheidungsbaum» (bitte nachfragen, dann male ich ihn als Baum an die Tafel) für eine quadratische Gleichung $ax^2 + bx + c = 0$ sieht wie folgt aus:

(a) $a \neq 0$
  (i) $D > 0$: zwei Lösungen (Mitternachtsformel $x_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$)
  (ii) $D = 0$: genau eine Lösung («Anfang» von Mitternachtsformel $x = \frac{-b}{2a}$)
  (iii) $D < 0$: keine Lösung

(b) $a = 0$
  (i) $b \neq 0$: genau eine Lösung $x = -\frac{c}{b}$
  (ii) $b = 0$
    i. $c \neq 0$: keine Lösung
    ii. $c = 0$: Lösungsmenge $\mathbb{R}$

Ergänzen Sie das folgende Programm so, dass es alle Lösungen der quadratischen Gleichung $ax^2 + bx + c = 0$ ausgibt.

(a) Nehmen Sie zunächst an, dass $a \neq 0$ gilt. Das Programm soll dann einerseits ausgeben, ob die Gleichung keine, genau eine oder genau zwei Lösungen hat (dazu ist die Diskriminante $D = b^2 - 4ac$ verwenden); andererseits soll es alle Lösungen ausgeben.
Testen Sie Ihr Programm mit mindestens drei quadratischen Gleichungen, die alle Fälle abdecken, etwa $5(x-2)(x+3) = 0$ (genau zwei Lösungen), $3(x+7)^2 = 0$ (genau eine Lösung), $(x-2)^2 = -3$ (keine Lösung).

(b) Schreiben Sie das Programm nun so um, dass es auch im Fall $a = 0$ funktioniert. In diesem Fall ist also die Lösung der linearen Gleichung $0x^2 + bx + c = bx + c = 0$ zu untersuchen.
Testen Sie Ihr Programm! Funktioniert es auch korrekt für die «neuen» Fälle, also etwa für die Gleichungen $3x - 2 = 0$ (genau eine Lösung), $0x - 2 = 0$ (keine Lösung), $0x + 0 = 0$ (unendliche viele Lösungen, Lösungsmenge $\mathbb{R}$)?

```
print("Ich kann jede quadratische Gleichung ax^2+bx+c=0 lösen.")
a = int(input("Gib die Zahl a ein: "))
b = int(input("Gib die Zahl b ein: "))
c = int(input("Gib die Zahl c ein: "))
D = b**2 - 4*a*c                          # Diskriminante D
print(f'Die Lösungen der Gleichung {a}x^2+{b}x+{c}=0 sind ...')
# Hier ist Code
# zu ergänzen.
```

1.3 Strings = Zeichenketten, Listen, Dictionaries

Tafelerklärungen + Python:
Strings, Addition, Multiplikation mit einer Zahl;
Zugriff auf einzelne Zeichen;
Loop (= Schleife) über Zeichen eines Strings;
Funktionen `ord()` und `chr()` (ASCII-Code); per Schleife diverse Zeichen ausgeben; `ord('A')=65`, `ord('z')=122`;
Zugriff auf gewisse «Unterstrings»: Slicing (start, end, step), auch rückwärts
https://www.w3schools.com/python/python_strings.asp

Aufgabe A6 Slalomtext: Schreibe ein Programm mit der folgenden Ausgabe. Die Anzahl der ausgegebenen Zeilen soll von einer Variablen abhängen.

```
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
  Schleifen sind cool!
 Schleifen sind cool!
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
```

Wenn die Ausgabe verzögert werden soll: Verwende die Bibliothek `time`. Der Befehl `time.sleep(0.05)` verzögert die Programmausführung um 0.05 s.

```
import time
time.sleep(0.05)
```

Aufgabe A7 Definiere wie folgt einen String (sollte per 'copy and paste' kopierbar sein).

(Wenn man den String mit drei Anführungzeichen beginnt und beendet, wird er samt der Zeilenumbrüche gespeichert. Hier haben wir einfache Anführungszeichen aussen verwendet, damit wir im inneren doppelte Anführungszeichen verwenden können.)

```
text = '''Drei Chinesen mit dem Kontrabass,
die sassen auf der Strasse und erzählten sich was.
Da kam die Polizei: "Ja, was ist denn das?"
Drei Chinesen mit dem Kontrabass.'''
```

Erzeuge daraus jeweils einen neuen String auf die angegebene Weise und gibt ihn aus.

Die ersten drei Teilaufgaben sind «von Hand» zu lösen, also ohne Stringmethoden (= Funktionen der Datenstruktur String) oder Slicing; diese dürfen dann in der letzten Teilaufgabe verwendet werden.

(a) Ersetze alle Vokale durch einen fest gewählten Vokal, etwa durch 'o'.

(b) Verwandle alle Grossbuchstaben in Kleinbuchstaben; alle anderen Zeichen sind beizubehalten.
Hinweis: Verwende die Funktionen `ord()` und `chr()`.
- `ord(c)` weist einem Character/Zeichen `c` (= einzelnes Zeichen, etwa ein Buchstabe oder ein Satzzeichen) eine Zahl zu, den sogenannten ASCII-Code des Zeichens.
- `chr(x)` weist umgekehrt einer Zahl `x` das zugehörige Zeichen zu.

(c) Gib den Text rückwärts aus.

(d) Löse nun alle vorherigen Teilaufgaben mit Hilfe der folgenden String-Methoden bzw. Slicing:
- `replace`, siehe etwa https://www.w3schools.com/python/python_strings_methods.asp
- `lower`, dieselbe Referenz
- Slicing strings; hier wichtig: der dritte Parameter (Schrittweite) darf auch negativ sein: https://www.w3schools.com/python/python_strings_slicing.asp

Aufgabe A8 (Caesar-Verschlüsselung)

Die sogenannte Caesar-Verschlüsselung eines Textes funktioniert wie folgt:

- Wähle eine natürliche Zahl n als Verschiebungsparameter (meist liegt n zwischen 0 und 25).

- Dann wird jeder Buchstabe des Alphabets durch den Buchstaben ersetzt, der n Positionen weiter hinten im Alphabet steht. Wenn man bei 'z' ankommt, fängt man wieder vorne bei 'a' an.
  (So etwas nennt man eine zyklische Vertauschung: Wenn man sich das Alphabet kreisförmig (= zyklisch) aufgeschrieben vorstellt, geht man jeweils n Positionen weiter.)

Beispiel: Das Wort `'hallo'` wird per Verschiebung um $n = 3$ zu `'kdoor'`.

Der folgende Text wurde mit der Caesar-Verschlüsselung verschlüsselt (der Einfachheit halber besteht er nur aus Kleinbuchstaben und enthält keine Umlaute oder Sonderzeichen). Entschlüssele ihn.

Hinweis: Der Buchstabe 'e' kam im Originaltext am häufigsten vor. (In deutschen Texten ist 'e' mit etwa 17 % der Buchstaben der häufigste Buchstabe.) Welcher Buchstabe kommt im codierten Text am häufigsten vor?

```
jviyhi wglsiriv ksixxivjyroir
xsglxiv eyw ipmwmyq
amv fixvixir jiyivxvyroir
lmqqpmwgli himr limpmkxlyq
himri deyfiv fmrhir amihiv
aew hiv qshi wglaivh kixlimpx
fixxpiv aivhir jyivwxirfvyihiv
as himr werjxiv jpyikip aimpx
wimh yqwglpyrkir qmpppmsrir
hmiwir oyww hiv kerdir aipx
fvyihiv yifivq wxivrirdipx
qyww imr pmifiv zexiv aslrir
```

⚒ Aufgabe A9 Häufigkeitsanalyse

Lade einen längeren Text als `txt`-Datei aus dem Internet, etwa den «Faust 1» von Johann Wolfgang von Goethe (https://www.gutenberg.org/files/2229/2229-0.txt), und speichere ihn in demselben Verzeichnis, in dem du das Python-Programm für diese Aufgabe schreiben wirst.

Wie man eine Text-Datei (= Datei mit Dateinamenserweiterung `.txt` in Python öffnet und den Inhalt in einem String speichert, geht aus dem folgenden Beispiel hervor. (Um das Schliessen der Datei muss man sich mit der `with`-Konstruktion nicht kümmern.)

```
with open("faust.txt", 'r') as datei:
    text = datei.read()
```

(a) 'dictionaries' kennenlernen: https://www.w3schools.com/python/python_dictionaries.asp.

(b) Speichere in einem Dictionary für jeden Buchstaben, wie oft er vorkommt. (Unterscheide nicht zwischen Klein- und Grossschreibung; ignoriere alle Sonderzeichen und Leerzeichen.)
Gib für jeden Buchstaben an, wie häufig er vorkommt. Beispiel: Der Buchstabe 'e' kommt mit Häufigkeit 0.17 vor, wenn 17% aller Buchstaben 'e's sind.
Stelle das Resultat graphisch dar. Dabei hilft dir das folgende Beispielprogramm. (Vermutlich musst du die beiden Bibliotheken matplotlib und numpy installieren.)

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["a", "b", "c", "d"])
y = np.array([0.2, 0.7, 0.1, 1])

plt.bar(x,y)
plt.show()
```

Bemerkung: Ideen für andere Häufigkeitsanalysen und Diagramme findet man beispielsweise auf Wikipedia: Buchstabenhäufigkeit (z. B.: welcher Buchstabe kommt am häufigsten am Wortanfang vor?).

(c) Zerlege deinen Text in eine Liste von (kleingeschriebenen) Wörtern.
Hinweis: Es ist vermutlich geschickt, zuerst alle Sonderzeichen durch Leerzeichen zu ersetzen.

- Welche Palindrome (= Wörter, die von vorne wie von hinten gelesen dasselbe Wort ergeben) kommen in deinem Text vor?
- Wie oft kommt das Wort 'und' vor?
- Welches Wort kommt am häufigsten vor?
- Welche zehn Wörter kommen am häufigsten vor?
- Welche Wortlänge kommt wie häufig vor? Stelle das Resultat ähnlich wie oben graphisch dar.

Ausbau von Aufgabe A7

- Allgemein empfohlene Seite, um Python zu lernen: https://www.w3schools.com/python
- Slicing: http://www.w3schools.com/python/python_strings_slicing.asp
  Schreibe eine Funktion `unterstring(s, start, ende)`, die als Argumente einen String `s` und natürliche Zahlen `start` und `ende` entgegennimmt und als Rückgabewert den Unterstring von `s` liefert, der an der Position `start` beginnt und ein Zeichen vor der Position `ende` endet. Dabei ist es verboten, slicing zu verwenden.
  Lösung:

```
# Variante A: Mit einer while-Schleife
def unterstringA(s, start, ende):
    t = ''
    i = start
    while i < ende:
        t = t+s[i]
        i = i + 1
    return t

# Variante B: Mit einer for-Schleife
def unterstringB(s, start, ende):
    t = ''
    for i in range(start, ende):
        t = t+s[i]
    return t

# Variante C: Mit Angabe der Datentypen der Parameter
def unterstringC(s: str, start: int, ende: int) -> str:
    t = ''
    for i in range(start, ende):
        t = t+s[i]
    return t

# Variante D: Mit optionalen Parametern
def unterstringD(s, start, ende, schrittweite = 1):
    t = ''
    i = start
    while i < ende:
        t = t+s[i]
        i = i + schrittweite
    return t

text = 'Drei Chinesen mit dem Kontrabass'

# Slicing
print(text[5:21])
# Slicing von Hand
print(unterstringA(text, 5, 21))
print(unterstringB(text, 5, 21))
print(unterstringC(text, 5, 21))

# Slicing mit Schrittweite
print(text[5:21:2])
# Slicing von Hand mit Schrittweite
print(unterstringD(text, 5, 21, 2))
# ohne Angabe der Schrittweite
print(unterstringD(text, 5, 21))
```

  Zu den Begriffen «Parameter» bzw. «Argument»:
  - In der Definition einer Funktion heissen die auftauchenden Variablen **Parameter**. In der Funktionsdefinition

    ```
    def unterstringA(s, start, ende):
    ```

    im obigen Beispiel sind also `s`, `start` und `ende` Parameter.
  - Wenn man die Funktion aufruft, nennt man die an die Funktion übergebenen Werte **Argumente**. Im Funktionsaufruf

    ```
    unterstringA(text, 5, 21)
    ```

    im obigen Beispiel sind die Argumente (der Wert der Variablen) `text` und die beiden Zahlen `5` und `10`.

  In der Praxis werden diese beiden Begriffe (leider) oft nicht genau unterschieden.
- String methods: http://www.w3schools.com/python/python_strings_methods.asp
  - Counting: http://www.w3schools.com/python/ref_string_count.asp

Schreibe eine Funktion `anzahl(s, t)`, die als Argumente zwei Strings `s` und `t` entgegennimmt und berechnet, wie oft der String `t` im String `s` auftaucht. Dabei darf die String-Methode `count` nicht verwendet werden. Slicing darf verwendet werden.

Lösung:

