



1 Programmierkonzepte (unplugged)

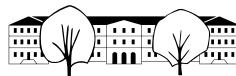
1.1 Steuerung und Regelung

☒ **Aufgabe A1** Im Schulzimmer wird eine «Ladestation» definiert. Ein «Roboter» soll diese aufsuchen, um sich wieder aufzuladen.

- a) (Gesamte Klasse) Der Roboter wird von einem Schüler gespielt. Seine Aufgabe ist es, die erhaltenen Befehle möglichst exakt umzusetzen (und nicht das umzusetzen, was vermutlich gemeint war). Unsinnige oder gefährliche Befehle dürfen auch einfach mit «Fehler: Unbekannter Befehl» quittiert werden. Ein anderer Schüler ist der «Programmierer». Er erteilt dem Roboter mündlich möglichst wenige kurze, einfache und eindeutige Befehle, um den Roboter zur Ladestation zu führen.
- b) Arbeiten Sie nun in Zweiergruppen, wobei eine Person den Roboter spielt und die andere für die Programmierung zuständig ist. Legen Sie eine Startposition des Roboters fest und geben Sie dem Roboter einen Zettel mit schriftlichen Anweisungen, die dieser dann Schritt für Schritt abarbeitet, so dass er hoffentlich bei der Ladestation ankommt.
Tauschen Sie dann die Rollen und verbessern bzw. erweitern Sie das Programm (d.h. die schriftlichen Anweisungen).
- c) Damit nicht jeder Roboter einen eigenen Befehlssatz hat, möchten wir einen «standardisierten Befehlsatz» vereinbaren. Dieser soll einerseits «universell» genug sein, aber auch möglichst klein.

| | |
|--------|---------|
| Befehl | Wirkung |
|--------|---------|

- d) Versuchen Sie, eine Roboterprogrammierung aufzuschreiben, mit der der Roboter den Weg zur Ladestation finden kann, unabhängig davon, wo er startet und in welche Richtung er anfangs schaut.
Eventuell müssen Sie dafür den Befehlssatz erweitern. Diskutieren Sie eventuell mit Kollegen, welche Erweiterungen sinnvoll sind.
- e) Wer hat das «kürzeste» universelle Programm (d. h. am wenigsten Befehle)?
- f) Wer hat das «effizienteste» Programm (d. h. jenes, mit dem der Roboter im Durchschnitt mit möglichst wenig Programmschritten das Ziel erreicht)?



Aufgabe A2 Ein Roboter startet in einem Irrgarten (Labyrinth) im Feld oben links und soll den Weg zum Ziel, dem Feld unten rechts, finden. Schreiben Sie mehrere Programme auf, mit denen der Roboter den Weg zum Ziel findet. Die «Programmiersprache» für den Roboter soll folgende Elemente enthalten: Mit «Nachbarfeld» sind nur solche Nachbarfelder gemeint, die der Roboter von seiner aktuellen Position aus erreichen kann (keine Wand dazwischen).

Allgemeine Anweisungen:

- Programm beenden mit Meldung
- Aus mehreren Möglichkeiten eine auswählen

Kontrollstrukturen:

- Solange ..., etwas wiederholen
- Wenn ..., dann ... (sonst ...)

Markierungen:

- Ein Symbol auf das aktuelle Feld schreiben
- Symbol vom aktuellen Feld/Nachbarfeld lesen

Bewegungen:

- vorwärts gehen
- nach Norden/Osten/Süden/Westen drehen
- auf ein bestimmtes Nachbarfeld gehen
- drehen

Tests (für Bedingungen):

- Test, ob der Roboter am Ziel/Start steht.
- Präsenz einer Wand (links/vorne/rechts)
- Wand nördlich/östlich/südlich/westlich?
- Präsenz und/oder Art des Symbols auf einem Feld

Testen Sie Ihre Programme (oder die Ihres Kollegen) mit Hilfe der folgenden Testlabyrinthe.

```
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
|     | |     | |     | |     | |     | |     | |     |
+---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
|     | |     | |     | |     | |     | |     | |     |
+ + + +---+ + + + +---+ + + + +---+ + + + +---+ + + +
|     | |     | |     | |     | |     | |     | |     |
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
```

```
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
|     | |     | |     | |     | |     | |     | |     |
+---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
|     | |     | |     | |     | |     | |     | |     |
+ +---+ +---+ + + + +---+ + + + +---+ + + + +---+ + +
|     | |     | |     | |     | |     | |     | |     |
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
|     | |     | |     | |     | |     | |     | |     |
+---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
|     | |     | |     | |     | |     | |     | |     |
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
```

```
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
|     | |     | |     | |     | |     | |     | |     |
+ + + +---+ +---+ + + + +---+ + + + +---+ + + + +---+
|     | |     | |     | |     | |     | |     | |     |
+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
|     | |     | |     | |     | |     | |     | |     |
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
|     | |     | |     | |     | |     | |     | |     |
+---+ + + + +---+ +---+ + + + +---+ + + + +---+ + + +
|     | |     | |     | |     | |     | |     | |     |
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
```

```
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
|     | |     | |     | |     | |     | |     | |     |
+---+ + + + +---+ +---+ + + + +---+ + + + +---+ + + +
|     | |     | |     | |     | |     | |     | |     |
+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
|     | |     | |     | |     | |     | |     | |     |
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
|     | |     | |     | |     | |     | |     | |     |
+---+ + + + +---+ +---+ + + + +---+ + + + +---+ + + +
|     | |     | |     | |     | |     | |     | |     |
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
```



Aufgabe A3 Jetzt sollen Irrgärten ohne «Roboter» gelöst und generiert werden. Das Programm sieht jetzt das ganze Labyrinth auf einmal (globale Sichtweise im Gegensatz zur lokalen Sicht des Roboters) und es kann nach Feldern gesucht werden, die bestimmte Bedingungen erfüllen.

Schreiben Sie ein Programm, das Sackgassen füllt und so einen Weg übrig lässt/markiert.

Schreiben Sie ein Programm, das immer den kürzesten Weg findet.

Aufgabe A4 Generieren Sie selbst ein Labyrinth auf die folgenden drei Arten. Bei den Varianten 1 und 2 sind am Anfang alle Mauern gesetzt.

