Die **range**-Anweisung definiert einen Bereich ganzer Zahlen.

range(10)	definiert den Bereich 0, 1, . . . , 9
range(4,21)	definiert den Bereich 4, 5, . . . , 20
range(4,21,3)	definiert den Bereich 4, 7, 10, . . . , 16, 19
range(-4,3)	definiert den Bereich -4, -3, -2, -1, 0, 1, 2

Allgemein gilt:

range(start, stop)
definiert den Bereich **start**, ... , **stop-1** ganzer Zahlen,

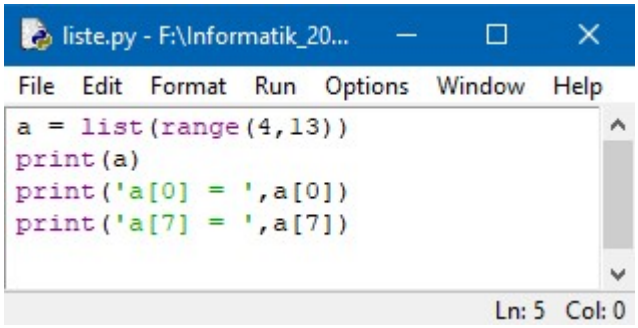
range(start, stop, step)
definiert den Bereich **start**, . . . mit der Schrittweite **step**, wobei die Zahl **stop** nicht mehr enthalten ist.

Erstellen einer Liste ganzer Zahlen

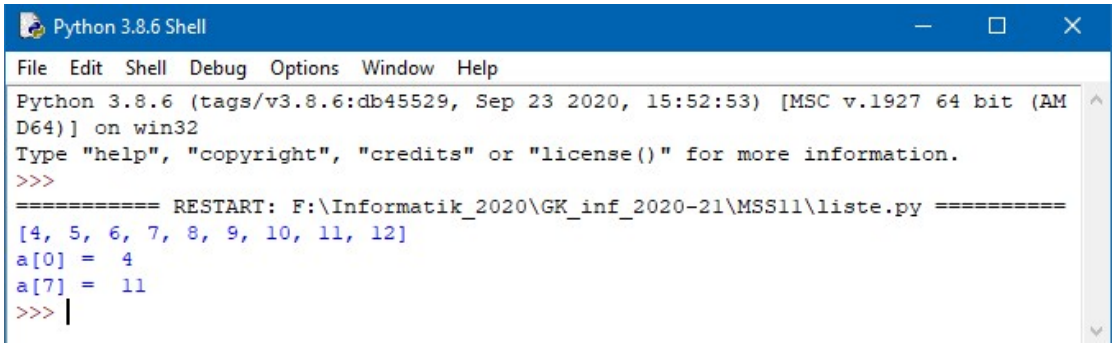
a = list(range(4,13)) erzeugt die Liste
[4, 5, 6, 7, 8, 9, 10, 11, 12];
die (in diesem Fall 9) Elemente dieser Liste heißen auch Komponenten, auf die man mit **a[0], a[1], . . . , a[8]** zugreifen kann.

Bemerkung: *Unter einem **Feld** oder einem **array** verstehen wir eine Folge von Variablen gleichen Typs; mit vorstehendem Beispiel haben wir also ein array **a** ganzer Zahlen erzeugt mit den Komponenten a[0], a[1], . . . , a[8].*

Python-Quelltext:



Ausgabe:



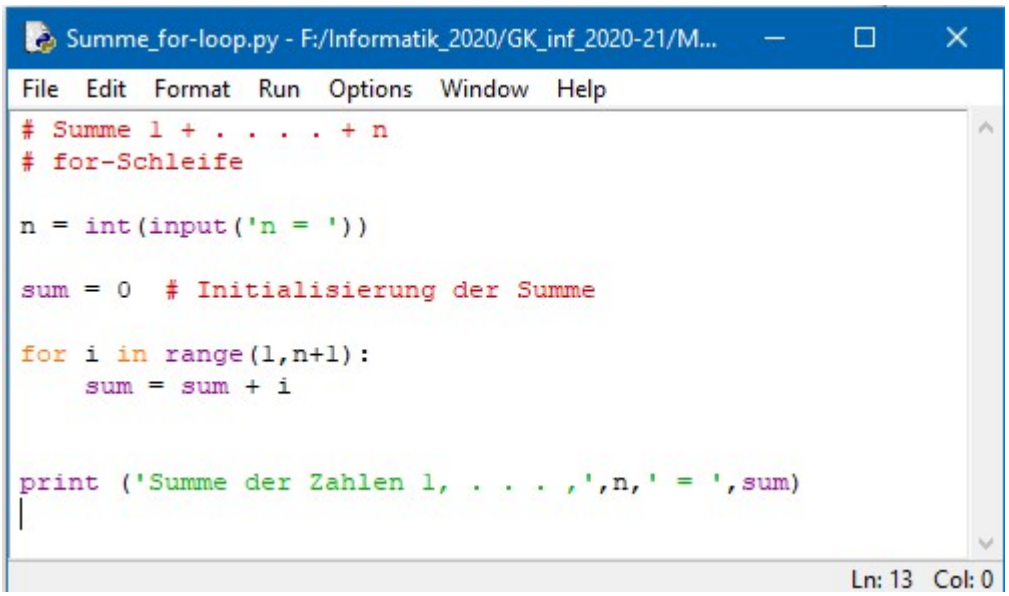
For-Schleife

Das Python-Programm

```
n = int(input('n = '))

for i in range(1,n):
    print(i)
```

gibt nach Eingabe der natürlichen Zahl n die Zahlen 1, 2, . . . , n-1 aus; probiert es aus!

Algorithmus zur Bestimmung der Summe der Zahlen 1, ... , n mit Verwendung einer for-Schleife:



Arbeitsaufträge:

1. Schreibe und teste ein Python-Programm, um die ersten 20 Zahlen der Siebener-Reihe auszugeben (also die Zahlen 7, 14, 21, . . .).

2. Erstelle gemäß dem vorstehenden screenshot den Python-Programmtext zur Berechnung der Summe 1 + . . . + n und teste das Programm mit unterschiedlichen Eingaben.

3. Formuliere und teste ein Python-Programm, welches nach Eingabe der natürlichen Zahl k die Summe der ersten k ungeraden natürlichen Zahlen bestimmt, und zwar mit Verwendung einer for-Schleife.

