

## Grundlagen der Programmierung

Wir wollen die Grundlagen der Programmierung mit der Programmiersprache Python an einem Projekt zur Bruchrechnung kennenlernen. Die Grundlagen des Rechnens mit Brüchen sind allen im Kurs bekannt und es ließ sich sehr einfach klären, dass es nicht um ein triviales Problem handelt, für das schon eine Lösung in Python vorliegt. Außerdem kommen bei der Lösung schon zwei wesentliche Konzepte der Informatik vor, nämlich Algorithmen und Klassen bzw. Objekte.

### Algorithmen

Ein Algorithmus ist eine Abfolge von Schritten zur Lösung eines Problems. Typische Algorithmen, mit denen ihr euch schon beschäftigt habt, sind Bedienungsanleitungen für Geräte. Wenn ich die auszugsweise rechts dargestellte Bedienungsanleitung unseres Kaffeeautomaten betrachte, stelle ich fest, dass sie erheblich komplizierter ist, als viele der von uns im Unterricht behandelten Algorithmen.

### Klassen und Objekte

Trotzdem ist dieser Text ein gutes Beispiel, weil er neben der Beschreibung des einzuhaltenden Ablaufes auch die an diesem Ablauf beteiligten Objekte beschreibt.

### Aufgabe:

Finde aus der Anleitung die beteiligten Objekte heraus, schreibe sie auf und versuche Zusammenhänge zu erkennen und sie zu gliedern.

### Beispiel Bruch


Bei unserem Bruchprogramm ist das zunächst sehr viel einfacher. Es gibt hier nur eine Art von Objekten, eben die Brüche. In der Klasse Bruch wird das allgemeine Verhalten von Bruchobjekten beschrieben.

Man beschreibt,

- welche Eigenschaften sie haben und
- welches Verhalten sie haben.

### Eigenschaften

Die Eigenschaften, die man üblicherweise eher mit dem Fremdwort **Attribute** angibt, sind bei einem Bruch sehr einfach, es sind der Zähler und der Nenner. Um nicht mit dem Zeichensatz Problem zu bekommen, verwenden wir beim Namen des Attributes Zähler nicht den Umlaut, sondern schreiben zaehler. Der Kopf unserer Klassendefinition ist damit schon anzugeben:  
[Einrückungen beachten!]

 Nach jedem Reinigungsprozess sollte der Einfülltrichter für vorgemahlene Kaffee gereinigt werden (9).

- **DISPLAY** ☐ leuchtet, ✖ blinkt
- Drücken Sie die Pfeiletaste ⏏ (4):
- **DISPLAY** ✖ leuchtet, ☐ leuchtet
- Leeren Sie die Schale (18).
- **DISPLAY** ☐ blinkt, ✖ leuchtet
- Setzen Sie die Schale (18) wieder sorgfältig ein.
- Stellen Sie ein 1/2 Litergefäß (Fig. 12) unter den höhenverstellbaren Kaffeeauslauf (15).
- **DISPLAY** ⏏ leuchtet, ✖ leuchtet
- Drücken Sie die Pfeiletaste ⏏ (4).
- **DISPLAY** ✖ leuchtet
- Gerät reinigt.
- **DISPLAY** ⏏ leuchtet, ✖ leuchtet, ⏏ leuchtet
- Werfen Sie die Tablette ein (siehe Fig. 15).
- Drücken Sie die Pfeiletaste ⏏ (4).
- **DISPLAY** ✖ leuchtet
- Gerät reinigt.
- **DISPLAY** ✖ leuchtet, ☐ leuchtet
- Leeren Sie die Schale (18).
- **DISPLAY** ☐ blinkt, ✖ leuchtet
- Geben Sie die Schale (18) wieder sorgfältig ein.
- **DISPLAY** ☐ leuchtet
- Die Reinigung ist erfolgreich abgeschlossen.

```
class Bruch:
    def __init__(self, zaehler, nenner):
        # Konstruktor: wird beim Erzeugen eines Objektes ausgefuehrt.
        self.zaehler=zaehler
        self.nenner=nenner
```

Damit können wir schon Bruchobjekte erzeugen, allerdings können die noch nichts außer die Werte von Zähler und Nenner in ihren Attributen `zaehler` und `nenner` zu speichern.