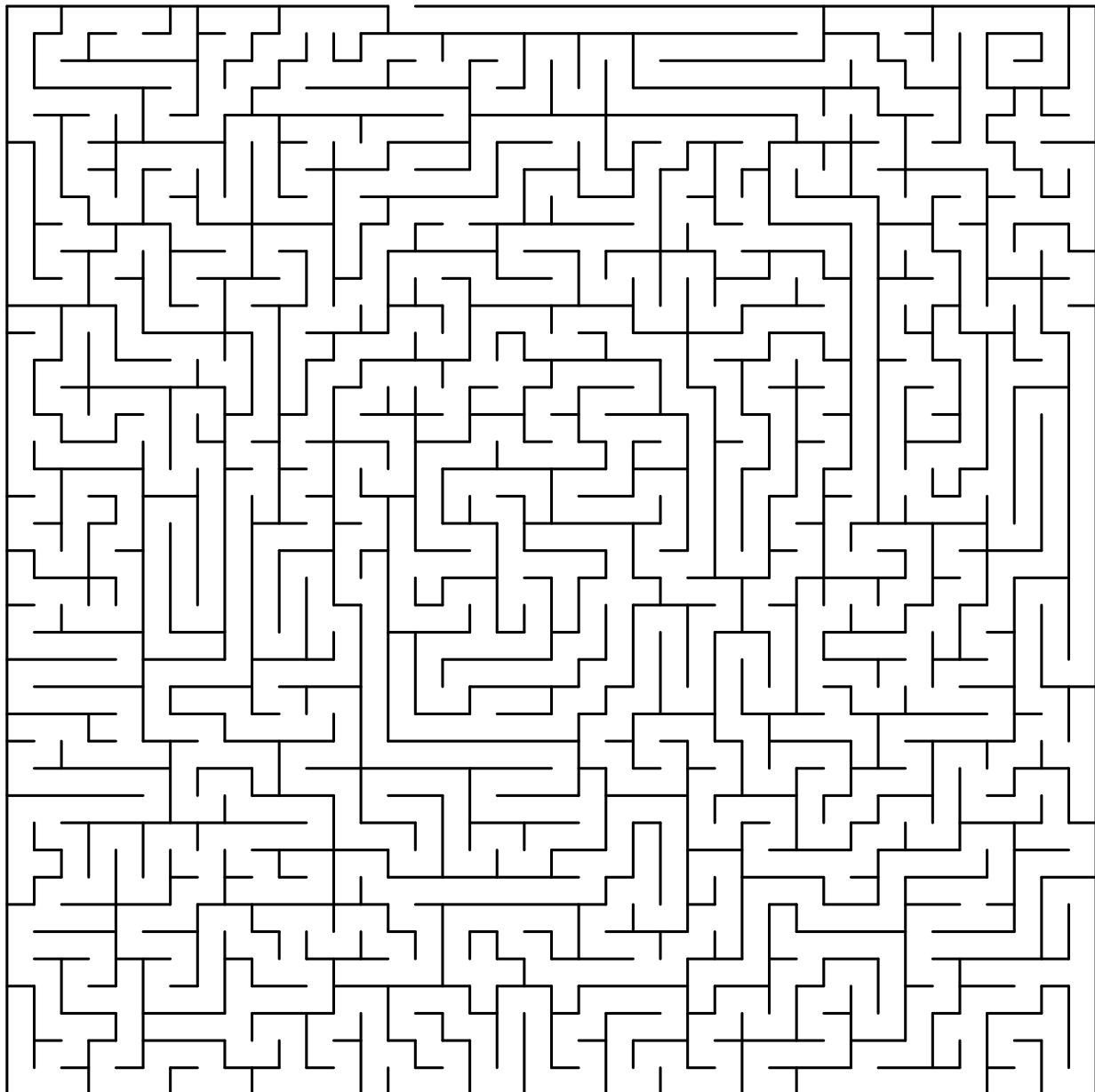
Variante 1: Wenden Sie die Tiefensuche an (wählen Sie die Richtungen jeweils zufällig) um Mauern zu öffnen.
Hinweis: Tiefensuche ist in der Lösung von Aufgabe A2 erklärt.

Variante 2: Entfernen Sie Mauern (d. h. verbinden Sie zwei Nachbarfelder), solange Sie damit keinen Zyklus erzeugen.

Variante 3: Keine Mauern außer den Randmauern sind gesetzt. Man lässt vom Rand her «Korallenbäume» aus Mauern wachsen, die aber sich selbst und andere Mauern nie berühren dürfen.

