1.3 Verzweigungen und Schleifen

Die bisher betrachteten Programme bestanden aus Anweisungen,
die der Reihe nach von der ersten bis zur letzten ausgeführt werden.

Nun werden wir sehen, wie man diese Reihenfolge beeinflussen kann.

Abhängig von *Bedingungen* (Ausdrücke vom Typ `bool`) können

- *Anweisungs-Blöcke* ausgeführt oder nicht ausgeführt werden (*Verzweigungen*),
- *Anweisungs-Blöcke* mehrfach ausgeführt werden (*Schleifen*).

Mit Verzweigungen und Schleifen werden wir
Versuchsreihen mit Würfelexperimenten machen und
ein Problem lösen, das Euler gerne gelöst hätte.

Vorlesung 02

1 Verzweigungen – `if`

In der letzten Vorlesung hatten wir ein Programm zur Lösung der quadratischen Gleichung $a \cdot x^2 + b \cdot x + c = 0$ mit der Mitternachtsformel geschrieben:

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Bei der Mitternachtsformel muss man beachten:

1. Falls $a = 0$, kann die Formel nicht benutzt werden.
2. Falls $b^2 - 4 \cdot a \cdot c < 0$, kann die Formel nicht benutzt werden.
3. Falls $b^2 - 4 \cdot a \cdot c = 0$ und $a \neq 0$, hat die quadratische Gleichung nur eine Lösung.

Beim Lauf des Programmes `mitternacht.py` haben wir das nicht beachtet
und dadurch treten „unschöne“ Ereignisse auf:

Bei 1. und 2. stürzt das Programm ab.

Bei 3. wird zweimal die gleiche Lösung ausgegeben.

1 Verzweigungen – `if`

In der letzten Vorlesung hatten wir ein Programm zur Lösung der quadratischen Gleichung $a \cdot x^2 + b \cdot x + c = 0$ mit der Mitternachtsformel geschrieben:

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Bei der Mitternachtsformel muss man beachten:

1. Falls $a = 0$, kann die Formel nicht benutzt werden.
2. Falls $b^2 - 4 \cdot a \cdot c < 0$, kann die Formel nicht benutzt werden.
3. Falls $b^2 - 4 \cdot a \cdot c = 0$ und $a \neq 0$, hat die quadratische Gleichung nur eine Lösung.

Beim Lauf des Programmes `mitternacht.py` haben wir das nicht beachtet
und dadurch treten „unschöne“ Ereignisse auf:

- Bei 1. und 2. stürzt das Programm ab.
- Bei 3. wird zweimal die gleiche Lösung ausgegeben.

1 Verzweigungen – `if`

In der letzten Vorlesung hatten wir ein Programm zur Lösung der quadratischen Gleichung $a \cdot x^2 + b \cdot x + c = 0$ mit der Mitternachtsformel geschrieben:

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Bei der Mitternachtsformel muss man beachten:

1. Falls $a = 0$, kann die Formel nicht benutzt werden.
2. Falls $b^2 - 4 \cdot a \cdot c < 0$, kann die Formel nicht benutzt werden.
3. Falls $b^2 - 4 \cdot a \cdot c = 0$ und $a \neq 0$, hat die quadratische Gleichung nur eine Lösung.

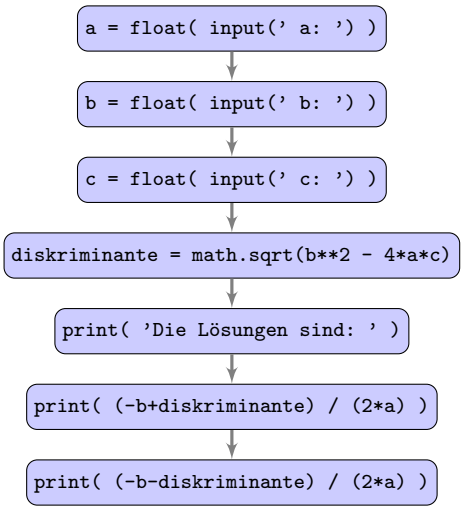
Beim Lauf des Programmes `mitternacht.py` haben wir das nicht beachtet
und dadurch treten „unschöne“ Ereignisse auf:

Bei 1. und 2. stürzt das Programm ab.

Bei 3. wird zweimal die gleiche Lösung ausgegeben.

Der Programmfluss . . .

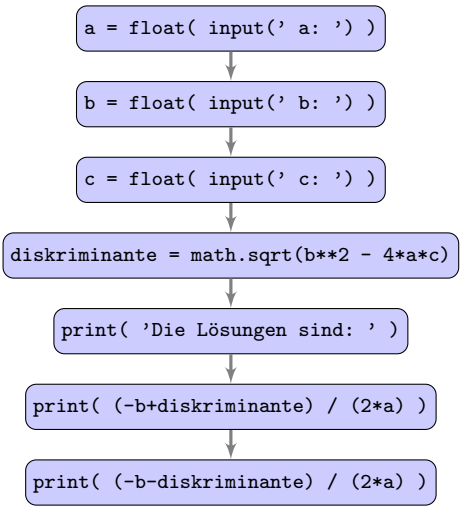
. . . dieses Programms besteht bei jeder Eingabe aus der gleichen Folge von Anweisungen.




```
#-------------------------------------------------------
# mitternacht.py
# Liest float-Werte a, b und c ein und gib die
# durch die Mitternachtsformel bestimmten Lösungen aus.
#-------------------------------------------------------
import math

# Lies float-Werte a, b und c ein.
a = float( input('  a: ') )
b = float( input('  b: ') )
c = float( input('  c: ') )

# Berechne die Diskriminante.
diskriminante = math.sqrt(b**2 - 4 * a * c)

# Gib die Ergebnisse für x aus.
print('Die Lösungen sind:')
print( (-b + diskriminante) / (2*a) )
print( (-b - diskriminante) / (2*a) )
#-------------------------------------------------------
```

Der Programmfluss . . .

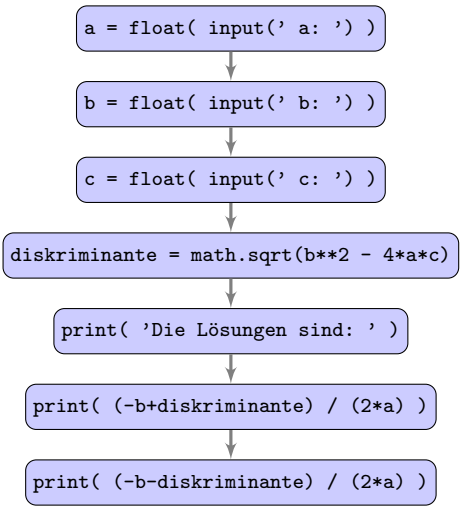
. . . dieses Programms besteht bei jeder Eingabe aus der gleichen Folge von Anweisungen.




```
mundhenk@ma3:~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 12
  b: 3456
  c: 78
Die Lösungen sind:
-0.022571213401647583
-287.97742878659835
mundhenk@ma3:~/Python/VL01$
```

Der Programmfluss . . .

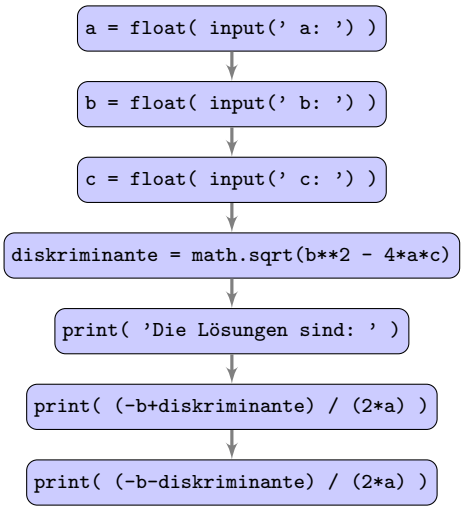
. . . dieses Programms besteht bei jeder Eingabe aus der gleichen Folge von Anweisungen.




```
mundhenk@ma3: ~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 0
  b: 2
  c: 4
Die Lösungen sind:
Traceback (most recent call last):
  File "mitternacht.py", line 18, in <module>
    print( (-b + diskriminante) / (2*a) )
ZeroDivisionError: float division by zero
mundhenk@ma3:~/Python/VL01$
```

Der Programmfluss . . .

. . . dieses Programms besteht bei jeder Eingabe aus der gleichen Folge von Anweisungen.

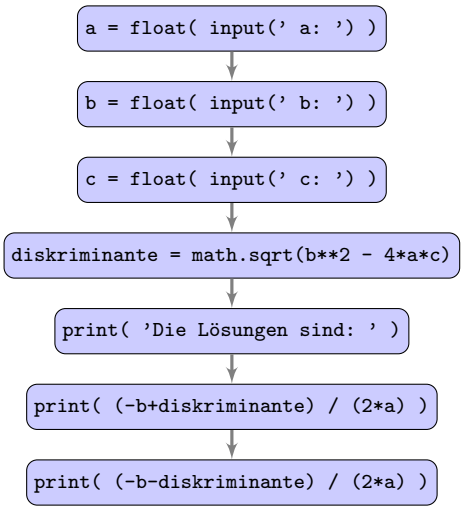



```
mundhenk@ma3:~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 0
  b: 2
  c: 4
Die Lösungen sind:
Traceback (most recent call last):
  File "mitternacht.py", line 18, in <module>
    print( (-b + diskriminante) / (2*a) )
ZeroDivisionError: float division by zero
mundhenk@ma3:~/Python/VL01$
```

Idee: das Programm soll wie bisher laufen, *wenn* $a \neq 0$.

Der Programmfluss . . .

. . . dieses Programms besteht bei jeder Eingabe aus der gleichen Folge von Anweisungen.



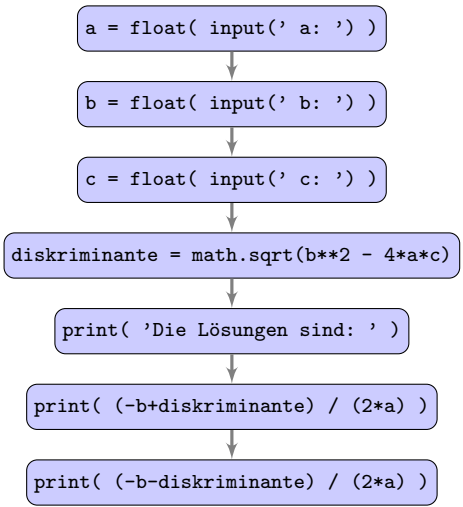

```
mundhenk@ma3:~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 0
  b: 2
  c: 4
Die Lösungen sind:
Traceback (most recent call last):
  File "mitternacht.py", line 18, in <module>
    print( (-b + diskriminante) / (2*a) )
ZeroDivisionError: float division by zero
mundhenk@ma3:~/Python/VL01$
```

Idee: das Programm soll wie bisher laufen, *wenn* $a \neq 0$.

Sonst soll eine eigene „Fehlermeldung“ ausgegeben werden.

Der Programmfluss . . .

. . . dieses Programms besteht bei jeder Eingabe aus der gleichen Folge von Anweisungen.



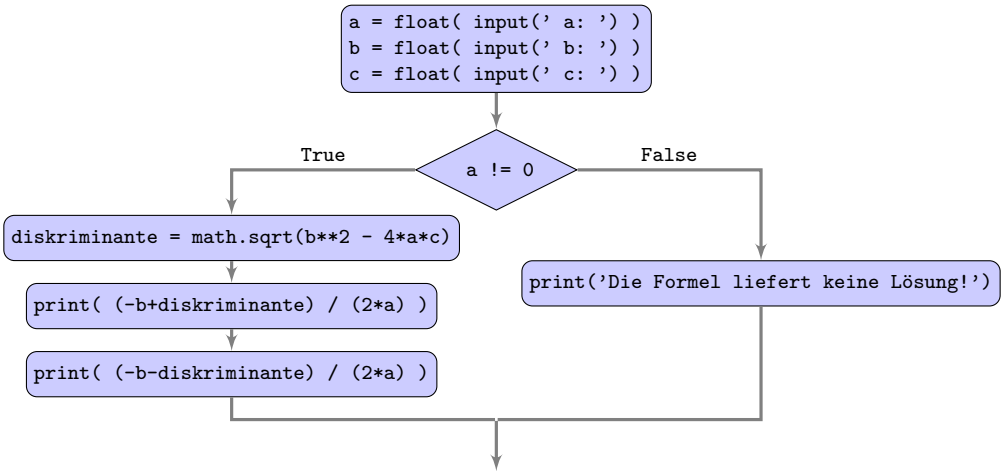

```
mundhenk@ma3: ~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht_v2.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 0
  b: 2
  c: 4
Die Formel liefert keine Lösung!
mundhenk@ma3:~/Python/VL01$
```

Idee: das Programm soll wie bisher laufen, *wenn* $a \neq 0$.

Sonst soll eine eigene „Fehlermeldung“ ausgegeben werden.