```
def zaehle(s, t):
    anzahl = 0
    laenge = len(t)

    # Warum ist das "+1" in der folgenden Zeile nötig?
    for i in range(len(s)-laenge+1):
        if s[i:i+laenge] == t:
            anzahl = anzahl+1
    return anzahl

text = 'Fischers Fritz fischt frische Fische, frische Fische fischt Fischers Fritz'

# Python count-Methode
print(text.count('F'))
# von Hand
print(zaehle(text, 'F'))

print(text.count('Fritz'))
print(zaehle(text, 'Fritz'))

print(text.count('isch'))
print(zaehle(text, 'isch'))

print(text.count(''))
print(zaehle(text, ''))
```

- Für weitere String-Methoden siehe http://www.w3schools.com/python/python_strings_methods.asp Programmiere einige der folgenden Methoden selbst als Funktion (mit einem geeigneten deutschen Namen).

  a) `find` (ist fast dasselbe wie `index`)
  b) `replace`
  c) `swapcase`
  d) `center`

Für Fortgeschrittene: Nonsense-Texter

Aufgabe A10 (Nonsense-Texter oder "next letter prediction" und "next word prediction"; schwierig)

Aufgabe motiviert durch «Kapitel 2: Texte bauen mit Markow» im empfehlenswerten Buch https://www.maschinennah.de/ki-buch/ (elektronisch in der Kanti-Bibliothek). Dort gibt es diverse Ausprobierseiten.

(a) Lies einen längeren Text ein, etwa Goethes Faust, und speichere ihn als String (aus Kleinbuchstaben; eventuell Satzzeichen durch Leerzeichen ersetzen).
Wähle eine «Abschnittslänge» n (zum Beispiel $n = 4$). Speichere für jeden Unterstring der Länge n des Textes, welche Zeichen danach wie oft vorkommen (dafür eignet sich ein dictionary of dictionaries).
Beispiel: Nach dem Unterstring 'herz' kommen im Faust 1 die folgenden Zeichen so oft vor, wie die Zahl nach dem jeweiligen Zeichen angibt:
{' ': 24, 'e': 32, ',': 6, 't': 2, 'l': 7, 'new line': 2, 'i': 1, 'u': 1}
Das 'l' könnte zum Beispiel von 'herzlich' kommen, das 'e' von 'Herzen'.
Nutze diese Information, um einen „Nonsense-Text" zu erzeugen. Wähle einen String der Länge n, der im Text vorkommt (etwa die ersten n Zeichen).
Ergänze diesen String zum Beispiel 1000 Mal am Ende um ein Zeichen, das (im Ausgangstext) auf die letzten n Zeichen folgt. Dieses neue Zeichen kann zufällig aus den erlaubten Kandidaten ausgewählt werden; sinnvoll ist es auch, dieses Zeichen gemäss der Auftretenshäufigkeiten zufällig auszuwählen. Im obigen Beispiel bedeutet dies, dass man nach der Zeichenfolge 'herz' in etwa 32 von 75 Fällen ein 'e' als nächstes Zeichen wählt.

(b) Dasselbe Verfahren zum Nonsense-Texten kann man auch mit Wortfolgen der Länge n statt mit Zeichenfolgen der Länge n durchführen. Schreibe ein entsprechendes Programm.

Bemerkung: Dies ist ein recht einfaches Modell zum Erzeugen sinnvoll klingender Unsinnstexte. Diese Art der "next word prediction" wird oft verwendet, um die Funktionsweise von Sprachmodellen wie ChatGPT zu veranschaulichen. Solche Sprachmodelle lernen mit Hilfe neuronaler Netze Muster aus sehr vielen Texten und produzieren auf Grund dieser Muster Antworten (ohne wahres «Verständnis»; wobei man sich natürlich fragen mag, wo unser menschliches «Verständnis» naturwissenschaftlich beheimatet ist und ob wir Menschen nicht auch nur «ohne wahres Verständnis nachplappern»).

Anregungen für Fortgeschrittene:

- Sieb des Eratosthenes (siehe Wikipedia) programmieren (verwende eine Liste von Booleschen Werten).
- Turtle-Graphik: Objektorientiert nachprogrammieren (Empfehlung: `pygame` für Grafik verwenden).
- Wie kann man einen Irrgarten erzeugen? Wie kann man den Weg durch einen Irrgarten finden?
- Game of Life

1.4 Basics einüben

Schleifen (`for` und `while`)

⚒ Aufgabe A11 Schleifen: `for`- und `while`-Schleifen

Schreibe ein Programm, dass genau die folgende Ausgabe erzeugt (Hinweise unten):

```
Mit for-Schleife und range(end): Natürliche Zahlen von 0 bis 20:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
Mit while-Schleife: Natürliche Zahlen von 0 bis 20:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
Mit for-Schleife und range(start, end): Ganze Zahlen von -10 bis 10:
-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
Mit while-Schleife: Ganze Zahlen von -10 bis 10:
-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
Mit for-Schleife und range(start, end, step): Jede dritte natürliche Zahl von 11 bis 24:
11, 14, 17, 20, 23,
Mit while-Schleife: Jede dritte natürliche Zahl von 11 bis 24:
11, 14, 17, 20, 23,
Mit while-Schleife: Reelle Zahlen von -1 bis 1 in Schritten von 0.1:
0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0,
```

Hinweise:

- Lerne aus https://www.w3schools.com/python/python_for_loops.asp, was eine `for`-Schleife über eien `range` (Bereich, Intervall ganzer Zahlen) ist.
- Lerne aus https://www.w3schools.com/python/python_while_loops.asp, was eine `while`-Schleife ist.
- Normalerweise erzeugt der `print`-Befehl eine Ausgabe und der Cursor springt danach in die nächste Zeile. Man kann das Springen in die nächste Zeile mit Hilfe eines optionalen Arguments unterbinden: Der Befehl `print(i, end='x ')` sorgt beispielsweise dafür, dass nach der Ausgabe (des Werts) der Variablen `i` die Zeichenkette ' x' ausgegeben wird und der Cursor am Ende der Ausgabe stehen bleibt, also nicht in die nächste Zeile wechselt.
- Der f-String (= formatted string) `f'{x:.1f}'` besteht aus der Zahl `x` mit genau einer Nachkommastelle. (Das `f` am Ende des f-Strings steht für «floating point number» = «float» = «Gleitkommazahl», was grob gesagt eine Kommazahl meint.)

⚒ Aufgabe A12 Schreibe ein Programm, dass vom Benutzer eine natürliche Zahl n erfragt und danach mit Hilfe einer Schleife die folgenden Zahlen berechnet und ausgibt:

- die Summe aller Zahlen von 0 bis n;
- die Summe aller Quadratzahlen von 0 bis n^2;
- die Summe aller dritten Potenzen von 0 bis n^3;

Zusätzlich sollen die folgenden drei Zahlen ausgegeben werden:

$$\frac{n(n+1)}{2} \qquad \frac{n(n+1)(2n+1)}{6} \qquad \left(\frac{n(n+1)}{2}\right)^2$$

Der Dialog bei Eingabe von $n = 9$ soll genau wie folgt aussehen:

```
Gib eine Zahl ein: 9
Die Summe aller Zahlen von 0 bis 9 beträgt 45.
Die Summe aller Quadratzahlen von 0 bis 9**2=81 beträgt 285.
Die Summe aller Kubikzahlen von 0 bis 9**3=729 beträgt 2025.

Drei Zahlen: 45.0, 285.0, 2025.0.
```

Aufgabe A13 Kleines Einmaleins:

Schreibe ein Programm, das die folgende Multiplikationstabelle ausgibt.

Hinweise:

- Verwende zwei ineinander verschachtelte Schleifen.
- Der f-String (= formatted string) `f'{i:4}` erzeugt einen String der Länge 4, in dem die ganze Zahl `i` rechtsbündig steht (falls genug Platz 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 |
```

Verzweigung (`if`)

Aufgabe A14 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.
```

Hinweis: Auf https://www.w3schools.com/python/python_conditions.asp kannst du lernen, wie if-(elif-else)-statements/Fallunterscheidungen in Python programmiert werden.

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.

Funktionen

Aufgabe A15 Lerne, was Funktionen in Python sind: https://www.w3schools.com/python/python_functions.asp

Definiere in einem Python-Programm die folgenden Funktionen und teste sie durch Funktionsaufrufe:

- `begruesse(name)`: Diese Funktion nimmt einen String `name` entgegen (= den Namen einer Person) und begrüsst diese Person. Beispielsweise soll

  ```
  begruesse('Pinocchio')
  ```

  die folgende Ausgabe liefern.

  ```
  Hallo Pinocchio!
  ```

- `wurzel(n)`: Diese Funktion die (Quadrat-)Wurzel $\sqrt{n}$ als Funktionswert zurück.
- `betrag(x)`: Diese Funktion liefert den Betrag $|x|$ von x als Funktionswert zurück.
- `ist_durch_drei_teilbar(x)`: Diese Funktion liefert `True` zurück, wenn die natürliche Zahl x durch 3 teilbar ist, sonst `False`.
- `arithmetisches_mittel(a,b)`: Diese Funktion liefert das arithmetische Mittel (= den Durchschnitt) $\frac{a+b}{2}$ zweier Zahlen zurück.
- `minium(a,b)`: Diese Funktion liefert das Minimum (= die kleinere) der beiden Zahlen a und b zurück.

⚒ Aufgabe A16 Schreibe eine Funktion `prim(n)`, die eine positive natürliche Zahl n als Argument entgegennimmt, alle Teiler von n ausgibt, ausserdem die Anzahl aller Teiler ausgibt und mitteilt, ob n eine Primzahl ist (= genau zwei Teiler hat, nämlich 1 und n). Der Rückgabewert der Funktion soll `True` sein, wenn n prim ist, sonst `False`.

Beispielsweise soll

```
print(teiler(12))
```

die folgende Ausgabe liefern.

```
Teiler von 12: 1, 2, 3, 4, 6, 12,
Anzahl der Teiler: 6
12 ist keine Primzahl
True
```

Strings = Zeichenketten

⚒ Aufgabe A17 Definiere am Anfang eines Python-Programms einen String (etwa `s='Drei Chinesen mit dem Kontrabass ...'`).

- Gib die Länge des Strings aus.
- Gib den String Zeichen für Zeichen aus (jedes Zeichen in einer neuen Zeile).
- Gib alle Unterstrings der Länge 5 aus (in derselben Reihenfolge, in der sie in `s` erscheinen; jeder Unterstring in einer neuen Zeile).
  Hinweis: Verwende *slicing*.
- Erzeuge einen neuen String `ersetzt`, der aus `s` entsteht, indem man alle Vokale durch 'o' ersetzt, und gibt `ersetzt` aus.
- Erzeuge einen neuen String `rueckwaerts`, der `s` rückwärts gelesen ist, und gib ihn aus.

Listen

⚒ Aufgabe A18 Lerne die Basics zu Listen: https://www.w3schools.com/python/python_lists.asp

Schreibe ein neues Python-Programm, das mit der Definition einer Liste von Zahlen (etwa Noten) startet, etwa

```
zahlenliste = [4, 6, 3, 5, 4, 2, 5, 6, 5, 5, 4, 4]
```

Das Programm soll ausgeben:

- die Anzahl der Elemente der Liste;
- die Summe aller Zahlen der Liste;
- den Durchschnitt aller Zahlen der Liste;
- die kleinste und die grösste Zahl der Liste.

⚒ Aufgabe A19 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` jeweils 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"]
```

Aufgabe A20 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 A21 Sieb des Eratosthenes

Das **Sieb des Eratosthenes** ist der folgende (sehr effiziente) Algorithmus zum Erstellen einer Liste aller Primzahlen bis zu einer gegebenen positiven natürlichen Zahl n.

```
Schreibe alle Zahlen von 0 bis n auf und streiche 0 und 1 durch (Liste der Primzahlkandidaten).
Initialisiere eine leere Liste (sie enthält die bis jetzt gefundenen Primzahlen).
Lass i in einer while-Schleife alle Zahlen von 2 bis n durchlaufen:
    Falls i nicht durchgestrichen ist:
        Nimm i in die Liste der Primzahlen auf.
        Lass j in einer while-Schleife alle Vielfachen von i zwischen 2 * i und n durchlaufen:
            Streiche j durch.
Gib die Liste der Primzahlen aus.
```

Die hier verwendete Beschreibungssprache nennt man **Pseudo-Code**. Es handelt sich um eine Art Zwischensprache zwischen verbaler Beschreibung und echtem Programm-Code.

