# LU01d - Ist Klasse gleich Klasse?

#### **TODO SuMa - Python-Beispiel korrigieren**

Warum stellen wir uns diese Frage?

Sie haben im (Modul 319) einen ersten Einblick erhalten. Dabei haben Sie eine Klasse als Ansammlung vieler Attribute kennengelernt. Man nennt solche Klassen (informell) auch Datenklassen (POPO - Plain Old Python Object).

In einigen Programmiersprachen oder Frameworks wird der Begriff "Datenklasse" verwendet, um eine Klasse zu beschreiben, die hauptsächlich dazu dient, Daten zu speichern, ohne viel Verhalten zu haben. Solche Klassen haben normalerweise private Datenfelder (Variablen) und öffentliche Getterund Setter-Methoden, um auf die Daten zuzugreifen und sie zu setzen. Sie enthalten normalerweise keine komplexe Geschäftslogik oder spezielle Verhaltensmethoden. Eine "allgemeine" Klasse kann ein komplexeres Objekt mit verschiedenen Verhaltensmethoden enthalten.

### **Beispiel: Dataklasse und Standardklasse**

Human	«have»	Address

Abb. Beschreibung einer Person mit deren Adresse

Wir finden in Abbildung die zwei Klassen Human und Address, die in einer einseitigen Beziehung "Mensch besitzt Adresse" stehen.

Die Klasse Human beschreibt eine Person mit all ihren Eigenschaften und Fähigkeiten, was wohl erkennbar eine sehr komplexe Klasse sein wird, während Address die Adresse der Person beschreibt. Die Adresse wird einige wenige Attribute halten wie

- street
- number
- postal\_code
- city

die durch set- und get-Methoden geschrieben (set) bzw. gelesen (get) werden. Weitere Methoden werden kaum benötigt. Es handelt sich hier um eine typische Datenklasse.

Dagegen wird die Person durch eine Vielzahl von Eigenschaften wie z.B.

- size
- hair\_color
- weight
- ...

und Methoden wie

breath()

Last		
update: 2024/07/01	modul:m320_2024:learningunits:lu01:gleichartig https://wiki.bzz.ch/modul/	m320_2024/learningunits/lu01/gleichartig?rev=1719841346
15.42		

- eat()
- go()
- ...

beschrieben. Zu diesen Attributen finden sich dann auch wieder set- und get-Methoden.

## Was aber bedeutet das für die Programmierung in Python?

Wir betrachten uns hier die beiden Klassen Human und Address sowie deren Umsetzung in Code. Für die Person werden einige wenige Attribute und Methoden stellvertretend angeschrieben.

#### Beispiel: Codierung einer Datenklasse und einer "standard" Klasse

Address	Human
Audi C33	Iluliidii

https://wiki.bzz.ch/ Printed on 2025/11/14 09:00

```
from dataclasses import dataclass
@dataclass
class Address:
    Represents a residential address
    street: str = None
    number: str = None # String since 11a or
similar are possible,
    postal_code: int = 0
    city: str = None
    # Durch den Decorator @dataclass wird unter
anderem der Konstruktor generiert.
    @property
    def street(self):
       return self. street
    @street.setter
    def street(self, value):
        self. street = value
    @property
    def number(self):
        return self. number
    @number.setter
    def number(self, value):
        self._postal_code = value
    @property
    def postal_code(self):
        return self._postal_code
    @postal_code.setter
    def postal_code(self, value):
        self. number = value
    @property
    def city(self):
        return self._city
    @city.setter
    def city(self, value):
        self. city = value
```

```
class Human:
    Represents a human being with some
attributs and abilities.
    # Hier wird ein expliziter Konstruktor
angeschrieben.
   # Mehr zur Bedeutung des Konstruktors folgt
später.
    def
          _init__(self, name, weight, eye_color,
one_address):
        self._name = name
        self._weight = weight
        self._eye_color = eye_color
        # hier können ganz viele weitere
Attribute stehen, die durch
        # den Konstruktor initialisiert werden.
        self. address = one address # eine
Referenz zu einem Address-Objekt
    def __repr__(self):
    return f'Human(name=\'{self.name}\',
weight={self.weight},
eye_color=\'{self.eye_color}\',
address={self.address}'
    # Attribute werden über get-Methoden
gelesen.
    # Python kennt dazu den Dekorator
@property.
    # Hier als Beispiel drei Methoden.
    @property
    def weight(self):
       return self._weight
    @property
    def name(self):
        return self._name
    @property
    def eye color(self):
        return self._eye_color
    @property
    def address(self):
        return self._address
    # Attribute werden über set-Methoden
gesetzt.
    # ACHTUNG: Es gibt Attribute, die einmalig
bei der Initialisierung
    # gesetzt werden, hier z.B. die Augenfarbe.
    # Python nutzt hier den @.setter Dekorator.
    # Hier als Beispiel wieder nur eine
Methode.
    @address.setter
    def address(self, new_address):
        self._address = new_address
    # Und hier folgen Methoden, die das
Verhalten des Objekts
    # beeinflussen.
    def eat(self, what ever):
        self. weight += 0.3
```