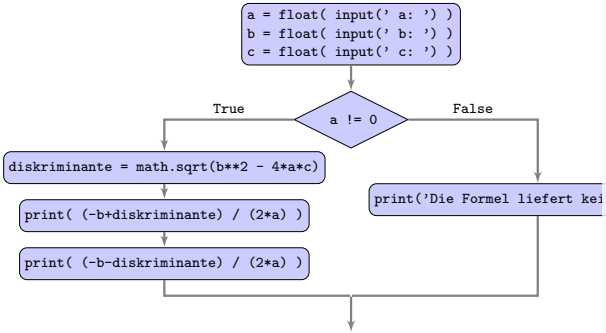
Die Struktur des verbesserten Programmes

Verzweigung des Programmflusses – `if ... else`



Das verbesserte Programm mitternacht_v2.py

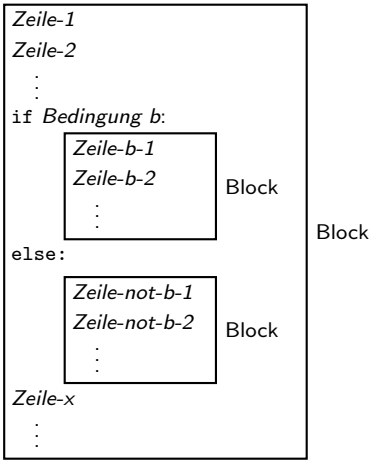
Verzweigung des Programmflusses – if ... else




```
# mitternacht_v2.py
#-------------------------------------------------
import math

# Lies float-Werte a, b und c ein.
a = float( input('  a: ') )
b = float( input('  b: ') )
c = float( input('  c: ') )

# Verzweige abhängig von a!=0 zwischen Berechnung
# von Lösungen und Ausgabe einer Fehlermeldung.
if a!=0:
  # Berechne die Diskriminante.
  diskriminante = math.sqrt(b**2 - 4*a*c)

  # Gib die Ergebnisse für x aus.
  print( (-b + diskriminante) / (2*a) )
  print( (-b - diskriminante) / (2*a) )
else:
  print('Die Formel liefert keine Lösung!')
```

Die Blockstruktur einer `if`-Anweisung

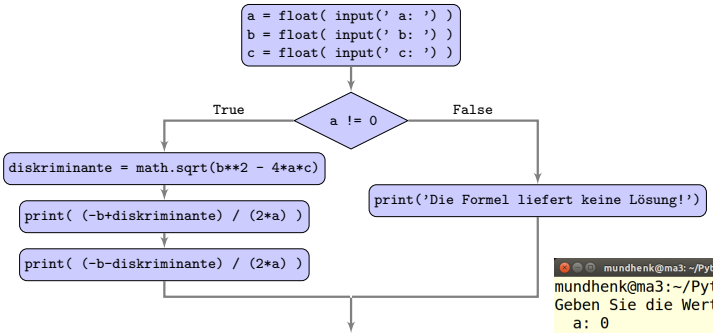
Verzweigung des Programmflusses – `if ... else`



Blockstruktur einer `if`-Anweisung

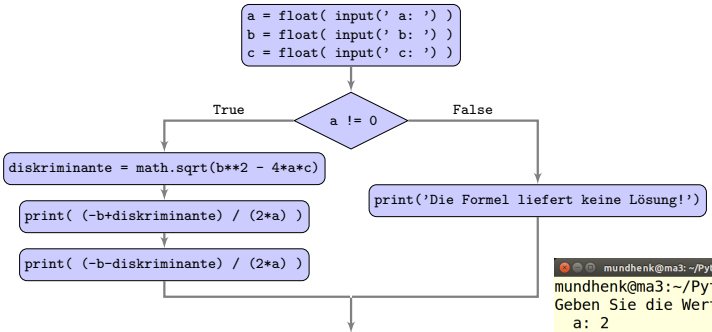
```
# mitternacht_v2.py
#-------------------------------------------------
import math

# Lies float-Werte a, b und c ein.
a = float( input('  a: ') )
b = float( input('  b: ') )
c = float( input('  c: ') )

# Verzweige abhängig von a!=0 zwischen Berechnung
# von Lösungen und Ausgabe einer Fehlermeldung.
if a!=0:
  # Berechne die Diskriminante.
  diskriminante = math.sqrt(b**2 - 4*a*c)

  # Gib die Ergebnisse für x aus.
  print( (-b + diskriminante) / (2*a) )
  print( (-b - diskriminante) / (2*a) )
else:
  print('Die Formel liefert keine Lösung!')
```

Die nächste Verbesserungsidee



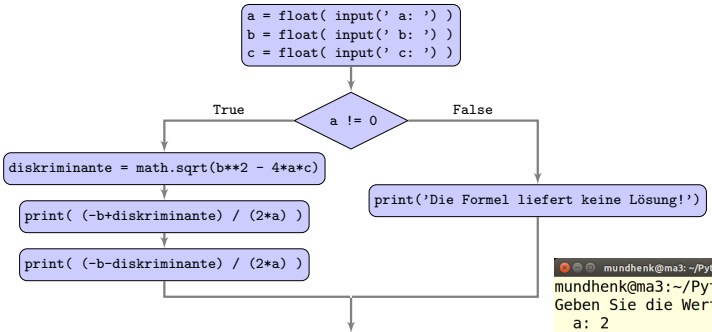

```
mundhenk@ma3: ~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht_v2.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 0
  b: 2
  c: 4
Die Formel liefert keine Lösung!
mundhenk@ma3:~/Python/VL01$
```

Die nächste Verbesserungsidee



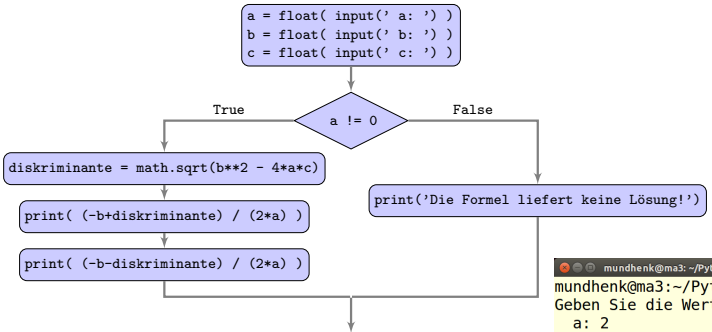

```
mundhenk@ma3: ~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht_v2.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 2
  b: 0
  c: 2
Traceback (most recent call last):
  File "mitternacht_v2.py", line 16, in <module>
    diskriminante = math.sqrt(b**2 - 4 * a * c)
ValueError: math domain error
mundhenk@ma3:~/Python/VL01$
```

Die nächste Verbesserungsidee




```
mundhenk@ma3: ~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht_v3.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 2
  b: 0
  c: 2
Die Formel liefert keine Lösung!
mundhenk@ma3:~/Python/VL01$
```

Die nächste Verbesserungsidee




```
mundhenk@ma3: ~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht_v3.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 2
  b: 0
  c: 2
Die Formel liefert keine Lösung!
mundhenk@ma3:~/Python/VL01$
```

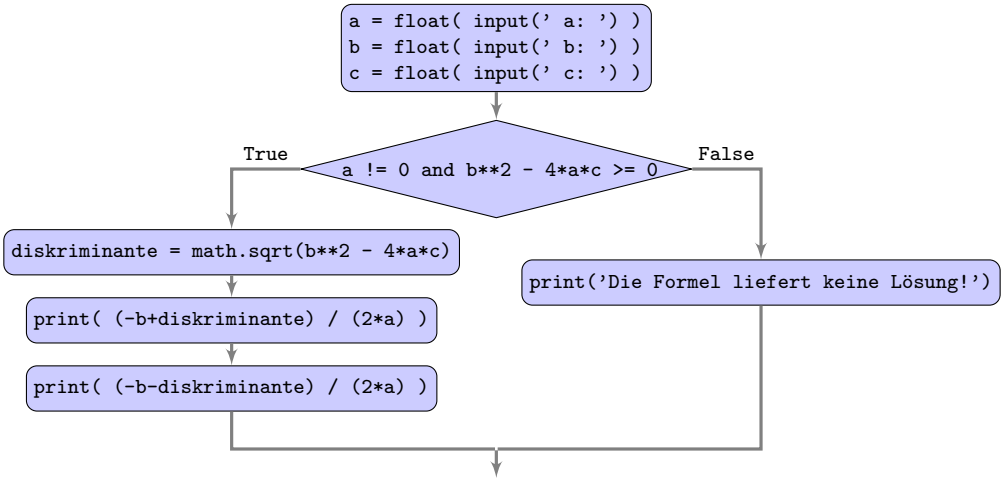
Wenn `b**2 - 4*a*c >= 0`,

dann soll das Programm laufen wie bisher,

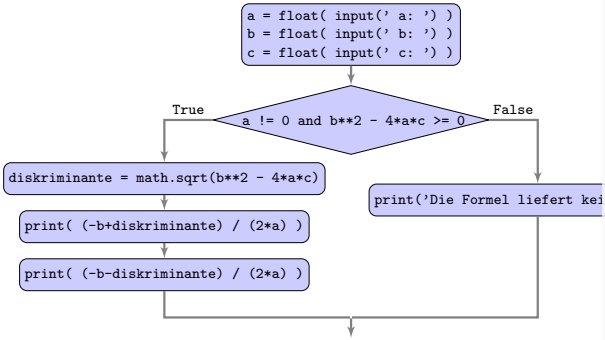
sonst

soll die Fehlermeldung ausgegeben werden.

Die Struktur des verbesserten Programmes

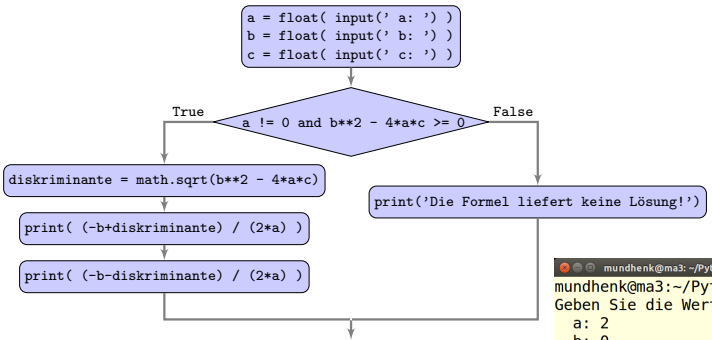


Das verbesserte Programm mitternacht_v3.py



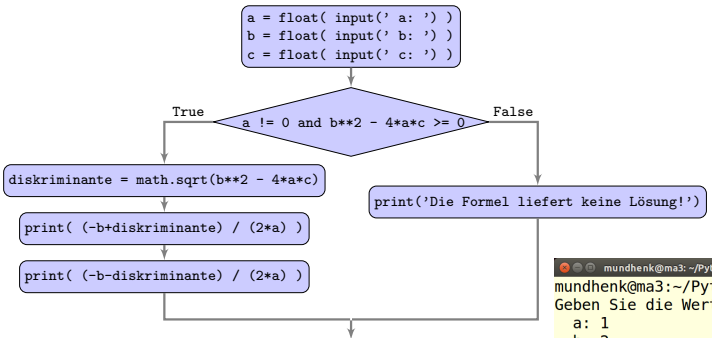

```python
# mitternacht_v3.py
#-----------------------------------------------
import math

# Lies float-Werte a, b und c ein.
a = float( input('  a: ') )
b = float( input('  b: ') )
c = float( input('  c: ') )

# Verzweige abhängig von
# a!=0 und b**2-4*a*c>=0 zwischen Berechnung
# von Lösungen und Ausgabe einer Fehlermeldung.
if a != 0  and  b**2-4*a*c >= 0:
  # Berechne die Diskriminante.
  diskriminante = math.sqrt(b**2 - 4*a*c)

  # Gib die Ergebnisse für x aus.
  print( (-b + diskriminante) / (2*a) )
  print( (-b - diskriminante) / (2*a) )
else:
  print('Die Formel liefert keine Lösung!')
```

Die letzte Verbesserungsidee



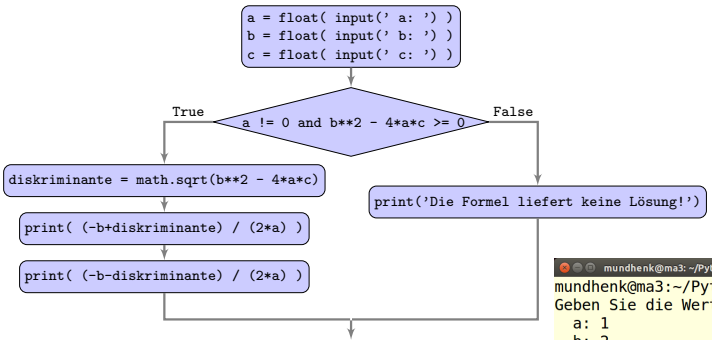

```
mundhenk@ma3: ~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht_v3.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 2
  b: 0
  c: 2
Die Formel liefert keine Lösung!
mundhenk@ma3:~/Python/VL01$
```

Die letzte Verbesserungsidee




```
mundhenk@ma3: ~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht_v3.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 1
  b: 2
  c: 1
-1.0
-1.0
mundhenk@ma3:~/Python/VL01$
```

Die letzte Verbesserungsidee




```
mundhenk@ma3:~/Python/VL01
mundhenk@ma3:~/Python/VL01$ python3 mitternacht_v4.py
Geben Sie die Werte von a,b und c für die Mitternachtsformel ein.
  a: 1
  b: 2
  c: 1
-1.0
mundhenk@ma3:~/Python/VL01$
```

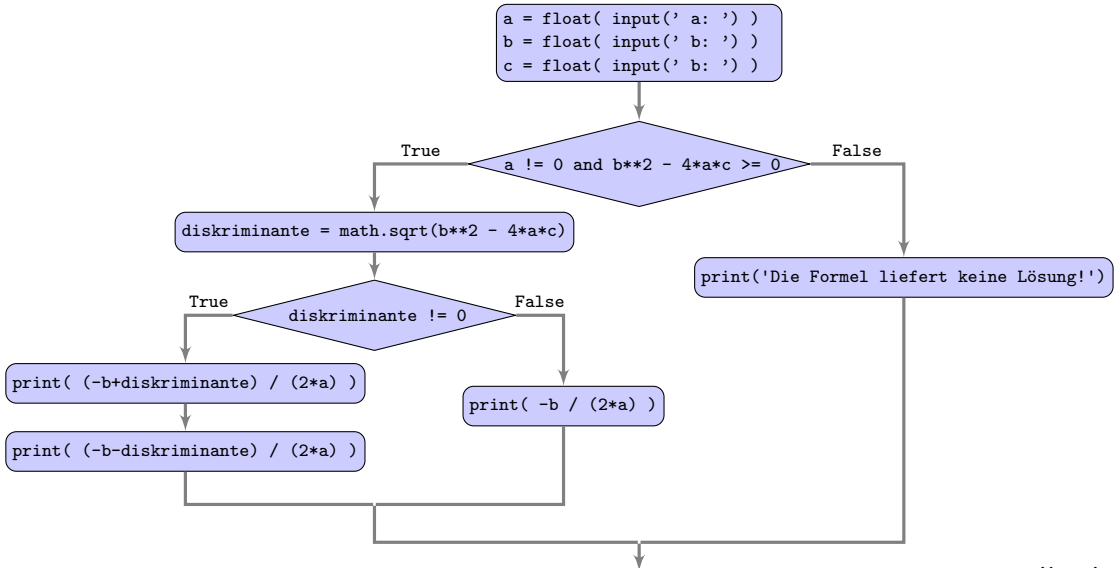
Wenn `diskriminante != 0`,

dann soll das Programm laufen wie bisher,

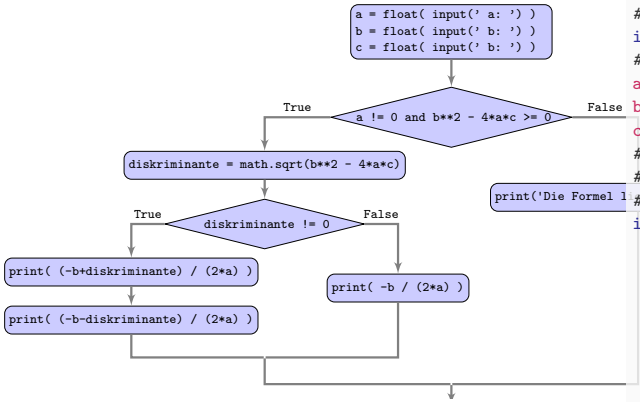
sonst

soll nur eine Lösung ausgegeben werden.

