Klassen II

Klassen beschreiben den Aufbau eines Objektes aus dem beliebig viele Instanzen mit allen beschrieben Attributen erstellt werden können. Nun kann es sein, dass man verschiedene Arten von Objekten erstellen will, die unterschiedliche Eigenschaften haben aber auch viele Gemeinsamkeiten. Damit man hier nicht Dinge doppelt und dreifach schreiben muss, wird hier u.a. das Konzept der Vererbung vorgestellt.

1 Statische Variablen (Klassenvariablen)

Wird eine Variable in der `init()`-Methode mit `self.farbe = farbe` angelegt, so ist dies eine Instanzvariable. Jede Instanz hat diese Variable und kann diese **unabhängig** von allen anderen Instanzen ändern.
Eine Klassenvariable hingegen wird über alle Instanzen dieser Klasse geteilt. Wird diese Klassenvariable an einer Stelle geändert, so wird diese für **ALLE** instanzen geändert.

> Beispiel: In einem Teich schwimmen Fische, jeder Fisch hat ein Gewicht, je nach dem ist er dicker oder dünner wenn er mehr zu fressen bekommt. Wird die Wassertemparatur geändert, so betrifft das alle Fische gleichermaßen.

Klassenvariablen werden außerhalb der `init()`-Methode, am besten oben direkt nach der Klassendefinition, angelegt:

```
1  class Fisch:
2    wasser_temparatur = 20
3
4    def __init(self, gewicht):
5      self.gewicht = gewicht
```

Will man eine Klassenvariable abfragen, so tut man dies nicht über eine Instanz-variable. Stattdessen wird der Klassennamen verwendet. Dadurch ist sofort ersichtlich, dass es sich um eine Klassenvariable handelt.

```
1  fisch1 = Fisch(20)
2  fisch2 = Fisch(24)
3
4  # Instanzvariablen abfragen
5  print("Gewicht 1:", fisch1.gewicht) # Gewicht 1: 20
6  print("Gewicht 1:", fisch2.gewicht) # Gewicht 2: 24
7
8  # Die Klassenvariablen abfragen/setzen
9  print(Fisch.wasser_temparatur) # 20
10 Fisch.wasser_temparatur = 21
```

Klassenvariablen werden manchmal auch als statische Variablen bezeichnet. Hier noch ein weiteres Beispiel, wie man einen Zähler hat, der Instanzübergreifend zählt.

```python
class Zaehler:
  gemeinsamer_zaehler = 0

  def init(self):
    self.eigener_zaehler = 0

  def zaehle(self):
    self.eigener_zaehler += 1
    Zaehler.gemeinsamer_zaehler += 1


z1 = Zaehler()
z2 = Zaehler()

z1.zaehle()
print(z1.eigener_zaehler, Zaehler.gemeinsamer_zaehler) # 1, 1

z2.zaehle()
print(z2.eigener_zaehler, Zaehler.gemeinsamer_zaehler) # 1, 2
```

2 Vererbung

Wir wollen verschiedene geometrische Formen erstellen und derren Fläche und Umfang ausgeben. Jede Form hat eine andere Formel, wie um die Fläche/Umfang zu berechnen. Hier bietet es sich an, von Vererbung gebrauch zu machen. Klassen können von einander erben. D.h. sie bekommen alle Variablen und Funktionen ihrer Elternklasse zur Verfügung gestellt!
Als Beispiel erstellen wir nun einen Elternklasse, die noch nicht so viele Funktionalität besitzt.

```python
class Form:

  def init(self):
    # hier passiert nichts
    pass

  def umfang(self):
    print("Nicht implementiert!")
    return 0

  def flaeche(self):
    print("Nicht implementiert!")
    return 0

  def info(self):
      print("Diese Figur hat eine Fläche von",
        self.flaeche(), "und einen Umfang von",
```

```
18          self.umfang())
```

Diese Klasse ist lediglich eine Beschreibung einen abstrakten Form. Beim Programmieren muss man oft über 'abstrakte' Konzepte nachdeneken. Man weiß im Vorraus nicht, weas die korrrekte Berechnungsvorschrift ist, um z.B. die Fläche auszurechnen. Stattdessen legt man eine Platzhalter Methode an, die später ausgefüllt werden kann.

```
1  class Kreis(Form): # (Ich will von Form erben)
2
3    # Eine statische Variable
4    PI = 3.1415926
5
6    def init(self, radius):
7      # WICHTIG: Das Eltern-__init__() aufrufen
8      Form.__init__(self)
9
10     self.radius = radius
11
12   def umfang(self):
13     return 2 self.radius * Kreis.PI
14
15   def flaeche(self):
16     return self.radius * self.radius * Kreis.PI
```

Kreis erbt von Form, deshalb muss in der `__init__()`-Methode aud die `__init__()`-Methode der Elternklasse aufgerufen werden mit allen nötigen Parameter. Parameter sind `self`, und danach weitere Extraparameter, falls gefordert. Danach werden erneut die Funktionen `umfang()` und `flaeche()` angelegt. Dies nennt man überschreiben. Es gibt diese beiden Funktionen nun *zweimal*, einmal in der Eltern- und einmal in der Kindklasse. Ruft man diese nun auf, so wird immer die Funktion aus der Kindklasse gewählt!

```
1  class Rechteck(Form): # (Ich will von Form erben)
2
3    def init(self, breite, hoehe):
4      # WICHTIG: Das Eltern-__init__() aufrufen
5      Form.__init__(self)
6
7      self.breite = breite
8      self.hoehe = hoehe
9
10   def umfang(self):
11     return 2 (self.breite + self.hoehe)
12
13   def flaeche(self):
14     return self.breite * self.hoehe
```

Die `Rechteck`-Klasse wird analog zu `Kreis` angelegt, nur mit anderen Formeln.
Nun können wir verschiedene Formen erstellen und derren WErte berechnen lassen.

```
rechteck = Rechteck(4, 3)
kreis = Kreis(1)

# Beide haben .info() von Form geerbt
rechteck.info() # ... 12 ... 14
kreis.info()# PI ... 2PI
```