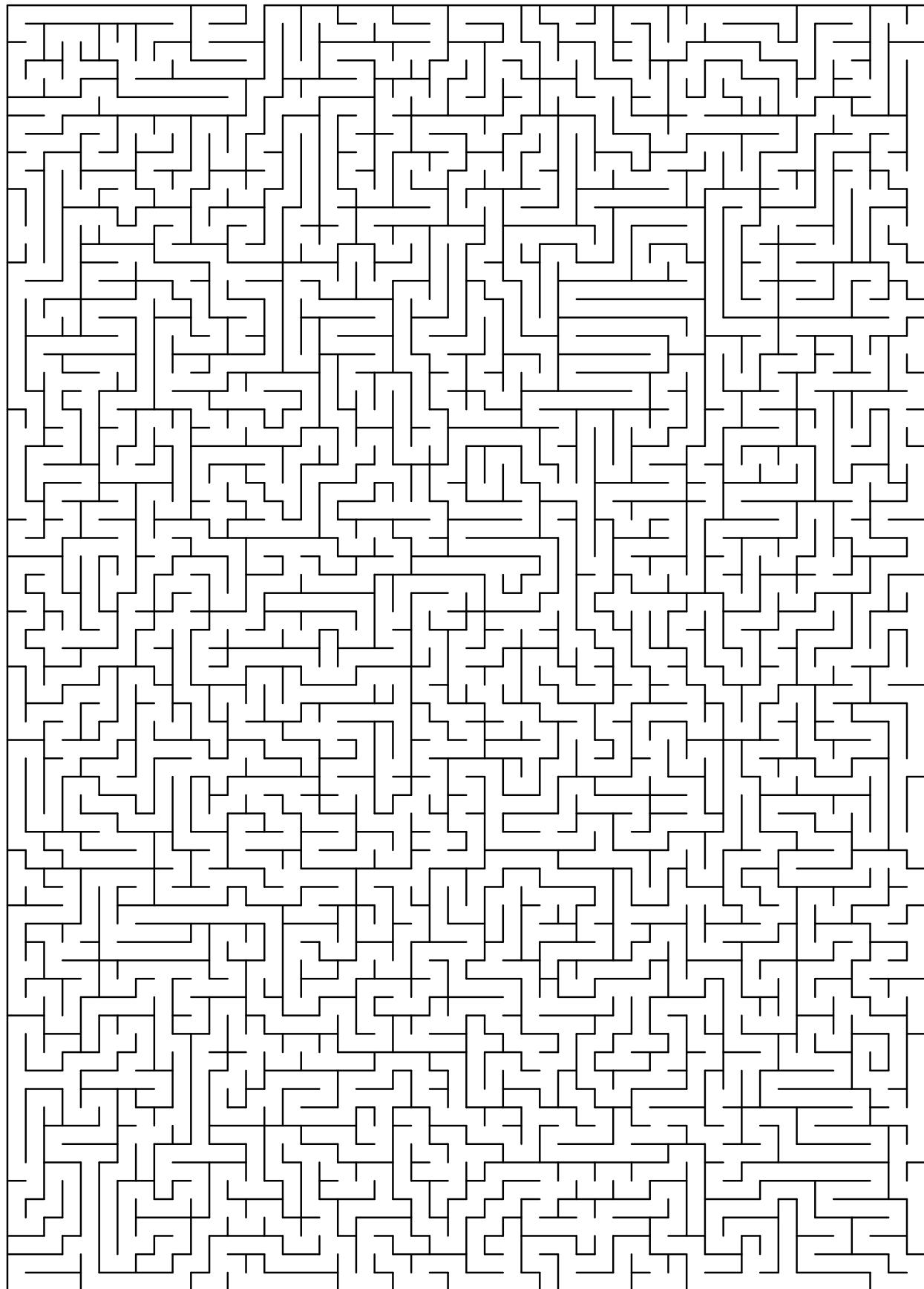


Just for fun: Markieren Sie alle Felder auf dem Lösungsweg, um ein Bild zu erhalten.





Markieren Sie alle Felder auf dem Weg, um ein Bild zu erhalten.



Merke 1.1.1 Steuerung vs. Regelung

In Aufgabe A1 wurde der Roboter zuerst in Echtzeit ferngesteuert und danach für einen fixen Weg vorausprogrammiert. Der Roboter wurde **gesteuert**. Das Programm kann nicht auf Unvorhergesehenes oder neue Startpositionen reagieren.

Am Schluss wurde ein universelles Programm entworfen, das auf Unvorhergesehenes reagieren kann. Man spricht dann von **Regelung**.

Aufgabe A5 Sind folgende Dinge eher gesteuert oder geregt? Argumentieren Sie!

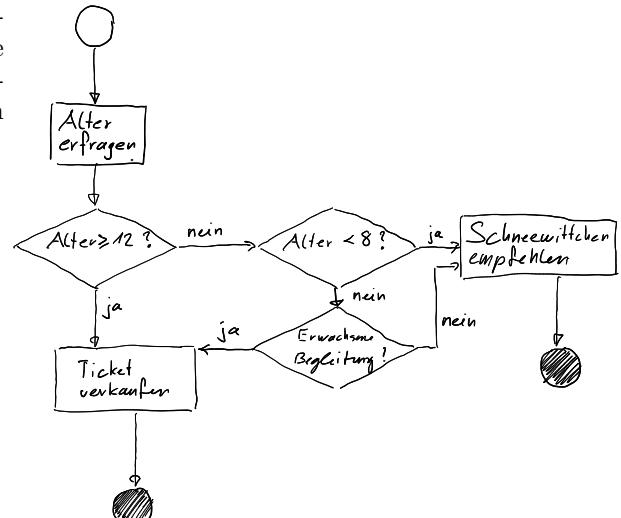
- | | | |
|-------------------------------------|----------------|--------------------------|
| a) Liftposition bezüglich Stockwerk | b) Lifttüren | c) Heizung |
| d) Herdplatte | e) Kühlschrank | f) Wohnzimmerbeleuchtung |

1.2 Flussdiagramme

Flussdiagramme (= Programmablaufpläne) sind zwar sehr einfach zu verstehen, für komplexere Programme oder Abläufe aber eher ungeeignet und darum kaum gebräuchlich. Dies soll mit den folgenden Aufgaben verständlich gemacht werden.

Aufgabe A6

Sie haben sich für einen Ferienjob an der Kinokasse beworben und diesen auch erhalten. Am ersten Tag verkaufen Sie nur Billette für den Film «Flipper im Flussdiagramm». Dazu erhalten Sie die Skizze rechts als Anweisung. Formulieren Sie diese Einlassregel in zwei einfachen Sätzen.


Merke 1.2.1 Elemente eines Flussdiagramms

In einem Flussdiagramm kommen 4 Elemente vor, die mit Pfeilen miteinander verbunden sind:

Start Symbolisiert durch einen leeren Kreis.

Anweisung Rechteck mit genau einem ausgehenden Pfeil.

Verzweigung Raute mit Bedingung und zwei Ausgängen «ja» und «nein»

Ende Symbolisiert durch einen vollen Kreis.

Aufgabe A7 Sie haben den ersten Tag Ihres Ferienjobs an der Kinokasse gut gemeistert und verkaufen nun Tickets zu drei Filmen:

- «Schneewittchen» ohne Altersbegrenzung.
- «Flipper im Flussdiagramm» (wie in A6).
- «Der Spaghetti-Code» ab 16 Jahren, in Begleitung ab 13 Jahren.

Zeichnen Sie den Ablauf als Flussdiagramm.

☒ **Aufgabe A8** Im Kino gibt es einen Getränke-Automaten, der nur Mineralwasser der Marke «Trivial» zum Preis von CHF 3.50 verkauft.

- Beschreiben Sie das Verhalten des Getränkeautomaten mit einem Flussdiagramm. Das Diagramm soll keinen Endknoten haben, denn der Automat soll einfach auf den nächsten Verkauf warten.
- Testen Sie Ihr Flussdiagramm in Zweiergruppen und ergänzen bzw. korrigieren Sie es, falls nötig.
- Der Automat wird nun auf 3 Getränke ausgebaut, es kommen «Trivial Orange» für CHF 4.- und «Trivial Cola» für CHF 4.50 dazu. Zeichnen Sie das Flussdiagramm und testen Sie es wieder zu zweit.