Die Struktur des verbesserten Programms

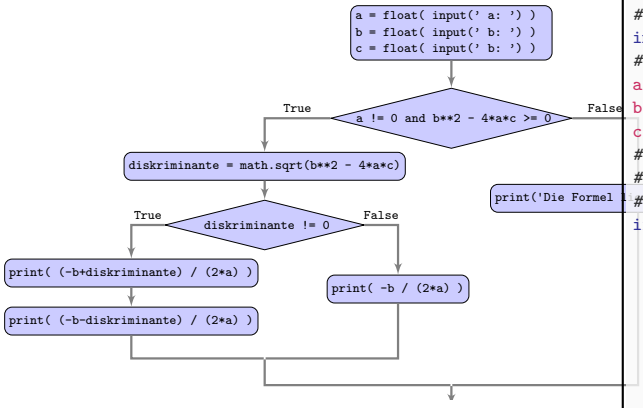


Das verbesserte Programm mitternacht_v4.py



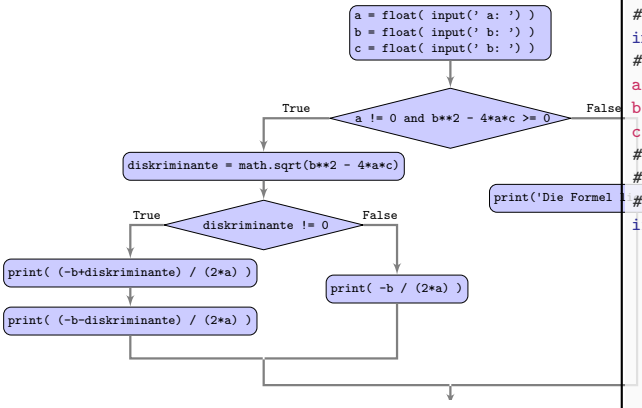

```
# mitternacht_v4.py
#--------------------------------------------------
import math
# Lies float-Werte a, b und c ein.
a = float( input('  a: ') )
b = float( input('  b: ') )
c = float( input('  c: ') )
# Verzweige abhängig von
# a!=0 und b**2-4*a*c>=0 zwischen Berechnung
# von Lösungen und Ausgabe einer Fehlermeldung.
if a != 0  and  b**2-4*a*c >= 0:
  # Berechne die Diskriminante.
  diskriminante = math.sqrt(b**2 - 4*a*c)
  # Verzweige abhängig von diskriminante!=0 zwischen
  # der Ausgabe von zwei oder einem Ergebnis.
  if diskriminante!=0:
    # Gib die beiden Ergebnisse für x aus.
    print( (-b + diskriminante) / (2*a) )
    print( (-b - diskriminante) / (2*a) )
  else:
    # Gib das einzige Ergebnis aus.
    print( -b / (2*a) )
else:
  print('Die Formel liefert keine Lösung!')
```

Das verbesserte Programm mitternacht_v4.py



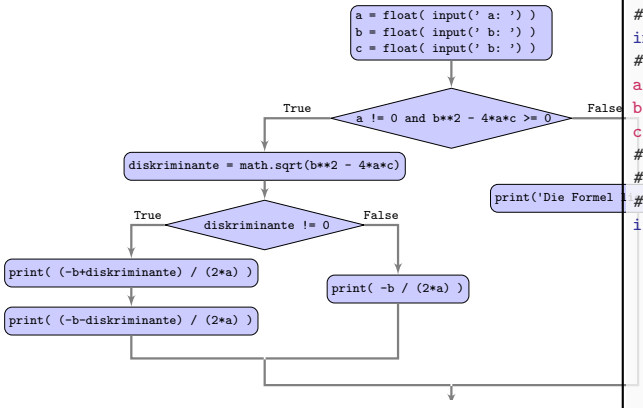

```python
# mitternacht_v4.py
#-------------------------------------------------
import math
# Lies float-Werte a, b und c ein.
a = float( input('  a: ') )
b = float( input('  b: ') )
c = float( input('  c: ') )
# Verzweige abhängig von
# a!=0 und b**2-4*a*c>=0 zwischen Berechnung
# von Lösungen und Ausgabe einer Fehlermeldung.
if a != 0  and  b**2-4*a*c >= 0:
  # Berechne die Diskriminante.
  diskriminante = math.sqrt(b**2 - 4*a*c)
  # Verzweige abhängig von diskriminante!=0 zwischen
  # der Ausgabe von zwei oder einem Ergebnis.
  if diskriminante!=0:
    # Gib die beiden Ergebnisse für x aus.
    print( (-b + diskriminante) / (2*a) )
    print( (-b - diskriminante) / (2*a) )
  else:
    # Gib das einzige Ergebnis aus.
    print( -b / (2*a) )
else:
  print('Die Formel liefert keine Lösung!')
```

Das verbesserte Programm mitternacht_v4.py




```
# mitternacht_v4.py
#-------------------------------------------------
import math
# Lies float-Werte a, b und c ein.
a = float( input('  a: ') )
b = float( input('  b: ') )
c = float( input('  c: ') )
# Verzweige abhängig von
# a!=0 und b**2-4*a*c>=0 zwischen Berechnung
# von Lösungen und Ausgabe einer Fehlermeldung.
if a != 0  and  b**2-4*a*c >= 0:
    # Berechne die Diskriminante.
    diskriminante = math.sqrt(b**2 - 4*a*c)
    # Verzweige abhängig von diskriminante!=0 zwischen
    # der Ausgabe von zwei oder einem Ergebnis.
    if diskriminante!=0:
        # Gib die beiden Ergebnisse für x aus.
        print( (-b + diskriminante) / (2*a) )
        print( (-b - diskriminante) / (2*a) )
    else:
        # Gib das einzige Ergebnis aus.
        print( -b / (2*a) )
else:
    print('Die Formel liefert keine Lösung!')
```

Das verbesserte Programm `mitternacht_v4.py`




```python
# mitternacht_v4.py
#-------------------------------------------------
import math
# Lies float-Werte a, b und c ein.
a = float( input('  a: ') )
b = float( input('  b: ') )
c = float( input('  c: ') )
# Verzweige abhängig von
# a!=0 und b**2-4*a*c>=0 zwischen Berechnung
# von Lösungen und Ausgabe einer Fehlermeldung.
if a != 0  and  b**2-4*a*c >= 0:
    # Berechne die Diskriminante.
    diskriminante = math.sqrt(b**2 - 4*a*c)
    # Verzweige abhängig von diskriminante!=0 zwischen
    # der Ausgabe von zwei oder einem Ergebnis.
    if diskriminante!=0:
        # Gib die beiden Ergebnisse für x aus.
        print( (-b + diskriminante) / (2*a) )
        print( (-b - diskriminante) / (2*a) )
    else:
        # Gib das einzige Ergebnis aus.
        print( -b / (2*a) )
else:
    print('Die Formel liefert keine Lösung!')
```

- Verzweigungen des Programmflusses werden mit `if ... else` gesteuert
- der `else`-Block kann weggelassen werden
- die Blockstruktur wird durch Einrücken des Programmtextes beschrieben
- es gibt einfachere Schreibweise (`elif`) für einseitige Mehrfachverzweigungen

2 Schleifen – `while`

Üblicherweise bestehen Programme aus Anweisungen,
die häufig wiederholt werden.

Die `while`-Schleife erlaubt die Wiederholung eines Anweisungs-Blocks,
solange eine Bedingung erfüllt ist.

Die `for`-Schleife erlaubt die Wiederholung eines Anweisungs-Blocks
mit mitlaufender Veränderung eines Zählers (siehe Abschnitt 3).

Programmierauftrag:
berechne die Rückzahlungsdauer eines Kredits

X nimmt einen Kredit über 1000 € mit einem Jahreszzinssatz von 5% auf.
Am Ende jedes Jahres werden die Zinsen ausgerechnet und X zahlt 175 € zurück.
Wieviele Jahre muss X zahlen, bis die Schulden abbezahlt sind?

Die Idee

Rechne für ein Jahr nach dem anderen den aktuellen Schuldenstand aus,
solange noch Schulden da sind.
Schaue dann nach, wieviele Jahre vergangen sind.

Im Beispiel (1000€ Kredit, 5% Zinsen, 175€ jährliche Rückzahlung) könnte man das in folgender Tabelle aufschreiben:

	Zinsen	alte Schulden + Zinsen - Rate = Schuldenstand
Anfang 2020		1000.00
Ende 2020	50.00	875.00
Ende 2021	43.75	743.75
Ende 2022	37.19	605.94
Ende 2023	30.30	461.23
Ende 2024	23.06	309.30
Ende 2025	15.46	149.76
Ende 2026	7.49	-17.75

Die Rückzahlungsdauer beträgt also 7 Jahre.

Die Idee

Rechne für ein Jahr nach dem anderen den aktuellen Schuldenstand aus,
solange noch Schulden da sind.
Schaue dann nach, wieviele Jahre vergangen sind.

Im Beispiel (1000€ Kredit, 5% Zinsen, 175€ jährliche Rückzahlung) könnte man das in folgender Tabelle aufschreiben:

	Zinsen	alte Schulden + Zinsen - Rate = Schuldenstand
Anfang 2020		1000.00
Ende 2020	50.00	875.00
Ende 2021	43.75	743.75
Ende 2022	37.19	605.94
Ende 2023	30.30	461.23
Ende 2024	23.06	309.30
Ende 2025	15.46	149.76
Ende 2026	7.49	-17.75

Die Rückzahlungsdauer beträgt also 7 Jahre.

Die Idee

Rechne für ein Jahr nach dem anderen den aktuellen Schuldenstand aus,
solange noch Schulden da sind.
Schaue dann nach, wieviele Jahre vergangen sind.

Im Beispiel (1000€ Kredit, 5% Zinsen, 175€ jährliche Rückzahlung) könnte man das in folgender Tabelle aufschreiben:

	Zinsen	alte Schulden + Zinsen - Rate = Schuldenstand
Anfang 2020		1000.00
Ende 2020	50.00	875.00
Ende 2021	43.75	743.75
Ende 2022	37.19	605.94
Ende 2023	30.30	461.23
Ende 2024	23.06	309.30
Ende 2025	15.46	149.76
Ende 2026	7.49	-17.75

Die Rückzahlungsdauer beträgt also 7 Jahre.

Die Idee

Rechne für ein Jahr nach dem anderen den aktuellen Schuldenstand aus,
solange noch Schulden da sind.
Schaue dann nach, wieviele Jahre vergangen sind.

Im Beispiel (1000€ Kredit, 5% Zinsen, 175€ jährliche Rückzahlung) könnte man das in folgender Tabelle aufschreiben:

	Zinsen	alte Schulden + Zinsen - Rate = Schuldenstand
Anfang 2020		1000.00
Ende 2020	50.00	875.00
Ende 2021	43.75	743.75
Ende 2022	37.19	605.94
Ende 2023	30.30	461.23
Ende 2024	23.06	309.30
Ende 2025	15.46	149.76
Ende 2026	7.49	-17.75

Die Rückzahlungsdauer beträgt also 7 Jahre.

Die Idee

Rechne für ein Jahr nach dem anderen den aktuellen Schuldenstand aus,
solange noch Schulden da sind.
Schaue dann nach, wieviele Jahre vergangen sind.

Im Beispiel (1000€ Kredit, 5% Zinsen, 175€ jährliche Rückzahlung) könnte man das in folgender Tabelle aufschreiben:

	Zinsen	alte Schulden + Zinsen - Rate = Schuldenstand
Anfang 2020		1000.00
Ende 2020	50.00	875.00
Ende 2021	43.75	743.75
Ende 2022	37.19	605.94
Ende 2023	30.30	461.23
Ende 2024	23.06	309.30
Ende 2025	15.46	149.76
Ende 2026	7.49	-17.75

Die Rückzahlungsdauer beträgt also 7 Jahre.

Die Idee

Rechne für ein Jahr nach dem anderen den aktuellen Schuldenstand aus,
solange noch Schulden da sind.
Schaue dann nach, wieviele Jahre vergangen sind.