(a) Führe dieses Verfahren für $n = 20$ auf einem Blatt Papier durch.
(b) Prüfe mit Hilfe der Animation auf https://de.wikipedia.org/wiki/Sieb_des_Eratosthenes, ob du das Verfahren richtig angewendet hast.
(c) Schreibe ein Programm, das diesen Algorithmus realisiert.

Empfehlung: Statt eine Liste aller Zahlen von 0 bis n zum Durchstreichen zu verwenden, verwende eine Liste `kandidat` der Länge $n + 1$ von Booleschen Werten (= Wahrheitswerten, also `True` oder `False`).

Idee: Der Wert von `kandidat[i]` ist

- `True`, falls die Zahl i bisher nicht durchgestrichen wurde (also noch ein Kandidat für eine Primzahl ist);
- `False`, falls die Zahl i bereits durchgestrichen wurde.

Du kannst diese Liste wie folgt erstellen (im Fall $n = 100$):

```
n = 100
kandidat = (n + 1) * [True]
# alternativ per list comprehension: kandidat = [True for i in range(0, x + 1)]
kandidat[0] = False     # Durchstreichen der 0
kandidat[1] = False     # Durchstreichen der 1
```

to be done (vielleicht schon im Python-Kurs): Kreditkartennummer als String eingeben. In Liste von Zahlen/-Ziffern umwandeln. Prüfziffer berechnen und ausgeben, ob korrekte Nummer; siehe /scientific-american

Dictionaries = Datenstruktur aus «durch Strings indizierter Werte»

Aufgabe A22 Lerne, was ein «dictionary» ist, etwa per
https://www.w3schools.com/python/python_dictionaries.asp

In Aufgabe A19 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); bei jeder Aufgabe ist sowohl die Frage als auch die Antwort gespeichert.

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.

1.5 Einführung in die Grafik- und Spielprogrammierung mit Pygame

Aufgabe A23 Auf der Dokuwiki-Seite des Freifachs findest du als Vorlage das Programm

`pygame-kennenlernen-vorlage.py`

Versuche, das Programm zu verstehen und zum Laufen zu bringen (dafür musst du vermutlich die Bibliothek Pygame installieren).

Ergänze es wie folgt:

(a) Zeichne die beiden Diagonalen im Zeichenfenster ein, mit Weiss als Zeichenfarbe.
(b) Zeichne einen blau gefüllten Viertelkreis in der linken oberen Bildschirmecke, Radius 50 Pixel.
(c) Zeichne ein gelbes Rechteck im linken unteren Viertel des Zeichenfensters.

Aufgabe A24 Chaos-Spiel (chaos game) – der springende Punkt:

Auf der Dokuwiki-Seite des Freifachs findest du als Vorlage das Programm

`chaos-game-vorlage.py`

Es zeichnet zufällig einige Punkte auf dem Bildschirm.

Schreibe darauf aufbauend ein Programm, das das sogenannte Chaos-Spiel simuliert:

(a) Zeichne im Zeichenfenster ein gleichseitiges Dreieck ABC.
(b) Wähle zufällig einen Punkt P.
(c) Lass diesen Punkt wiederholt «springen», indem du die folgenden Anweisungen sehr oft wiederholst:
- Wähle zufällig einen der Eckpunkte A, B, C (jeweils Wahrscheinlichkeit $\frac{1}{3}$).
- Der «neue» Punkt P sei der Mittelpunkt zwischen dem «alten» Punkt P und dem zufällig gewählten Eckpunkt.
- Markiere diesen neuen Punkt P.

Zusatzaufgabe (eventuell zu eigener Aufgabe machen): Verändere dein Programm (oder eines der in den Lösungen angegebenen Programme) so, dass die in
https://en.wikipedia.org/wiki/Chaos_game#Restricted_chaos_game
abgebildeten Bilder entstehen.

Bemerkung: Mehr Informationen und nette weitere Anregungen findest du beispielsweise hier: https://en.wikipedia.org/wiki/Chaos_game

Aufgabe A25 Pong solo: Computerspiel programmieren

Auf der Dokuwiki-Seite des Freifachs findest du als Vorlage das Programm

`pong-solo-vorlage.py`

Versuche, das Programm zu verstehen.

Verändere das Programm gemäss der folgenden Vorschläge:

(a) Das Paddle (das der Spieler steuert) wird mit Hilfe der Pfeiltasten nach links und rechts bewegt. Es verlässt den Bildschirm nicht. Es befindet sich genau am unteren Rand des Zeichenfensters.
(b) Der Ball wird an allen Rändern des Zeichenfensters korrekt reflektiert.
(c) Nun kannst du versuchen, Pong als Solo-Spiel zu programmieren:
- 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.

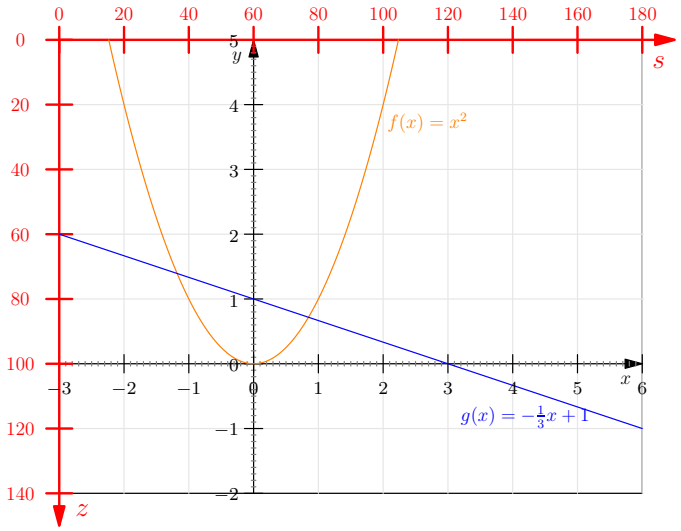
(d) Erweitere das Spiel in kreativer Weise. Einige Ideen:
- Geschwindigkeit des Balls wächst mit der Zeit;
- Winkel des Balls kann verändert werden, wenn der Ball am Rand des Paddle reflektiert wird;
- das Treffen gewisser zufällig erzeugter Objekte liefert Bonusleben oder Bonuspunkte;
- zweiter Spieler steuert anderes Paddle;

- mehrere Bälle;
- etc.

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

⚒ **Aufgabe A26** Zeichne mit Pygame ein Koordinatensystem und die Graphen einiger beliebiger, von dir gewählter Funktionen (etwa der Funktionen $f(x) = x^2$ und $g(x) = -\frac{1}{3}x + 1$ wie im dargestellten Koordinatensystem (das zugegebenermassen nicht mit Python, sonder mit Asymptote gezeichnet wurde)).

Beachte die Hinweise unter der Zeichnung.



Wie in der Zeichnung dargestellt, gibt es zwei Koordinatensysteme:

- Das Koordinatensystem des Zeichenfensters: Die horizontale s-Achse und die vertikale, nach unten zeigende z-Achse (hier steht s für Spalte und z für Zeile).
- Das schwarz gezeichnete Koordinatensystem mit üblicher x- und y-Achse.

Mathematische Vorüberlegungen (eventuell gemeinsam). Bevor man losprogrammiert, sollte man sich überlegen, wie die «Umrechnungsfunktionen» zwischen den Koordinatensystemen aussehen.

- Sei $s(x)$ die lineare Umrechnungsfunktion, die aus der x-Koordinate x eines Punktes seine s-Koordinate (= Spaltenkoordinate) berechnet. Zum Beispiel gilt $s(5) = 160$.
  Überlege dir, wie $s(x)$ aussieht für die in der Abbildung gezeigten Koordinatensysteme. Teste dein Ergebnis, indem du $s(-3)$, $s(0)$ und $s(6)$ berechnest.
- Überlege dir ebenfalls, wie die lineare Umrechnungsfunktion $z(y)$ aussieht, die aus der y-Koordinate y eines Punktes seine z-Koordinate (= Zeilenkoordinate) berechnet.

Nun geht's ans Programmieren.

«Einfache» Variante:

- Verwende ein Zeichenfenster der Breite 180 und der Höhe 140.
- Zeichne die beiden schwarzen Koordinatenachsen ein (eventuell mit Pfeilspitzen).
- Markiere alle ganzen Zahlen auf den Achsen (mit einer Schleife; ohne Beschriftung).
- Zeichne die Graphen einiger Funktionen ein (am besten eine Funktion `f(x)` und eine Funktion `g(x)` definieren).
- Wer mag, kann auch das hellgraue Raster einzeichnen und diverse Beschriftungen vornehmen (Achsen mit x und y beschriften; Markierungen an den Achsen mit Zahlen beschriften; «kleine» Markierungen an den Achsen anbringen; eventuell Graphen beschriften (wobei die mathematische Notation vermutlich nicht so einfach zu erhalten ist (KI fragen?)).
- Wer mag, kann auch Graphen von Funktionen einzeichnen, die nicht auf ganz $\mathbb{R}$ definiert sind, etwa $w(x) = \sqrt{x}$ oder $h(x) = \frac{1}{x}$. Was muss man dabei beachten?

Schwierigere Variante:

- Definiere am Anfang fünf Variablen:
  - `minx`: minimale x-Koordinate, in der Zeichnung -3;
  - `maxx`: maximale x-Koordinate, in der Zeichnung 6;
  - `miny`: minimale y-Koordinate, in der Zeichnung -2;
  - `maxy`: maximale y-Koordinate, in der Zeichnung 5;
  - `pixel_pro_einheit`: Gibt an, wie lang eine Einheit des Koordinatensystems in Pixeln ist; in der Zeichnung 20.
- Folge nun den Anweisungen in der einfachen Variante.
  Das Koordinatensystem soll so im Zeichenfenster liegen, wie von den obigen fünf Variablen spezifiziert. Wenn man diese Variablen ändert, soll sich die Zeichnung entsprechend ändern.

to be done: Zeigeruhr zeichnen drei Zeigern und Stunden- und Minutenmarkierungen, mit pygame, Zeit per Library abfragen

1.6 Sortieren von Daten

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 recurrere zurücklaufen)

1.8.3. Die Idee ist, dass man ein Problem in einfachere Teilprobleme **derselben Art** zerlegt, diese dann (rekursiv) löst und die Teilösungen 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

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

ist dadurch definiert, dass sie mit den beiden Folgengliedern 0 und 1 startet und danach jedes Folgeglied 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 mit 1 Monat erwachsen und wirft ab dem Alter von 2 Monaten (also bereits im Alter von 2 Montaten) 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))
```

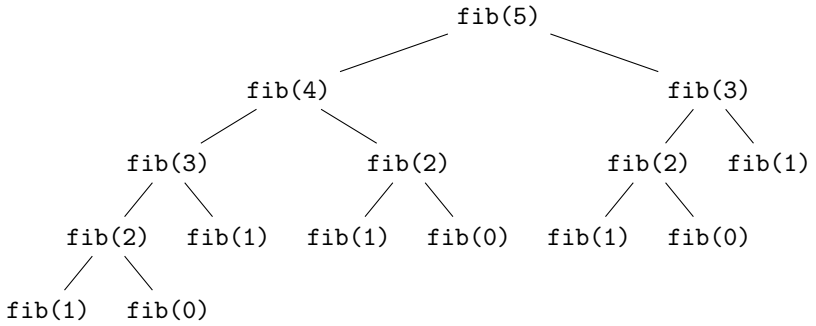
Hier ist ein iteratives Programm zur Brechnung von f_n (die Laufvariable i wird nicht verwendet; es geht nur darum, dass die Schleife $n-1$ mal ausgeführt wird):

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

print(f(0))
print(f(1))
print(f(2))
print(f(10))
print(f(38))
```

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

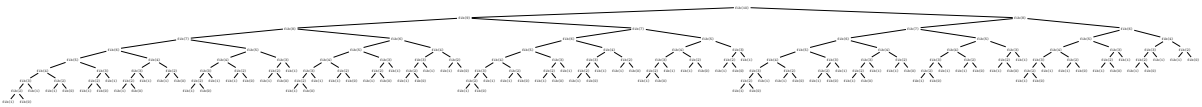
```
def fi(n):
    fib = [0, 1]
    for i in range(n-1):
        fib.append(fib[-1]+fib[-2])
    return fib

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 viele Funktionswerte mehrfach berechnet werden. Dies verdeutlicht der folgende Baum, der die Funktionsaufrufe zur Berechnung von `fib(5)` darstellt.



Das Problem ist, dass viele Funktionswerte mehrfach rekursiv berechnet werden, etwa f_3 (zweimal) und f_2 (dreimal). (Dass f_0 und f_1 mehrfach vorkommen, ist 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 Anzahlen der Aufrufe einer bestimmten Funktion, etwa von `fib(2)`, sind wohl wieder Fibonacci-Zahlen.)



1.8.6. Warum es keine gute Idee ist, die Fibonacci-Folge rekursiv zu berechnen (S. 29 im Buch von Al Sweigart).

⚒ Aufgabe A27 Die Folge

$$0, 1, 2, 4, 11, 25, 59, 142, 335, 796, 1892, 4489, \ldots$$

ist wie folgt definiert.

- Die Anfangsglieder sind 0, 1, 2.
- Jedes nachfolgende Folgenglied ist die Summe aus
  - dem vorhergehenden Folgeglied und

– dem Doppelten des vorvorhergehenden Folgeglieds und
– dem Dreifachen des vorvorvorhergehenden Folgeglieds.

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

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

Beispiel 1.8.7. Das folgende Programm berechnet die Summe der Zahlen von 1 bis n iterativ, also $1+2+3+\cdots+10$.