1.3 Variablen

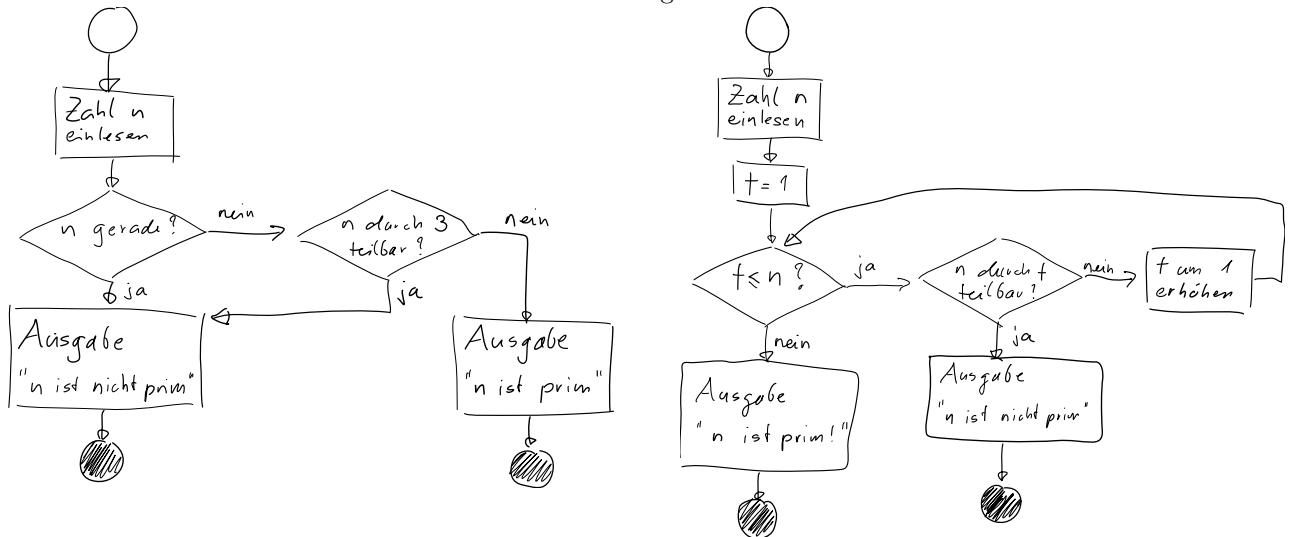
In obigen Programmen mussten diverse Dinge gespeichert werden, wie z. B. das erfragte Alter im Kino, das gewählte Getränk oder der bereits eingeworfene Betrag. Diese Größen wurden im Programm (bzw. Flussdiagramm) als Wort genannt (z. B. «Alter» oder «Betrag»).

Diese mit einem Wert verbundenen Namen nennen wir **Variablen**. Sie speichern Zahlen, Text oder beliebige komplexere Objekte (wie z.B. Koordinaten, Bilder, etc.).

Variablen, wie ihr Name schon impliziert, können sich während des Programmablaufs ändern (z.B. wird der eingeworfene Betrag nach dem Einwurf einer Münze nachgeführt).

☒ **Aufgabe A9**

- Die beiden folgenden Flussdiagramme sollen entscheiden, ob eine gegebene natürliche Zahl n eine Primzahl ist oder nicht. Leider sind sie fehlerhaft. Finden Sie jeweils den «ersten» Fehler, den das Programm macht. Für welche natürlichen Zahlen n «funktioniert» das Programm bereits korrekt?



- Entwerfen Sie selbst ein korrektes Flussdiagramm.

1.4 Programme als reiner Text

Computerprogramme werden meist als reiner Text geschrieben, wobei Leerschläge und Zeilenumbrüche die einzigen formatierenden Eingaben sind.


Merke 1.4.1 Reine Text-Dateien

Text-Dateien haben einige gewichtige Vorteile:

Menschenlesbar: Die Dateien können von Menschen gelesen, verstanden und verändert werden.

Maschinell lesbar: Problemlos mit verschiedensten Programmen les- und bearbeitbar (vorausgesetzt, die Dateien befolgen eine Struktur).

Ein Nachteil ist, dass Text-Dateien oft mehr Speicherplatz benötigen als z.B. eine binäre Codierung. Dieser Nachteil kann aber durch den Gebrauch von Kompressionsalgorithmen verringert werden.

Fun fact: Dieses Dokument wurde auch aus einer reinen Text-Datei generiert (mit Hilfe von L^AT_EX). Z.B. sieht die obige Box im Original wie folgt aus:

```
\begin{merke}[Reine Text-Dateien]
  Text-Dateien haben einige gewichtige Vorteile:
  \begin{description}
    \item[Menschenlesbar:] Die Dateien können von Menschen gelesen,
      verstanden und verändert werden.
    \item[Maschinell lesbar:] Problemlos mit verschiedenen Programmen les-
      und bearbeitbar.
  \end{description}
  Ein Nachteil ist, dass Text-Dateien oft mehr Speicherplatz benötigen als
  z.B. eine binäre Codierung. Dieser Nachteil kann aber durch den gebrauch von
  Kompressionsalgorithmen verringert werden.
\end{merke}
```

❖ **Aufgabe A10** Versuchen Sie, den Ablauf des einfachen Getränkeautomaten (nur ein Getränk) mit nur Buchstaben, Ziffern und einigen Sonderzeichen (wie Satzzeichen, Operationszeichen, Klammern aufzuschreiben). Als einzige Gestaltungselemente sind Leerschläge und Zeilenumbrüche erlaubt.

Welche Probleme treten dabei auf und wie lösen Sie diese?

1.5 Python-Konventionen

Merke 1.5.1 Python-Konventionen

- Aufeinander folgende Anweisungen werden gleich weit eingerückt untereinander geschrieben (d. h. gleich viele Leerschläge am Zeilenanfang).
- Verzweigungen werden wie folgt notiert (wobei der «Sonst»-Teil auch weggelassen werden kann). Man beachte jeweils den Doppelpunkt am Ende der Zeilen mit «Falls» und «Sonst».

```
Falls Bedingung:
  tu dies
  tu das
Sonst:
  tu jenes
  tu anderes
Tu das danach auf jeden Fall
```

- Wiederholungen werden wie folgt notiert (ebenfalls ein Doppelpunkt am Ende der Zeile mit «Wiederhole»). Die eingerückten Anweisungen werden solange ausgeführt, wie die Bedingung erfüllt ist.

```
Wiederhole solange Bedingung:
  tu dies
  tu noch etwas
  # hier wird zur Wiederholung zurückgesprungen, wird normalerweise nicht
  aufgeschrieben
Tu das nach den Wiederholungen nur einmal
```



☒ Aufgabe A11

- (a) Stellen Sie die folgenden beiden Programme als Flussdiagramme dar.

Programm 1

```
Lies eine natuerliche Zahl n ein.
Speichere 1 in (der Variablen) i
Speichere 0 in s
Wiederhole solange i kleiner gleich n:
    Berechne s+i und speichere das Resultat in s
    Erhoehe i um 1
Gib (den Wert von) s aus
```

Programm 2

```
Lies eine natuerliche Zahl n ein.
Wiederhole solange n ungleich 1:
    Falls n gerade:
        Berechne n/2 und speichere das Resultat in n
    Sonst:
        Berechne 3*n+1 und speichere das Resultat in n
Gib n aus
```

- (b) Was produzieren die Programme für verschiedene Eingaben der Zahl n ?

Zu erwähnen: Die obigen «Programme» sind in sogenanntem «Pseudocode» geschrieben. Pseudocode verwendet natürliche Sprache, ist aber schon recht nahe an echten Programmiersprachen (Code = Programm in einer Programmiersprache). Eventuell Begriff «Trace table» (an Tafel) erklären, danach einfachere Variante. Ein Computer führt genau so eine Tabelle mit den Variablenwerten (und Zeilennummern).