Im Beispiel (1000€ Kredit, 5% Zinsen, 175€ jährliche Rückzahlung) könnte man das in folgender Tabelle aufschreiben:

	Zinsen	alte Schulden + Zinsen - Rate = Schuldenstand
Anfang 2020		1000.00
Ende 2020	50.00	875.00
Ende 2021	43.75	743.75
Ende 2022	37.19	605.94
Ende 2023	30.30	461.23
Ende 2024	23.06	309.30
Ende 2025	15.46	149.76
Ende 2026	7.49	-17.75

Die Rückzahlungsdauer beträgt also 7 Jahre.

Die Idee

Rechne für ein Jahr nach dem anderen den aktuellen Schuldenstand aus,
solange noch Schulden da sind.
Schaue dann nach, wieviele Jahre vergangen sind.

Im Beispiel (1000€ Kredit, 5% Zinsen, 175€ jährliche Rückzahlung) könnte man das in folgender Tabelle aufschreiben:

	Zinsen	alte Schulden + Zinsen - Rate = Schuldenstand
Anfang 2020		1000.00
Ende 2020	50.00	875.00
Ende 2021	43.75	743.75
Ende 2022	37.19	605.94
Ende 2023	30.30	461.23
Ende 2024	23.06	309.30
Ende 2025	15.46	149.76
Ende 2026	7.49	-17.75

Die Rückzahlungsdauer beträgt also 7 Jahre.

Die Idee

Rechne für ein Jahr nach dem anderen den aktuellen Schuldenstand aus,
solange noch Schulden da sind.
Schaue dann nach, wieviele Jahre vergangen sind.

Im Beispiel (1000€ Kredit, 5% Zinsen, 175€ jährliche Rückzahlung) könnte man das in folgender Tabelle aufschreiben:

	Zinsen	alte Schulden + Zinsen - Rate = Schuldenstand
Anfang 2020		1000.00
Ende 2020	50.00	875.00
Ende 2021	43.75	743.75
Ende 2022	37.19	605.94
Ende 2023	30.30	461.23
Ende 2024	23.06	309.30
Ende 2025	15.46	149.76
Ende 2026	7.49	-17.75

Die Rückzahlungsdauer beträgt also 7 Jahre.

Die Idee

Rechne für ein Jahr nach dem anderen den aktuellen Schuldenstand aus,
solange noch Schulden da sind.
Schaue dann nach, wieviele Jahre vergangen sind.

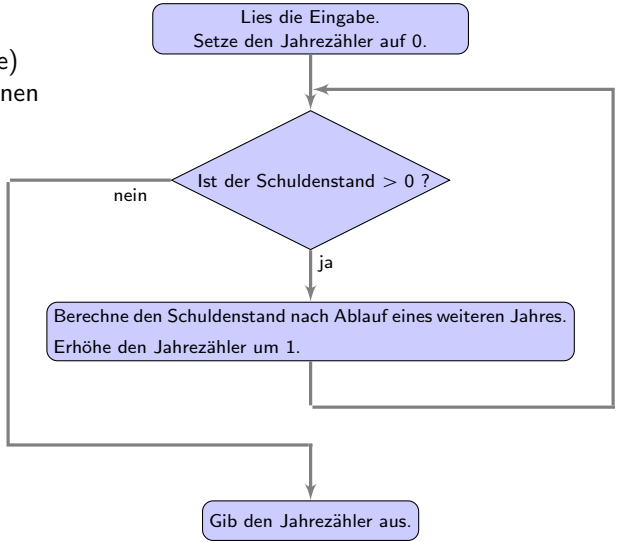
Im Beispiel (1000€ Kredit, 5% Zinsen, 175€ jährliche Rückzahlung) könnte man das in folgender Tabelle aufschreiben:

	Zinsen	alte Schulden + Zinsen - Rate = Schuldenstand
Anfang 2020		1000.00
Ende 2020	50.00	875.00
Ende 2021	43.75	743.75
Ende 2022	37.19	605.94
Ende 2023	30.30	461.23
Ende 2024	23.06	309.30
Ende 2025	15.46	149.76
Ende 2026	7.49	-17.75

Die Rückzahlungsdauer beträgt also 7 Jahre.

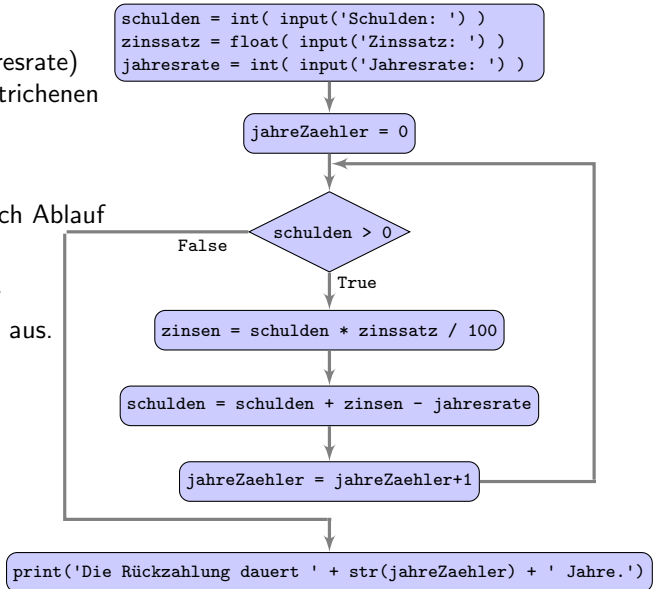
Die grobe Struktur des Programms

1. Lies die Eingabe (Schulden, Zinssatz, Jahresrate) und setze einen Zähler für die bereits verstrichenen Jahre auf 0.
2. Solange der Schuldenstand > 0 ist:
   - **2.1** berechne den Schuldenstand nach Ablauf eines weiteren Jahres und speichere, dass ein weiteres Jahr verstrichen ist.
3. Gib den Zähler für die verstrichenen Jahre aus.



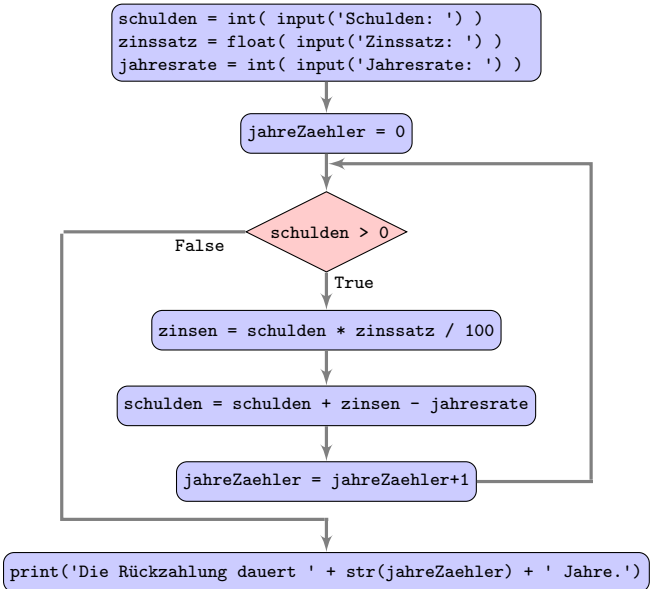
Die etwas feinere Struktur des Programms

1. Lies die Eingabe (Schulden, Zinssatz, Jahresrate) und setze einen Zähler für die bereits verstrichenen Jahre auf 0.
2. Solange die Schulden > 0 sind:
   - **2.1** berechne den Stand der Schulden nach Ablauf eines weiteren Jahres und speichere, dass ein weiteres Jahr verstrichen ist.
3. Gib den Zähler für die verstrichenen Jahre aus.



Das Programm `rueckzahlungsdauer.py`

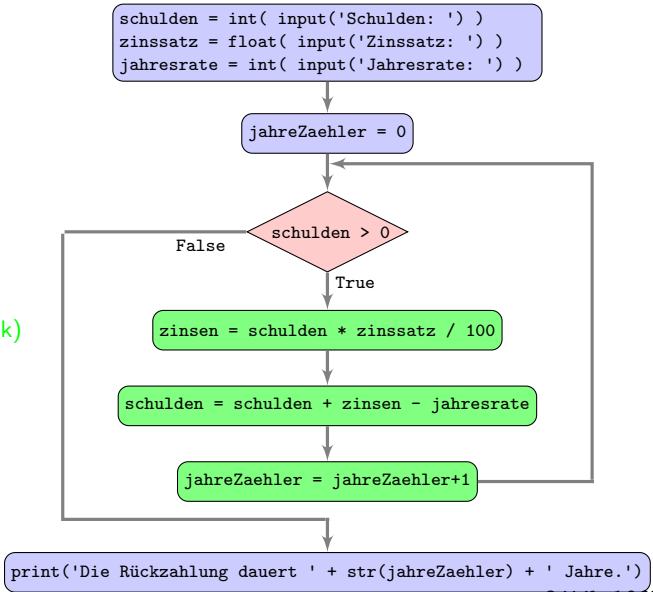
Schleifenbedingung



Das Programm `rueckzahlungsdauer.py`

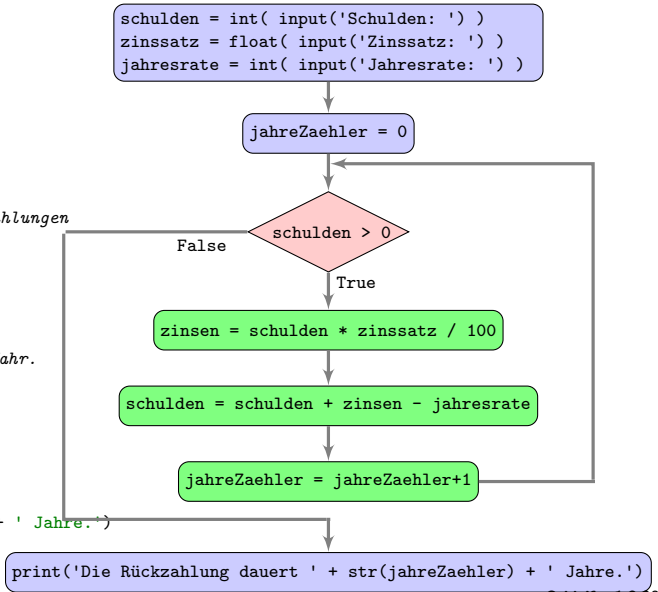
Schleifenbedingung

Schleifenrumpf (Block)



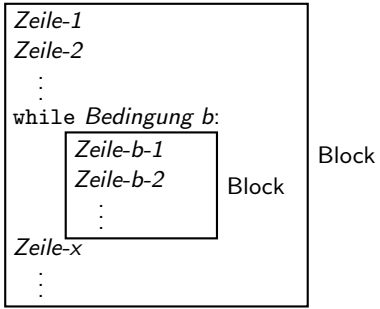
Das Programm rueckzahlungsdauer.py

```
# Lies Schulden, Zinssatz und Jahresrate ein.
schulden = int( input('Schulden: ') )
zinssatz = float( input('Zinssatz: ') )
jahresrate = int( input('Jahresrate: ') )

jahreZaehler = 0   # Zähler für die Jahre mit Ratenzahlungen

# Solange noch Schulden da sind, wird ihr Stand
# nach einem weiteren Jahr berechnet.
# Die Anzahl dieser Jahre wird mitgezählt.
while schulden > 0 :
   # Berechne den Schuldenstand nach einem weiteren Jahr.
   zinsen = schulden * zinssatz / 100
   schulden = schulden + zinsen - jahresrate
   # Zähle den Jahrezähler 1 weiter.
   jahreZaehler = jahreZaehler + 1

# Die Dauer der Rückzahlung steht in jahreZaehler.
print('Die Rückzahlung dauert ' + str(jahreZaehler) + ' Jahre.')
```



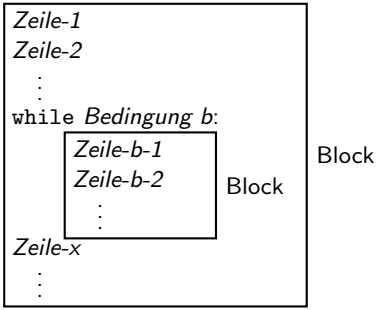
Blockstruktur von `while`-Schleifen

```
# rueckzahlungsdauer.py
#-------------------------------------------------------------------------
# Das Programm liest von der Kommandozeile die Werte
# schulden (int), zinssatz (float) und jahresrate (int).
# Es wird berechnet, wieviele Jahre die jahresrate bezahlt werden muss,
# bis die schulden beim angegebenen zinssatz beglichen sind.
#-------------------------------------------------------------------------
# Lies Schulden, Zinssatz und Jahresrate ein.
schulden = int( input('Schulden: ') )
zinssatz = float( input('Zinssatz: ') )
jahresrate = int( input('Jahresrate: ') )

jahreZaehler = 0   # Zähler für die Jahre mit Ratenzahlungen

# Solange noch Schulden da sind, wird ihr Stand nach einem weiteren Jahr
# berechnet. Die Anzahl dieser Jahre wird mitgezählt.
while schulden > 0 :
   # Berechne den Schuldenstand nach einem weiteren Jahr.
   zinsen = schulden * zinssatz / 100
   schulden = schulden + zinsen - jahresrate
   # Zähle den Jahrezähler 1 weiter.
   jahreZaehler = jahreZaehler + 1

# Die Dauer der Rückzahlung steht in jahreZaehler.
print('Die Rückzahlung dauert ' + str(jahreZaehler) + ' Jahre.')
#-------------------------------------------------------------------------
```