4. Formuliere und teste ein Python-Programm, welches nach Eingabe der natürlichen Zahl n das Produkt der Zahlen 1, . . . , n zu berechnet (for-Schleife!).

Informatik 11
14.01.2021

Wir kennen bereits die numerischen Datentypen **float** und **integer**.

Beispiele:

```
>>> print(11 / 6)         Quotient zweier ganzer Zahlen
1.8333333333333333

>>> print(2 ** 0.5)       Wurzel aus 2
1.4142135623730951

>>> print(27 / 4)         Quotient zweier ganzer Zahlen
6.75

>>> print(27 // 4)        ganzzahliger Quotient (27 : 4 = 6 Rest 3)
6

>>> print(27 % 4)         Rest bei ganzzahliger Division
3

>>> print(7 * 12)         Produkt ganzer Zahlen
84

>>> print(0.8 * (-7.5))   Produkt zweier Kommazahlen
-6.0
```

Datentyp **boolean**

Eine Boolesche Variable oder ein Boolescher Ausdruck (Term) nimmt nur zwei Werte an:
`True` oder **`False`**
(oder abkürzend: 1 oder 0; in Python sind **`True`** oder **`False`** zu verwenden)

Insbesondere sind folgende Terme Boolesche Ausdrücke, deren Wert sich auch einer Variablen zuweisen läßt:

8 > 5 hat den Wert **`True`**
7 == 8 hat den Wert **`False`**
7 != 8 hat den Wert **`True`**
x hat den Wert **`True`** nach der Wertzuweisung x = 7 < 12
x hat den Wert **`False`** nach der Wertzuweisung x = (0 == 6)

Wir definieren die Verknüpfungen **`and`** und **`or`** sowie die Operation **`not`** jeweils über eine Wahrheitstafel:

a	b	a **or** b
False	**False**	**False**
False	**True**	**True**
True	**False**	**True**
True	**True**	**True**

a	b	a **and** b
False	**False**	**False**
False	**True**	**False**
True	**False**	**False**
True	**True**	**True**

a	**not** a
False	**True**
True	**False**

Aufgabenblatt Nr. 3 inf11 18.01.2021

Hinweis:
Wiederholungen können wahlweise als while- oder for-Schleife formuliert werden.

7. Der Algorithmus **SCHALTJAHR** verlangt als Eingabe eine Jahreszahl und gibt aus, ob das eingegebene Jahr ein Schaltjahr ist.
Regeln:

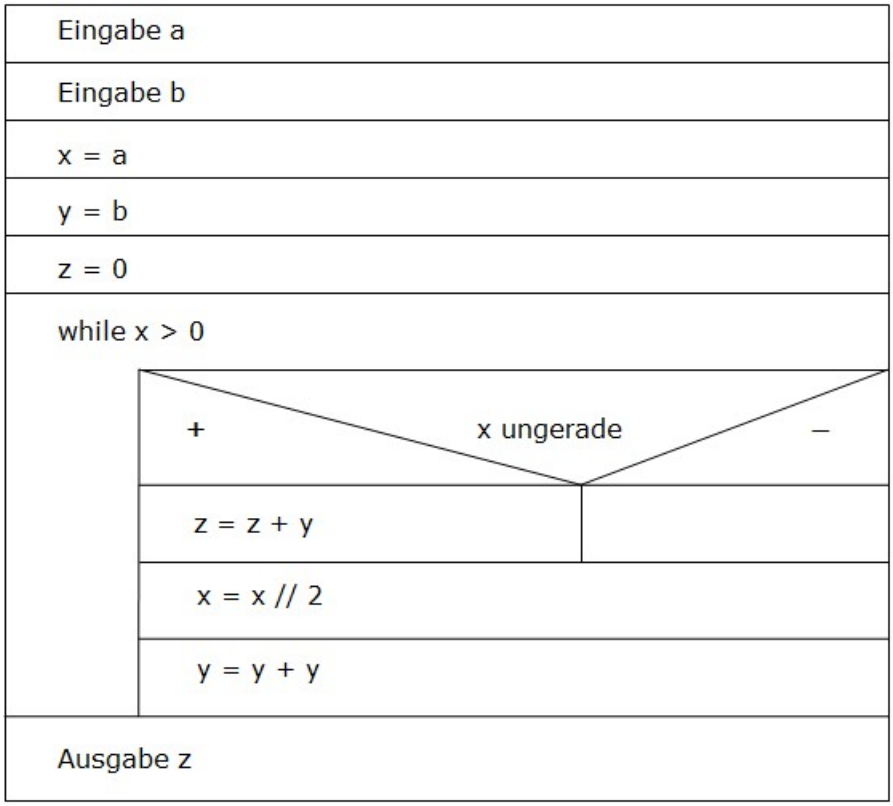
Jahreszahl	
- nicht durch 4 teilbar:	kein Schaltjahr
- durch 4 teilbar:	Schaltjahr
- durch 100 teilbar:	kein Schaltjahr
- durch 400 teilbar:	Schaltjahr

Formuliere den Algorithmus als Struktogramm und als Python-Programm.

8. Erstelle ein Python-Programm, welches nach Eingabe einer natürlichen Zahl n die Summe sum mit

sum = 1 + 1/2 + 1/3 + 1/4 + ... + 1/n berechnet, und teste das Programm für unterschiedliche Eingaben.

9. Ein Algorithmus, der die natürlichen Zahlen a und b als Eingabe verlangt und als Ergebnis die Zahl z ausgibt, ist durch folgendes Struktogramm gegeben:



Schreibe diesen Algorithmus als Python-Programm und teste ihn mit unterschiedlichen Eingaben; was bewirkt der Algorithmus vermutlich?

Hinweise: *Unter x // 2 verstehen wir den ganzzahligen Quotient bei der Division von x durch 2.*

Eine Zahl x ist genau dann gerade, wenn sie durch 2 ohne Rest teilbar ist, d. h. wenn gilt: x % 2 = 0.

10. Der Algorithmus **PRIMZAHLTEST**

Definition: Eine natürliche Zahl n heißt Primzahl genau dann, wenn sie nur durch 1 und durch sich selbst jeweils ohne Rest teilbar ist.

Aufgabe: Konzipiere einen Algorithmus (als Struktogramm und als Python-Programm), der nach Eingabe einer natürlichen Zahl n entscheidet, ob n die Primzahleigenschaft hat.

Hinweis: Teste für alle in Frage kommenden Teiler (Divisoren) t, ob n % t gleich 0 ist.