from address import Address

```
11 11 11
Hier folgt die main-Methode über die das
Programm ausgeführt werden kann.
if __name__ == '__main__':
    # Instanzieren eines Objektes über den
Konstruktor.
    # my home ist eine Objekt-Variable, während
Address(...) den Konstruktor
    # der Klasse ausführt, um das Objekt zu
    my home = Address(street='Musterstrasse',
                      number='11A',
                      postal_code=9999,
                      city='Musterdorf')
    my_home.postal_code = 1001 # nutzen der
setter-Methode
    # Ausgabe der Werte über impliziten Aufruf
der jeweiligen property-Methoden.
    print('Werte des Objekts address:')
    print(f'{my_home.street} {my_home.number} '
          f'{my_home.postal_code}
{my_home.city}') # nutzen der get-Methoden
    print(my_home) # Die Methode __repr__
wurde durch @dataclass erzeugt
Ausgabe:
Werte des Objekts address:
Musterstrasse 11A 1001 Musterdorf
Address(street='Musterstrasse', number='11A',
postal_code=1001, city='Musterdorf')
```

```
Und nun ein Stück Programmcode, um den Effekt
zu testen
if
    name == '
                 main ':
    address = Address(street='Musterstrasse',
                      number='11A',
                      postal_code=9999,
                      city='Musterdorf')
    milli = Human('Milli', 50, 'blue', address)
# Initialgewicht ist 50
    milli.eat('chips') # wer isst nimmt zu
    print(f'Name: {milli.name}')
    print(f'Adresse: {milli.address}')
    print(f'Gewicht: {milli.weight}') # das
Ergebnis sieht man ;-)
    print(milli) # Ruft die Methode __repr__
```

Ausgabe: Name: Milli Adresse: Address(street='Musterstrasse', number=9999, postal\_code=9999, city='Musterdorf') Gewicht: 50.3 Human(name='Milli', weight=50.3, eye color='blue', address=Address(street='Musterstrasse', number=9999, postal\_code=9999, city='Musterdorf')

Sie haben nun an Hand der Klasse Human den Aufbau einer Klasse mit all ihren Details gesehen. Ebenso könenn Sie feststellen, dass eine "Datenklasse" in Python mittels dem @dataclass Dekorator um ein vielfaches einfacher zu erstellen ist, da der Konstruktor verschiedene weitere Methoden notwendig sind.

Anmerkung: Sie können natürlich auch einer "Datenklasse" eigene Methoden für <u>init</u>, <u>repr</u>, etc. hinzufügen.



No sa © René Probst, bearbeitet durch Marcel Suter

https://wiki.bzz.ch/ - BZZ - Modulwiki

Permanent link:

https://wiki.bzz.ch/modul/m320 2024/learningunits/lu01/gleichartig?rev=1719841346

Last update: 2024/07/01 15:42



https://wiki.bzz.ch/ Printed on 2025/11/14 09:00