Blockstruktur von `while`-Schleifen

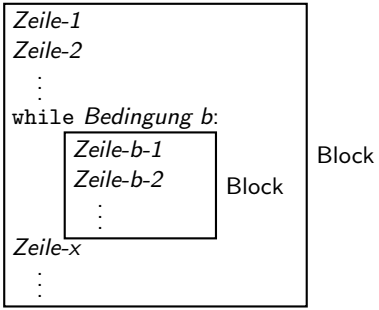
```
# rueckzahlungsdauer.py
#-------------------------------------------------------------------------
# Das Programm liest von der Kommandozeile die Werte
# schulden (int), zinssatz (float) und jahresrate (int).
# Es wird berechnet, wieviele Jahre die jahresrate bezahlt werden muss,
# bis die schulden beim angegebenen zinssatz beglichen sind.
#-------------------------------------------------------------------------
# Lies Schulden, Zinssatz und Jahresrate ein.
schulden = int( input('Schulden: ') )
zinssatz = float( input('Zinssatz: ') )
jahresrate = int( input('Jahresrate: ') )

jahreZaehler = 0   # Zähler für die Jahre mit Ratenzahlungen

# Solange noch Schulden da sind, wird ihr Stand nach einem weiteren Jahr
# berechnet. Die Anzahl dieser Jahre wird mitgezählt.
while schulden > 0 :
   # Berechne den Schuldenstand nach einem weiteren Jahr.
   zinsen = schulden * zinssatz / 100
   schulden = schulden + zinsen - jahresrate
   # Zähle den Jahrezähler 1 weiter.
   jahreZaehler = jahreZaehler + 1

# Die Dauer der Rückzahlung steht in jahreZaehler.
print('Die Rückzahlung dauert ' + str(jahreZaehler) + ' Jahre.')
#-------------------------------------------------------------------------
```




```
mundhenk@ma3:~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 rueckzahlungsdauer.py
Schulden: 1000
Zinssatz in Prozent: 5
Jahresrate: 175
Die Rückzahlung dauert 7 Jahre.
mundhenk@ma3:~/Python/VL02$
```

Blockstruktur von `while`-Schleifen

```
# rueckzahlungsdauer.py
#-------------------------------------------------------------------------
# Das Programm liest von der Kommandozeile die Werte
# schulden (int), zinssatz (float) und jahresrate (int).
# Es wird berechnet, wieviele Jahre die jahresrate bezahlt werden muss,
# bis die schulden beim angegebenen zinssatz beglichen sind.
#-------------------------------------------------------------------------
# Lies Schulden, Zinssatz und Jahresrate ein.
schulden = int( input('Schulden: ') )
zinssatz = float( input('Zinssatz: ') )
jahresrate = int( input('Jahresrate: ') )

jahreZaehler = 0   # Zähler für die Jahre mit Ratenzahlungen

# Solange noch Schulden da sind, wird ihr Stand nach einem weiteren Jahr
# berechnet. Die Anzahl dieser Jahre wird mitgezählt.
while schulden > 0 :
   # Berechne den Schuldenstand nach einem weiteren Jahr.
   zinsen = schulden * zinssatz / 100
   schulden = schulden + zinsen - jahresrate
   # Zähle den Jahrezähler 1 weiter.
   jahreZaehler = jahreZaehler + 1

# Die Dauer der Rückzahlung steht in jahreZaehler.
print('Die Rückzahlung dauert ' + str(jahreZaehler) + ' Jahre.')
#-------------------------------------------------------------------------
```




```
mundhenk@ma3:~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 rueckzahlungsdauer.py
Schulden: 1000
Zinssatz in Prozent: 5
Jahresrate: 175
Die Rückzahlung dauert 7 Jahre.
mundhenk@ma3:~/Python/VL02$ python3 rueckzahlungsdauer.py
Schulden: 50000
Zinssatz in Prozent: 4
Jahresrate: 6000
Die Rückzahlung dauert 11 Jahre.
mundhenk@ma3:~/Python/VL02$
```

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also '++++–+––––+'.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also '++++-+------+'.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also '++++-+------+'.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also '++++-+------+'.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also '++++-+------+'.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also '++++-+------+'.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also '++++-+------+'.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also '++++-+------+'.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also '++++–+––––+'.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also '++++–+––––+'.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also `'++++–+––––+'`.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also `'++++-+-----+'`.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also `'++++-+------'`.

Programmier-Auftrag: bestimme alle Teiler einer Zahl und gib sie als String aus + und – aus

Die Teiler von 12:

	1	2	3	4	5	6	7	8	9	10	11	12
Ergebnis-String	+	+	+	+	–	+	–	–	–	–	–	+

Bei Eingabe 12 ist das Ergebnis also `'++++-+-----+'`.

Die Idee

Zur Bestimmung aller Teiler einer ganzen Zahl *a*

geht man alle Zahlen *z* von 1 bis *a* durch

und hängt dabei ein `'+'` an den Ergebnisstring an, falls *z* ein Teiler von *a* ist

bzw. hängt dabei ein `'-'` an den Ergebnisstring an, falls *z* kein Teiler von *a* ist

(Der Ergebnisstring war am Anfang leer.)

Die grobe Struktur des Programms

1. Lies die Eingabe `a` und setze den Ergebnisstring auf seinen Anfangswert `''`.
2. Benutze eine Variable `zaehler`, um alle Werte `1...a` durchzuzählen.
3. Solange `zaehler <= a`, wiederhole:
   - **3.1** Abhängig davon, ob `zaehler` ein Teiler von `a` ist,
     wird ein `+` bzw. ein `-` an den Ergebnisstring angehängt.
     Anschließend wird der Wert von `zaehler` wird um 1 erhöht.
4. Gib den Ergebnisstring aus.

Das Programm alleteiler.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
zaehler = 1
while zaehler <= a:
   # Falls zaehler ein Teiler von a ist,
   #   dann wird ein + an ergebnis angehängt,
   # sonst wird ein - an ergebnis angehängt.
   if a % zaehler == 0:
      ergebnis = ergebnis + '+'
   else:
      ergebnis = ergebnis + '-'

   # zaehler wird um 1 erhöht.
   zaehler = zaehler + 1

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Das Programm alleteiler.py

Die Blockstruktur des Programms.

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
zaehler = 1
while zaehler <= a:
   # Falls zaehler ein Teiler von a ist,
   #   dann wird ein + an ergebnis angehängt,
   # sonst wird ein - an ergebnis angehängt.
   if a % zaehler == 0:
      ergebnis = ergebnis + '+'
   else:
      ergebnis = ergebnis + '-'

   # zaehler wird um 1 erhöht.
   zaehler = zaehler + 1

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Das Programm alleteiler.py

Die Blockstruktur des Programms.

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
zaehler = 1
while zaehler <= a:
   # Falls zaehler ein Teiler von a ist,
   #   dann wird ein + an ergebnis angehängt,
   # sonst wird ein - an ergebnis angehängt.
   if a % zaehler == 0:
      ergebnis = ergebnis + '+'
   else:
      ergebnis = ergebnis + '-'

   # zaehler wird um 1 erhöht.
   zaehler = zaehler + 1

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Das Programm alleteiler.py

Die Blockstruktur des Programms.

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
zaehler = 1
while zaehler <= a:
  # Falls zaehler ein Teiler von a ist,
  #   dann wird ein + an ergebnis angehängt,
  # sonst wird ein - an ergebnis angehängt.
  if a % zaehler == 0:
    ergebnis = ergebnis + '+'
  else:
    ergebnis = ergebnis + '-'

  # zaehler wird um 1 erhöht.
  zaehler = zaehler + 1

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Das Programm alleteiler.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
zaehler = 1
while zaehler <= a:
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an ergebnis angehängt,
    # sonst wird ein - an ergebnis angehängt.
    if a % zaehler == 0:
        ergebnis = ergebnis + '+'
    else:
        ergebnis = ergebnis + '-'

    # zaehler wird um 1 erhöht.
    zaehler = zaehler + 1

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Die Blockstruktur des Programms.

```
mundhenk@ma3: ~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 alleteiler.py
Berechne alle Teiler von 9
+-+-----+
mundhenk@ma3:~/Python/VL02$
```

Das Programm alleteiler.py

Die Blockstruktur des Programms.

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
zaehler = 1
while zaehler <= a:
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an ergebnis angehängt,
    # sonst wird ein - an ergebnis angehängt.
    if a % zaehler == 0:
        ergebnis = ergebnis + '+'
    else:
        ergebnis = ergebnis + '-'

    # zaehler wird um 1 erhöht.
    zaehler = zaehler + 1

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

```
mundhenk@ma3: ~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 alleteiler.py
Berechne alle Teiler von 9
+-+-----+
mundhenk@ma3:~/Python/VL02$ python3 alleteiler.py
Berechne alle Teiler von 27
+-+-----+-----------------+
mundhenk@ma3:~/Python/VL02$
```

Das Programm alleteiler.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
zaehler = 1
while zaehler <= a:
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an ergebnis angehängt,
    # sonst wird ein - an ergebnis angehängt.
    if a % zaehler == 0:
        ergebnis = ergebnis + '+'
    else:
        ergebnis = ergebnis + '-'

    # zaehler wird um 1 erhöht.
    zaehler = zaehler + 1

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Die Blockstruktur des Programms.

```
mundhenk@ma3: ~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 alleteiler.py
Berechne alle Teiler von 9
+-+-----+
mundhenk@ma3:~/Python/VL02$ python3 alleteiler.py
Berechne alle Teiler von 27
+-+-----+-----------------+
mundhenk@ma3:~/Python/VL02$ python3 alleteiler.py
Berechne alle Teiler von 37
+-----------------------------------+
mundhenk@ma3:~/Python/VL02$
```

Das Programm alleteiler.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
zaehler = 1
while zaehler <= a:
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an ergebnis angehängt,
    # sonst wird ein - an ergebnis angehängt.
    if a % zaehler == 0:
        ergebnis = ergebnis + '+'
    else:
        ergebnis = ergebnis + '-'

    # zaehler wird um 1 erhöht.
    zaehler = zaehler + 1

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Die Blockstruktur des Programms.

Anweisungen der Form

```
zaehler = zaehler + 1
```

kann man abgekürzt schreiben als

```
zaehler += 1.
```

Das Programm alleteiler.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
zaehler = 1
while zaehler <= a:
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an ergebnis angehängt,
    # sonst wird ein - an ergebnis angehängt.
    if a % zaehler == 0:
        ergebnis = ergebnis + '+'
    else:
        ergebnis = ergebnis + '-'

    # zaehler wird um 1 erhöht.
    zaehler = zaehler + 1

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Die Blockstruktur des Programms.

Anweisungen der Form

`ergebnis = ergebnis + '+'`

kann man abgekürzt schreiben als

`ergebnis += '+'`.

3 Schleifen – `for`

In `alleteiler.py` wurde die Schleife benutzt,
um die Variable `zaehler` von 1 bis a durchzuzählen.

Es gab der Reihe nach
einen Schleifendurchlauf, bei dem `zaehler` Wert 1 hatte,
einen Schleifendurchlauf, bei dem `zaehler` Wert 2 hatte,
einen Schleifendurchlauf, bei dem `zaehler` Wert 3 hatte,
...
einen Schleifendurchlauf, bei dem `zaehler` Wert a hatte.

Schleifen dieser Art werden häufig verwendet.
Deshalb gibt es eine spezielle Schleife dafür: die `for`-Schleife.

3 Schleifen – `for`

In `alleteiler.py` wurde die Schleife benutzt,
um die Variable `zaehler` von `1` bis `a` durchzuzählen.

Es gab der Reihe nach
einen Schleifendurchlauf, bei dem `zaehler` Wert 1 hatte,
einen Schleifendurchlauf, bei dem `zaehler` Wert 2 hatte,
einen Schleifendurchlauf, bei dem `zaehler` Wert 3 hatte,
...
einen Schleifendurchlauf, bei dem `zaehler` Wert `a` hatte.

Schleifen dieser Art werden häufig verwendet.
Deshalb gibt es eine spezielle Schleife dafür: die `for`-Schleife.

Das Programm alleteiler-for.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende
# an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
for zaehler in range(1,a+1):
   # Falls zaehler ein Teiler von a ist,
   #   dann wird ein + an ergebnis angehängt,
   # sonst wird ein - an ergebnis angehängt.
   if a % zaehler == 0:
      ergebnis += '+'
   else:
      ergebnis += '-'

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Das Programm alleteiler-for.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende
# an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
for zaehler in range(1,a+1):
   # Falls zaehler ein Teiler von a ist,
   #   dann wird ein + an ergebnis angehängt,
   # sonst wird ein - an ergebnis angehängt.
   if a % zaehler == 0:
      ergebnis += '+'
   else:
      ergebnis += '-'

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Die Blockstruktur des Programms.

Das Programm alleteiler-for.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende
# an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
for zaehler in range(1,a+1):
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an ergebnis angehängt,
    # sonst wird ein - an ergebnis angehängt.
    if a % zaehler == 0:
        ergebnis += '+'
    else:
        ergebnis += '-'

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Das Programm alleteiler-for.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende
# an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
for zaehler in range(1,a+1):
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an ergebnis angehängt,
    # sonst wird ein - an ergebnis angehängt.
    if a % zaehler == 0:
        ergebnis += '+'
    else:
        ergebnis += '-'

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

Das Programm alleteiler-for.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende
# an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
for zaehler in range(1,a+1):
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an ergebnis angehängt,
    # sonst wird ein - an ergebnis angehängt.
    if a % zaehler == 0:
        ergebnis += '+'
    else:
        ergebnis += '-'

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

`zaehler in range(1,9)` durchläuft für `zaehler` die Werte 1, 2, 3, 4, 5, 6, 7, 8.

Das Programm alleteiler-for.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende
# an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
for zaehler in range(1,a+1):
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an ergebnis angehängt,
    # sonst wird ein - an ergebnis angehängt.
    if a % zaehler == 0:
        ergebnis += '+'
    else:
        ergebnis += '-'

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

`zaehler in range(1,a+1)` durchläuft für `zaehler` die Werte 1, 2, 3, ..., a.