```
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.

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

(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 iterativ, einmal rekursiv. 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 1.8.8. 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 durch «Zerteilung der Liste in zwei Hälften» wie in dem obigen Beispiel zur rekursiven Berechnung des Maximums einer Liste von Zahlen.

Beispiel 1.8.9. 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 1.8.10 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 Implementiere den Algorithmus Mergesort 1.8.10 in Python (und teste dein Programm mit einigen zufällig erzeugten Listen).

Hinweis: Das Beispiel 1.8.8 dürfte beim Teilen der Liste in zwei Teillisten helfen.

Algorithmus 1.8.11 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;
            groesser enthält alle Elemente von a, die groesser 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 1.8.11 in Python (und teste dein Programm mit einigen zufällig erzeugten Listen).

1.9 Fraktale zeichnen (rekursiv mit der `turtle`-Bibliothek)

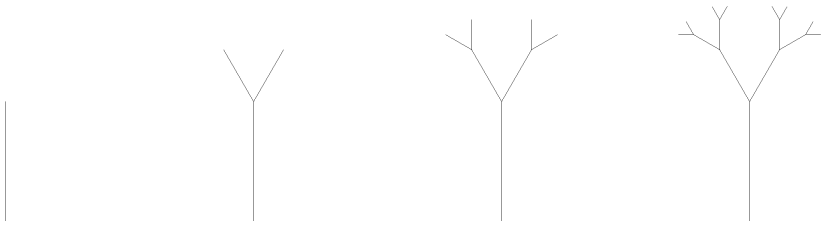
1.9.1. Beim Zeichnen mit «Turtle-Graphik» steuert man eine imaginäre Schildkröte über den Bildschirm, indem man ihr beispielsweise sagt, wie weit sie gehen soll oder um welchen Winkel sie sich drehen soll. Die Schildkröte zeichnet mit einem Stift ihren Weg auf, sofern der Zeichenstift nicht angehoben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

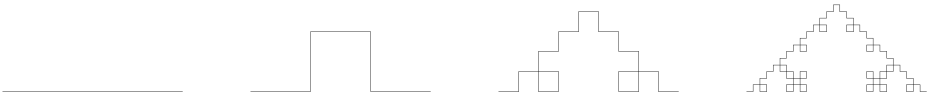


Schreibe eine Funktion `baum(tiefe, laenge)`, so dass die folgenden Funktionsaufrufe den angegebenen Effekt haben.

- `baum(0, laenge)` zeichnet das erste Bild: Die Turtle geht um `laenge` Pixel vorwärts und dann wieder genausoweit 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.

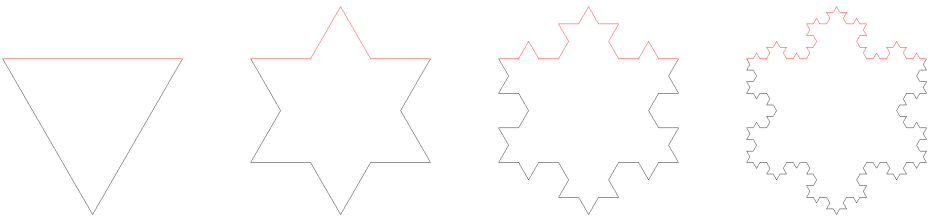


Schreibe eine Funktion `hutkurve(tiefe, laenge)`, so dass die folgenden Funktionsaufrufe den angegebenen Effekt haben.

- `baum(0, laenge)` zeichnet das erste Bild: Die Turtle geht um `laenge` Pixel vorwärts.
- `baum(1, laenge)` zeichnet das zweite Bild: Die Turtle ruft fünfmal `baum(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.)
- `baum(n, laenge)`: Die Turtle ruft fünfmal `baum(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.

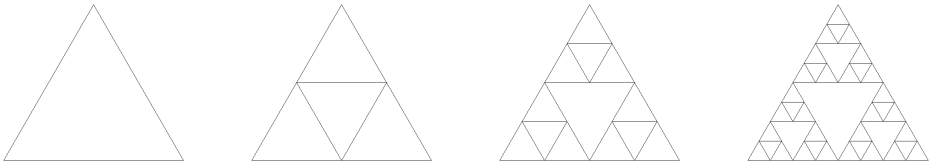


Schreibe eine rekursive Funktion `kochkurve(tiefe, laenge)`, die für `tiefe`= 0 die rote Kurve in der linken Zeichnung zeichnet, für `tiefe`= 1 die rote Kurve in der zweiten Zeichnung von links etc.

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

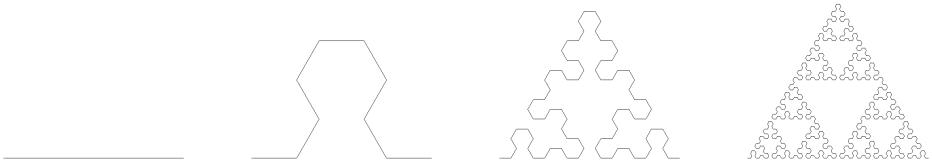
Aufgabe A37 (Sierpinski-Dreieck)

(a) Vom ersten zum zweiten Bild wird das Ausgangsdreieck durch drei Dreiecke der halben Seitenlänge ersetzt, und dies geht dann genaus 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.)

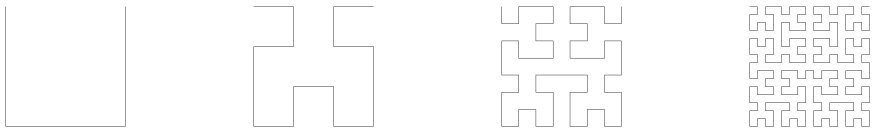


Schreibe eine rekursive Funktion `sierpinski_variante` mit geeigneten Parametern, die für geeignete Argumente die obigen Zeichnungen anfertigt.

(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 Turtle-Funktionen selbst schreiben: Zuerst naiv, dann objektorientiert

1.10.1. Idee: Selbst Turtle programmieren. Statt die Ausgabe in einem Grafikfenster anzuzeigen (etwa mit PyGame), erstellen wir eine svg-Datei, die das gezeichnete Bild enthält.
Beispiel 1.10.2. Gemeinsam programmiert: Ein- und Ausgabe in Datei:

```
dateiname = 'ausgabe.txt'

with open(dateiname, 'w') as datei:
    datei.write("Hello World!\n")
    datei.write(f"Er sagte: 'Drei hoch 10 ist {3**10}'")

with open(dateiname) as datei:
    for zeile in datei:
        print(zeile, end="")

print()
with open(dateiname) as datei:
    inhalt = datei.readlines()
    print(inhalt)

with open('ein-und-ausgabe-mit-dateien.py') as datei:
    for zeile in datei:
        print(zeile, end="")
```

1.10.3. Vorgegebenes Programmgerüst: Nur die Zeilen, die mit `fenster` beginnen, muss man verstehen.

```
from random import randrange

def svg_linie_als_string(x1, y1, x2, y2, farbe, dicke):
    return f'<line x1="{x1}" y1="{y1}" x2="{x2}" y2="{y2}" style="stroke:rgb({farbe[0]}, {farbe[1]}, {farbe[2]});stroke-width:{dicke}" />'

class zeichenfenster():
    def __init__(self, breite, hoehe):
        self.breite = breite
        self.hoehe = hoehe
        self.liste_strecken = []

    def strecke(self, x1, y1, x2, y2, farbe, dicke=1):
        self.liste_strecken.append((x1, y1, x2, y2, farbe, dicke))

    def speichere_als_svg(self, name):
        with open(name, 'w') as datei:
            datei.write(f'<svg width="{self.breite}" height="{self.hoehe}" xmlns="http://www.w3.org/2000/svg">\n')
            datei.write(f'<rect x="0" y="0" width="{self.breite}\n" height="{self.hoehe}" fill="white" stroke="blue"/>\n')
            for s in self.liste_strecken:
                datei.write(svg_linie_als_string(*s)+'\n')
            datei.write('</svg>\n')

fenster = zeichenfenster(4000, 4000)

fenster.strecke(0, 0, 4000, 4000, (0, 255, 0), 100)
fenster.strecke(0, 4000, 4000, 0, (0, 0, 255), 200)

for i in range(10):
    x1 = randrange(4000)
    y1 = randrange(4000)
    x2 = randrange(4000)
    y2 = randrange(4000)
    r = randrange(256)
    g = randrange(256)
    b = randrange(256)
    d = randrange(20)
    fenster.strecke(x1, y1, x2, y2, (r, g, b), d)

fenster.speichere_als_svg("output.svg")
```

Aufgaben:

- Verwende das obige Programm, um eine svg-Datei zu erstellen, in der das Haus des Nikolaus gespeichert ist.
- Simuliere selbst eine Turtle/Schildkroete:
  - erstelle Funktionen `vorwaerts`, `rueckwaerts`, `links`, `rechts`, `stifthoch`, `stiftrunter`, `setze_farbe`, `setze_dicke` etc., mit denen du auf das svg zeichnest. (Du musst die Position und die Blickrichtung der Schildkroete speichern, und zusätzlich ...; eventuell Trigonometrie an Tafel erklären.)
  - Vermutlich/hoffentlich hast du ein «naives» Programm der folgenden Art geschrieben.

```
from math import sin, cos, pi
from random import randrange

def svg_linie_als_string(x1, y1, x2, y2, farbe, dicke):
    return f'<line x1="{x1}" y1="{y1}" x2="{x2}" y2="{y2}" style="stroke:rgb({farbe[0]}, {farbe[1]}, {farbe[2]});stroke-width:{dicke}" />'

class zeichenfenster():
    def __init__(self, breite, hoehe):
        self.breite = breite
        self.hoehe = hoehe
        self.liste_strecken = []

    def strecke(self, x1, y1, x2, y2, farbe, dicke=1):
        self.liste_strecken.append((x1, y1, x2, y2, farbe, dicke))

    def speichere_als_svg(self, name):
```

```
        with open(name, 'w') as datei:
            datei.write(f'<svg width="{self.breite}" height="{self.hoehe}" xmlns="http://www.w3.org/2000/svg">\n')
            datei.write(f'<rect x="0" y="0" width="{self.breite}\n" height="{self.hoehe}" fill="white" stroke="blue"/>\n')
            for s in self.liste_strecken:
                datei.write(svg_linie_als_string(*s)+'\n')
            datei.write('</svg>\n')

################################################
# Diverse Funktionen zur Steuerung der Kröte #
################################################

def vorwaerts(distanz):
    global x, y, alpha
    vx = distanz*cos(alpha*pi/180)
    vy = distanz*sin(alpha*pi/180)
    if stiftunten:
        fenster.strecke(x, y, x+vx, y+vy, farbe, dicke)
    x += vx
    y += vy

def rueckwaerts(distanz):
    vorwaerts(-distanz)

def links(winkel):
    global alpha
    alpha -= winkel

def rechts(winkel):
    links(-winkel)

def stifthoch():
    global stiftunten
    stiftunten = False

def stiftrunter():
    global stiftunten
    stiftunten = True

def setze_farbe(f):
    global farbe
    farbe = f

def setze_dicke(d):
    global dicke
    dicke = d

def baum(tiefe, laenge):
    setze_farbe((randrange(256), randrange(256), randrange(256)))
    vorwaerts(laenge)
    if tiefe > 0:
        links(30)
        baum(tiefe-1, laenge/2)
        rechts(60)
        baum(tiefe-1, laenge/2)
        links(30)
    stifthoch()
    rueckwaerts(laenge)
    stiftrunter()

##############################
# Beginn des Hauptprogramms #
##############################

fenster = zeichenfenster(5000, 5000)

############################################################
# Zustandsdaten der Kröte in separaten Variablen speichern #
############################################################

x = 2500
y = 5000
alpha = -90
farbe = (255, 0, 0)
dicke = 1
stiftunten = True

setze_dicke(10)
baum(9, 2000)

fenster.speichere_als_svg("output.svg")
```