11. **BOOLESCHE VARIABLE** oder **BOOLESCHE TERME** können nur zwei Werte annehmen: **`True`** oder **`False`**.
Die Verknüpfungen **`and`** und **`or`** sowie die Operation **`not`** werden bekanntlich jeweils über eine Wahrheitstafel definiert.

Wir verwenden folgende abkürzende Schreibweisen (a, b, c sind Boolesche Variable):

`a and b = a · b = a b`
`a or b = a + b`
`not a = ¬ a`

Dabei gelte auch die aus der Algebra bekannte Vereinbarung: "Punkt vor Strich", d. h.
`a + (b · c) = a + b · c = a + b c`

Eine Auswahl von Rechenregeln für Boolesche Variable:

Kommutativgesetze

`(1) a + b = b + a` **`(1') a · b = b · a`**

Assoziativgesetze

`(2) a + (b + c) = (a + b) + c` **`(2') a · (b · c) = (a · b) · c`**

Distributivgesetze

`(3) a · (b + c) = a · b + a · c` **`(3') a + b · c = (a + b)·(a + c)`**

Beweis von (3):

a	b	c	b + c	a(b + c)	ab	ac	ab + ac
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Da die Spalten zu **`a(b + c)`** und **`ab + ac`** übereinstimmen, gilt: **`a(b + c) = ab + ac`** .

Aufgaben: a) Beweise die Rechengesetze (2) und (3').

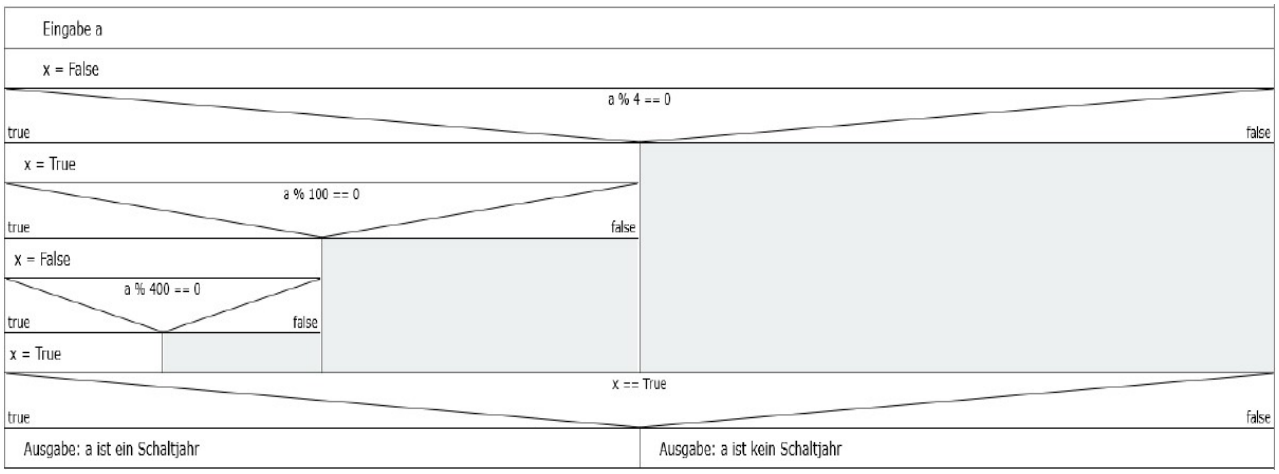
b) Zeige: $(a \cdot b) + c \neq a \cdot (b + c)$

Informatik 11 10.02.2021

Lösungen zu den Aufgabenblättern **Nr. 3** (18.01.2021) und **Nr. 4** (26.01.2021)

Aufgabe 7 (Algorithmus Schaltjahrbestimmung)

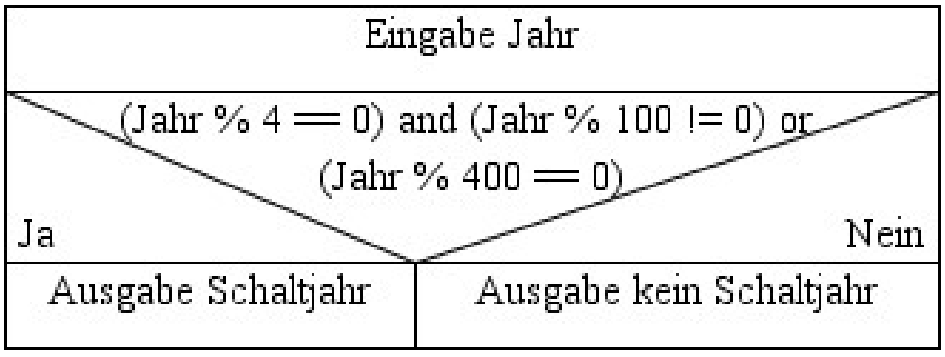
Struktogramm I (Christian):



Python-Quelltext I:

```
# Schaltjahrbestimmung

a = int(input('Jahreszahl = '))
x = False

if a % 4 == 0:
    x = True
    if a % 100 == 0:
        x = False
        if a % 400 == 0:
            x = True

if x == True:
    print(a,' ist Schaltjahr')
else:
    print(a,' ist kein Schaltjahr')
```

Struktogramm II (Marvin):



Python-Quelltext II:

```
# Schaltjahresrechner

jahr = int(input('Geben Sie das Jahr an: '))

if (jahr % 4 == 0) and (jahr % 100 != 0) or (jahr % 400 == 0):
    print(jahr, 'ist ein Schaltjahr')
else:
    print(jahr, 'ist kein Schaltjahr')
```

Hinweis:
Zu Version II beachte unbedingt die Bemerkung innerhalb der Lösung zu Aufgabe 11.b)!