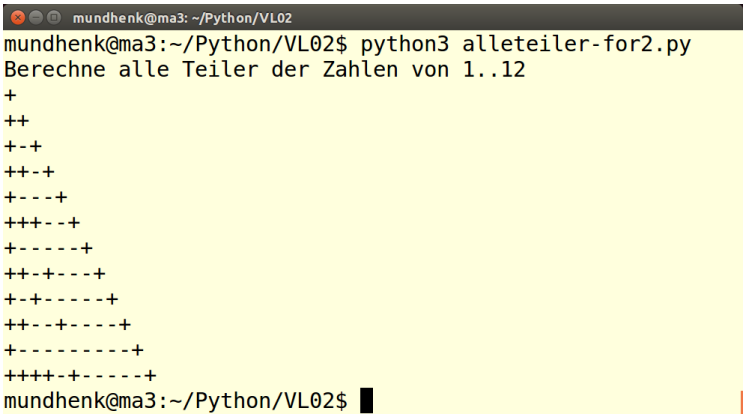
Das Programm alleteiler-for.py

```
# Lies a (int) von der Konsole.
a = int( input('Berechne alle Teiler von ') )

# ergebnis ist der String, der am Ende
# an Stelle i ein + hat, falls i Teiler von a ist, und ein - sonst.
ergebnis = ''

# zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
for zaehler in range(1,a+1):
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an ergebnis angehängt,
    # sonst wird ein - an ergebnis angehängt.
    if a % zaehler == 0:
        ergebnis += '+'
    else:
        ergebnis += '-'

# Das Ergebnis wird ausgegeben.
print(ergebnis)
```

`zaehler in range(a)` durchläuft für `zaehler` die Werte 0, 1, 2, ..., a-1.

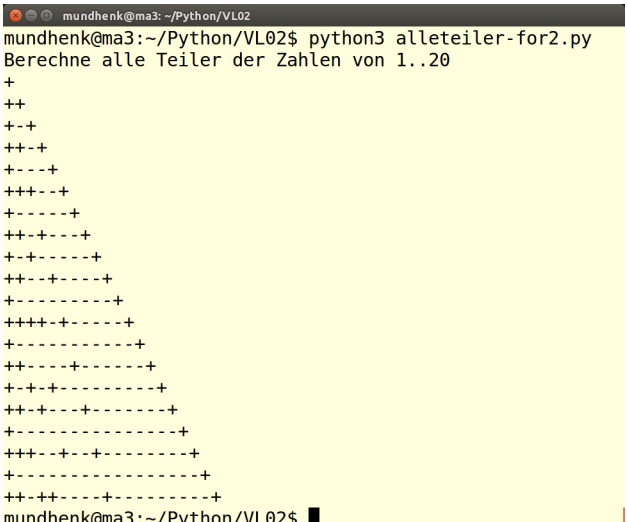
Programmier-Auftrag: bestimme für jede Zahl `1...m` alle Teiler

Schleifen ineinander schachteln



Programmier-Auftrag: bestimme für jede Zahl `1...m` alle Teiler

Schleifen ineinander schachteln

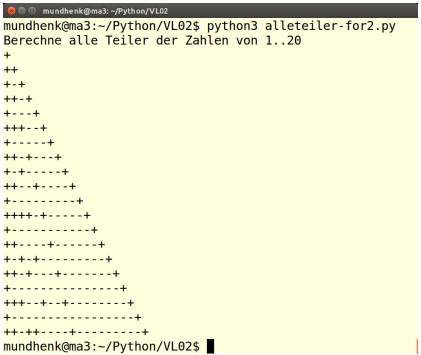


Die Idee

Gehe alle Zahlen von `1` bis `m` durch

und gib für jede dieser Zahlen alle Teiler aus

(das kann man wie mit `alleteiler-for.py` machen).



Die grobe Struktur des Programmes

1. Lies die Eingabe `m` ein.
2. Für jedes a im Bereich `1...m`:
   - **2.1** berechne die Teiler von a und gib sie aus
     wie in `alleteiler-for.py`

Das Programm alleteiler-for2.py

```
# alleteiler-for2.py
#-------------------------------------------------------------------------
# Das Programm liest int-Wert m von der Konsole ein.
# Es gibt für alle Zahlen a von 1..m alle Teiler von a als +/--String aus.
#-------------------------------------------------------------------------
# Lies m (int) von der Konsole.
m = int( input('Berechne alle Teiler der Zahlen von 1..') )
# a soll alle Zahlen im Bereich 1..m durchlaufen.
for a in range(1,m+1):
  # ergebnis ist der String aus +/-, der alle Teiler von a darstellt.
  ergebnis = ''

  # zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
  for zaehler in range(1,a+1):
     # Falls zaehler ein Teiler von a ist,
     #   dann wird ein + an erg angehängt,
     # sonst wird ein - an erg angehängt.
     if a % zaehler == 0:
        ergebnis += '+'
     else:
        ergebnis += '-'

  # Das Ergebnis wird ausgegeben.
  print(ergebnis)
#-------------------------------------------------------------------------
```

Das Programm alleteiler-for2.py

```
# alleteiler-for2.py
#-------------------------------------------------------------------------
# Das Programm liest int-Wert m von der Konsole ein.
# Es gibt für alle Zahlen a von 1..m alle Teiler von a als +/--String aus.
#-------------------------------------------------------------------------
# Lies m (int) von der Konsole.
m = int( input('Berechne alle Teiler der Zahlen von 1..') )
# a soll alle Zahlen im Bereich 1..m durchlaufen.
for a in range(1,m+1):
  # ergebnis ist der String aus +/-, der alle Teiler von a darstellt.
  ergebnis = ''

  # zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
  for zaehler in range(1,a+1):
     # Falls zaehler ein Teiler von a ist,
     #   dann wird ein + an erg angehängt,
     # sonst wird ein - an erg angehängt.
     if a % zaehler == 0:
        ergebnis += '+'
     else:
        ergebnis += '-'

  # Das Ergebnis wird ausgegeben.
  print(ergebnis)
#-------------------------------------------------------------------------
```

Die Blockstruktur des Programmes.

Das Programm alleteiler-for2.py

```
# alleteiler-for2.py
#-------------------------------------------------------------------------
# Das Programm liest int-Wert m von der Konsole ein.
# Es gibt für alle Zahlen a von 1..m alle Teiler von a als +/--String aus.
#-------------------------------------------------------------------------
# Lies m (int) von der Konsole.
m = int( input('Berechne alle Teiler der Zahlen von 1..') )
# a soll alle Zahlen im Bereich 1..m durchlaufen.
for a in range(1,m+1):
   # ergebnis ist der String aus +/-, der alle Teiler von a darstellt.
   ergebnis = ''

   # zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
   for zaehler in range(1,a+1):
      # Falls zaehler ein Teiler von a ist,
      #   dann wird ein + an erg angehängt,
      # sonst wird ein - an erg angehängt.
      if a % zaehler == 0:
         ergebnis += '+'
      else:
         ergebnis += '-'

   # Das Ergebnis wird ausgegeben.
   print(ergebnis)
#-------------------------------------------------------------------------
```

Die Blockstruktur des Programmes.

Das Programm alleteiler-for2.py

```
# alleteiler-for2.py
#-------------------------------------------------------------------------
# Das Programm liest int-Wert m von der Konsole ein.
# Es gibt für alle Zahlen a von 1..m alle Teiler von a als +/--String aus.
#-------------------------------------------------------------------------
# Lies m (int) von der Konsole.
m = int( input('Berechne alle Teiler der Zahlen von 1..') )
# a soll alle Zahlen im Bereich 1..m durchlaufen.
for a in range(1,m+1):
  # ergebnis ist der String aus +/-, der alle Teiler von a darstellt.
  ergebnis = ''

  # zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
  for zaehler in range(1,a+1):
     # Falls zaehler ein Teiler von a ist,
     #   dann wird ein + an erg angehängt,
     # sonst wird ein - an erg angehängt.
     if a % zaehler == 0:
        ergebnis += '+'
     else:
        ergebnis += '-'

  # Das Ergebnis wird ausgegeben.
  print(ergebnis)
#-------------------------------------------------------------------------
```

Die Blockstruktur des Programmes.

Das Programm alleteiler-for2.py

```
# alleteiler-for2.py
#-------------------------------------------------------------------------
# Das Programm liest int-Wert m von der Konsole ein.
# Es gibt für alle Zahlen a von 1..m alle Teiler von a als +/--String aus.
#-------------------------------------------------------------------------
# Lies m (int) von der Konsole.
m = int( input('Berechne alle Teiler der Zahlen von 1..') )
# a soll alle Zahlen im Bereich 1..m durchlaufen.
for a in range(1,m+1):
  # ergebnis ist der String aus +/-, der alle Teiler von a darstellt.
  ergebnis = ''

  # zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen.
  for zaehler in range(1,a+1):
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an erg angehängt,
    # sonst wird ein - an erg angehängt.
    if a % zaehler == 0:
      ergebnis += '+'
    else:
      ergebnis += '-'

  # Das Ergebnis wird ausgegeben.
  print(ergebnis)
#-------------------------------------------------------------------------
```

Die Blockstruktur des Programmes.

Das Programm alleteiler-for2.py

```
# alleteiler-for2.py
#-------------------------------------------------------------------------
# Das Programm liest int-Wert m von der Konsole ein.
# Es gibt für alle Zahlen a von 1..m alle Teiler von a als +/--Str
#-------------------------------------------------------------------------
# Lies m (int) von der Konsole.
m = int( input('Berechne alle Teiler der Zahlen von 1..') )
# a soll alle Zahlen im Bereich 1..m durchlaufen.
for a in range(1,m+1):
  # ergebnis ist der String aus +/-, der alle Teiler von a darstel
  ergebnis = ''

  # zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen
  for zaehler in range(1,a+1):
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an erg angehängt,
    # sonst wird ein - an erg angehängt.
    if a % zaehler == 0:
      ergebnis += '+'
    else:
      ergebnis += '-'

  # Das Ergebnis wird ausgegeben.
  print(ergebnis)
#-------------------------------------------------------------------------
```

```
mundhenk@ma3:~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 alleteiler-for2.py
Berechne alle Teiler der Zahlen von 1..20
+
++
+-+
++-+
+---+
+++--+
+-----+
++-+---+
+-+-----+
++--+----+
+---------+
++++-+-----+
+----------+
++----+------+
+-+-+---------+
++-+---+-------+
+---------------+
+++--+--+--------+
+-----------------+
++-++----+---------+
mundhenk@ma3:~/Python/VL02$
```

Das Programm alleteiler-for2.py

```
# alleteiler-for2.py
#-----------------------------------------------------------------
# Das Programm liest int-Wert m von der Konsole ein.
# Es gibt für alle Zahlen a von 1..m alle Teiler von a als +/--Str
#-----------------------------------------------------------------
# Lies m (int) von der Konsole.
m = int( input('Berechne alle Teiler der Zahlen von 1..') )
# a soll alle Zahlen im Bereich 1..m durchlaufen.
for a in range(1,m+1):
  # ergebnis ist der String aus +/-, der alle Teiler von a darstel
  ergebnis = ''

  # zaehler soll alle möglichen Teiler im Bereich 1..a durchlaufen
  for zaehler in range(1,a+1):
    # Falls zaehler ein Teiler von a ist,
    #   dann wird ein + an erg angehängt,
    # sonst wird ein - an erg angehängt.
    if a % zaehler == 0:
      ergebnis += '+'
    else:
      ergebnis += '-'

  # Das Ergebnis wird ausgegeben.
  print(ergebnis)
#-----------------------------------------------------------------
```

```
mundhenk@ma3: ~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 alleteiler-for2.py
Berechne alle Teiler der Zahlen von 1..26
+
++
+-+
++-+
+---+
+++--+
+-----+
++-+---+
+-+-----+
++--+----+
+---------+
++++-+-----+
+-----------+
++----+------+
+-+-+---------+
++-+---+-------+
+---------------+
+++--+--+--------+
+-----------------+
++-++----+---------+
+-+---+-------------+
++--------+----------+
+---------------------+
++++-+-+---+-----------+
+---+-------------------+
++----------+------------+
mundhenk@ma3:~/Python/VL02$
```

Programmier-Auftrag: berechne die Wahrscheinlichkeit, mit zwei Würfeln Augensumme 7 zu würfeln

Mit zwei Würfeln kann man Zahlen im Bereich $2 \ldots 12$ würfeln.

Aus Erfahrung weiß man, dass die Summe 2 seltener gewürfelt wird als die Summe 7.

Wie groß ist die Wahrscheinlichkeit, die Summe 7 zu würfeln?

Die Idee

Statt die Wahrscheinlichkeit exakt auszurechnen,
simulieren wir ein Experiment mit einem Programm.

Das Experiment besteht aus dem Wurf von zwei Würfeln.
Das Experiment ist erfolgreich, wenn die Würfelsumme 7 ist.
Anderenfalls ist das Experiment nicht erfolgreich.

Dieses Experiment wiederholen wir z.B. 100000-mal
und zählen, wie oft das Experiment erfolgreich ist.

Die experimentell bestimmte Wahrscheinlichkeit, die Augensumme 7 zu würfeln,
ist dann der Quotient

$$\frac{\text{Anzahl der erfolgreichen Experimente}}{\text{Anzahl aller Experimente}}\ .$$

Je öfter das Experiment wiederholt wird,
desto wahrscheinlicher ist es, die „richtige“ Wahrscheinlichkeit herauszubekommen.

Die grobe Struktur des Programms

1. Lies die Anzahl n der Wiederholungen des Experiments ein.
2. Wiederhole n-mal:
   - **2.1** Experiment: würfele mit zwei Würfeln
   - **2.2** Auswertung: das Experiment war erfolgreich, falls Augensumme 7 gewürfelt wurde
3. Gib „*Anzahl erfolgreicher Experimente* / n“ aus.