### Verhalten

Das Verhalten wird in den Prozeduren und Funktionen beschrieben, in der objektorientierten Programmierung nennt man sie einheitlich **Methoden**. Bisher haben wir:

```
def zeige(self):
    print '(' ,self.zaehler, "/", self.nenner, ")"

def kuerze(self):
    teiler=ggt(self.zaehler, self.nenner)
    self.zaehler=self.zaehler/teiler
    self.nenner=self.nenner/teiler

def multipliziere(self, faktor):
    return Bruch(self.zaehler*faktor.zaehler
, self.nenner*faktor.nenner)
```

Hier war noch ein Fehler in der Kursversion

sonst ist der ggt-Wert hier falsch

kein wirklicher Umbruch

Hinter jedem Namen einer Funktion oder Prozedur muss ein Paar von Klammern kommen, das gilt auch für die Methoden eines Objektes. In dieser Klammer werden die Parameter angegeben werden, welche sie zu ihrer Arbeit braucht. Die Methode eines Objektes muss als ersten Parameter zusätzlich den Parameter **self** haben.

Wir haben in unser Programm außerdem noch eine Funktion eingebaut, die nicht Teil der Klasse ist.

```
def ggt (grosse, kleine):
    if kleine>grosse:
        kleine, grosse=grosse, kleine
    rest=grosse%kleine
    while rest>0:
        grosse=kleine
        kleine=rest
        rest=grosse%kleine
    return kleine
```

Das erkennen wir [und Python] einmal daran, dass der Text nicht eingerückt ist und damit nicht zu dem Programmblock der Klasse `Bruch` gehört. Andererseits hat die Funktion nicht `self` als ersten Parameter. Sie muss also nicht den `Bruch` kennen, der `Bruch` verwendet sie.

Mit diesem Programm können wir nun schon einiges tun:

```
>>> b1=Bruch(1, 2)
>>> b2=Bruch(2, 5)
>>> b3=b1.multipliziere(b2)
>>> b3.zeige()
( 2 / 10 )
>>> b3.kuerze()
>>> b3.zeige()
( 1 / 5 )
```

>>>

Besonders, sehr ungewohnt, ist die Art der Multiplikation, da man nicht  $b3=b1*b2$  schreiben kann. Statt dessen wird das Objekt  $b1$  aufgefordert, die Methode `multipliziere` mit dem Bruchobjekt  $b2$  auszuführen.

Übrigens führt die Berechnung mit ausgetauschten  $b1$  und  $b2$  natürlich zum selben Ergebnis:

```
>>> b3=b2.multipliziere(b1)
>>> b3.zeige()
( 2 / 10 )
>>> b3.kuerze()
>>> b3.zeige()
( 1 / 5 )
>>>
```

### Addition ist schwieriger

Die Addition von Brüchen ist normalerweise sehr viel komplizierter. Da Computer (speziell unter Python) aber keine Probleme mit großen Zahlen haben, kann man hier sehr „brutal“ vorgehen. Man lässt die addiere-Methode zunächst folgendermaßen rechnen

$$\frac{1}{10} + \frac{4}{20} = \frac{1 \cdot 20}{10 \cdot 20} + \frac{4 \cdot 10}{20 \cdot 10} = \frac{20}{200} + \frac{40}{200} = \frac{60}{200}$$

und erhält nach dem Kürzen das gewünschte Ergebnis  $\frac{3}{10}$ , das von unserem

Programm dann als  $( 3 / 10 )$  angezeigt werden kann. Der Programmcode wird von Abdul in der Stunde vorgestellt.

### Fertig?

Das war es eigentlich schon. Wir wollen unserer Klasse `Bruch` aber eine grafische Oberfläche hinzufügen und dafür verwenden wir `wxPython`, die Erweiterung von Python um tools für grafische Oberflächen [GUI = graphical user interface].

Als Elemente der Oberfläche verwenden wir Buttons, Beschriftungen und Textfelder. Da wir das Prinzip schon von Mediator kennen, habe ich eine solche Oberfläche einmal mit Mediator erstellt und das Bild hier eingefügt. Das Ergebnis ist so noch sehr unbefriedigend, weil ihr eure eigene Oberfläche entwickeln sollt.