Aufgabe 8 (Harmonische Reihe)

Python-Quelltext (Max):

```
# KW 3 Aufgabe 8
# Eingabe

n = int(input('Geben Sie eine natürlichen Zahl ein: '))

# Berechnung

i = 1
sum = 0

while (i <= n):
    sum = sum + (1/i)
    i = i + 1

# Ausgabe

print('1  + . . . .  +  1 /',n,'  = ', sum)

# Max Yurchak
```

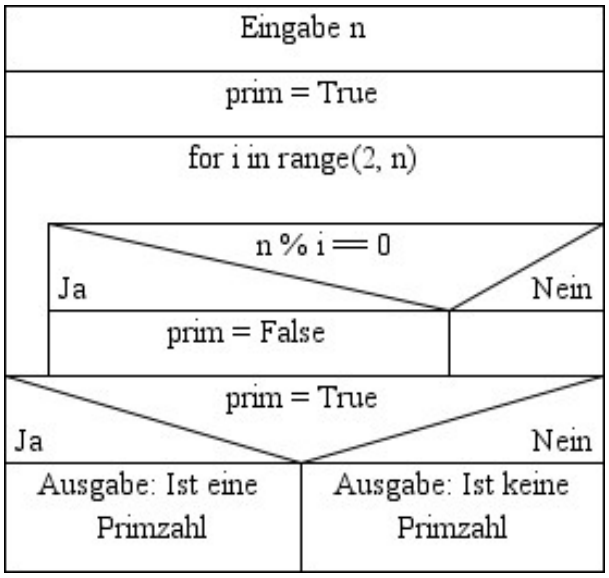
Aufgabe 9 (Algorithmus zur Multiplikation zweier natürlicher Zahlen a und b)

Python-Quelltext (Marvin):

```
a = int(input('a = '))
b = int(input('b = '))
x = a
y = b
z = 0
while x > 0:
    if x % 2 != 0:
        z = z + y
    x = x // 2
    y = y + y
print('z = ', z)
```

Aufgabe 10 (Algorithmus Primzahltest)

Struktogramm I (Marvin):



Python-Quelltext I (Laura):

```
# Laura primzahltest
n = int(input('n = '))

prim = True

if n > 2:
    for i in range(2,n):
        if n % i == 0:
            prim = False

if prim == True:
    print('primzahl')
else:
    print('keine primzahl')
```

Anmerkung: Statt **`if prim == True:`** schreibe kürzer: **`if prim:`**

Python-Quelltext II

Die Laufzeit des Algorithmus können wir optimieren (insbesondere bei großen Zahlen n), indem die zu prüfende Zahl n nur durch solche Divisoren i ganzzahlig geteilt wird, deren Quadrat nicht größer als n ist (die also nicht größer sind als die Wurzel aus n).

Außerdem wird berücksichtigt, daß die Zahl 1 definitionsgemäß keine Primzahl ist.

```
n = int(input('n = '))

if n == 1:
    prim = False
else:
    prim = True
    i = 2

    while i * i < n + 1:
        if n % i == 0:
            prim = False
        i = i + 1

if prim:
    print(n,' ist eine Primzahl')
else:
    print(n,' ist keine Primzahl')
```

Aufgabe 11 (Boolesche Variable und Boolesche Terme)
Lösungen zu Teil a) (Rechengesetz (3'): Christian) und Teil b)

Lösung zu Aufgabe 11.a) von Aufgabenblatt Nr. 4

3'

a	b	c	b * c	a + b * c	(a + b)	(a + c)	(a + b)(a + c)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Da die Spalten zu a + b * c und (a + b)(a + c) übereinstimmen, gilt: a + b * c = (a + b)(a + c)

Lösung zu Aufgabe 11.b) von Aufgabenblatt Nr. 4

a	b	c	a and b	(a and b) or c	b or c	a and (b or c)
0	0	0	0	0	0	0
0	0	1	0	1	1	0
0	1	0	0	0	1	0
0	1	1	0	1	1	0
1	0	0	0	0	0	0
1	0	1	0	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

(a and b) or c ≠ a and (b or c)

a and b or c ≠ a and (b or c)

a · b + c ≠ a · (b + c)

Bemerkung zum Algorithmus „Schaltjahr“ (Nr. 7 Blatt 3):

j%4==0 **a**	j%100!=0 **b**	j%400==0 **c**	a and b	(a and b) or c	b or c	a and (b or c)	j teilbar
0	*0*	*0*	*0*	*0*	*0*	*0*	*nicht durch 4 durch 100 nicht durch 400*
0	*0*	*1*	*0*	*1*	*1*	*0*	*nicht durch 4 durch 100 durch 400*
0	1	0	0	0	1	0	*nicht durch 4 nicht durch 100 nicht durch 400*
0	*1*	*1*	*0*	*1*	*1*	*0*	*nicht durch 4 nicht durch 100 durch 400*
1	0	0	0	0	0	0	*durch 4 durch 100 nicht durch 400*
1	0	1	0	1	1	1	*durch 4 durch 100 durch 400*
1	1	0	1	1	1	1	*durch 4 nicht durch 100 nicht durch 400*
1	*1*	*1*	*1*	*1*	*1*	*1*	*durch 4 nicht durch 100 durch 400*

Das Jahr **j** ist Schaltjahr genau dann, wenn gilt:

[(j%4==0 and j%100!=0) or j%400==0] == True

Da nur die nicht kursiv geschriebenen Zeilen in Frage kommen können, gilt hier:

(j%4==0 and j%100!=0) or j%400==0 = j%4==0 and (j%100!=0 or j%400==0)

12. Die Datenstruktur „array" (siehe auch Arbeitsblatt vom 06.01.2021; unter einem array verstehen wir eine Folge von Variablen gleichen Typs) läßt sich in Python als Liste erzeugen. Wiederhole durch selbständiges Üben:
    - „range"-Anweisung
    - Vereinbaren eines arrays a mit der „list"-Anweisung
    - Zugriff (Ausgabe und Zuweisung von Werten) auf die Komponenten a[0], a[1], . . des als Liste a definierten arrays
    - „append"-Anweisung
    - Erzeugen von Zufallszahlen und deren Zuweisung an die Komponenten des arrays

    Man orientiere sich an den screenshots der letzten BBB-Konferenz, die am 11.02.2021 gemailt wurden.

13. Die Algorithmen **MinSuche** und **MaxSuche**
    Nach Eingabe einer natürlichen Zahl n werden den n Komponenten eines arrays a Zufallszahlen aus dem Bereich (1,100000) zugewiesen; MinSuche bestimmt die kleinste Zahl und gibt diese aus.

    a) Erstelle und teste ein Python-Programm zu MinSuche!
    Beispiel für n = 10 (Benedikt):

```
from random import randint
a = list(range(1,11))

for i in range(0,10):
    a[i] = randint(1,100)

min = a[0]

for i in range(1,10):
    if min > a[i]:
        min = a[i]

print('a =',a)
print('kleinstes Element =',min)
```

Ausgabe:

```
a = [25, 70, 55, 79, 48, 36, 2, 23, 71, 37]
kleinstes Element = 2
```

    b) Erstelle und teste ein Python-Programm zu MaxSuche!