Die grobe Struktur des Programms

1. Die Experimentreihe wird vorbereitet.
   Lies die Anzahl n der Wiederholungen des Experiments ein.
2. Die Experimentreihe wird durchgeführt.
   Wiederhole n-mal:
   - **2.1** Experiment: würfele mit zwei Würfeln
   - **2.2** Auswertung: das Experiment war erfolgreich, falls Augensumme 7 gewürfelt wurde
3. Die Experimentreihe wird ausgewertet.
   Gib „*Anzahl erfolgreicher Experimente* / n“ aus.

Wie kann man mit Python würfeln?

Das Modul `random` stellt Funktionen für zufällige Auswahlen bereit.

`randint(a,b)`	ein zufällig gewählter `int`-Wert aus dem Bereich `a`...`b` für `int`-Werte `a` und `b`
`randrange(a,b)`	ein zufällig gewählter `int`-Wert aus dem Bereich `a`...`b-1` für `int`-Werte `a` und `b`
`random()`	ein zufällig gewählter `float`-Wert aus dem Intervall `0`...`1` (ohne 1).

Die Funktionswerte sind gleichverteilt.

Ein Wurf mit einem Würfel entspricht einem Ausdruck

`random.randint(1,6)` oder

`random.randrange(1,7)`.

Das Programm siebenwuerfeln.py

```
import random

### Die Experimentreihe wird vorbereitet.
# Die Anzahl der Experimente wird von der Kommandozeile eingelesen.
n = int( input('Anzahl der Wiederholungen des Experiments: ') )

erfolgsZaehler = 0       # zählt die erfolgreichen Experimente

### Die Experimentreihe wird durchgeführt.
for i in range(n):     # es werden n Experimente durchgeführt
  # Das Experiment: zwei Würfel werden geworfen.
  wuerfel1 = random.randint(1,6)
  wuerfel2 = random.randint(1,6)
  # Auswertung des Experiments: Das Experiment ist erfolgreich, wenn die Würfelsumme 7 ist.
  if wuerfel1 + wuerfel2 == 7:
        erfolgsZaehler += 1

### Das Ergebnis der Experimentreihe wird ausgegeben.
erfolgsWkeit = erfolgsZaehler/n
print('Die Wahrscheinlichkeit für ein erfolgreiches Experiment ist ' + str(erfolgsWkeit) + '.' )
print('D.h. etwa einer von ' + str(round(1/erfolgsWkeit)) + ' Würfen ist erfolgreich.' )
```

```
#-------------------------------------------------------------------------------
# siebenwuerfeln.py
#-------------------------------------------------------------------------------
import random

### Die Experimentreihe wird vorbereitet.
# Die Anzahl der Experimente wird von der Kommandozeile eingelesen.
n = int( input('Anzahl der Wiederholungen des Experiments: ') )

erfolgsZaehler = 0       # zählt die erfolgreichen Experimente

### Die Experimentreihe wird durchgeführt.
for i in range(n):      # es werden n Experimente durchgeführt
  # Das Experiment: zwei Würfel werden geworfen.
  wuerfel1 = random.randint(1,6)
  wuerfel2 = random.randint(1,6)
  # Auswertung des Experiments: Das Experiment ist erfolgreich, wenn die Würfelsumme 7 ist.
  if wuerfel1 + wuerfel2 == 7:
        erfolgsZaehler += 1

### Das Ergebnis der Experimentreihe wird ausgegeben.
erfolgsWkeit = erfolgsZaehler/n
print('Die Wahrscheinlichkeit für ein erfolgreiches Experime
print('D.h. etwa einer von ' + str(round(1/erfolgsWkeit)) +
```

```
mundhenk@ma3: ~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 siebenwuerfeln.py
Anzahl der Wiederholungen des Experiments: 100
Die Wahrscheinlichkeit für ein erfolgreiches Experiment ist 0.25.
D.h. etwa einer von 4 Würfen ist erfolgreich.
mundhenk@ma3:~/Python/VL02$
```

```
#-------------------------------------------------------------------------------
# siebenwuerfeln.py
#-------------------------------------------------------------------------------
import random

### Die Experimentreihe wird vorbereitet.
# Die Anzahl der Experimente wird von der Kommandozeile eingelesen.
n = int( input('Anzahl der Wiederholungen des Experiments: ') )

erfolgsZaehler = 0       # zählt die erfolgreichen Experimente

### Die Experimentreihe wird durchgeführt.
for i in range(n):      # es werden n Experimente durchgeführt
  # Das Experiment: zwei Würfel werden geworfen.
  wuerfel1 = random.randint(1,6)
  wuerfel2 = random.randint(1,6)
  # Auswertung des Experiments: Das Experiment ist erfolgreich, wenn die Würfelsumme 7 ist.
  if wuerfel1 + wuerfel2 == 7:
        erfolgsZaehler += 1

### Das Ergebnis der Experimentreihe wird ausgegeben.
erfolgsWkeit = erfolgsZaehler/n
print('Die Wahrscheinlichkeit für ein erfolgreiches Experime
print('D.h. etwa einer von ' + str(round(1/erfolgsWkeit)) +
```

```
mundhenk@ma3: ~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 siebenwuerfeln.py
Anzahl der Wiederholungen des Experiments: 100
Die Wahrscheinlichkeit für ein erfolgreiches Experiment ist 0.14.
D.h. etwa einer von 7 Würfen ist erfolgreich.
mundhenk@ma3:~/Python/VL02$
```

```
#-------------------------------------------------------------------------------
# siebenwuerfeln.py
#-------------------------------------------------------------------------------
import random

### Die Experimentreihe wird vorbereitet.
# Die Anzahl der Experimente wird von der Kommandozeile eingelesen.
n = int( input('Anzahl der Wiederholungen des Experiments: ') )

erfolgsZaehler = 0       # zählt die erfolgreichen Experimente

### Die Experimentreihe wird durchgeführt.
for i in range(n):      # es werden n Experimente durchgeführt
  # Das Experiment: zwei Würfel werden geworfen.
  wuerfel1 = random.randint(1,6)
  wuerfel2 = random.randint(1,6)
  # Auswertung des Experiments: Das Experiment ist erfolgreich, wenn die Würfelsumme 7 ist.
  if wuerfel1 + wuerfel2 == 7:
        erfolgsZaehler += 1

### Das Ergebnis der Experimentreihe wird ausgegeben.
erfolgsWkeit = erfolgsZaehler/n
print('Die Wahrscheinlichkeit für ein erfolgreiches Experime
print('D.h. etwa einer von ' + str(round(1/erfolgsWkeit)) +
```

```
mundhenk@ma3: ~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 siebenwuerfeln.py
Anzahl der Wiederholungen des Experiments: 100
Die Wahrscheinlichkeit für ein erfolgreiches Experiment ist 0.14.
D.h. etwa einer von 7 Würfen ist erfolgreich.
mundhenk@ma3:~/Python/VL02$ python3 siebenwuerfeln.py
Anzahl der Wiederholungen des Experiments: 1000000
Die Wahrscheinlichkeit für ein erfolgreiches Experiment ist 0.166608.
D.h. etwa einer von 6 Würfen ist erfolgreich.
mundhenk@ma3:~/Python/VL02$
```

```
#-------------------------------------------------------------------------------
# siebenwuerfeln.py
#-------------------------------------------------------------------------------
import random

### Die Experimentreihe wird vorbereitet.
# Die Anzahl der Experimente wird von der Kommandozeile eingelesen.
n = int( input('Anzahl der Wiederholungen des Experiments: ') )

erfolgsZaehler = 0       # zählt die erfolgreichen Experimente

### Die Experimentreihe wird durchgeführt.
for i in range(n):      # es werden n Experimente durchgeführt
  # Das Experiment: zwei Würfel werden geworfen.
  wuerfel1 = random.randint(1,6)
  wuerfel2 = random.randint(1,6)
  # Auswertung des Experiments: Das Experiment ist erfolgreich, wenn die Würfelsumme 7 ist.
  if wuerfel1 + wuerfel2 == 7:
        erfolgsZaehler += 1

### Das Ergebnis der Experimentreihe wird ausgegeben.
erfolgsWkeit = erfolgsZaehler/n
print('Die Wahrscheinlichkeit für ein erfolgreiches Experime
print('D.h. etwa einer von ' + str(round(1/erfolgsWkeit)) +
```

```
mundhenk@ma3: ~/Python/VL02
mundhenk@ma3:~/Python/VL02$ python3 siebenwuerfeln.py
Anzahl der Wiederholungen des Experiments: 100
Die Wahrscheinlichkeit für ein erfolgreiches Experiment ist 0.14.
D.h. etwa einer von 7 Würfen ist erfolgreich.
mundhenk@ma3:~/Python/VL02$ python3 siebenwuerfeln.py
Anzahl der Wiederholungen des Experiments: 1000000
Die Wahrscheinlichkeit für ein erfolgreiches Experiment ist 0.166608.
D.h. etwa einer von 6 Würfen ist erfolgreich.
mundhenk@ma3:~/Python/VL02$ python3 siebenwuerfeln.py
Anzahl der Wiederholungen des Experiments: 1000000
Die Wahrscheinlichkeit für ein erfolgreiches Experiment ist 0.166549.
D.h. etwa einer von 6 Würfen ist erfolgreich.
mundhenk@ma3:~/Python/VL02$
```

4 Programmier-Auftrag:
finde ganze Zahlen $a, b, c, d, e \geqslant 1$, so dass $a^5 + b^5 + c^5 + d^5 = e^5$

Euler hat 1769 vermutet (u.a.), dass es keine solchen Zahlen gibt.

Diese Vermutung wurde 1966 von L. J. Lander und T. R. Parkin widerlegt.
Sie schrieben ein Programm, das solche Zahlen gefunden hat.

Wir können das jetzt auch.

Die erste Idee

Gibt es ganze Zahlen $a, b, c, d, e \geqslant 1$ mit $a^5 + b^5 + c^5 + d^5 = e^5$?

Gehe systematisch alle möglichen Kombinationen von $a, b, c, d, e \geqslant 1$ durch,
teste für jede Kombination, ob $a^5 + b^5 + c^5 + d^5 = e^5$,
und stoppe das Programm, wenn eine solche Kombination gefunden wurde.

Mein „brute force“ Programm durchläuft etwa 65 Milliarden Kombinationen a, b, c, d, e,
bis es die Lösung gefunden hat.

Das dauert länger als 28 Stunden (auf einem normalen PC).

Die erste Idee

Gibt es ganze Zahlen $a, b, c, d, e \geqslant 1$ mit $a^5 + b^5 + c^5 + d^5 = e^5$?

Gehe systematisch alle möglichen Kombinationen von $a, b, c, d, e \geqslant 1$ durch,
teste für jede Kombination, ob $a^5 + b^5 + c^5 + d^5 = e^5$,
und stoppe das Programm, wenn eine solche Kombination gefunden wurde.

Mein „brute force“ Programm durchläuft etwa 65 Milliarden Kombinationen a, b, c, d, e,
bis es die Lösung gefunden hat.

Das dauert länger als 28 Stunden (auf einem normalen PC).

Die zweite Idee

Gibt es ganze Zahlen $a, b, c, d, e \geqslant 1$ mit $a^5 + b^5 + c^5 + d^5 = e^5$?

Können wir die Frage umformulieren,
so dass man sie „schneller“ lösen kann?

„Es gibt ganze Zahlen $a, b, c, d, e \geqslant 1$ mit $a^5 + b^5 + c^5 + d^5 = e^5$“

ist gleichbedeutend mit

„es gibt ganze Zahlen $a, b, c, d \geqslant 1$,
so dass $a^5 + b^5 + c^5 + d^5$ die 5te Potenz einer ganzen Zahl ist“.

Um das festzustellen, reicht es, alle möglichen Kombinationen von a, b, c, d zu durchsuchen.

Ob $e = \sqrt[5]{a^5 + b^5 + c^5 + d^5}$ eine ganze Zahl ist, kann berechnet werden.

Die zweite Idee

Gibt es ganze Zahlen $a, b, c, d, e \geqslant 1$ mit $a^5 + b^5 + c^5 + d^5 = e^5$?

Können wir die Frage umformulieren,
so dass man sie „schneller“ lösen kann?

„Es gibt ganze Zahlen $a, b, c, d, e \geqslant 1$ mit $a^5 + b^5 + c^5 + d^5 = e^5$“

ist gleichbedeutend mit

„es gibt ganze Zahlen $a, b, c, d \geqslant 1$,
so dass $a^5 + b^5 + c^5 + d^5$ die 5te Potenz einer ganzen Zahl ist“.

Um das festzustellen, reicht es, alle möglichen Kombinationen von a, b, c, d zu durchsuchen.

Ob $e = \sqrt[5]{a^5 + b^5 + c^5 + d^5}$ eine ganze Zahl ist, kann berechnet werden.

Die zweite Idee

Gibt es ganze Zahlen $a, b, c, d, e \geqslant 1$ mit $a^5 + b^5 + c^5 + d^5 = e^5$?

Gehe systematisch alle möglichen Kombinationen von $a, b, c, d \geqslant 1$ durch,
teste für jede Kombination, ob $\sqrt[5]{a^5 + b^5 + c^5 + d^5}$ eine ganze Zahl ist,
und stoppe das Programm, wenn eine solche Kombination gefunden wurde.

Die Umsetzung der zweiten Idee

Berechne, ob $\sqrt[5]{x}$ eine ganze Zahl ist

Die Funktion `round(y)` rundet `y` auf die nächste ganze Zahl.

Eine `float`-Variable `y` ist also eine ganze Zahl, wenn `round(y)==y` Wert `True` hat.

Beim Rechnen mit `float`-Werten sind die Ergebnisse oft etwas ungenau (Rundungsfehler).
Z.B. ist mathematisch $\sqrt[5]{7776} = 6$, aber `7776**0.2` hat `float`-Wert `6.000000000000001`.

