Informatik

inf12 08.10.2020

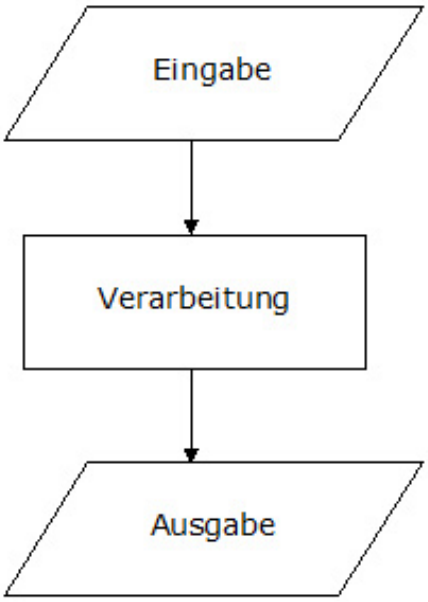
Definition:
Unter einem **Algorithmus** verstehen wir ein aus endlich vielen Anweisungen bestehendes allgemeines Verfahren, welches eine Klasse von Problemen in endlich vielen Schritten löst.

Wir beschreiben einen Algorithmus, unabhängig von der Programmiersprache, in der er codiert wird, durch ein Flußdiagramm oder ein Struktogramm („Nassi-Shneiderman"-Diagramm).

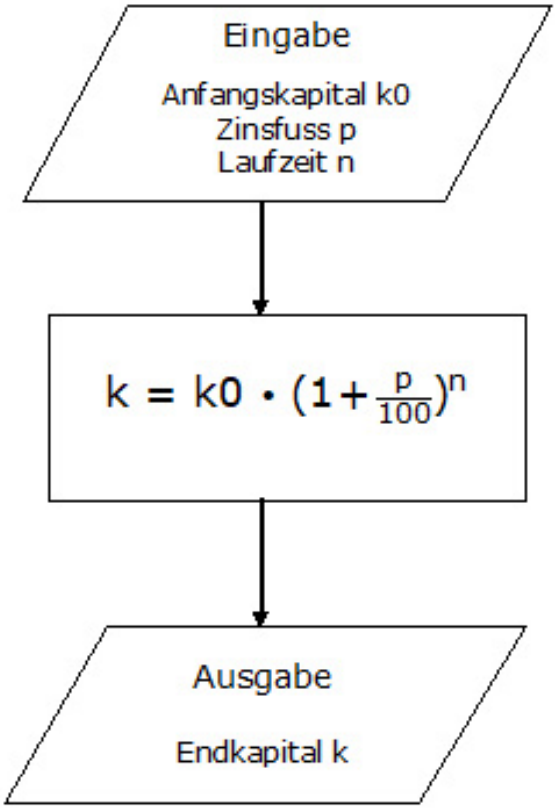
1. Lineare Algorithmen

Wenn bei der Abarbeitung eines Algorithmus die einzelnen Anweisungen sich längs eines einzigen Pfades aneinanderreihen, sprechen wir von einem linearen Algorithmus; insbesondere gibt es hier keine Verzweigungen. Beispiel: Zinseszinsberechnung.

Allgemeines Flußdiagramm eines Algorithmus:



Flußdiagramm des Algorithmus „Zinseszins":