14. Der Algorithmus **Zulassung** ermittelt, ob jemand zur Jahrgangsstufe 12 zugelassen wird (Bestimmungen: MSS-Broschüre MSS_2022_G9_WEB.pdf, pp. 24 – 26).

    a) Welche Daten sind zu erfassen?

    b) Überlege eine geeignete Datenstruktur für die zu erfassenden und auszuwertenden Daten.

    c) Erstelle ein Struktogramm.

    d) Schreibe und teste ein Python-Programm.

Informatik 11 **18.02.2021**

Für die folgenden Algorithmen **MinSuche**, **MinSuche2**, **SelectionSort** gilt:

Nach Eingabe einer natürlichen Zahl n werden den n Komponenten
a[0], a[1], ... , a[n-1] einer Liste a Zufallszahlen zugewiesen.

Die Quelliste und die verarbeitete Liste werden jeweils ausgegeben.

Algorithmus **MinSuche**

Der Algorithmus **MinSuche** bestimmt das kleinste Element der Liste a und weist es der Komponente a[0] zu.

Quelltext in Python:

```
#Eingabe
n = int(input('Anzahl der Datenelemente = '))

# Erzeugen der Liste a mit
# den n Komponenten a[0], . . . , a[n-1]
a = list(range(1,n+1))

# Zuweisung von Zufallszahlen
from random import randint
for i in range(0,n):
    a[i] = randint(1,1000)

# Ausgabe der Quelliste
print('Quelliste:')
print(a)

# Bestimmung des kleinsten Elements
min = a[0]
for i in range(1,n):
    if a[i] < min:
        min = a[i]
        a[i] = a[0]
        a[0] = min

# Ausgabe
print('verarbeitete Liste:')
print(a)
```

Beispiel für n = 10:

```
Anzahl der Datenelemente = 10
Quelliste:
[285, 972, 236, 304, 306, 40, 662, 621, 820, 636]
verarbeitete Liste:
[40, 972, 285, 304, 306, 236, 662, 621, 820, 636]
```

Algorithmus **MinSuche2**

Der Algorithmus **MinSuche2** bestimmt die zwei kleinsten Elemente der Liste a und weist diese den Komponenten a[0] und a[1] zu.

Quelltext in Python:

```
n = int(input('Anzahl der Datenelemente = '))

a = list(range(1,n+1))

from random import randint
for i in range(0,n):
    a[i] = randint(1,1000)

# Ausgabe der Quelliste
print('Quelliste:')
print(a)

# Bestimmung des kleinsten Elements
min = a[0]
for i in range(1,n):
    if a[i] < min:
        min = a[i]
        a[i] = a[0]
        a[0] = min

# Bestimmung des zweitkleinsten Elements
min = a[1]
for i in range(2,n):
    if a[i] < min:
        min = a[i]
        a[i] = a[1]
        a[1] = min

# Ausgabe
print('verarbeitete Liste:')
print(a)
```

Beispiel für n = 10:

```
Anzahl der Datenelemente = 10
Quelliste:
[597, 81, 284, 703, 131, 891, 263, 989, 940, 904]
verarbeitete Liste:
[81, 131, 597, 703, 284, 891, 263, 989, 940, 904]
```

Algorithmus **SelectionSort**
(„Sortieren durch direkte Auswahl")

Der Algorithmus **SelectionSort** bestimmt

- das kleinste Element der Liste a[0], a[1], ... , a[n-1] und weist dieses der Komponente a[0] zu,
- das kleinste Element der Liste a[1], ... , a[n-1] und weist dieses der Komponente a[1] zu,
- das kleinste Element der Liste a[2], ... , a[n-1] und weist dieses der Komponente a[2] zu,
- . . . . . .
- . . . . . .
- das kleinste Element der Liste a[n-2], ... , a[n-1] und weist dieses der Komponente a[n-2] zu.

Auf diese Weise gelingt es, die Komponenten der Liste a der Größe nach zu sortieren.

Quelltext in Python:

```
n = int(input('Anzahl der Datenelemente = '))

a = list(range(1,n+1))

from random import randint
for i in range(0,n):
    a[i] = randint(1,1000)

# Ausgabe der Quelliste
print('Quelliste:')
print(a)

# Sortieren
for j in range(0,n-1):
    min = a[j]
    for i in range(j+1,n):
        if a[i] < min:
            min = a[i]
            a[i] = a[j]
            a[j] = min

# Ausgabe
print('verarbeitete Liste:')
print(a)
```

Beispiel für n = 10:

```
Anzahl der Datenelemente = 10
Quelliste:
[237, 833, 72, 84, 128, 599, 16, 67, 906, 75]
verarbeitete Liste:
[16, 67, 72, 75, 84, 128, 237, 599, 833, 906]
```

```
# Sortieren von Namen (Zeichenketten)

# Erzeugen eines arrays mit dem Variablennamen a
# und der Komponente a[0]
a = list(range(0,1))

# Vereinbaren einer Booleschen Variabeln "condition"
# zum Steuern der Eingabe
# i = Indexvariable der Komponenten von a
condition = True
i = 0

# Eingabe der zu sortierenden Namen
while condition:
    name = input('Name: ')
    if i == 0:
        a[i] = name
    else:
        a.append(name)
    answer = input('weiter? <y> <n> ')
    condition = answer == 'y'
    i +=1

# Ausgabe der eingegebenen Liste (Quelliste)
print()
print('Eingegebene Liste:')
print()
for i in range(0,len(a)):
    print(a[i])


# Sortieren der Quelliste:

for j in range(0,len(a)-1):
    min = a[j]
    for i in range(j+1,len(a)):
       if min > a[i]:
           min = a[i]
           a[i] = a[j]
           a[j] = min


# Ausgabe der sortierten Liste:
print()
print('sortierte Liste:')
print()
for i in range(0,len(a)):
    print(a[i])
```

```
Name: Sören
weiter? <y> <n> y
Name: Maximilian
weiter? <y> <n> n

Eingegebene Liste:

Marvin
Nils
Nele
Philipp
Christian
Pascal
Noel
Benedikt
Laura
Jan
Kerem
Ivo
Laura
Sören
Maximilian

sortierte Liste:

Benedikt
Christian
Ivo
Jan
Kerem
Laura
Laura
Marvin
Maximilian
Nele
Nils
Noel
Pascal
Philipp
Sören
```

Nach erfolgter, d. h. mit „Enter" abgeschlossener Eingabe einer der Variablen **name** zugewiesenen Zeichenkette erfolgt jeweils die Abfrage, ob die Datenerfassung fortgesetzt werden soll; die Antwort wird als character (Zeichen; hier: **y** oder **n**) in der Variablen **answer** gespeichert.