– Es folgen drei Programme, die mehr oder weniger dasselbe machen, jedoch einmal «naiv», einmal mit einem Dictionary und einmal objektorientiert.
Versuche, diese Programme zu verstehen. (Wohl gut, hier alle Programme kurz zu erklären.)
«Naiv»:

```
def vorstellung():
    s = f'\nIch heisse {name}, bin {alter} Jahre alt und '
    if not erwachsen:
        s += 'nicht '
    s += f'erwachsen.\nDie Liste meiner Noten ist {notenliste}.\n'
    s += f'Mein Notendurchschnitt ist {notenschnitt()}.\n'
    return s

def speichere_note(note):
    notenliste.append(note)

def geburtstag():
    global alter
    alter += 1
    if alter >= 18:
        erwachsen = True
```

```
def notenschnitt():
    if len(notenliste) > 0:
        return sum(notenliste)/len(notenliste)
    else:
        return 'undefiniert'

#########################################################
# Daten des Schülers in separaten Variablen speichern #
#########################################################

name = "Pinocchio"
alter = 16
notenliste = []
erwachsen = alter >= 18

print(vorstellung())
speichere_note(4.0)
speichere_note(4)
speichere_note(6.0)
geburtstag()
geburtstag()
print(vorstellung())
```

Mit Dictionary:

```
def vorstellung():
    s = f'\nIch heisse {A["name"]}, bin {A["alter"]} Jahre alt und '
    if not A["erwachsen"]:
        s += 'nicht '
    s += f'erwachsen.\nDie Liste meiner Noten ist {A["notenliste"]}.\n'
    s += f'Mein Notendurchschnitt ist {notenschnitt()}.\n'
    return s

def speichere_note(note):
    A["notenliste"].append(note)

def geburtstag():
    A["alter"] += 1
    if A["alter"] >= 18:
        A["erwachsen"] = True

def notenschnitt():
    if len(A["notenliste"]) > 0:
        return sum(A["notenliste"])/len(A["notenliste"])
    else:
        return 'undefiniert'

#########################################################
# Daten des Schülers in einem Dictionary speichern #
#########################################################

A = {
    "name" : "Pinocchio",
    "alter" : 16,
    "notenliste" : []
}
A["erwachsen"] = A["alter"] >= 18

print(vorstellung())
speichere_note(4.0)
speichere_note(4)
speichere_note(6.0)
geburtstag()
geburtstag()
print(vorstellung())
```

Objektorientiert:

```
#########################################################################################################
# Bündelung der Schülerdaten und zugehörigen Methoden in einer Klasse (objektorientiertes Programmieren) #
#########################################################################################################

class Schueler():

    def __init__(self, name, alter, notenliste):
        self.name = name
        self.alter = alter
        self.notenliste = notenliste
        self.erwachsen = alter >= 18

    def __repr__(self):
        s = f'\nIch heisse {self.name}, bin {self.alter} Jahre alt und '
        if not self.erwachsen:
            s += 'nicht '
        s += f'erwachsen.\nDie Liste meiner Noten ist {self.notenliste}.\n'
        s += f'Mein Notendurchschnitt ist {self.notenschnitt()}.\n'
        return s

    def speichere_note(self, note):
        self.notenliste.append(note)
```

```
    def geburtstag(self):
        self.alter += 1
        if self.alter >= 18:
            self.erwachsen = True

    def notenschnitt(self):
        if len(self.notenliste) > 0:
            return sum(self.notenliste)/len(self.notenliste)
        else:
            return 'undefiniert'
A = Schueler("Pinocchio", 16, [])
B = Schueler("Cinderella", 8, [])

print(A)
A.speichere_note(4.0)
A.speichere_note(4)
A.speichere_note(6.0)
A.geburtstag()
A.geburtstag()
print(A)

print(B)
B.geburtstag()
B.speichere_note(6)
B.speichere_note(6)
B.speichere_note(6)
print(B)
```

– Schreibe nun das Schildkrötprogramm auf zwei Arten um:
  * mit einem Dictionary, das alle Schildkrötendaten gebündelt speichert;
  * objektorientiert.

  (Auf den folgenden Seiten folgt eine mögliche Lösung.)

Mit Dictionary:

```
from math import sin, cos, pi
from random import randrange

def svg_linie_als_string(x1, y1, x2, y2, farbe, dicke):
    return f'<line x1="{x1}" y1="{y1}" x2="{x2}" y2="{y2}" style="stroke:rgb({farbe[0]}, {farbe[1]}, {farbe[2]});stroke-width:{dicke}" />'

class zeichenfenster():
    def __init__(self, breite, hoehe):
        self.breite = breite
        self.hoehe = hoehe
        self.liste_strecken = []

    def strecke(self, x1, y1, x2, y2, farbe, dicke=1):
        self.liste_strecken.append((x1, y1, x2, y2, farbe, dicke))

    def speichere_als_svg(self, name):
        with open(name, 'w') as datei:
            datei.write(f'<svg width="{self.breite}" height="{self.hoehe}" xmlns="http://www.w3.org/2000/svg">\n')
            datei.write(f'<rect x="0" y="0" width="{self.breite}\n" height="{self.hoehe}" fill="white" stroke="blue"/>\n')
            for s in self.liste_strecken:
                datei.write(svg_linie_als_string(*s)+'\n')
            datei.write('</svg>\n')

##########################################
# Diverse Funktionen zur Steuerung der Kröte #
##########################################

def vorwaerts(distanz):
    vx = distanz*cos(kroete["alpha"]*pi/180)
    vy = distanz*sin(kroete["alpha"]*pi/180)
    if kroete["stiftunten"]:
        fenster.strecke(kroete["x"], kroete["y"], kroete["x"]+vx, kroete["y"]+vy, kroete["farbe"], kroete["dicke"])
    kroete["x"] += vx
    kroete["y"] += vy

def rueckwaerts(distanz):
    vorwaerts(-distanz)

def links(winkel):
    kroete["alpha"] -= winkel

def rechts(winkel):
    links(-winkel)

def stifthoch():
    kroete["stiftunten"] = False

def stiftrunter():
    kroete["stiftunten"] = True

def setze_farbe(farbe):
    kroete["farbe"] = farbe

def setze_dicke(dicke):
    kroete["dicke"] = dicke

def baum(tiefe, laenge):
    setze_farbe((randrange(256), randrange(256), randrange(256)))
    vorwaerts(laenge)
    if tiefe > 0:
        links(30)
        baum(tiefe-1, laenge/2)
        rechts(60)
        baum(tiefe-1, laenge/2)
        links(30)
    stifthoch()
    rueckwaerts(laenge)
    stiftrunter()

############################
# Beginn des Hauptprogramms #
############################

fenster = zeichenfenster(5000, 5000)

#########################################################
# Zustandsdaten der Kröte in einem Dictionary speichern #
#########################################################

# 1. Möglichkeit:

kroete = {
    "x" : 2500,
    "y" : 5000,
    "alpha" : -90,
    "farbe" : (255, 0, 0),
    "dicke" : 1,
    "stiftunten" : True,
}

# 2. Möglichkeit:

# kroete = {}
# kroete["x"] = 2500
# kroete["y"] = 5000
# kroete["alpha"] = -90
# kroete["farbe"] = (255, 0, 0)
# kroete["dicke"] = 1
# kroete["stiftunten"] = True

setze_dicke(10)
baum(9, 2000)

fenster.speichere_als_svg("output.svg")
```

Objektorientiert:

```
from math import sin, cos, pi
from random import randrange

def svg_linie_als_string(x1, y1, x2, y2, farbe, dicke):
    return f'<line x1="{x1}" y1="{y1}" x2="{x2}" y2="{y2}" style="stroke:rgb({farbe[0]}, {farbe[1]}, {farbe[2]});stroke-width:{dicke}" />'

class zeichenfenster():
    def __init__(self, breite, hoehe):
        self.breite = breite
        self.hoehe = hoehe
        self.liste_strecken = []

    def strecke(self, x1, y1, x2, y2, farbe, dicke=1):
        self.liste_strecken.append((x1, y1, x2, y2, farbe, dicke))

    def speichere_als_svg(self, name):
        with open(name, 'w') as datei:
            datei.write(f'<svg width="{self.breite}" height="{self.hoehe}" xmlns="http://www.w3.org/2000/svg">\n')
            datei.write(f'<rect x="0" y="0" width="{self.breite}\n" height="{self.hoehe}" fill="white" stroke="blue"/>\n')
            for s in self.liste_strecken:
                datei.write(svg_linie_als_string(*s)+'\n')
            datei.write('</svg>\n')

#############################################################################################################
# Bündelung der Turtle-Daten und zugehörigen Methoden in einer Klasse (objektorientiertes Programmieren) #
#############################################################################################################

class Schildkroete():

    def __init__(self, x=0, y=0, alpha=0, farbe=(255, 0, 0), dicke=1, stiftunten=True):
        self.x = x
        self.y = y
        self.alpha = -alpha
        self.farbe = farbe
        self.dicke = dicke
        self.stiftunten = stiftunten

    def vorwaerts(self, distanz):
        vx = distanz*cos(self.alpha*pi/180)
        vy = distanz*sin(self.alpha*pi/180)
        if self.stiftunten:
            fenster.strecke(self.x, self.y, self.x+vx, self.y+vy, self.farbe, self.dicke)
        self.x += vx
        self.y += vy

    def rueckwaerts(self, distanz):
        self.vorwaerts(-distanz)

    def links(self, winkel):
        self.alpha -= winkel

    def rechts(self, winkel):
        self.links(-winkel)

    def stifthoch(self):
        self.stiftunten = False

    def stiftrunter(self):
        self.stiftunten = True

    def setze_farbe(self, farbe):
        self.farbe = farbe

    def setze_dicke(self, dicke):
        self.dicke = dicke

    def baum(self, tiefe, laenge):
        self.setze_farbe((randrange(256), randrange(256), randrange(256)))
        self.vorwaerts(laenge)
        if tiefe > 0:
            self.links(30)
            self.baum(tiefe-1, laenge/2)
            self.rechts(60)
            self.baum(tiefe-1, laenge/2)
            self.links(30)
        self.stifthoch()
        self.rueckwaerts(laenge)
        self.stiftrunter()

fenster = zeichenfenster(5000, 5000)

kroete = Schildkroete(2500, 5000, 90)
kroete.setze_dicke(10)
kroete.baum(9, 2000)

fenster.speichere_als_svg("output.svg")
```

1.11 Spiel- oder Entscheidungsbaum

Nächstes Mal: Lass Gelb von unten nach oben ziehen, wie beim richtigen Schach.

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

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

https://www.youtube.com/watch?v=l-hh51ncgDI

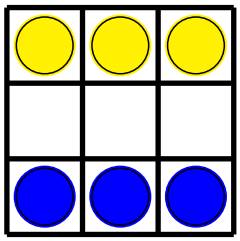
https://stackoverflow.com/questions/14826451/extending-minimax-algorithm-for-multiple-opponents

https://de.wikipedia.org/wiki/Hex_(Spiel)

https://de.wikipedia.org/wiki/Gel%C3%B6ste_Spiele

Wir betrachten das Spiel «Mini-Schach» oder «3 × 3-Schach» (ich folge Ideen, die ich von Jens Gallenbach gelernt habe). Es hat im Gegensatz zu vielen anderen Spielen den Vorteil, dass der Spielbaum so gross ist, dass man ihn gut von Hand zeichnen kann.

