

Serie 3

1. Schreibe ein Python-Programm `quad.py`, das die quadratische Gleichung $ax^2 + bx + c = 0$ für beliebige (`float`)-Zahleingaben für a , b und c löst. Eingabe mit: `a = float(input("a = "))`, etc.

Beachte insbesondere den Spezialfall $a = 0$ und die verschiedenen Fälle der Diskriminante $D = b^2 - 4ac$.

2. Gegeben ist das Python-Programm `collatz.py`:

```
n = int(input("Natuerliche Zahl: "))
while n != 1:
    if n % 2 == 0:
        n = n / 2
    else:
        n = 3*n+1
    print (int(n))
```

- a) Tippe das Programm ein. Erkläre die einzelnen Zeilen. Was leisten sie?
- b) Erweitere und verändere das Python-Program, so dass bei allen Zahlen n von 1 bis 100 gezählt wird, wie viele Schritte benötigt werden, um bei der Zahl 1 anzukommen (= `coll(n)`). (Tipp: `for`-Schleife verwenden!)
- c) Schreibe eine Funktion `coll` (mit `def coll(n):`), so dass man `coll` für eine beliebige Zahl n aufrufen kann: z.B. `coll(17)=12`
- d) Verbessere das Programm bei b) mit Hilfe der Funktion `coll`.
3. Schreibe ein Programm `raten.py`, bei welchem das Programm zunächst mit dem Zufallsgenerator eine ganze Zahl x zwischen 0 und 1000 bestimmt. Der Benutzer wird nun immer wieder aufgefordert, die Zahl x zu erraten und einzugeben. Als Antwort wird angegeben, ob die eingebene Zahl y stimmt, zu klein oder zu gross ist. Das Programm bricht ab, wenn der Spieler die Zahl erraten hat und erfährt, wie oft er probieren musste.
4. Schreibe eine Python-Funktion `faktor(n)`, welche für eine beliebige natürliche Zahl n die Primfaktorzerlegung bestimmt. (Beispiel: $1111111111 = 11 \cdot 41 \cdot 271 \cdot 9091$)

Mögliches Vorgehen:

- Teile die Zahl so oft es geht durch 2 (Tipp: `while m % 2 != 0:`)
- Teile die verbliebene Zahl so oft es geht durch 3
- Teile die verbliebene Zahl so oft es geht durch 5
- Teile die verbliebene Zahl so oft es geht durch 7, etc.

- a) Weshalb ist der Algorithmus nicht besonders effizient?
- b) Bis zu welchen Teilern muss der Algorithmus "probieren"?