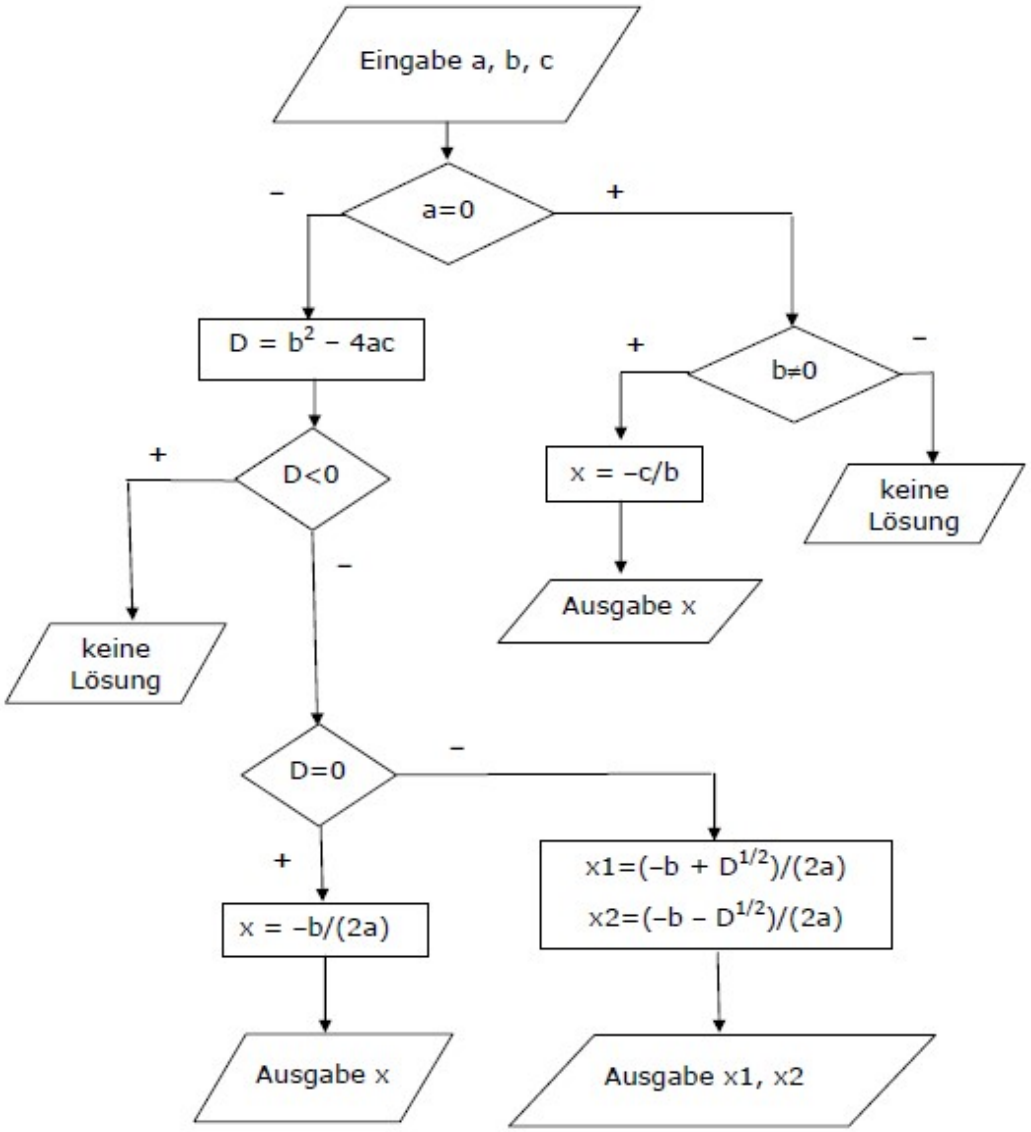


2. Verzweigte Algorithmen

Ein Algorithmus, bei dessen Abarbeitung unterschiedliche Anweisungsblöcke durchlaufen werden können, heißt verzweigter Algorithmus; dabei entscheidet die Abfrage einer Bedingung (in Form einer booleschen Variablen oder eines booleschen Ausdrucks) darüber, welcher Zweig durchlaufen wird.