Startaufstellung:



Regeln

- Gelb beginnt.
- Die beiden Spieler ziehen abwechselnd.
- Alle Figuren ziehen wie die Bauern beim Schach:
  - entweder ein Feld nach vorne (wenn dieses frei ist; blau nach oben, gelb nach unten).
  - oder ein Feld diagonal nach vorne, aber nur, wenn dort eine gegnerische Figur steht, die geschlagen wird.
  - kein Schlagzwang.
  - es ist nicht erlaubt, zwei Felder auf einmal nach vorne zu gehen (und somit auch kein «en passant-Schlagen»)
- Spielende:
  - Ein Spieler, der die gegnerische Grundlinie erreicht, gewinnt (und der andere verliert).
  - Ein Spieler, der nicht ziehen kann, verliert (zum Beispiel, weil er keine Figuren mehr hat) (und der andere gewinnt).

⚒ **Aufgabe A40** Vervollständige die folgende Vorlage zu einem vollständigen Python-Programm.

(An Tafel erklären, wie das Spielfeld gespeichert werden soll.)

Datei mit Unicode-Zeichen macht aktuell Probleme.

https://fginfo.ksbg.ch/dokuwiki/lib/exe/fetch.php?media=lehrkraefte:snr:informatik:freifach-2025:minischach-vorlage.py

Aufgabe (alte Version; neue Version oberhalb)

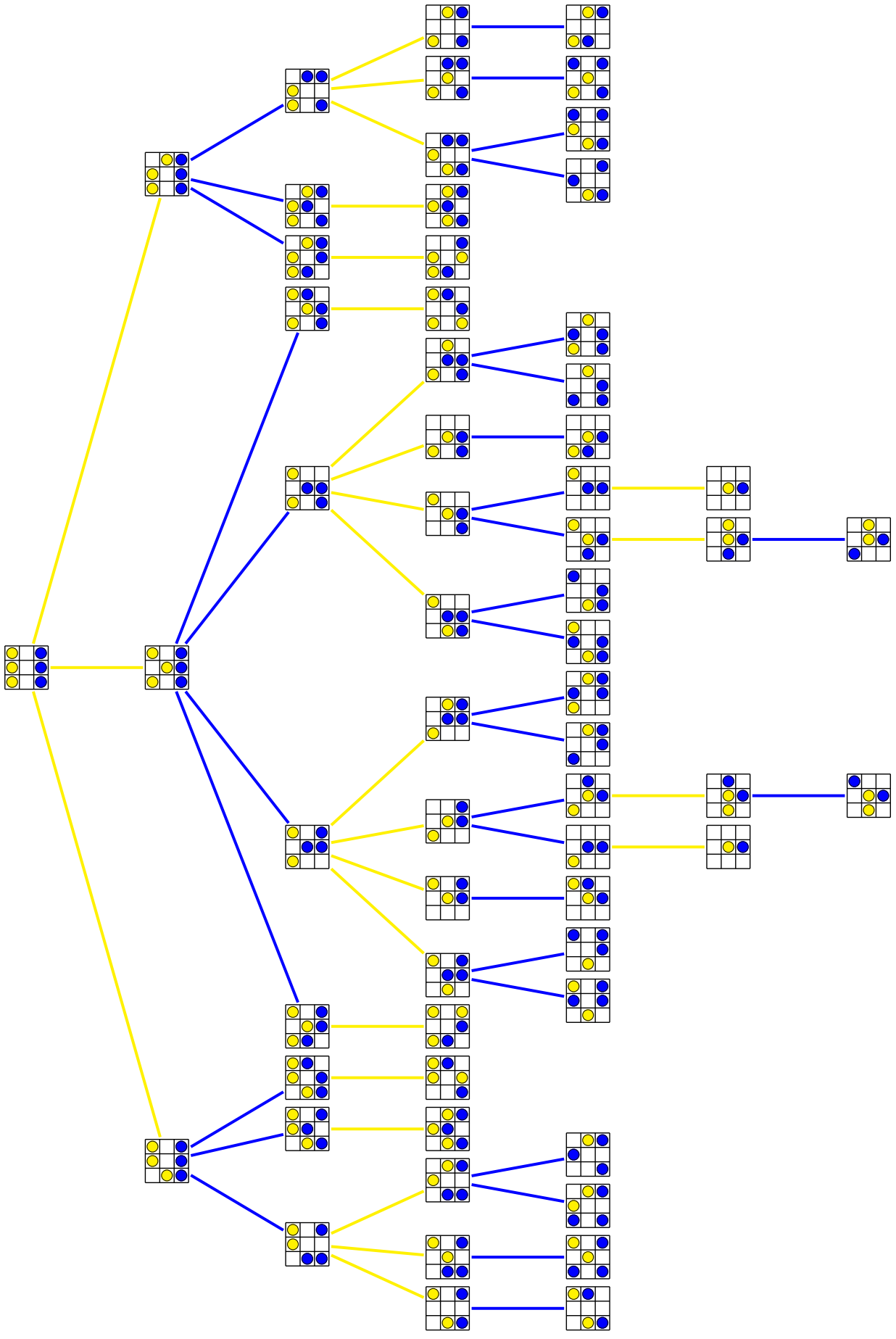
- Schreibe ein Programm, so dass zwei menschliche Spieler dieses Spiel am Computer gegeneinander spielen können.
  - die Eingabe der Züge soll geschehen durch Eingabe des Startfeldes und des Zielfeldes im «Schachkoordinatensystem», also etwa «b3 nach b2».

  - Speichere das Spielfeld in einem 5×5-Array (= Liste von Listen), Einträge sind

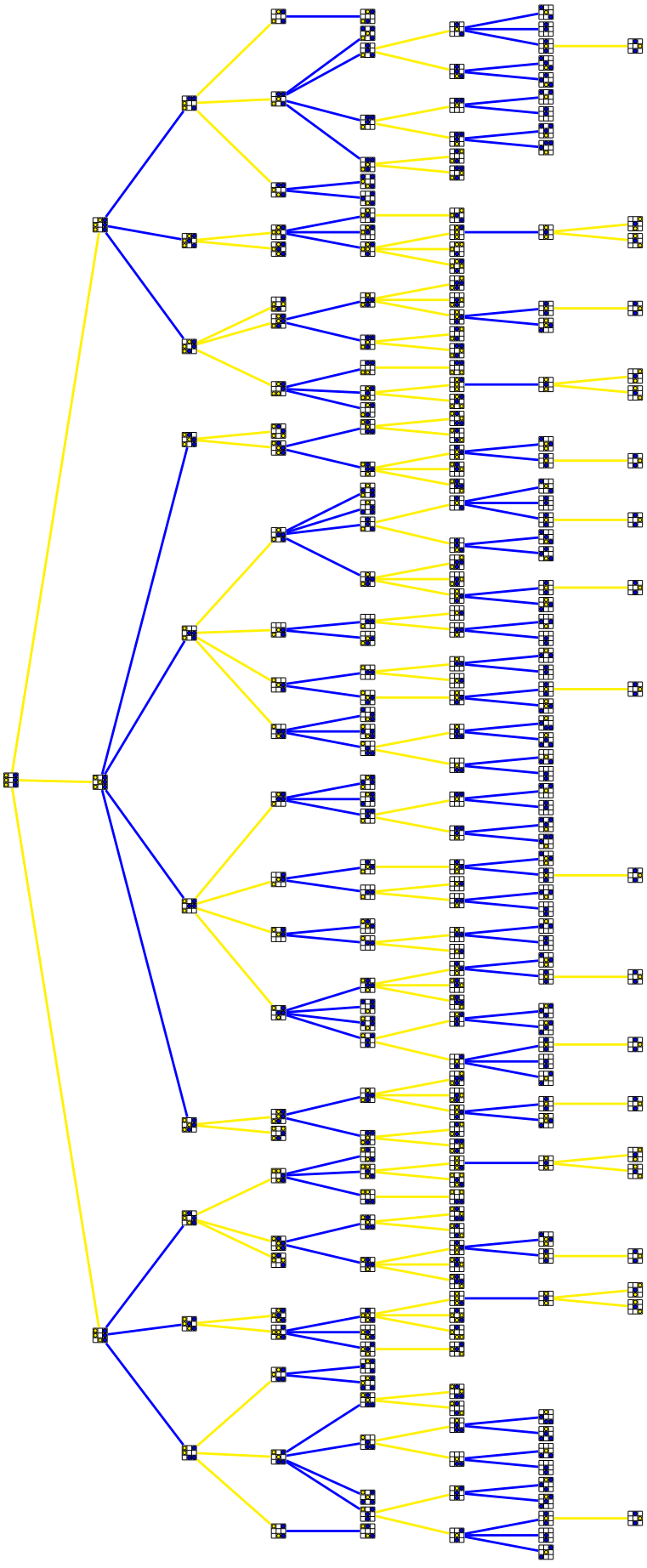
```
gelb= '🟡'
blau = '🔵'
leer = '⬜'
rand = '⬛'
```

    Das erleichtert auch die Ausgabe des Spielfeldes als Text.

- Der gesamte Spielbaum zu diesem Spiel besteht aus 252 Stellungen.
  Damit es nicht zu aufwändig wird: Wenn es von einer Stellung aus einen Gewinnzug gibt (für den Spieler, der dran ist), so zeichne man von dieser Stellung aus nur die Gewinnzüge ein (mitsamt der erhaltenen Endstellungen).
  Dieser Spielbaum besteht nur noch aus 62 Stellungen; genauer:
  - 1. Zeile: 1 Stellung
  - 2. Zeile: 3 Stellungen
  - 3. Zeile: 10 Stellungen
  - 4. Zeile: 20 Stellungen
  - 5. Zeile: 22 Stellungen
  - 6. Zeile: 4 Stellungen
  - 7. Zeile: 2 Stellungen
- Finde mit Hilfe des Spielbaums heraus, ob das Spiel stets unentschieden ausgeht (bei geschicktem Spiel beider Spieler) oder ob einer der beiden Spieler eine Gewinnstrategie hat.
- später: besten Zug per Computer finden



Wenn Gewinnzug/-züge möglich, so nur dieser/diese gezeigt:



Gesamter Suchbaum:

Für Minimax, wohl guter Suchbaum. https://de.wikipedia.org/wiki/Alpha-Beta-Suche#Der_Algorithmus

Fragen:

- Wie viele Stellungen werden bei Minimax bewertet? Bei Minimiax mit alpha-beta-pruning?
- Kann man dies noch verbessern, indem man zuerst erfolgversprechendere Züge berechnet?
- Wer gewinnt stets beim 3×3-Schach? Wer beim 3×4-Schach? Wer beim 4×4-Schach etc.
- Bei grösseren Spielfeldern braucht das Berechnen des komplessen Spielbaums sehr lange. Finde gute Bewertungsfunktionen, die in gewisser Suchtiefe abbrechen.
- Programmiere ein anderes (kleines) Spiel, etwa Tic-Tac-Toe oder ein kleines 4-gewinnt oder Nim oder ...

https://de.wikipedia.org/wiki/Satz_von_Sprague-Grundy

Reversi? Vier gewinnt?

https://math.stackexchange.com/questions/5103789/minimum-of-black-squares-to-guarantee-uniqueness-of-l

https://stackoverflow.com/beta/challenges/79768869/complete-code-challenge-8-word-ladder

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

https://de.wikipedia.org/wiki/TRIZ

https://de.wikipedia.org/wiki/Heuristik

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

Weitere Ideen

- Gültigkeitsbereich von Variablen?
- immutable variables

Spiele aus Zeitungen lösen (auskommentiert ist Screenshot der Rätselseite der NYT: Two Not Touch, KenKen, Crossword)

Niklaus Wirth Video auf Wikipedi: In der Neuauflage seines Oberon-Buches wird wohl von der Hardware übers Betriebssystem alles selbst gemacht. Er verwendet fpga's = field programmable gate array. Liefert wohl 100 Dollar Computer.