Merk 1.5.2 Python-Konventionen

- Zuweisungen von Variablen (d.h. einen Wert in einer Variablen speichern) werden mit **einem Gleichheitszeichen** wie folgt notiert:

```
a = 6
b = 7*a
c = -7
x = (-b + (b**2 - 4*a*c)**0.5)/(2*a)
```

- Anstatt «Falls» wird englisch **if** notiert. Aus «Sonst» wird **else**.
- Anstatt «Wiederhole solange» wird englisch **while** notiert.

```
i = 1
while i<10:
    print(i)
    i=i+1
print("Fertig!")
```

- Anstatt «Ausgabe» wird englisch **print(was)** geschrieben.
- Anstatt «n Einlesen» wird englisch **n=input()** geschrieben.
- Auf Gleichheit wird mit **doppeltem Gleichheitszeichen** geprüft:

```
if b==42:
    print("Die Antwort!")
else:
    print("Nicht korrekt")
```

- Kleiner, Kleiner gleich, Grösser gleich und grösser werden mit **<**, **<=**, **>=** und **>** notiert.
- Ungleich (**≠**) wird mit **!=** notiert.

```
if x!=0:
    print("x ist nicht Null")
```

☒ Aufgabe A12

Schreiben Sie die Programme aus Aufgabe A11 mit den neuen Python-Konventionen. Der Test, ob eine Zahl x gerade ist, geschieht per $x \% 2 == 0$, vgl. Merke 1.5.4.

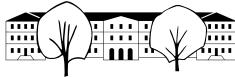
**Definition 1.5.3** Algorithmus

Ein **Algorithmus** ist ein Verfahren zur Lösung eines Problems. Üblicherweise verlangt man, dass ein Algorithmus die folgenden Eigenschaften hat:

- Finitheit (Endlichkeit): Das Verfahren ist als endliche Folge von genau definierten Anweisungen beschreibbar.
- Ausführbarkeit: Jeder Schritt des Verfahrens muss ausführbar sein (etwa von einem Computer).
- Terminierung: Das Verfahren endet nach endlich vielen Schritten.

Algorithmen nehmen in der Regel gewisse Eingaben entgegen und produzieren Ausgaben.

Algorithmen werden in der Regel durch Computerprogramme realisiert.



Aufgabe A13

- (a) Welche Ausgabe produziert das folgende Programm?

Hinweis: % berechnet den Rest einer Division und // den sogenannten «Ganzzahlquotient».

Beispiel: 53 % 10 ist 3 und 53 // 10 ist 5, denn «53 durch 10 ist 5 Rest 3» (siehe Merke 1.5.4).

```

a = 512
x = a % 10
b = a // 10
y = b % 10
z = b // 10
print(x)
print(y)
print(z)
print(100 * y + 10 * x + z)

```

- (b) Finden Sie heraus, was dieses Programm allgemein berechnet, wenn man in der ersten Zeile die Zahl 512 durch eine beliebige dreistellige Zahl ersetzt!

Aufgabe A14 Welche Ausgabe produziert das folgende Programm?

```
n = -1
s = 0
while n <= 10:
    print(n, s)
    n = n + 2
    s = s + n
```

Aufgabe A15

- (a) Spielen Sie Computer! Welche Ausgabe produziert das unten gegebene Programm für die Eingaben

- (i) $a = 31$ und $b = 10$; (ii) $a = 23$ und $b = 5$.

```
a = int(input("Gib eine natürliche Zahl a ein: "))
b = int(input("Gib eine natürliche Zahl b ein: "))
q = 0
while a >=b:
    a = a-b
    q = q+1
print(a)
print(q)
```

- (b) Finden Sie heraus, was dieses Programm allgemein berechnet!

Aufgabe A16

- (a) Spielen Sie Computer! Welche Ausgabe produziert das unten gegebene Programm für die Eingaben

- (i) $n = 24$; (ii) $n = 64$.

```
n = int(input("Gib eine natürliche Zahl a ein: "))
z = 1
while z*z < n:
    z = z+1
print(z*z)
```

- (b) Finden Sie heraus, was dieses Programm allgemein berechnet!



Aufgabe A17

- (a) Spielen Sie Computer! Welche Ausgabe produziert das unten gegebene Programm für die Eingaben

- (i) $z = 16$; (ii) $z = 15$; (iii) $z = 42$.

```
z = int(input("Gib eine natürliche Zahl ein: "))
while z>0:
    if z % 2 == 0:
        print(0)
        z = z/2
    else:
        print(1)
        z = (z-1)/2
```

- (b) Finden Sie heraus, was dieses Programm allgemein berechnet!



Programme ausführen

Merke 1.5.4 Divisionen in Python

In Python gibt es drei verschiedene Divisionszeichen:

- Normale Division

```
43 / 10
```

berechnet $\frac{43}{10} = 4.3$.

- Rest einer Division

```
43 % 10
```

berechnet den Rest der Ganzzahl-Division «43 durch 10», das Ergebnis ist 3.

- Division ohne Rest (Ganzzahlquotient)

```
43 // 10
```

berechnet, wie oft die 10 in die 43 hineinpasst, nämlich 4 Mal.

❖ Aufgabe A18

- (a) Spielen Sie Computer! Welche Ausgabe produziert das unten gegebene Programm für die Eingabe $x = 2004$, $y = 900$?

```
x = int(input("Gib eine natürliche Zahl x ein: "))
y = int(input("Gib eine natürliche Zahl y ein: "))

while y > 0:
    print("x =", x, "und y =", y)
    r = x % y
    x = y
    y = r
print("Das Ergebnis ist:", x)
```

- (b) Finden Sie heraus, was dieses Programm allgemein berechnet!

❖ Aufgabe A19

- (a) Spielen Sie Computer! Welche Ausgabe produziert das unten angegebene Programm für die Eingabe $x = 2$, $b = 8$.

```
a = int(input("Gib eine natürliche Zahl a ein (mindestens 2): "))
b = int(input("Gib eine grössere natürliche Zahl b ein: "))
x = a
while x < b:
    t = 2
    gefunden = False
    while t < x:
        if x % t == 0:
            gefunden = True
        t = t + 1
    if gefunden == False:
        print(x)
    x = x + 1
```

- (b) Finden Sie heraus, was dieses Programm allgemein berechnet!



Programme schreiben

❖ **Aufgabe A20** Schreiben Sie jeweils ein Programm in Python-Syntax, das

- eine Zahl n einliest und dann alle Teiler von n ausgibt.
- eine Zahl n einliest und dann die Summe aller Teiler (ausser n selbst) ausgibt.
- eine Zahl n einliest und dann alle Zahlen x zwischen 1 und n ausgibt, deren Teilersumme (wie in Aufgabe b) definiert) gleich der Zahl x ist.