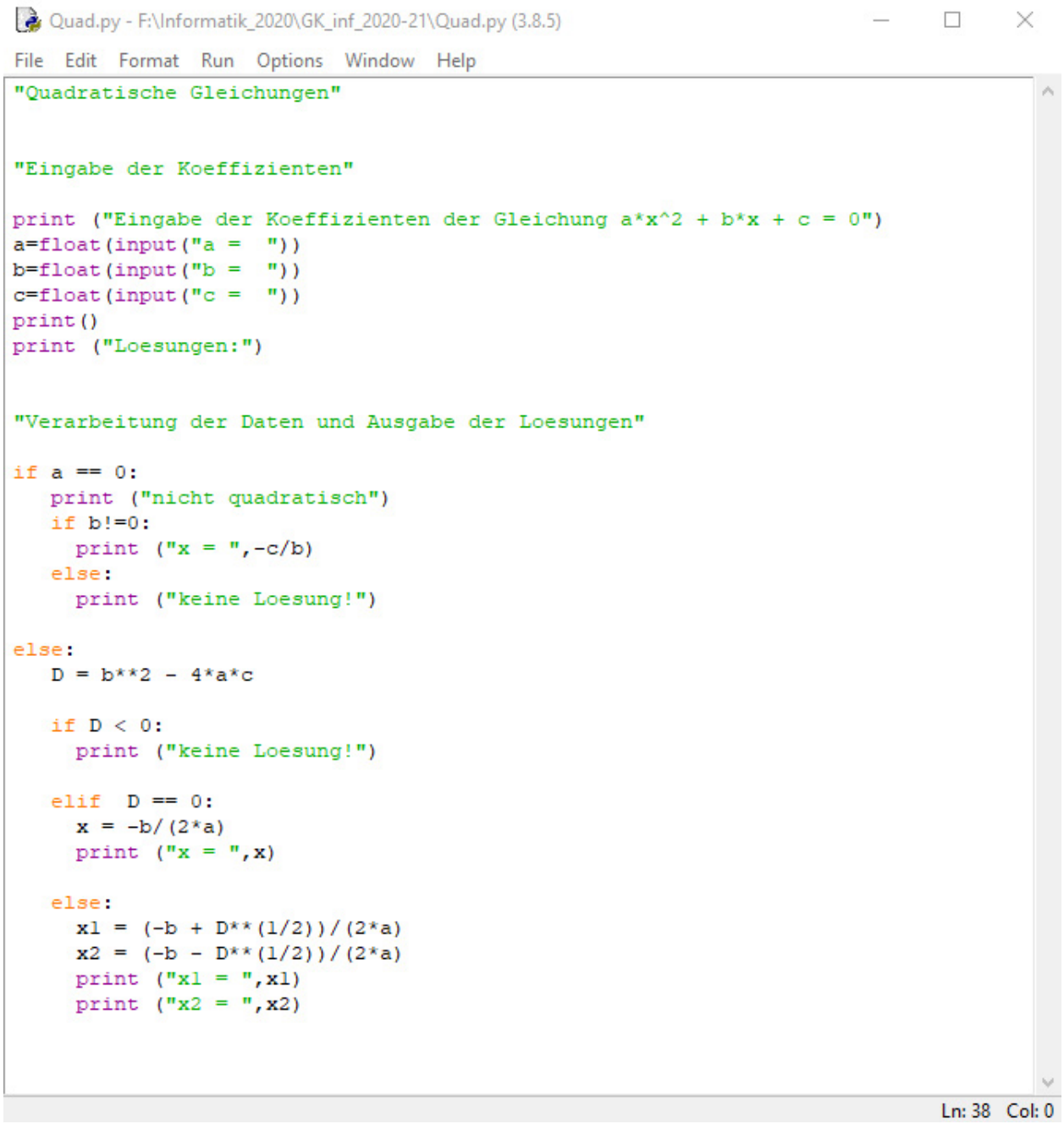
Beispiel:
Der Algorithmus „**QuadEquation**", der die Lösungsmenge einer quadratischen Gleichung bestimmt.

