Objekte und Klassen in Python

Mit Methoden hast du bereits ein Verfahren kennengelernt, um häufig benötigte Programmschritte innerhalb eines Programmes zusammenzufassen. Methoden haben aber noch weitere Vorteile:

- Sie vereinfachen die Wartbarkeit des Codes, da Fehler innerhalb einer Methode gut eingegrenzt werden können und die Fehlerkorrektur dann an allen Stellen im Programm wirkt, an denen sie zum Einsatz kommen
- Sie ermöglichen die Zusammenarbeit in einem Team, z.B. könnte man überlegen, welche Methoden mit welchen Parametern für eine Aufgabe benötigt werden und diese Methoden dann von unterschiedlichen Personen bearbeiten lassen

Objekte perfektionieren diese Vorteile noch einmal auf einem ganz anderen Level. Objekte und Klassen klingen als Begriffe erstmal sehr abstrakt, aber eigentlich sind diese intuitiv gut zugänglich. Dazu ein Beispiel:

In einer Schul**klasse** gibt es Schüler:innen. Man sagt: Schüler:innen sind **Objekte** der Klasse (vielleicht fühlst du dich manchmal ja auch als bloßes Objekt in der Schule).

Klassen erzeugst du in Python so (tut gar nicht weh):

oop01.py

```
class schueler:
   pass
```

Das „pass“ kannst du erstmal als Hilfe sehen. Es sagt dem Interpreter: „Ist noch nicht fertig, meckere vorerst nicht!“.

Schüler:innen haben bestimmte Eigenschaften. Eigenschaften bezeichnet man beim Programmieren auch als **Attribute**. Wir machen es uns einfach und nehmen uns einige Beispielattribute heraus, die sich gut für das Programmieren eignen.

- Name
- Geschlecht
- Haarfarbe
- Durchschnittsnote

Diese Attribute werden für alle Objekte einer Klasse festgelegt. Es ist guter Stil, dafür zu sorgen, dass keine Objekte ohne Attribute erzeugt werden können. Beim Bauen („Konstruieren“) eines Objekts kann man das durch einen Konstruktor beim Anlegen **init()** erzwingen (tut etwas mehr weh).

oop02.py

```
class schueler:
   # Attribute definieren ("self" nicht vergessen!")
   def __init__(self, name, geschlecht, haarfarbe, durchschnittsnote):
      self.name = name
      self.geschlecht = geschlecht
      self.haarfarbe = haarfarbe
```

```
        self.durchschnittsnote = durchschnittsnote
```

Jetzt können wir schon Schüler:innen erzeugen - das Schlüsselwort „**self**“ sagt, dass sich die laufende Operation auf das aktuelle Objekt bezieht (du könntest ja innerhalb eines Objektes auch auf andere Objekte zugreifen, s.u.).

Nochmal zum Mitdenken:

```
self.geschlecht = geschlecht
```

Heißt: Hallo Objekt! Dein eigenes Attribut („self“) „geschlecht“ setzt du auf den Wert, den dir dein Meister/deine Meisterin beim Anlegen übergeben hat. Jetzt bauen wird uns mal zwei Schüler:innen (tut fast nicht weh).

oop3.py

```
class schueler:
   # Attribute definieren
   def __init__(self, name, geschlecht, haarfarbe, durchschnittsnote):
      self.name = name
      self.geschlecht = geschlecht
      self.haarfarbe = haarfarbe
      self.durchschnittsnote = durchschnittsnote

# main
schueler01 = schueler("Maik", "m", "grau", 2.3)
schueler02 = schueler("Kerstin", "w", "braun", 1.1)
```

Na, toll, jetzt haben wir zwei Schüler:innen erzeugt. Man sollte damit aber auch etwas anfangen können. Man kann z.B. die Attribute eines Objekts ausgeben, indem man das gewünschte Attribut mit einem Punkt hinter den Namen des Schuelerobjekts hängt. In anderen Programmiersprachen müsstest du dafür „Getter“-Methoden definieren, Python macht das freundlicherweise automatisch für dich.

oop3.py

```
class schueler:
   # Attribute definieren
   def __init__(self, name, geschlecht, haarfarbe, durchschnittsnote):
      self.name = name
      self.geschlecht = geschlecht
      self.haarfarbe = haarfarbe
      self.durchschnittsnote = durchschnittsnote

# main
schueler01 = schueler("Maik", "m", "grau", 2.3)
```

```
schueler02 = schueler("Kerstin", "w", "braun", 1.1)

print(schueler01.name)
```

Man kann aber auch Attribute von Objekten vergleichen:

oop4.py

```
class schueler:
   # Attribute definieren
   def __init__(self, name, geschlecht, haarfarbe, durchschnittsnote):
      self.name = name
      self.geschlecht = geschlecht
      self.haarfarbe = haarfarbe
      self.durchschnittsnote = durchschnittsnote

# main
schueler01 = schueler("Maik", "m", "grau", 2.3)
schueler02 = schueler("Kerstin", "w", "braun", 1.1)

if schueler01.durchschnittsnote < schuler02.durchschnittsnote:
   print(f'{schueler01.name} ist der bessere Schüler!')
else:
   print(f'{schueler02.name} ist der bessere Schüler!')
```

Attribute verändern

Durchschnittsnote und Haarfarbe können sich im Leben mal ändern. Hat man ein Objekt erstmal erzeugt, kommt man so ohne Weiteres nicht mehr an die Attribute heran. Man kann diese nicht einfach überschreiben, es sei denn, das Objekt selbst bietet uns dafür eine sogenannte „Setter“-Methode an.

oop5.py

```
class schueler:
   # Attribute definieren
   def __init__(self, name, geschlecht, haarfarbe, durchschnittsnote):
      self.name = name
      self.geschlecht = geschlecht
      self.haarfarbe = haarfarbe
      self.durchschnittsnote = durchschnittsnote

   # Settermethoden definieren
   def setGeschlecht(self, geschlecht):
      self.geschlecht = geschlecht
```

```
   def setHaarfarbe(self, haarfarbe):
      self.haarfarbe = haarfarbe

# main
schueler01 = schueler("Maik", "m", "grau", 2.3)

print(schueler01.haarfarbe)

schueler01.setHaarfarbe("getönt")

print(schueler01.haarfarbe)
```

Auf die Methoden eines Objekts greifst du genau so zu, wie auf die Attribute. Du machst quasi einen ganz normalen Methodenaufruf, sagst aber vorher, auf welches Objekt sich das beziehen soll. Genau wie Attribute stehen Methoden für alle Objekte einer Klasse zur Verfügung.

Kapselung

Der Trick hinter der Setter-Methode ist das Prinzip der Kapselung. Derjenige, der ein Objekt verwendet, muss nur wissen, welche Attribute und Methoden es gibt, aber nicht zwingend, wie diese programmiert sind, wenn sie denn fehlerfrei funktionieren. Man sagt: Die Prozeduren in einem Objekt sind **„gekapselt“**.

In einer Klasse ist alles Attribut oder Methode

Es gibt in einer Klasse kein Hauptprogramm. Nur Attribut- und Methodendefinitionen sind zulässig!