17. OOP Konzepte (3) Methoden- und Operatorüberladung

17.1. Methodenüberladung

- unter Methodenüberladung versteht man in der objektorientierten Programmierung, dass es in einer Klasse mehrere Methoden mit demselben Namen gibt, sie aber verschiedene Parameter erwarten - welche Methode genau aufgerufen wird, wird dann bei jedem Aufruf anhand der Parameter und ihrer Datentypen automatisch bestimmt
- in Python können Methoden nicht überladen werden
- in Python lassen sich Methoden in abgeleiteten Klassen überschreiben
- in der Vererbungshierarchie soll damit sichergestellt werden, dass alle Objekte aus dieser Hierarchie sich ähnlich verhalten – die Methode der Elternklasse wird überschrieben

- Beispiel:

```
# Deklaration der Klassen
class Tier:
    def __init__(self):
        self.gewicht = 0

    # diese Methode wird in jeder Klasse an den Typ
    def ausgeben(self):
        print("Ich bin ein Tier")

class Fisch(Tier):
    def __init__(self):
        super().__init__()
        self.kiemengroesse = 0

    def ausgeben(self):
        print("Ich bin ein Fisch")

class Saeugetier(Tier):
    def __init__(self):
        super().__init__()
        self.lungenvolumen = 0

    def ausgeben(self):
        print("Ich bin ein Säugetier")

# Deklaration einer Liste von Tieren
tierliste = [Tier(), Saeugetier(), Fisch()]

# mit der Schleifenvariablen t wird durch die Liste iteriert
# entsprechend dem Typ wird die richtige Methode ausgerufen
for t in tierliste:
    t.ausgeben()
```

```
Ich bin ein Tier
Ich bin ein Säugetier
Ich bin ein Fisch.5 10
```

17.2. Operatorüberladung

- ein Operator ist eine Vorschrift, die aus einer Reihe von Operanden einen neuen Wert berechnet
- in Abhängigkeit vom Datentyp der Operanden hat der Operator unterschiedliche Auswirkungen - z.B. der Operator **+**
  - für int- und float-Instanzen werden die Werte addiert
  - für str-Instanzen werden die Zeichenfolgen verkettet
- die Mehrfachbelegung des Operators wird dadurch ermöglicht, dass intern eine spezielle Methode aufgerufen wird, die festlegt, was der Operator bewirken soll
- für den Operator **+** ist es die Methode **__add__(self, other)**
- diese speziellen Methoden werden auch als magische Methoden in Python bezeichnet
- weitere arithmetischen Operatoren und ihre magischen Methoden

Operator	**magische Methode**
-	__sub__(self, other)
*****	__mul__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
******	__pow__(self, other)

- Vergleichsoperatoren und ihre magischen Methoden

Operator	**magische Methode**
<	__lt__(self, other)
<=	__le__(self, other)
==	__eq__(self, other)
!=	__ne__(self, other)
>	__gt__(self, other)
>=	__ge__(self, other)

- all diese Operatoren lassen sich auch für die eigenen Klassen definieren, indem die magischen Methoden in diesen Klassen überschrieben werden
- Beispiel:

```
# Definition der Klasse Laenge
class Laenge:
    # das Dictionary umrechnung enthält die Faktoren für die Umrechnung
    umrechnung = {"m"    : 1,
                  "dm"   : 0.1,
                  "cm"   : 0.01,
                  "mm"   : 0.001,
                  "km"   : 1000,
                  "ft"   : 0.3048,    # Fuss
                  "in"   : 0.0254,    # Zoll/inch
                  "mi"   : 1609344    # Meilen
                  }
    def __init__(self, wert, einheit):
        self.zahlenwert = wert
        self.einheit = einheit

    def __str__(self):
        return f"{self.zahlenwert} {self.einheit}"
```

```
#   Überschreiben der magischen Methode __add__
    def __add__(self, other):
        z = round(self.zahlenwert * Laenge.umrechnung[self.einheit], 2)
        z += other.zahlenwert * Laenge.umrechnung[other.einheit]
        z /= Laenge.umrechnung[self.einheit]
        return Laenge(z, self.einheit)

#   Überschreiben der magischen Methode __sub__
    def __sub__(self, other):
        z = round(self.zahlenwert * Laenge.umrechnung[self.einheit], 2)
        z -= other.zahlenwert * Laenge.umrechnung[other.einheit]
        z /= Laenge.umrechnung[self.einheit]
        return Laenge(z, self.einheit)

# Deklaration
a1 = Laenge(5, "cm")
a2 = Laenge(3, "dm")

# Addition von einem Wert in cm und einem Wert in dm - Ergebnis in cm
print(a1 + a2)

# Subtraktion eines Wertes in cm von einem Wert in dm - Ergebnis in dm
print(a2 - a1)

# Vergleich eines Wertes in m mit einem Wert in mm - Ergebnis True oder False
print (Laenge(0.05, "m") == Laenge(50, "mm"))
```

```
35.0 cm
2.5 dm
True
```

Aufgabe

1. Klasse Konto
   ist zu definieren; sie soll die Attribute Inhaber, Kontonummer, Kontostand, maximalen Tagesumsatz haben; das Attribut Umsatz von heute soll in der Konstruktor-Methode mit 0 initialisiert werden
   Mit der magischen Methode __eq__ werden zwei Konten auf Gleichheit der Kontonummer geprüft – gleiche Kontonummer ergibt True, sonst False.
   Die Instanzen konto1 und konto2 sollen die gleiche Kontonummer haben, während konto3 mit einer anderen Kontonummer erstellt wird.
   Mit den Vergleichsanweisungen `print(konto1 == konto2)` und `print(konto1 == konto3)` wird dann True bzw. False ausgegeben.