Der Ablauf ergibt sich aus einem Flußdiagramm oder einem Struktogramm (bei letzterem fehlt die Abfrage b≠0):



<table>
<tr><td colspan="4">Eingabe a, b, c</td></tr>
<tr><td>+</td><td colspan="3">a=0 –</td></tr>
<tr><td>Ausgabe
"nicht quadratisch"</td><td colspan="3">D:=b*b - 4*a*c</td></tr>
<tr><td></td><td colspan="3">+ D<0 –</td></tr>
<tr><td></td><td>Ausgabe
"keine Lösung"</td><td colspan="2">+ D=0 –</td></tr>
<tr><td></td><td></td><td>x:= -b/(2*a)</td><td>x1:=(-b + sqrt(D))/(2*a)</td></tr>
<tr><td></td><td></td><td>Ausgabe
x</td><td>x2:=(-b - sqrt(D))/(2*a)
Ausgabe
x1; x2</td></tr>
</table>

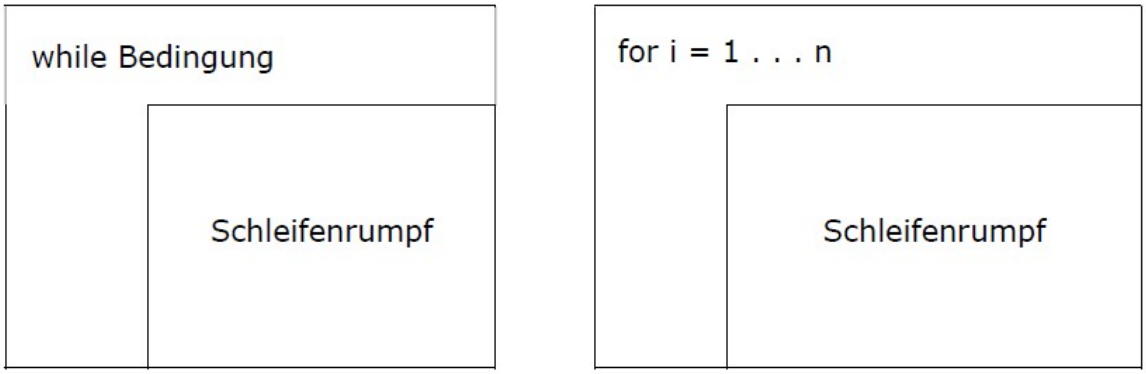
Quelltext des Algorithmus **QuadEquation** in Python codiert:



3. Schleifen

Soll ein Anweisungsblock innerhalb eines Algorithmus mehrmals durchlaufen werden, sprechen wir von einer Schleife; der wiederholt durchlaufene Anweisungsblock heißt auch Schleifenrumpf. Wenn die Anzahl der Durchläufe einer Schleife a priori (von vorneherein) feststeht, läßt sich eine **for**- oder **while**-**Schleife** verwenden; hat z. B. die Abfrage einer Bedingung (in Form eines Booleschen Ausdrucks) innerhalb des Schleifenrumpfs Einfluß auf die Anzahl der Durchläufe, kommt nur die **while-Schleife** in Frage.

Struktogramme:



Der Algorithmus „**Quadratzahlen**" gibt die Quadrate der ganzen Zahlen aus dem Intervall [a, b] aus:

Quellcode in Python, realisiert mit einer while-Schleife:

*Quadratzahlen_while-Schleife.py - F:\Informatik_2020\GK_inf_202...

File Edit Format Run Options Window Help

```
# Tabelle Quadratzahlen (while-Schleife)
# i = Schleifenindex
# a = kleinste ganze Zahl, deren Quadrat berechnet wird
# b = grösste ganze Zahl, deren Quadrat berechnet wird

print('In welchem Bereich soll das Quadrat berechnet werden?')
a=int(input('untere Grenze = '))
b=int(input('obere Grenze  = '))
print()

# Initialisierung des Schleifenindicis i
i=a

# while-loop
while i<=b:
    print(i,'^2 = ',i**2)
    i=i+1
    # alternativ: i += i
```

Ln: 19 Col: 0

Quellcode in Python, realisiert mit einer for-Schleife:

*Quadratzahlen_for-Schleife.py - F:/Informatik_2020/GK_inf_2020-...