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

Für Schnelle/Interessierte/Motivierte:

- https://en.wikipedia.org/wiki/Sliding_puzzle
  insbesondere https://en.wikipedia.org/wiki/Klotski
- neuronale Netze
  - Zufallsparameter und ausprobieren; was liefert XOR am besten?
  - Sehr empfohlenes Video:
    https://www.youtube.com/watch?v=pauPCy_s0Ok
  - vermutlich gut (wie vieles von diesem Autor), aber noch nicht angeschaut:
    https://www.youtube.com/watch?v=hfMk-kjRv4c
  - evtl. pdf meines Vortrags, siehe Homepage-Freifach
- lineare Algebra; graphisch illustrieren?
- geometrische Algorithmen:
  - konvexe Hülle einer Punktmenge;
  - Punkt in Polygon? Satz von Pick.
  - minimal spanning tree? (etwa Punktmenge in $\mathbb{R}^2$ gegeben, oder abstrakt)
  - Bezier-Kurven: https://en.wikipedia.org/wiki/B%C3%A9zier_curve
    vielleicht davor string art: https://en.wikipedia.org/wiki/String_art
  - https://en.wikipedia.org/wiki/Computational_geometry#List_of_algorithms

- Strategiespiele? 3x3-Schach, Reversi,
  https://www.youtube.com/watch?v=1-hh51ncgDI
- Labyrinthe und Irrgärten kreieren und lösen
- andere Spiele: Sudoku lösen oder kreieren
- Schiebepuzzle lösen
- Pit Noack-Buch (in Bibliothek);
- Al Sweigart Bücher
- https://realpython.com/tic-tac-toe-python/
  https://realpython.com/tic-tac-toe-ai-python/
- https://amankharwal.medium.com/130-python-projects-with-source-code-61f498591bb
- Ray tracing
  https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

Minted package: siehe minted-python.tex (lief aber hier reinkopiert nicht...?)

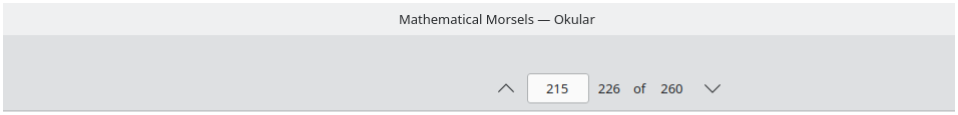
Danach:

- recursive book of recursion anschauen!
- für Fortgeschrittene: Droste-Bilder? Habe Programm geschrien, benötige jedoch besseres Bild mit Person vor Bilderrahmen ...
- Fraktale mit Turtle-Grafik
- objektorientiert Programmieren lernen: Turtle-Grafik-Bibliothek selbst schreiben! Output in SVG?
- Dynamische Fraktale?

Hüllkurven! Z. B. im D54: Ellipse, Hyperbel, Parabel

Per Fadenbild Fotos erzeugen: Den Faden einzeichnen, der Grauwerte am besten verbessert (oder so ähnlich).

Zwei Bücher von Jeremy Kun, vielleicht nett. Siehe https://www.jeremykun.com/



2. In order to find the last 3 digits of F_{73} we shall use, without proof, the following two remarkable discoveries:

(i) the square of a natural number and its 22nd power always end in the same two digits:

$$n^{22} \equiv n^2 \pmod{100};$$

(ii) the cube of a natural number and its 103rd power always end in the same three digits:

$$n^{103} \equiv n^3 \pmod{1000}.$$

1.12 Ideen, im Entstehen

Ambigramme erzeugen? https://de.wikipedia.org/wiki/Ambigramm

Wort-Palindrome:

- Is it crazy how saying sentences backwards creates backwards sentences saying how crazy it is?
- King, are you glad you are king?
- Kein Palindrom im ganz strengen Sinne, aber laut Wikipedia:Palindrom ist auch Regal/Lager ein Palindrom, aber auch sinnvoll von hinten:
  Are you as bored as me?

siehe https://english.stackexchange.com/questions/532216/word-for-sentences-that-make-sense-when-read-ba
554185

Habe Buch von Sedgewick et alius in meinem books-buecher-Verzeichnis

https://de.wikipedia.org/wiki/Liste_von_Algorithmen

oder auf Englisch:

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

Buch von Armin Barth: Algorithmik für Einsteiger, in Verzeichnis books-buecher

Foldology, Faltpuzzle

https://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic

Zeilen eines Programms permutieren. S müssen die Zeilen in die richtige Reihenfolge bringen.

Dreidimensionalen Graphen zeichnen: Kleine Vierecke zeichnen.

Dreidimensionalen Graphen drehen: Am besten wohl per Raytracing

Raytracing/Ray tracing: https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

Wärmeleitungsgleichung und Wellengleichung!

https://fginfo.ksbg.ch/dokuwiki/doku.php?id=lehrkraefte:snr:informatik:glf4-24

Bildbearbeitung:

- RGB-Farben vertauschen, etwa zu GRB.
- nur Grauwerte
- Kantenfilter drüberlaufenlassen.
- Bildkomprimierung (derselbe Farbwert nur in 10x10-Feldern)

Huffmann-Kodierung

Pygame, einfaches Spiel

Koralle

springender Ball

String art https://en.wikipedia.org/wiki/String_art

Programmiere Hangman/Galgenmännchen (Wort aus Liste extrahieren)

andere Buchstabenspiele (Mastermind = Wortiger; Wörter finden im Rechteck (wie an Geburtstag))

Interessantes:

Goethe-Fortsetzung: Autor erkennen an Wortpaaren oder ähnlichem, Wortverwendung.

Sprache erkennen an Buchstabenhäufigkeit

statt Caesar-Verschlüsselung: Permutationsverschlüsselung, so lange herumprobieren (bei wahrscheinlichen), bis korrekte Wörter entstehen.

Programmiere Simulation: Münze werfen oder würfeln, bis gewisse

Versickerung

Wärmeleitungsgleichung

Wellengleichung

https://de.wikipedia.org/wiki/Nagel-Schreckenberg-Modell

Game of Life

Pythagoräische Tripel erzeugen (ausprobieren, per ggt, der Grösse nach ordnen)

Pythagoräische Quadrupel?

REKURSION

https://inventwithpython.com/recursion/

- Fakultät (iterativ und rekursiv)
- Fibonacci
- Summe einer Liste
- String/Liste umdrehen
- Palindrom erkennen: isPalindrome
  - level
  - race car
  - taco cat
  - a man, a plan, a canal ... Panama
- Permutationen!
- meine Idee: Maximum einer Liste finden
- Aufgabe: Liste per Mergesort sortieren, Graphik siehe S. 105 Sweigart
- Fraktale! Dann könnte danach objektorientiert Programmieren!
- Droste Effekt
- weitere Ideen in Sweigarts Buch;

einfaches jump and run programmieren

(wie hiess das von mir programmierte Spiel?)

objektorientiert Programmieren: Turtle

deploy everywhere

Zeichenprogramm (Maussteuerung)

https://de.wikipedia.org/wiki/Icosian_Game

Sortieren!

Median bestimmen: https://rcoh.me/posts/linear-time-median-finding/

schau in meinem DokuWiki nach netten Aufgaben

Grafik

- Chaos-Spiel
- Fraktale
- Korallenwachstum
- Graph einer Funktion zeichnen
- Bezier-Kurven: https://en.wikipedia.org/wiki/B%C3%A9zier_curve
  vielleicht davor string art: https://en.wikipedia.org/wiki/String_art

Matplotlib https://matplotlib.org/cheatsheets/

```
Drei Chinesen mit dem Kontrabass
sassen auf der Straße und erzählten sich was.
Da kam die Polizei, fragt: "Was ist denn das?"
Drei Chinesen mit dem Kontrabass.
```

- Wort ersetzen
- jeden 5.ten Buchstaben ausgeben, dasselbe von hinten
- Zeilen in Listen extrahieren
- Dictionary: Bei jedem Wort Anzahl speichern.
- Substrings suchen, wie oft
- Slalomtext
- Wörter finden und jedes Wort in sich permutieren
- Goethes Faust: Kreiere ähnlichen Text durch Häufigkeits-Ergänzung (KI) (autocomplete)
- https://fginfo.ksbg.ch/dokuwiki/doku.php?id=lehrkraefte:snr:informatik:fachdidaktik:list-comprehens[]=list&s[]=comprehension

Aus Datei lesen.

In Datei speichern.

dir, type, help

Text vorlesen:

https://stackoverflow.com/questions/1614059/how-to-make-python-speak

Vermischtes

⚒ **Aufgabe A41** Schreiben Sie eine Funktion `ist_prim(n)`, die den booleschen Wert (= Wahrheitswert) `True` zurückliefert, wenn n eine Primzahl ist, und sonst `False`.

Verwenden Sie Ihre Funktion, um eine Liste aller Primzahlen von 1 bis 100 auszugeben.

⚒ **Aufgabe A42** Am 1. Januar des Jahres 2025 hat Herr Huber 100'000 Franken Schulden bei einer Bank. Er tilgt jedes Jahr jeweils am 1. Juni 5'000 Franken der Schuld (bzw. den noch ausstehenden Restbetrag).

Der Kreditzins beträgt 2%. Am 31. Dezember wird die aktuelle Schuld dementsprechend erhöht.

Erweitern Sie das unten angegebene Programm zu einem Programm, das die folgenden Fragen beantwortet:

- In welchem Jahr ist die Schuld komplett zurückgezahlt?
- Wie viele Tilgungen hat Herr Huber bis dahin vorgenommen?
- Wie viele Franken hat Herr Huber bis dahin der Bank insgesamt überwiesen?

Das Programm soll auch dann funktionieren, wenn die Startwerte der Variablen `schuldbetrag`, `kreditzins`, `tilgung`, `startjahr` verändert werden.

```
schuldbetrag = 100000
kreditzins = 0.02
tilgung = 5000
startjahr = 2025
```

Aufgabe A43 Folgender Python-Code erzeugt **ganzzahlige** Zufallszahlen zwischen 0 (einschliesslich) und 3 (ausschliesslich).

```
from random import randrange
n = 6
while n>0:
    z = randrange(3)
    print(z)
    n = n-1
```

```
1
0
1
2
2
1
```

Schreiben Sie ein Programm, das sich eine ganzzahlige Zufallszahl zwischen 0 und 100 ausdenkt (jeweils einschliesslich). Der Benutzer soll die Zahl erraten. Das Programm gibt bei jedem Tipp an, ob die Zahl zu klein, zu gross oder korrekt ist. Ein Beispieldialog könnte so aussehen:

```
Ich habe mir eine natürliche Zahl zwischen 0 und 100 ausgedacht. Welche Zahl ist es?
Dein Tipp: 21
Zu klein.
Dein Tipp: 84
Zu gross.
Dein Tipp: 42
Korrekt. Herzlichen Glückwunsch!
Benötigte Versuche: 3
```

Testen Sie Ihr Programm auf einem Computer!

1.13 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("Kontostand Jahr ", n, ":", kontostand)
    n = n+1
```

```
Kontostand Jahr  1 : 6060.0
Kontostand Jahr  2 : 12180.6
Kontostand Jahr  3 : 18362.406
Kontostand Jahr  4 : 24606.03006
Kontostand Jahr  5 : 30912.090360600003
Kontostand Jahr  6 : 37281.21126420601
Kontostand Jahr  7 : 43714.02337684807
Kontostand Jahr  8 : 50211.163610616546
Kontostand Jahr  9 : 56773.27524672271
Kontostand Jahr  10 : 63401.00799918994
Kontostand Jahr  11 : 70095.01807918184
Kontostand Jahr  12 : 76855.96825997366
Kontostand Jahr  13 : 83684.52794257339
Kontostand Jahr  14 : 90581.37322199912
Kontostand Jahr  15 : 97547.18695421911
Kontostand Jahr  16 : 104582.6588237613
Kontostand Jahr  17 : 111688.48541199892
Kontostand Jahr  18 : 118865.3702661189
Kontostand Jahr  19 : 126114.0239687801
Kontostand Jahr  20 : 133435.1642084679
Kontostand Jahr  21 : 140829.5158505526
Kontostand Jahr  22 : 148297.81100905812
Kontostand Jahr  23 : 155840.7891191487
Kontostand Jahr  24 : 163459.1970103402
Kontostand Jahr  25 : 171153.7889804436
Kontostand Jahr  26 : 178925.32687024804
Kontostand Jahr  27 : 186774.58013895052
Kontostand Jahr  28 : 194702.32594034003
Kontostand Jahr  29 : 202709.34919974342
Kontostand Jahr  30 : 210796.44269174084
Kontostand Jahr  31 : 218964.40711865824
Kontostand Jahr  32 : 227214.05118984482
Kontostand Jahr  33 : 235546.19170174326
Kontostand Jahr  34 : 243961.6536187607
Kontostand Jahr  35 : 252461.27015494832
```

⚒ 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')
vorletztes_glied = 0
letztes_glied = 1
nummer = 0
print(nummer, vorletztes_glied)
nummer = nummer + 1
print(nummer, letztes_glied)
while nummer < maxn:
    summe = vorletztes_glied + letztes_glied
    vorletztes_glied = letztes_glied
    letztes_glied = summe
    nummer = nummer + 1
    print(nummer, letztes_glied)
```