*Ein Beispiel für eine solche Zahl ist 6. (Teilersumme $1 + 2 + 3 = 6$). Eine Zahl mit dieser Eigenschaft heisst **perfekt**.*

❖ **Aufgabe A21** Gegeben ist folgendes Programm:

```
a = int(input()) # Ganzzahl einlesen
b = int(input()) # Ganzzahl einlesen
print("a = ", a, " b = ", b)
# Hier fehlt Code
print("a = ", a, " b = ", b)
```

Ohne die `print`-Anweisungen zu verändern, ergänzen Sie den Code so, dass die Werte von a und b vertauscht werden.

Bonus 1: Lösen Sie die Aufgabe ohne zusätzliche Variablen und nur mit Additionen und Subtraktionen.

Bonus 2: Lösen Sie die Aufgabe mit 3 Variablen a , b und c die dann zyklisch vertauscht werden.

❖ **Aufgabe A22** Was macht das folgende Programm mit den Werten der Variablen a und b ? Wir nehmen an, es seien Zahlen in diesen Variablen gespeichert:

```
a = a-b
b = a+b
a = b-a
```

Merke 1.5.5 Mehrfachzuweisung in Python

In Python können mehrere Variablen gleichzeitig zugewiesen werden, z.B.

```
x,y = 23, 24 # Achtung, nicht mit Dezimalpunkt verwechseln!
```

Damit können die Werte von Variablen sehr einfach vertauscht werden:

```
a,b = b,a # Vertauschen Variante lazy
```

❖ **Aufgabe A23** Testen Sie Ihre Python-Programme von den Aufgaben A20, A21 und A22 auf Ihrem Computer.

Die Anleitung zur Installation der nötigen Applikation finden Sie online.

❖ **Aufgabe A24** 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 `jahre=35` definiert und dann für jedes Jahr den Kontostand am 31. Dezember ausgibt.

❖ **Aufgabe A25** 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 = $\frac{\text{links}+\text{rechts}}{2}$` und vergleicht sein Quadrat `mittel2` 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 A26 Die Zahlenfolge

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

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

Schreiben Sie ein Programm, 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 A27 Folgender Python-Code erzeugt Zufallszahlen zwischen 0 und 1:

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

```
0.8696981969050243
0.7626787554309876
0.6196171861591572
0.17116177170210767
0.3170744431396274
0.49269143898456613
```

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 berechnen? Wie genau ist das Resultat? Hinweis: Die Kreiszahl π kann man wie folgt ausgeben:

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

Testen Sie Ihr Programm auf einem Computer!



❖ **Aufgabe A28** 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
```

Verzweigungen = if-Statements

Oft werden manche Teile eines Programms nur unter bestimmten Bedingungen ausgeführt. Hier einige Beispiele. Man beachte die Einrückungen.

```
if b==42:
    print("Die Antwort!")
```

```
if b==42:
    print("Die Antwort!")
else:
    print("Nicht korrekt")
```

Man kann if-Statements auch verschachteln.

```
if b % 4 == 0:
    print("Die Zahl b ist durch 4 teilbar.")
else:
    if b % 2 == 0:
        print("Die Zahl b ist durch 2 teilbar, aber nicht durch 4.")
    else:
        print("Die Zahl b ist nicht durch 2 teilbar.")
```

Das folgenden Programme macht genau dasselbe wie das gerade betrachtete Programm: Das *Schlüsselwort elif* ist eine Abkürzung für «else if».

```
if b % 4 == 0:
    print("Die Zahl b ist durch 4 teilbar.")
elif b % 2 == 0:
    print("Die Zahl b ist durch 2 teilbar, aber nicht durch 4.")
else:
    print("Die Zahl b ist nicht durch 2 teilbar.")
```

❖ Aufgabe A29

```
minuten = int(input("Wie viele Minuten Tageslicht hattest du heute bereits? "))
#
# Hier ist Code
# zu ergänzen.
#
```

Erweitern Sie das obige Programm wie folgt:

Falls der Benutzer 120 oder mehr antwortet, soll er gelobt werden. Sonst wird ihm gesagt, wie viele Minuten er noch draussen verbringen sollte, damit er mindestens 2 Stunden draussen ist.

Beispiel: Der Benutzer gibt ein, dass er 35 Minuten Tageslich hatte. Dann soll der Computer Folgendes ausgeben:

```
Bitte gehe noch 85 Minuten nach draussen.
```

Testen Sie Ihr Programm mit Eingaben wie 30, 170, -10, 2400.



Bonus: Wenn der Benutzer etwas Unsinniges eingibt, etwa eine negative Zahl oder eine Zahl, die grösser als $24 \cdot 60$ ist: Melden Sie dies.

Hintergrundinformation: Zwei Stunden Tageslicht pro Tag minimieren laut wissenschaftlicher Studien das Risiko, kurzstichtig zu werden, siehe Deutschlandfunk Kultur: <https://www.deutschlandfunkkultur.de/sehstoerung-des-auges-immer-mehr-kinder-werden-kurzsichtig-100.html>

Aufgabe A30 Ergänzen Sie das folgende Programm so, dass es alle Lösungen der folgenden linearen Gleichung ausgibt.

$$ax + b = 0$$

Das Programm soll auch im Fall $a = 0$ funktionieren und dann ausgeben, ob jede reelle Zahl eine Lösung ist oder ob die Gleichung keine Lösung hat.

```
print("Ich kann jede lineare Gleichung ax+b=0 lösen.")
a = int(input("Gib eine Zahl a ein: "))
b = int(input("Gib eine Zahl b ein: "))
print(f'Die Lösungen der Gleichung {a}x+{b}=0 sind ...')
#
# Hier ist Code
# zu ergänzen.
#
```

Aufgabe A31 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:

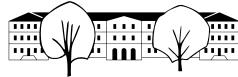
```
Ich habe mir eine natürliche Zahl zwischen 0 und 100 ausgedacht. Welche Zahl ist es?
Dein Tipp: 21
Zu klein.
Dein Tipp: 84
Zu gross.
Dein Tipp: 42
Korrekt. Herzlichen Glückwunsch!
Benötigte Versuche: 3
```

Testen Sie Ihr Programm auf einem Computer!

Aufgabe A32 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 (Diskriminante $D = b^2 - 4ac$ verwenden); andererseits soll es alle Lösungen ausgeben.
- (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 (vgl. vorige Aufgabe).

```
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.
#
```

**☒ Aufgabe A33** Die Collatz-Folge ist wie folgt definiert:(Schritt 1) Starte mit einer beliebigen natürlichen Zahl $n > 0$.(Schritt 2) Wenn n gerade ist: Ersetze n durch $\frac{n}{2}$.Sonst: Ersetze n durch $3n + 1$.(Schritt 3) Wiederhole den vorherigen Schritt, solange $n > 1$ gilt.

Beispiel: Die mit 13 startende Collatz-Folge sieht so aus: 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Passen Sie das folgende Programm so an, dass es ausgehend von einer Startzahl die Collatz-Folge ausgibt.