File Edit Format Run Options Window Help

```
# Tabelle Quadratzahlen (for-Schleife)
# i = Schleifenindex
# a = kleinste ganze Zahl, deren Quadrat berechnet wird
# b = grösste ganze Zahl, deren Quadrat berechnet wird

print('In welchem Bereich soll das Quadrat berechnet werden?')
a=int(input('untere Grenze = '))
b=int(input('obere Grenze  = '))
print()


# for-loop
for i in range(a,b+1):
    print(i,'^2 = ',i**2)
```

Ln: 11 Col: 0

Ausgabe der Quadratzahlen:

Python 3.8.6 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53
) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for m
ore information.
>>>
= RESTART: F:\Informatik_2020\GK_inf_2020-21\MSS12\Quadr
atzahlen_while-Schleife.py
In welchem Bereich soll das Quadrat berechnet werden?
untere Grenze = 11
obere Grenze  = 19

11 ^2 =  121
12 ^2 =  144
13 ^2 =  169
14 ^2 =  196
15 ^2 =  225
16 ^2 =  256
17 ^2 =  289
18 ^2 =  324
19 ^2 =  361
>>>
```

Ln: 18 Col: 4

Quadratwurzel aus einer positiven reellen Zahl

Der Algorithmus „**Wurzelberechnung**" approximiert **√a** für eine positive reelle Zahl **a**. Die Iteration bricht ab, sobald der Abstand zweier aufeinanderfolgender Folgenglieder kleiner als eine einzugebende Fehlerschranke **d** wird.

Der Abbruch erfolgt, sobald die Boolesche Variable **condition** innerhalb des Schleifenrumpfs den Wert **False** erhält, was eintritt, wenn **abs(y - z)** kleiner als **d** wird.

Anmerkung:
Der hier vorgestellte Algorithmus stützt sich darauf, daß die Folge $\{x_i\}$ *mit* $x_{i+1} = \frac{1}{2}(x_i + \frac{a}{x_i})$, $x_0 = a$, *gegen* $\sqrt{a}$ *konvergiert; dies sei hier ohne Beweis und ohne nähere Begründung mitgeteilt.*

*sqrt_while-loop_mit_abbruchbedingung.py - F:/Informatik_2020/...

File Edit Format Run Options Window Help

```
" Wurzelberechnung "

# Eingabe
print('Eingabe des Radikanden:')
a = float (input('a = '))
# Die Iteration bricht ab, sobald der Abstand
# zweier aufeinanderfolgender Iterationen kleiner als d wird
print('Eingabe Fehlerschranke d:')
d = float (input('d = '))
print()


# Verarbeitung und Ausgabe
condition = True    # Initialisierung der Bedingung
x = a               # Startwert; alternativ: x = 1
while condition:
    z = x
    x = 0.5*(x + a/x)
    y = x
    print(x)
    condition = (abs(y - z) >= d)
```

Ln: 19 Col: 8


Python 3.8.6 Shell
File Edit Shell Debug Options Window Help
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: F:/Informatik_2020/GK_inf_2020-21/MSS12/sqrt_while-loop_mit_abbruchbedingung.py
Eingabe des Radikanden:
a = 85
Eingabe Fehlerschranke d:
d = 0.0001
43.0
22.488372093023255
13.134051610110387
9.802889441169137
9.236901144433745
9.219560764417094
9.21954445730731
>>>
= RESTART: F:/Informatik_2020/GK_inf_2020-21/MSS12/sqrt_while-loop_mit_abbruchbedingung.py
Eingabe des Radikanden:
a = 85
Eingabe Fehlerschranke d:
d = 0.0000000000001
43.0
22.488372093023255
13.134051610110387
9.802889441169137
9.236901144433745
9.219560764417094
9.21954445730731
9.219544457292887
9.219544457292887
>>>
= RESTART: F:/Informatik_2020/GK_inf_2020-21/MSS12/sqrt_while-loop_mit_abbruchbedingung.py
Eingabe des Radikanden:
a = 25
Eingabe Fehlerschranke d:
d = 0.00000000000001
13.0
7.461538461538462
5.406026962727994
5.015247601944898
5.000023178253949
5.000000000053722
5.0
5.0
>>>
Ln: 48 Col: 4