Der Wert des Booleschen Terms **answer == 'y'** (**True** oder **False**) wird der Booleschen Variablen **condition** zu gewiesen; solange **condition** den Wert **True** hat, wird die Eingabe fortgesetzt (Schleifenrumpf der **while**-Schleife), und der für **i>0** neu eingegebene, in der Variablen **name** als Zeichenkette erfaßte Name wird mit **a.append(name)** der Liste **a** angefügt.

Falls man als Antwort auf die Frage, ob man weitermachen möchte, nicht **y** eingibt, nimmt **answer == 'y'** den Wert **False** an, die Eingabe wird abgebrochen. Nach der Ausgabe der eingegebenen Liste und dem Sortieren erfolgt die Ausgabe der sortierten Liste.

Aufwandsbetrachtung „Sortieren durch direkte Auswahl" (SelectionSort)

Wir formulieren einen Zusammenhang zwischen dem zeitlichen Aufwand, um eine Liste von n Datenelementen (z. B. Zufallszahlen, Namen) der Größe nach zu sortieren, und der Anzahl n der Datenelemente.

Wertzuweisungen, Abfragen und Rechenoperationen sind elementare Anweisungen, die eine bestimmte Rechenzeit erfordern; obwohl diese Rechenzeiten mit fortschreitender Leistungsfähigkeit der Hardware immer kürzer werden, gerät man rasch an Grenzen der praktischen Durchführbarkeit eines Algorithmus, wenn die Anzahl der abzuarbeitenden Anweisungen zu stark wächst.

Wesentlicher Baustein des Algorithmus „Sortieren durch direkte Auswahl" ist der Schleifenrumpf der inneren for-Schleife (hier: rot gekennzeichnet), der das kleinste Element innerhalb des arrays **`a[j], . . , a[n-1]`** ermittelt und dieses der Komponente **`a[j]`** zuweist:

```
for j in range(0,n-1):
    min = a[j]
    for i in range(j+1,n):
       if min > a[i]:
           min  = a[i]
           a[i] = a[j]
           a[j] = min
```

Dieser rot markierte Schleifenrumpf besteht aus 3 Wertzuweisungen und 1 Abfrage, die wir gedanklich als ganzes zum Anweisungsblock **A** zusammenfassen:

```
for j in range(0,n-1):
    min = a[j]
    for i in range(j+1,n):
            A
```

Für **j=0** nimmt der Schleifenindex i der inneren Schleife alle Werte von 1 bis n–1 an, folglich wird der Anweisungsblock **A** (n–1)-mal ausgeführt.
Für **j=1** nimmt der Schleifenindex i der inneren Schleife alle Werte von 2 bis n–1 an, folglich wird Block **A** (n–2)-mal ausgeführt.
Für **j=2** nimmt der Schleifenindex i der inneren Schleife alle Werte von 3 bis n–1 an, folglich wird Block **A** (n–3)-mal ausgeführt.

In der folgenden Tabelle notieren wir für jeden Wert von **j** jeweils den Bereich, den **i** durchläuft, und die daraus sich ergebende Anzahl **z(j)**, die angibt, wie oft der Anweisungsblock **A** durchlaufen wird:

Index j	**Index i**	**z(j)**
j = 0	1 ≤ i ≤ n-1	n-1
j = 1	2 ≤ i ≤ n-1	n-2
j = 2	3 ≤ i ≤ n-1	n-3
j = 3	4 ≤ i ≤ n-1	n-4
j = 4	5 ≤ i ≤ n-1	n-5
....	....	....
....	....	....
j = n-3	n-2 ≤ i ≤ n-1	2
j = n-2	n-1 ≤ i ≤ n-1	1

Gesamtzahl **z** der Abarbeitungen von Anweisungsblock **A**:

z = z(0) + z(1) + z(2) + z(3) + ... + z(n-3) + z(n-2)
z = (n-1) + (n-2) + ... + 2 + 1
z = ½ · (n – 1) · n
z = ½ · n^2 – ½ · n ≈ ½ · n^2 für große Werte von n

Ergebnis: Bei SelectionSort wächst der Rechenaufwand zum Sortieren einer aus n Elementen bestehenden Liste quadratisch mit n.

```
# SelectionSort
# Aufwand proportional zu n^2

from random import randint
import time

n = int(input('Anzahl der Datenelemente = '))
r = int(input('Wieviele Elemente sollen angezeigt werden? '))

a = list(range(1,n+1))

for i in range(0,n):
    a[i] = randint(1,1000000)

# Ausgabe der Quelliste:

for i in range(0,r):
    print(a[i])


# Sortieren der Quelliste:

start = time.time()

for j in range(0,n-1):
    min = a[j]
    for i in range(j+1,n):
       if min > a[i]:
           min = a[i]
           a[i] = a[j]
           a[j] = min

end = time.time()


# Ausgabe der sortierten Liste:
print()
print('sortierte Liste:')
for i in range(0,r):
    print(a[i])

print()
print('Zeitaufwand zum Sortieren von',n,'Elementen: {:7.3f} s'.format(end-start)


Anzahl der Datenelemente = 2000
Wieviele Elemente sollen angezeigt werden? 0

sortierte Liste:

Zeitaufwand zum Sortieren von 2000 Elementen:   0.422 s
>>>
====== RESTART: F:/Informatik_2021/info11/mail_27-03-2021/SelectionSort.py
Anzahl der Datenelemente = 4000
Wieviele Elemente sollen angezeigt werden? 0

sortierte Liste:

Zeitaufwand zum Sortieren von 4000 Elementen:   1.562 s
>>>
====== RESTART: F:/Informatik_2021/info11/mail_27-03-2021/SelectionSort.py
Anzahl der Datenelemente = 8000
Wieviele Elemente sollen angezeigt werden? 0

sortierte Liste:

Zeitaufwand zum Sortieren von 8000 Elementen:   6.275 s
```

Aufgabenblatt Nr. 6 inf11 22.03.2021
mit Lösungen zu Nr. 15 – Nr. 18 aus der BBB-Konferenz vom 25.03.2021