```
n = int(input("Gib eine natürliche Zahl > 0 ein: "))
while n>1:
    print(n)
    #
    # Hier ist Code
    # zu ergänzen.
    #
print(n)
```



1.6 for-Schleife

Merke 1.6.1 for-Schleife

Auch wenn `while` für die Wiederholung ausreichend ist, verwendet man oft eine `for`-Schleife, wenn die Anzahl Wiederholungen bekannt ist. Folgende Programme tun (fast) das Gleiche:

```
for i in range(5):
    print(i)
i = 0
while i<5:
    print(i)
    i = i+1
```

Die Variable *i* nennt man **Laufvariable**.

Der Unterschied zwischen beiden Varianten ist der Wert der Laufvariablen *i* nach der Wiederholung. Bei der for-Schleife hat *i* den Wert vom letzten Durchlauf, bei while eins mehr.

Python ist eine Ausnahme, wo die Gültigkeit der Laufvariablen nicht auf die For-Schleife begrenzt ist.

1.7 Arrays

Arrays sind Listen mit mehreren Werten, die so in einer Variable gespeichert werden können. Die einzelnen Werte (bzw. Einträge) können über ihren **Index** angesprochen werden, wobei von 0 weg indiziert wird.

```
a = [23, 17, 42, 99]      # direkte Definition
print(a[2], " ist die Antwort!")  # a[2] ist das dritte Element, also 42
a[1] = 999    # Das zweite Element ist jetzt 999
print("a hat ", len(a), " Elemente")  # 4 Elemente
a.append(1234)  # Neues Element hinten anfügen
print(a)    # Liefert [23, 999, 42, 99, 1234]
print(len(a))  # Liefert jetzt 5
```

❖ **Aufgabe A34** Was enthält das Array *a* jeweils bei Programmende?

```
a = []    # Leeres Array ohne Elemente
i = 5
while i<10:
    a.append(1+i*i)
    i = i+1
```

```
a = [3]
while a[len(a)-1]!=1:
    if a[len(a)-1]%2 == 0:
        a.append(a[len(a)-1]/2)
    else:
        a.append(a[len(a)-1]*3+1)
```

❖ **Aufgabe A35** Finden Sie jeweils das Lösungswort, das im Array *a* codiert ist.

```
a = [1,2,3,4]
a[1] = a[a[2]] * a[1]
a[0] = a[2]+10*a[0]
a[2] = a[2]-1
```

```
a = []
i = 0
while i<6:
    a.append(i)
    i = i+1
a[2] = a[2]**2
a.append(a[2])
a[2+1] = 3*a[1]
a[3] = a[5]*i//4
a[1] = (a[5]-1)*2
a[4] = a[i] + a[1] % a[i-1]
print(a)
```

```
a = [11]
a.append(a[0] % 7)
a.append(a[0])
a.append(a[0])
i = 0
while i<4:
    a.append(3+i*5)
    i = i+1
a[i+1] = a[len(a)-2]+1
a[len(a)-1] = a[1]
i = i % 2
```



```
a[i] = a[i]*2  
print(a)
```



1.8 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-schuelerroboter

☒ Lösung zu A2 ex-grid-laby-solver

Rechter Wand folgen, ohne Wegmarkierung:

Solange nicht am Ziel, wiederhole:

 Wenn rechts keine Wand:

 Drehe nach rechts

 Vorwärts

 Sonst:

 Wenn vorne keine Wand:

 Vorwärts

 Sonst:

 Drehe nach links

Programmende

Dieses Programm funktioniert bei beliebigem Startpunkt nur zuverlässig, wenn das Labyrinth keine Zyklen (Rundwege) hat. Bei unseren Beispillabyrinthen mit Startfeld links oben und Zielfeld rechts unten funktioniert es immer.

Ein Schüler hat den folgenden, äquivalenten Lösungsalgorithmus gefunden: Es ist eine gute Übung, sich zu überlegen, warum dieser Algorithmus «genau dasselbe» macht wie der oben angegebene.

Wiederhole, bis Ziel erreicht:

 Wenn keine Wand rechts:

 Drehe nach rechts

 Wenn Wand vorne:

 Drehe nach links

 Sonst:

 Gehe vorwärts

Programmende

Rückweg markieren (Tiefensuche)

Markiere Position mit 'S' (für Start)

Solange nicht am Ziel, wiederhole:

 Wenn es eine nicht markierte Nachbarposition gibt:

 Wähle eine solche Nachbarposition

 Markiere diese Nachbarposition mit einem Pfeil zur aktuellen Position.

 Gehe auf diese Nachbarposition

 Sonst:

 Wenn das Feld mit 'S' markiert ist:

 Melde «Keine Lösung»

 Programm Ende

 Sonst

 Gehe zum Nachbarfeld in Richtung des Pfeils



Programmende

Dieses Programm funktioniert auch zuverlässig, wenn das Labyrinth Zyklen aufweist.

Lässt man dieses Programm auf einem Labyrinth ohne jegliche Mauern (= einem leeren Rechteck) laufen (und hört erst auf, wenn alle Felder markiert sind), generiert dieser Algorithmus ein Labyrinth. Dies wird in einer Folgeaufgabe thematisiert.

✖ Lösung zu A3 ex-grid-laby-global

Programm Sackgassen füllen

Wiederhole:

 Merke dir, dass es noch keine Änderung gab.

 Gehe ein Feld nach dem anderen durch (ausser Start- und Zielfeld; etwa zeilenweise):

 Wenn das aktuelle Feld unmarkiert ist und von ihm genau ein unmarkiertes Nachbarfeld erreichbar ist:

 Markiere das aktuelle Feld mit 'X'

 Merke dir, dass es eine Änderung gab.

 Wenn es keine Änderung gab:

 Brich die Wiederholung ab

Programmende

Alte Version, die mir zu umständlich erscheint::

Wiederhole:

 Merke dir, dass es noch keine Änderung gab.

 Gehe ein Feld nach dem anderen durch (ausser Start- und Zielfeld; etwa zeilenweise):

 Wenn das aktuelle Feld unmarkiert ist:

 Wenn das aktuelle Feld genau ein unmarkiertes Nachbarfeld hat:

 Wenn die restlichen drei benachbarten Felder versperrt oder mit 'X' markiert sind:

 Markiere das aktuelle Feld mit 'X'

 Merke dir, dass es eine Änderung gab.

 Wenn es keine Änderung gab:

 Brich die Wiederholung ab

Programmende

Dieses Programm funktioniert im Folgenden Sinne nicht zuverlässig, wenn das Labyrinth einen Zyklus enthält: Man erhält dann alle möglichen Wege gleichzeitig (Felder ohne Markierung).

Damit es auch mit Zyklen funktioniert, müssen bei übrigbleibenden Kreuzungen jeweils alle bis auf einen Weg gesperrt werden, und dann wieder Sackgassen gefüllt werden:

Programm Weg markieren

Führe das Programm "Sackgassen füllen" aus.

Markiere das Start- und Zielfeld mit 'w' (Weg)

Wiederhole:

 Merke dir, dass es noch keine Änderung gab.

 Gehe ein Feld nach dem anderen durch:

 Wenn das Feld mit 'w' markiert ist:

 Wenn genau ein Nachbarfeld nicht markiert ist:

 Markiere dieses Nachbarfeld mit 'w'

 Merke dir, dass es eine Änderung gab.

 Wenn mehr als ein Nachbarfeld nicht markiert ist:

 Markiere eines davon mit 'w'

 Markiere die anderen mit 'X'

 Führe das Programm "Sackgassen füllen" aus.

 Merke dir, dass es eine Änderung gab.

 Wenn es keine Änderung gab:

 Brich die Wiederholung ab



Programmende

Programm kürzesten Weg finden (Breitensuche)

Markiere das Startfeld mit der Zahl 0

Merke dir die Zahl 0 in der Variablen DISTANZ

Wiederhole:

Wenn es Felder gibt, die mit (dem aktuellen Wert von) DISTANZ markiert sind:

Gehe alle Felder durch, die mit DISTANZ markiert sind:

Vom aktuellen Feld aus markiere alle nicht markierten Nachbarfelder mit (DISTANZ+1)

Berechne (DISTANZ+1) und merke dir den Wert in DISTANZ (d. h. erhöhe DISTANZ um 1)

Sonst:

Brich die Wiederholung ab

Programmende

✖ Lösung zu A4 ex-grid-laby-generation

✖ Lösung zu A5 ex-steuerung-oder-regelung

- a) Liftposition: Gesteuert, z.B. über mechanische Schalter auf der Liftschiene.
- b) Lifttüren: Geregelt, die Türen sollen nicht schliessen, wenn etwas im Weg ist.
- c) Heizung: Geregelt (Thermostat).
- d) Herdplatte: Gesteuert (normalerweise wird einfach eine bestimmte Heizleistung eingestellt).
- e) Kühlschrank: Geregelt (Thermostat).
- f) Wohnzimmerbeleuchtung: Meistens gesteuert, über einfache Lichtschalter. Kann heute aber auch geregelt sein (Bewegungsmelder).

✖ Lösung zu A6 ex-kino-einlass

✖ Lösung zu A7 ex-kino-drei-filme

✖ Lösung zu A8 ex-getraenke-automat

✖ Lösung zu A9 ex-logische-fehler-in-flussdiagrammen

✖ Lösung zu A10 ex-getraenkeautomat-als-text

✖ Lösung zu A11 ex-pseudocode-zu-flussdiagramm

✖ Lösung zu A12 ex-pseudocode-pythonisieren

Programm 1

