Kapitel 1

ALGORITHMIK

by RÉMY

1.1 Algorithmen und Programmierung

Definition

Algorithmen besitzen die folgenden charakteristischen Eigenschaften:

- Eindeutigkeit: ein Algorithmus darf keine widersprüchliche Beschreibung haben. Diese muss eindeutig sein.
- Ausführbarkeit: jeder Einzelschritt muss ausführbar sein.
- Finitheit (= Endlichkeit): die Beschreibung des Algorithmus muss endlich sein.
- Terminierung: nach endlich vielen Schritten muss der Algorithmus enden und ein Ergebnis liefern.
- Determiniertheit: der Algorithmus muss bei gleichen Voraussetzungen stets das gleiche Ergebnis liefern.
- Determinismus: zu jedem Zeitpunkt der Ausführung besteht höchstens eine Möglichkeit der Fortsetzung. Der Folgeschritt ist also eindeutig bestimmt.

Diese Eigenschaften können in der Mathematik genutzt werden, um Probleme zu lösen. Hierfür bedarf es einer einheitlichen Schreibweise.
Insbesondere vor dem Abitur stehen den Schülern mehrere Möglichkeiten zur Verfügung, die im Folgenden behandelt werden.

1.1.1 Pseudocode

Pseudocode ist ein Programmcode, der nicht zur maschinellen Interpretation, sondern lediglich zur Veranschaulichung eines Algorithmus dient. Meistens ähnelt er höheren Programmiersprachen, gemischt mit natürlicher Sprache und mathematischer Notation. Er ist leichter verständlich als realer Programmcode aber klarer und weniger missverständlich als eine Beschreibung in natürlicher Sprache.

Beispiel:
Ein Beispiel erübrigt sich.

1.1.2 Python

Programmiersprache, bekannt durch ihre einfach verständliche Syntax, sie gilt als höhere Sprache, was sie zu einer auf die (gesprochene) Sprache angepasste Sprache macht. Sie ist somit gerade für Einsteiger interessant und eignet sich dennoch für größere Projekte. Ein weiterer Vorteil ist die mitlerweile allgegenwärtige Präsenz der Sprache, denn sie wird auch für Apps und Web-Entwicklung verwendet.

Syntax

Folgendes macht die Syntax Pythons aus:

- gute Lesbarkeit des Quellcodes

- englische Schlüsselwörter
- wenig syntaktische Konstruktionen

Funktionsweise

Python verwendet Schleifen und Verzweigungen.

```
#Schleifen:
for: #(Wiederholung ueber Elemente einer Sequenz (Zahl, Liste, Zeit...))
   #Inhalt der Schleife
while: #(Wiederholung, solange ein logischer Ausdruck wahr ist)
   #Inhalt der Schleife

#Verzweigungen:
if: #(prueft,ob ein logischer Ausdruck wahr ist)
   #Inhalt der Verzweigung
elif: #(prueft,ob ein anderer logischer Ausdruck wahr ist)
   #Inhalt der Verzweigung
else: #(letzter Fall, tritt ein, wenn keine der obigen Bedingungen erfuellt wurde)
   #Inhalt der Verzweigung
```

Beispiel:
Man nehme den Algorithmus, der die Fibonacci-Sequenz bis zum n-ten Glied generiert:

```
a = 0
b = 1
n = 10
for iteration in range(n):
   print(a)
   a = a+b
   b = a-b
```

1.2 Algorithmen und mathematische Anwendungen

1.2.1 Iterationsverfahren

Unter Iteration versteht man ein Verfahren zur schrittweisen Annäherung an die Lösung einer Gleichung unter Anwendung eines sich wiederholenden Rechengangs. Das bedeutet, (wenn es möglich ist) aus einer Näherungslösung durch Anwenden eines Algorithmus zu einer besseren Näherungslösung zu kommen und die Lösung beliebig gut an die exakte Lösung heranzuführen. Man sagt dann, dass die Iteration konvergiert.
Beispiele für Verfahren dieser Art werden im Folgenden behandelt.

Newton-Rhapson Verfahren

Das Newton-Rhapson Verfahren, auch bekannt als Newtonmethode dient der Nullstellenbestimmung komplexer Polynome und allgemein jeder differenzierbaren Funktion.
Die Grundidee ist, die Nullstelle der Tangente an der Stelle x_0 von f zu nehmen und den Vorgang mit $f(\mathrm{NS}_T)$ zu wiederholen.
Eine mögliche Umsetzung in Python wäre:

```
def fx(x):
   return 3*x**3+5x**2+3          #beliebige Funktion (muss angegeben werden)

def f_strich(x):
   return 9*x**2+10*x             #muss auch manuell angegeben werden

def NS_Tangente(x):
```

```
    NS_T = x - fx(x)/f_strich(x)
    return NS_T

Startwert=1
altwert = NS_Tangente(Startwert)
Genauigkeit = 0.00001

while abs(altwert-NS_Tangente(altwert))>Genauigkeit:
    altwert=NS_Tangente(altwert)
print(altwert)
```

Heron-Verfahren

Es handelt sich hierbei um eine vereinfachte Version des Newton-Rhapson Verfahrens, da es zur Berechnung einer Näherung der Quadratwurzel einer reellen Zahl $a > 0$ dient.
Man erhält das gewünschte Ergebnis durch die Berechnung der Nullstelle einer Funktion $f(x) = x^2 - a$. Es gilt also: $f'(x) = 2x$.
Durch die Verwendung des Newton-Rhapson Verfahrens erhält man die Iterationsvorschrift:
$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \Leftrightarrow x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{x_n^2 + a}{2x_n} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$
Als kluger, ungefähr zutreffender Startwert gilt $x_0 = \frac{a+1}{2}$
Eine mögliche Umsetzung in Python wäre:

```
a=2                                          #Muss manuell angegeben werden
def fx(x):
    return x**2-a

def f_strich(x):
    return 2*x

def next_val(wert):
    return 0.5*(wert+(a/wert))

Startwert=(a+1)/2
altwert = next_val(Startwert)
Genauigkeit = 0.00001

while abs(altwert-next_val(altwert))>Genauigkeit:
    altwert=next_val(altwert)
print(altwert)
```