15. **BOOLESCHE TERME**

Schreibweisen für die Negation:
not a = ¬ a = $\overline{a}$

Wir ergänzen die in Aufgabe 11 (Blatt 4, 26.01.2021) für Boolesche Variable formulierten Rechenregeln
- Kommutativgesetze (1) und (1')
- Assoziativgesetze (2) und (2')
- Distributivgesetze (3) und (3')

um die beiden Gesetze von de Morgan:

(4) $\overline{a \cdot b} = \overline{a} + \overline{b}$ (4') $\overline{a + b} = \overline{a} \cdot \overline{b}$

Aufgabe: Beweise (4) *(Hinweis: Wahrheitstafel!)*

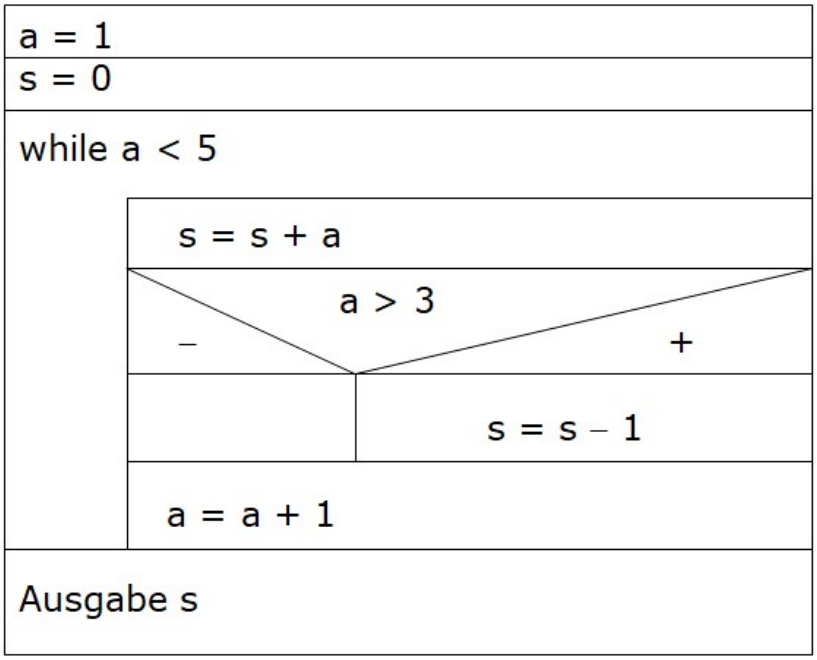
Lösung (Nils):

a	b	a · b	**not (a · b)**	not a	not b	**not a + not b**
0	0	0	**1**	1	1	**1**
0	1	0	**1**	1	0	**1**
1	0	0	**1**	0	1	**1**
1	1	1	**0**	0	0	**0**

Da die Spalten zu **not (a · b)** und **not a + not b** übereinstimmen, gilt:

not (a · b) = not a + not b

16. Erstelle einen in Python geschriebenen Quelltext zu folgendem Struktogramm:



Lösung (Philipp):

```
a = 1
s = 0

while a < 5:
    s = s + a  # oder:  s += a
    if a > 3:
        s = s-1
    a = a + 1

print(s)
```

17. In der Zusammenfassung ***MinSuche_SelectionSort.pdf*** werden die Algorithmen **MinSuche, MinSuche2** und **SelectionSort** noch einmal erläutert.

    Aufgabe: Formuliere bei

    a) MinSuche die for-Schleife zur Bestimmung des kleinsten Elements,

    b) SelectionSort die beiden for-Schleifen zum Sortieren

    jeweils als while-Schleifen und überprüfe die so erhaltenen Python-Programme anhand von Testläufen.

    Lösung zu a) (Benedikt):

```
n = int(input('Anzahl der Datenelemente = '))

a = list(range(1,n+1))

from random import randint
for i in range(0,n):
    a[i] = randint(1,1000)

print('Quelliste:')
print(a)

# Bestimmung des kleinsten Elements
min = a[0]
i = 1
while i < n:
    if a[i] < min:
        min = a[i]
        a[i] = a[0]
        a[0] = min
    i = i + 1

# Ausgabe
print('verarbeitete Liste:')
print(a)
```

Lösung zu b) (Christian):

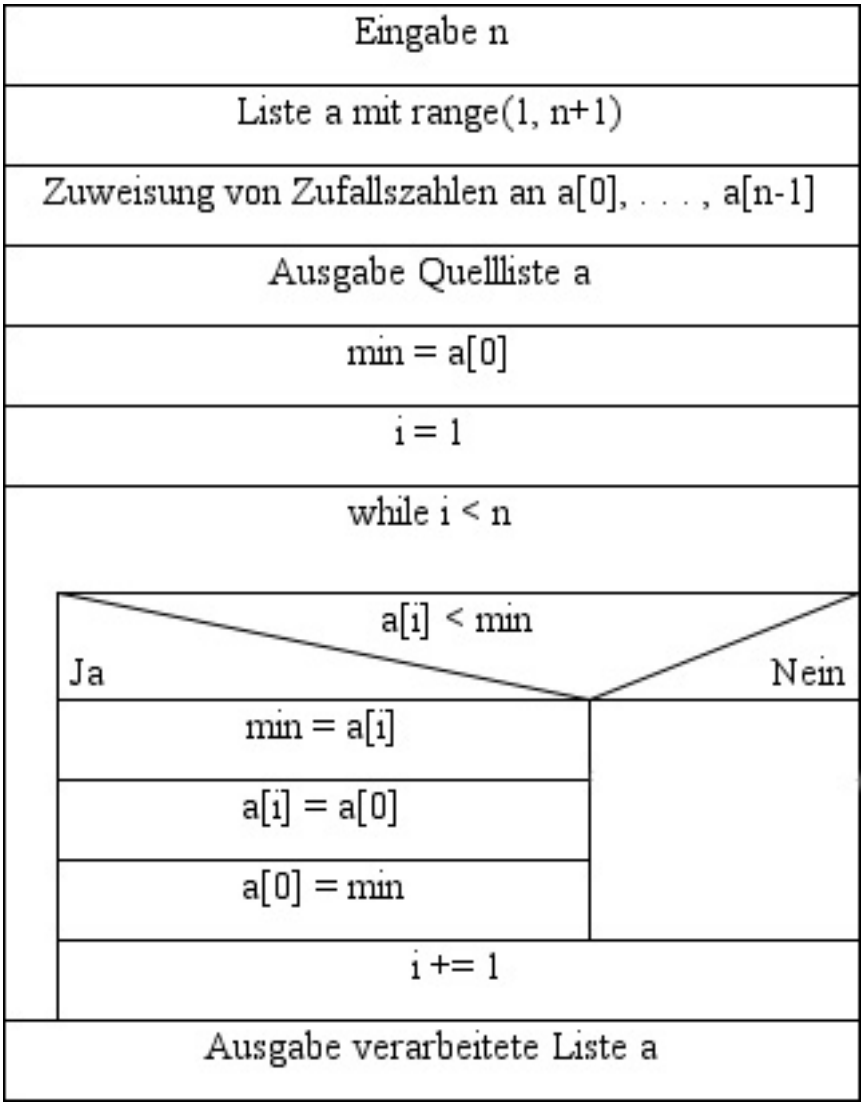
```
n = int(input('Anzahl der Datenelemente = '))

a = list(range(1,n+1))

from random import randint
for i in range(0,n):
    a[i] = randint(1,1000)

# Ausgabe der Quelliste
print('Quelliste:')
print(a)

# Sortieren
j = 0
while j < n - 1:
    i = j + 1
    min = a[j]
    while i < n:
        if a[i] < min:
            min = a[i]
            a[i] = a[j]
            a[j] = min
        i = i + 1
    j = j + 1

# Ausgabe
print('verarbeitete Liste:')
print(a)
```