Also hat `round(7776**0.2) == 7776**0.2` den Wert `False`.

Wie umgehen wir diesen Fehler?

Der Wert von `round(7776**0.2)**5 == 7776` ist `True`,
da die Ungenauigkeit bei `7776**0.2` „weggerundet" wird
und anschließend die Korrektheit des „Wegrundens" überprüft wird.

Allgemein:
$\sqrt[5]{x}$ ist eine ganze Zahl, falls `round(x**0.2)**5 == x` Wert `True` hat.

Die Umsetzung der zweiten Idee

Berechne, ob $\sqrt[5]{x}$ eine ganze Zahl ist

Die Funktion `round(y)` rundet `y` auf die nächste ganze Zahl.

Eine `float`-Variable `y` ist also eine ganze Zahl, wenn `round(y)==y` Wert `True` hat.

Beim Rechnen mit `float`-Werten sind die Ergebnisse oft etwas ungenau (Rundungsfehler).
Z.B. ist mathematisch $\sqrt[5]{7776} = 6$, aber `7776**0.2` hat `float`-Wert `6.000000000000001`.

Also hat `round(7776**0.2) == 7776**0.2` den Wert `False`.

Wie umgehen wir diesen Fehler?

Der Wert von `round(7776**0.2)**5 == 7776` ist `True`,
da die Ungenauigkeit bei `7776**0.2` „weggerundet“ wird
und anschließend die Korrektheit des „Wegrundens“ überprüft wird.

Allgemein:
$\sqrt[5]{x}$ ist eine ganze Zahl, falls `round(x**0.2)**5 == x` Wert `True` hat.

Die Umsetzung der zweiten Idee

Berechne, ob $\sqrt[5]{x}$ eine ganze Zahl ist

Die Funktion `round(y)` rundet `y` auf die nächste ganze Zahl.

Eine `float`-Variable `y` ist also eine ganze Zahl, wenn `round(y)==y` Wert `True` hat.

Beim Rechnen mit `float`-Werten sind die Ergebnisse oft etwas ungenau (Rundungsfehler).
Z.B. ist mathematisch $\sqrt[5]{7776} = 6$, aber `7776**0.2` hat `float`-Wert `6.000000000000001`.

Also hat `round(7776**0.2) == 7776**0.2` den Wert `False`.

Wie umgehen wir diesen Fehler?

Der Wert von `round(7776**0.2)**5 == 7776` ist `True`,
da die Ungenauigkeit bei `7776**0.2` „weggerundet“ wird
und anschließend die Korrektheit des „Wegrundens“ überprüft wird.

Allgemein:
$\sqrt[5]{x}$ ist eine ganze Zahl, falls `round(x**0.2)**5 == x` Wert `True` hat.

Die Umsetzung der zweiten Idee

Berechne, ob $\sqrt[5]{x}$ eine ganze Zahl ist

Die Funktion `round(y)` rundet `y` auf die nächste ganze Zahl.

Eine `float`-Variable `y` ist also eine ganze Zahl, wenn `round(y)==y` Wert `True` hat.

Beim Rechnen mit `float`-Werten sind die Ergebnisse oft etwas ungenau (Rundungsfehler).
Z.B. ist mathematisch $\sqrt[5]{7776} = 6$, aber `7776**0.2` hat `float`-Wert `6.000000000000001`.

Also hat `round(7776**0.2) == 7776**0.2` den Wert `False`.

Wie umgehen wir diesen Fehler?

Der Wert von `round(7776**0.2)**5 == 7776` ist `True`,
da die Ungenauigkeit bei `7776**0.2` „weggerundet“ wird
und anschließend die Korrektheit des „Wegrundens“ überprüft wird.

Allgemein:
$\sqrt[5]{x}$ ist eine ganze Zahl, falls `round(x**0.2)**5 == x` Wert `True` hat.

Die Umsetzung der zweiten Idee

Berechne, ob $\sqrt[5]{x}$ eine ganze Zahl ist

Die Funktion `round(y)` rundet `y` auf die nächste ganze Zahl.

Eine `float`-Variable `y` ist also eine ganze Zahl, wenn `round(y)==y` Wert `True` hat.

Beim Rechnen mit `float`-Werten sind die Ergebnisse oft etwas ungenau (Rundungsfehler).
Z.B. ist mathematisch $\sqrt[5]{7776} = 6$, aber `7776**0.2` hat `float`-Wert `6.000000000000001`.

Also hat `round(7776**0.2) == 7776**0.2` den Wert `False`.

Wie umgehen wir diesen Fehler?

Der Wert von `round(7776**0.2)**5 == 7776` ist `True`,
da die Ungenauigkeit bei `7776**0.2` „weggerundet“ wird
und anschließend die Korrektheit des „Wegrundens“ überprüft wird.

Allgemein:
$\sqrt[5]{x}$ ist eine ganze Zahl, falls `round(x**0.2)**5 == x` Wert `True` hat.

Die Umsetzung der zweiten Idee

Berechne, ob $\sqrt[5]{x}$ eine ganze Zahl ist

Die Funktion `round(y)` rundet `y` auf die nächste ganze Zahl.

Eine `float`-Variable `y` ist also eine ganze Zahl, wenn `round(y)==y` Wert `True` hat.

Beim Rechnen mit `float`-Werten sind die Ergebnisse oft etwas ungenau (Rundungsfehler).
Z.B. ist mathematisch $\sqrt[5]{7776} = 6$, aber `7776**0.2` hat `float`-Wert `6.000000000000001`.

Also hat `round(7776**0.2) == 7776**0.2` den Wert `False`.

Wie umgehen wir diesen Fehler?

Der Wert von `round(7776**0.2)**5 == 7776` ist `True`,
da die Ungenauigkeit bei `7776**0.2` „weggerundet“ wird
und anschließend die Korrektheit des „Wegrundens“ überprüft wird.

Allgemein:
$\sqrt[5]{x}$ ist eine ganze Zahl, falls `round(x**0.2)**5 == x` Wert `True` hat.

Die dritte Idee

Gibt es ganze Zahlen $a, b, c, d \geqslant 1$ mit $\sqrt[5]{a^5 + b^5 + c^5 + d^5}$ ist eine ganze Zahl ?

Die brutale Idee ist es, alle möglichen Kombinationen von a, b, c, d zu durchsuchen.
Das ist hier aber nicht nötig.
Zum Beispiel haben die beiden folgenden Kombinationen die gleichen Summen:

a	b	c	d	Summe
2	8	6	4	$2^5 + 8^5 + 6^5 + 4^5$
8	6	4	2	$8^5 + 6^5 + 4^5 + 2^5$

Nur die *Reihenfolge* der Summanden ist unterschiedlich.

Es reicht also, die Werte für a, b, c, d zu durchsuchen, bei denen $a \geqslant b \geqslant c \geqslant d$ ist.

Die dritte Idee

Gibt es ganze Zahlen $a, b, c, d \geqslant 1$ mit $\sqrt[5]{a^5 + b^5 + c^5 + d^5}$ ist eine ganze Zahl ?

Die brutale Idee ist es, alle möglichen Kombinationen von a, b, c, d zu durchsuchen.
Das ist hier aber nicht nötig.
Zum Beispiel haben die beiden folgenden Kombinationen die gleichen Summen:

a	b	c	d	Summe
2	8	6	4	$2^5 + 8^5 + 6^5 + 4^5$
8	6	4	2	$8^5 + 6^5 + 4^5 + 2^5$

Nur die *Reihenfolge* der Summanden ist unterschiedlich.

Es reicht also, die Werte für a, b, c, d zu durchsuchen, bei denen $a \geqslant b \geqslant c \geqslant d$ ist.

Die Umsetzung der dritten Idee

Durchlaufe alle Kombinationen $a, b, c, d \geqslant 1$ mit $a \geqslant b \geqslant c \geqslant d$

Für jedes $a = 1, 2, 3, \ldots$

  für jedes $b = 1 \ldots a$

    für jedes $c = 1 \ldots b$

      für jedes $d = 1 \ldots c$

        überprüfe die Kombination a, b, c, d

```
a  b  c  d
-----------
1  1  1  1
2  1  1  1
2  2  1  1
2  2  2  1
2  2  2  2
3  1  1  1
3  2  1  1
3  2  2  1
3  2  2  2
3  3  1  1
3  3  2  1
3  3  2  2
3  3  3  1
...
8  6  4  2
8  6  4  3
...
```

Die Umsetzung der dritten Idee

Durchlaufe alle Kombinationen $a, b, c, d \geqslant 1$ mit $a \geqslant b \geqslant c \geqslant d$

Für jedes $a = 1, 2, 3, \ldots$

  für jedes $b = 1 \ldots a$

    für jedes $c = 1 \ldots b$

      für jedes $d = 1 \ldots c$

        überprüfe die Kombination a, b, c, d

```
erfolg = False
a = 1
while not erfolg:
  for b in range(1,a+1):
    for c in range(1,b+1):
      for d in range(1,c+1):
        # teste die Euler-Bedingung für a,b,c,d
        # und setze erfolg = True im Erfolgsfall
        ...
  a += 1
```

a	b	c	d
1	1	1	1
2	1	1	1
2	2	1	1
2	2	2	1
2	2	2	2
3	1	1	1
3	2	1	1
3	2	2	1
3	2	2	2
3	3	1	1
3	3	2	1
3	3	2	2
3	3	3	1
...			
8	6	4	2
8	6	4	3
...			

Die dritte Idee

Gibt es ganze Zahlen $a, b, c, d, e \geqslant 1$ mit $a^5 + b^5 + c^5 + d^5 = e^5$?

Gehe systematisch alle möglichen Kombinationen von $a, b, c, d \geqslant 1$ mit $a \geqslant b \geqslant c \geqslant d$ durch,
teste für jede Kombination, ob $\sqrt[5]{a^5 + b^5 + c^5 + d^5}$ eine ganze Zahl ist,
und stoppe das Programm, wenn eine solche Kombination gefunden wurde.

Alles zusammen: das Programm `euler-vermutung.py`

```
zaehler = 0      # wir zählen, wieviele Zahlenkombinationen durchprobiert werden
erfolg = False   # wird auf True gesetzt, wenn die gesuchte Kombination gefunden wurde
a = 1

while not erfolg:    # teste alle Tupel a,b,c,d mit a>=b>=c>=d>=1, bis ein Test erfolgreich war
  print(a)           # damit wir sehen, wie weit das Programm gekommen ist
  for b in range(1,a+1):
    for c in range(1,b+1):
      for d in range(1,c+1):
          zaehler += 1
          summe = a**5 + b**5 + c**5 + d**5
          if round(summe**0.2)**5 == summe:
            print('Versuch ' + str(zaehler) + ' war erfolgreich: ')
            print(str(a) + '**5 + ' + str(b) + '**5 + ' + str(c) + '**5 + '
                                + str(d) + '**5 == ' + str(round(summe**0.2)) + '**5')
            print('Die Summe ist ' + str(summe) + '.')
            erfolg = True
  a += 1
```

```
# python3 euler-vermutung.py
# 1
# 2
# 3
# 4
# ...
# 130
# 131
# 132
# 133
# Versuch 13458163 war erfolgreich:
# 133**5 + 110**5 + 84**5 + 27**5 == 144**5
# Die Summe ist 61917364224.
```

Noch ein paar Bemerkungen ...

Die Vermutung von Euler wurde 1966 von L. J. Lander und T. R. Parkin widerlegt.

Ihr Programm fand genau die gleichen Zahlen für a, b, c, d, e wie wir.

Noch ein paar Bemerkungen . . .

Die Vermutung von Euler wurde 1966 von L. J. Lander und T. R. Parkin widerlegt.

Ihr Programm fand genau die gleichen Zahlen für a, b, c, d, e wie wir.

Das Programm lief auf einer CDC 6600. Sie war 1964–1969 der schnellste Rechner der Welt.

Das Tempo von Rechner misst man in megaFLOPS (10^6 floating point operations per second)
Die folgende Tabelle zeigt ein paar Beispiele.

CDC 6600:	3 megaFLOPS, d.h.	$3 \cdot 10^6$
heutiger PC:	100 gigaFLOPS, d.h.	$100 \cdot 10^9$
Summit (IBM):	122 petaFLOPS, d.h.	$122 \cdot 10^{15}$

2. F. P. Ramsey, *On a problem of formal logic*, Proc. London Math. Soc. (2) **30** (1930), 264–286.

DARTMOUTH COLLEGE

COUNTEREXAMPLE TO EULER'S CONJECTURE ON SUMS OF LIKE POWERS

BY L. J. LANDER AND T. R. PARKIN

Communicated by J. D. Swift, June 27, 1966

A direct search on the CDC 6600 yielded

$$27^5 + 84^5 + 110^5 + 133^5 = 144^5$$

as the smallest instance in which four fifth powers sum to a fifth power. This is a counterexample to a conjecture by Euler [1] that at least n nth powers are required to sum to an nth power, $n > 2$.

REFERENCE

1. L. E. Dickson, *History of the theory of numbers*, Vol. 2, Chelsea, New York, 1952, p. 648.

Bulletin of the American Mathematical Society, Volume 72, Number 6 (1966), 1079.

Beantwortung einer Frage von Euler 1.3.49

Zusammenfassung

- Wir haben if-Anweisungen, while-Schleifen und for-Schleifen kennengelernt.
- Wir kennen die Strukturierung von Programmen in Blöcke.
- Wir können einfache Programme mit
  - Eingabe von Argumenten (von der Konsole)
  - Abarbeitung von Anweisungs-Blöcken,
    - die in Schleifen wiederholt oder
    - durch if-Anweisungen ausgeführt oder übersprungen werden
  - Ausgabe von Ergebnissen (auf der Konsole)

  schreiben und deren Ausführung nachvollziehen.