```
Lies eine natürliche Zahl n ein.
i=1
s=0
while i<=n:
    s=s+i
    i=i+1
print(s)
```

Programm 2



```
Lies eine natuerliche Zahl n ein.
while n!=1:
    if n gerade:
        n = n/2
    else:
        n = 3*n+1
    print(n)
```

✖ Lösung zu A13 ex-ziffern-umordnen

✖ Lösung zu A14 ex-ungerade-und-quadratzahlen-ausgeben

✖ Lösung zu A15 ex-ganzzahlige-division

✖ Lösung zu A16 ex-groesste-quadratzahl-groesser-als-zahl

✖ Lösung zu A17 ex-zahl-binaer-ausgeben

- (a) •
- ```
x = 12 und y = 18
x = 18 und y = 12
x = 12 und y = 6
Das Ergebnis ist: 6
```
- ```
x = 12 und y = 21
x = 21 und y = 12
x = 12 und y = 9
x = 9 und y = 3
Das Ergebnis ist: 3
```
- ```
x = 13 und y = 21
x = 21 und y = 13
x = 13 und y = 8
x = 8 und y = 5
x = 5 und y = 3
x = 3 und y = 2
x = 2 und y = 1
Das Ergebnis ist: 1
```

(b) Das Programm berechnet den ggT( $x, y$ ) (= grösster gemeinsamer Teiler) von  $x$  und  $y$ .

Dies ist der sogenannte **Euklidische Algorithmus**.

✖ Lösung zu A19 ex-programm-verstehen-pl

- (a) Flussdiagramm
- (b) Ausgabe für  $a = 2, b = 8$ :

```
2
3
5
7
```

(c) Das Programm gibt alle Primzahlen zwischen  $a$  und  $b - 1$  aus.


**✖ Lösung zu A20** ex-teilerliste-summe-perfekte-zahlen-in-python

```
n = input()
teiler = 1
while teiler <= n:
 if n % teiler == 0:
 print(teiler)
 teiler = teiler+1
```

```
n = input()
summe = 0
teiler = 1
while teiler <= n/2:
 if n % teiler == 0:
 summe = summe + teiler
print(summe)
```

```
n=input()
x=1
while x <= n:
 summe = 0
 teiler = 1
 while teiler <= n/2:
 if n % teiler == 0:
 summe = summe + teiler
 if summe == x:
 print(x)
 x = x+1
```

**✖ Lösung zu A21** ex-swap-and-cycle

Vertauschen:

```
h = a # Wert von a sichern
a = b # a überschreiben
b = h # gesicherter Wert in b
```

Bonus 1:

```
a = a-b
b = a+b
a = b-a
```

oder

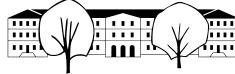
```
a = b-a
b = b-a
a = a+b
```

**✖ Lösung zu A22** ex-vertauschen-mit-subtraktion-verstehen

**✖ Lösung zu A23** ex-teiler-programme-testen

**✖ Lösung zu A24** ex-renten-sparen

```
betrag = 6000
zins = 0.01
jahre = 35
n = 1
kontostand = 0
while n<=jahre:
 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 : 148297.811009095812
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 A25 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 A26 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 A27 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 781 von 1000
Anteil 0.781
Pi ist ungefähr 3.124
```

### \* Lösung zu A28 ex-schulden-tilgen

### \* Lösung zu A29 ex-tageslicht

### \* Lösung zu A30 ex-lineare-gleichung

### \* Lösung zu A31 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)
```

### \* Lösung zu A32 ex-quadratische-gleichung



## ❖ Lösung zu A33 ex-collatz

```
n = int(input("Gib eine natürliche Zahl > 0 ein: "))
while n>1:
 print(n)
 if n gerade:
 n = n/2
 else:
 n = 3*n+1
print(n)
```

## ❖ Lösung zu A34 ex-array-fuellen-verstehen

## ❖ Lösung zu A35 ex-array-raetsel-code