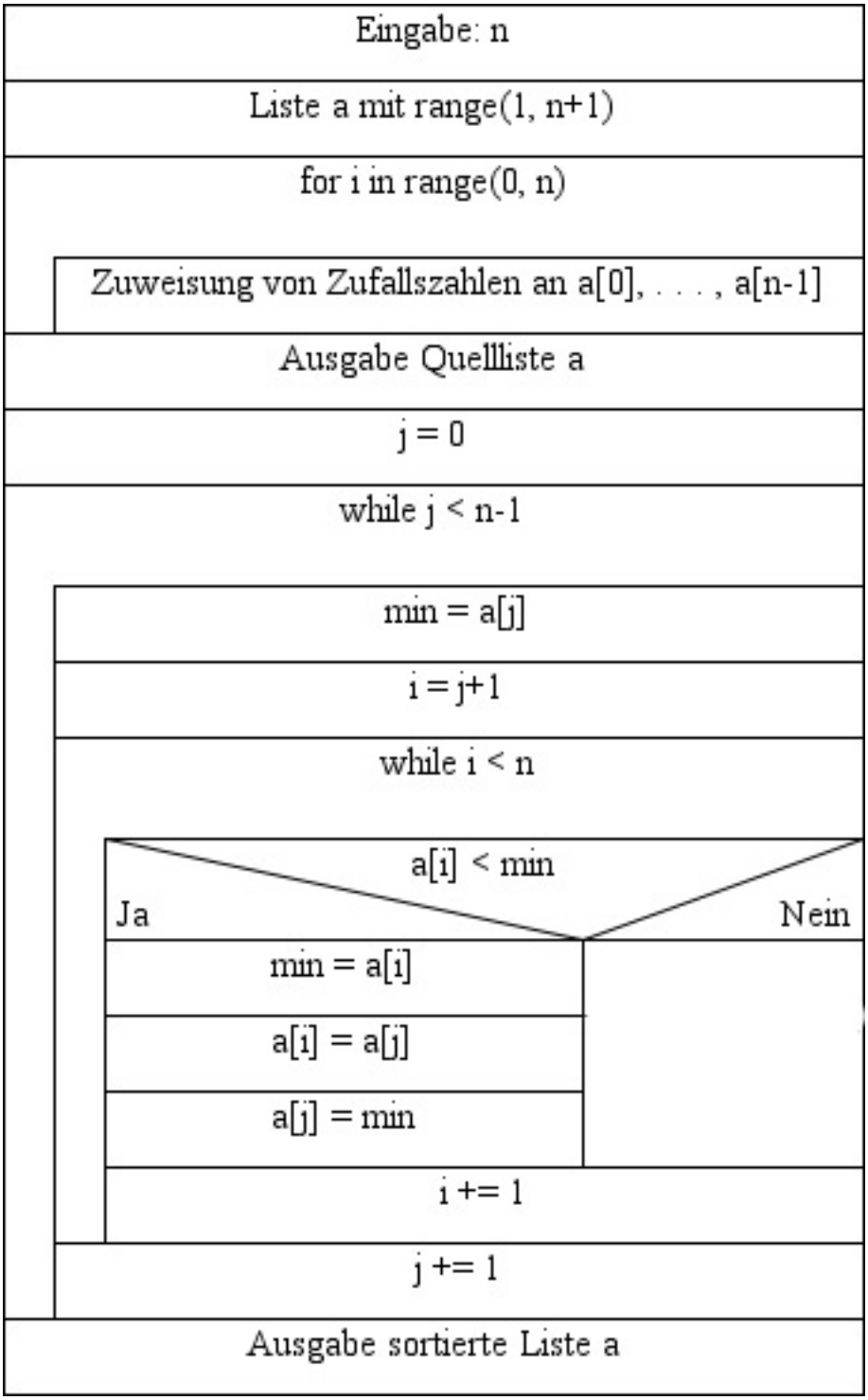
Testlauf (n = 20):

```
Anzahl der Datenelemente = 20
Quelliste:
[726, 236, 827, 926, 294, 774, 431, 133, 346, 434, 480, 167, 310, 690,
841, 344, 434, 411, 697, 301]
verarbeitete Liste:
[133, 167, 236, 294, 301, 310, 344, 346, 411, 431, 434, 434, 480, 690,
697, 726, 774, 827, 841, 926]
```

18. Erstelle jeweils ein Struktogramm zu **MinSuche** und **SelectionSort** (Hinweis: verwende bevorzugt die Versionen mit while-Schleife, Aufgabe 17; für die Zuweisung von Zufallszahlen an die Komponenten des arrays a genügt es zu schreiben: „Zuweisung von Zufallszahlen an a[0], . . . , a[n-1]“)

Struktogramme (Marvin):





19. Freiwillige Aufgabe:
Der Algorithmus SelectionSort (Quelltext: SelectionSort_04-03-2021.py) vertauscht die Inhalte der Speicherplätze a[j] und a[i], j < i < n, immer dann, wenn ein kleineres a[i] als a[j] gefunden wurde; hier gibt es noch Optimierungspotential, um die Rechenzeit insgesamt zu verkürzen. Ergreife diese Möglichkeit und teste! (Die Laufzeit zum Sortieren läßt sich für große Werte von n etwa halbieren.)