Für Nerds, etwas kürzer und rekursiv (was man bei Fibonacci eigentlich nicht machen sollte):

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

```
Drinnen 807 von 1000
Anteil 0.807
Pi ist ungefähr 3.228
```

Lösung zu A5 ex-quadratische-gleichung

Lösung zu A6 ex-slalomtext

```
n=100
import time
satz = "Schleifen sind cool!"
einrueckung = 0
```

```
veraenderung = 1
for i in range(n):
    print(einrueckung * " " + satz)
    einrueckung = einrueckung + veraenderung
    if einrueckung % 5 == 0:
        veraenderung = -veraenderung
    time.sleep(0.05)
```

```
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
  Schleifen sind cool!
 Schleifen sind cool!
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
  Schleifen sind cool!
 Schleifen sind cool!
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
  Schleifen sind cool!
 Schleifen sind cool!
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
  Schleifen sind cool!
 Schleifen sind cool!
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
  Schleifen sind cool!
 Schleifen sind cool!
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
  Schleifen sind cool!
 Schleifen sind cool!
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
  Schleifen sind cool!
 Schleifen sind cool!
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
```

```
  Schleifen sind cool!
 Schleifen sind cool!
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
  Schleifen sind cool!
 Schleifen sind cool!
Schleifen sind cool!
 Schleifen sind cool!
  Schleifen sind cool!
   Schleifen sind cool!
    Schleifen sind cool!
     Schleifen sind cool!
    Schleifen sind cool!
   Schleifen sind cool!
  Schleifen sind cool!
 Schleifen sind cool!
```

Lösung zu A7 ex-text-veraendern

Lösung zu A8 ex-text-caesar-verschluesseln

Lösung zu A9 ex-text-analysieren

Lösung zu A10 ex-nonsense-texter

Lösung zu A11 ex-basics-schleifen

```
print("Mit for-Schleife und range(end): Natürliche Zahlen von 0 bis 20:")
for x in range(21):
    print(x, end=', ')
print()

print("Mit while-Schleife: Natürliche Zahlen von 0 bis 20:")
x= 0
while x < 21:
    print(x, end=', ')
    x = x+1
print()

print("Mit for-Schleife und range(start, end): Ganze Zahlen von -10 bis 10:")
for x in range(-10, 11):
    print(x, end=', ')
print()

print("Mit while-Schleife: Ganze Zahlen von -10 bis 10:")
x= -10
while x <= 10:
    print(x, end=', ')
    x = x+1
print()

print("Mit for-Schleife und range(start, end, step): Jede dritte natürliche Zahl von 11 bis 24:")
for x in range(11, 25, 3):
    print(x, end=', ')
print()

print("Mit while-Schleife: Jede dritte natürliche Zahl von 11 bis 24:")
x= 11
while x <= 24:
    print(x, end=', ')
    x += 3
print()

print("Mit while-Schleife: Reelle Zahlen von -1 bis 1 in Schritten von 0.1:")
x= 0
```

```
while x <= 1:
    print(f'{x:.1f}', end=', ')
    x += 0.1
print()
```

Lösung zu A12 ex-zahlen-summieren

```
Gib eine Zahl ein: 9
Die Summe aller Zahlen von 0 bis 9 beträgt 45.
Die Summe aller Quadratzahlen von 0 bis 9**2=81 beträgt 285.
Die Summe aller Kubikzahlen von 0 bis 9**3=729 beträgt 2025.

Drei Zahlen: 45.0, 285.0, 2025.0.
```

Lösung zu A13 ex-multiplikationstabelle

Lösung zu A14 ex-basics-if-kopfrechentrainer

Lösung zu A15 ex-basics-funktionen

Lösung zu A16 ex-basics-funktionen-zahlentheorie

Lösung zu A17 ex-basics-strings

Lösung zu A18 ex-basics-listen

Lösung zu A19 ex-quiz

Lösung zu A20 ex-zahlen-in-liste-einlesen

Lösung zu A21 ex-primzahlen-sieb-des-eratosthenes

Lösung zu A22 ex-basics-dictionaries

Lösung zu A23 ex-zeichnen-mit-pygame

Lösung zu A24 ex-chaos-game

Variante 1: Einfachste Version

```
import pygame
from random import *

s = 800
breite = s
hoehe = s

weiss = (255, 255, 255)
schwarz = (0, 0, 0)
```

```
rot = (255, 0, 0)

pygame.init()
fenster = pygame.display.set_mode((breite+1, hoehe+1))
pygame.display.set_caption("Das Chaos-Spiel")

Ax = 0
Ay = s

Bx = s
By = s

Cx = s // 2
Cy = round(s-3**0.5/2*s)

fenster.fill(schwarz)

pygame.draw.line(fenster, rot, (Ax, Ay), (Bx, By), width=2)
pygame.draw.line(fenster, rot, (Ax, Ay), (Cx, Cy), width=2)
pygame.draw.line(fenster, rot, (Bx, By), (Cx, Cy), width=2)

pygame.display.update()

x = randrange(breite+1)
y = randrange(hoehe+1)

ende = False
while not ende:
    # Zeichnet ein Pixel am Punkt (x,y).
    # Koordinaten müssen ganze Zahlen sein.
    fenster.set_at((round(x),round(y)), weiss)
    pygame.display.update()

    zufall = randrange(3)
    if zufall == 0:
        # Springe halb zum Punkt A.
        x = (x+Ax)/2
        y = (y+Ay)/2
    elif zufall == 1:
        x = (x+Bx)/2
        y = (y+By)/2
    else:
        x = (x+Cx)/2
        y = (y+Cy)/2

    # Setze Variable ende auf True, falls Fenster per Maus geschlossen wird.
    for ereignis in pygame.event.get():
        if ereignis.type == pygame.QUIT:
            ende = True
pygame.quit()
```

Variante 2: Punkte als Listen mit zwei Einträgen, Listenelement mit Index 0 ist die x-Koordinate, Listenelement mit Index 1 ist die y-Koordinate.

```
import pygame
from random import *

s = 800
breite = s
hoehe = s

weiss = (255, 255, 255)
schwarz = (0, 0, 0)
rot = (255, 0, 0)

pygame.init()
fenster = pygame.display.set_mode((breite+1, hoehe+1))
pygame.display.set_caption("Das Chaos-Spiel")

A = [0, s]
B = [s, s]
C = [s // 2, round(s-3**0.5/2*s)]

fenster.fill(schwarz)

pygame.draw.line(fenster, rot, A, B, width=2)
pygame.draw.line(fenster, rot, A, C, width=2)
pygame.draw.line(fenster, rot, B, C, width=2)

pygame.display.update()

P = [randrange(breite+1), randrange(hoehe+1)]

ende = False
while not ende:
```

```
    fenster.set_at((round(P[0]),round(P[1])), weiss)
    pygame.display.update()

    zufall = randrange(3)
    if zufall == 0:
        P =[(P[0]+A[0])/2, (P[1]+A[1])/2]
    elif zufall == 1:
        P =[(P[0]+B[0])/2, (P[1]+B[1])/2]
    else:
        P =[(P[0]+C[0])/2, (P[1]+C[1])/2]

    for ereignis in pygame.event.get():
        if ereignis.type == pygame.QUIT:
            ende = True
pygame.quit()
```

Variante 3: Eckpunkte des Dreiecks als Liste von Punkten (die als Listen gespeichert werden wie bei Variante 2)

```
import pygame
from random import *

s = 800
breite = s
hoehe = s

weiss = (255, 255, 255)
schwarz = (0, 0, 0)
rot = (255, 0, 0)

pygame.init()
fenster = pygame.display.set_mode((breite+1, hoehe+1))
pygame.display.set_caption("Das Chaos-Spiel")

A = [0, s]
B = [s, s]
C = [s // 2, round(s-3**0.5/2*s)]

liste_eckpunkte = [A, B, C]

fenster.fill(schwarz)
pygame.draw.polygon(fenster, rot, liste_eckpunkte, width=2)
pygame.display.update()

P = [randrange(breite+1), randrange(hoehe+1)]

ende = False
while not ende:
    fenster.set_at((round(P[0]),round(P[1])), weiss)
    pygame.display.update()

    zufall = randrange(3)
    Q = liste_eckpunkte[zufall]
    P = [(P[0]+Q[0])/2, (P[1]+Q[1])/2]

    for ereignis in pygame.event.get():
        if ereignis.type == pygame.QUIT:
            ende = True
pygame.quit()
```

Variante 4: Mit numpy, einer sehr weit verbreiteten Bibliothek zum Rechnen mit Matrizen und Vektoren und für Numerik (und vieles andere).

```
import pygame
from random import *
import numpy as np

s = 800
breite = s
hoehe = s

weiss = (255, 255, 255)
schwarz = (0, 0, 0)
rot = (255, 0, 0)

pygame.init()
fenster = pygame.display.set_mode((breite+1, hoehe+1))
pygame.display.set_caption("Das Chaos-Spiel")

A = np.array([0, s])
B = np.array([s, s])
C = np.array([s // 2, round(s-3**0.5/2*s)])
```

```
liste_eckpunkte = [A, B, C]

fenster.fill(schwarz)
pygame.draw.polygon(fenster, rot, liste_eckpunkte, width=2)
pygame.display.update()

P = np.array([randrange(breite+1), randrange(hoehe+1)])

ende = False
while not ende:
    fenster.set_at((round(P[0]),round(P[1])), weiss)
    pygame.display.update()

    zufall = randrange(3)
    Q = liste_eckpunkte[zufall]
    P = (P+Q)/2

    for ereignis in pygame.event.get():
        if ereignis.type == pygame.QUIT:
            ende = True
pygame.quit()
```

Variante 5: Schnellere Version von Variante 4; der Bildschirminhalt wird nur nach jeweils 1000 gezeichneten Punkten erneuert.

```
import pygame
from random import *
import numpy as np

s = 800
breite = s
hoehe = s

weiss = (255, 255, 255)
schwarz = (0, 0, 0)
rot = (255, 0, 0)

pygame.init()
fenster = pygame.display.set_mode((breite+1, hoehe+1))
pygame.display.set_caption("Das Chaos-Spiel")

A = np.array([0, s])
B = np.array([s, s])
C = np.array([s // 2, round(s-3**0.5/2*s)])

liste_eckpunkte = [A, B, C]

fenster.fill(schwarz)
pygame.draw.polygon(fenster, rot, liste_eckpunkte, width=2)
pygame.display.update()

P = np.array([randrange(breite+1), randrange(hoehe+1)])

ende = False
i = 0
while not ende:
    fenster.set_at((round(P[0]),round(P[1])), weiss)
    i=i+1
    if i % 1000 == 0:
        pygame.display.update()
        i = 0

    zufall = randrange(3)
    Q = liste_eckpunkte[zufall]
    P = (P+Q)/2

    for ereignis in pygame.event.get():
        if ereignis.type == pygame.QUIT:
            ende = True
pygame.quit()
```

⚒ Lösung zu A25 ex-pong-solo

⚒ Lösung zu A26 ex-graphen-zeichnen

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

Lösung zu A40 ex-minischach-programmieren-computerzuege-zufaellig

Datei mit Unicode-Zeichen macht aktuell Probleme.

https://fginfo.ksbg.ch/dokuwiki/lib/exe/fetch.php?media=lehrkraefte:snr:informatik:freifach-2025:minischach.py

Lösung zu A41 ex-funktion-ist-prim

Lösung zu A42 ex-schulden-tilgen

Lösung zu A43 ex-zahlenraten

```
from random import randrange

zahl = randrange(101)
print("Ich habe mir eine natürliche Zahl zwischen 0 und 100 ausgedacht. Welche Zahl ist es?")

zahl = 42
geratene_zahl = -100
versuche = 0
while geratene_zahl != zahl:
    geratene_zahl = int(input("Dein Tipp: "))
    versuche = versuche + 1
    if zahl == geratene_zahl:
        print("Korrekt. Herzlichen Glückwunsch!")
    else:
        if geratene_zahl < zahl:
            print("Zu klein.")
        else:
            print("Zu gross.")
print("Benötigte Versuche:", versuche)
```