Objektorientiertes Programmieren (OOP)

sca, hof

TALIT

KSR

Toy-Auftrag

- Schreibe Software für Schule, um sämtliche Personen zu verwalten
- Wird in etwa so gemacht für ISY und EcoReader (Software im Hintergrund)
- Viele Personen an KSR, ca. 645
  - 530 Schüler:Innen
  - 92 Lehrpersonen
  - 32 Mitarbeiter:innen (Mensa, Hausdienst, Front Office, Assistenzen, …)
- Problem: Wie geht man in Code mit so vielen Personen um?

Schlechtes Beispiel 1

- Schlechter Code
  - Umständlich, Fehleranfällig, verliert schnell Überblich
- Probleme:
  - Versch. Infos zu einer Person in verschiedenen Variablen
  - Alle Personen sind praktisch gleich aufgebaut (Name, Alter, …), trotzdem muss gesamter Code für jede Person neugeschrieben werden

```
# LP 1
scb_name = "Bert Schreiner"
scb_address = "Hoehenweg 8, 8590 Romanshorn"
scb_birthday = "07.12.1969"
scb_type = "LP" # Lehrperson
scb_position = "HL" # Hauptlehrperson

# LP 2
hom_name = "Monika Hohler"
hom_address = "Seeblickstrasse 17, 9306 Freidorf"
hom_birthday = "24.05.1990"
hom_type = "LP"
hom_position = "LB1" # Lehrbeauftragte(r) 1

# LP lists with infos of all teachers
lp_names = [scb_name, hom_name]
lp_addresses = [scb_address, hom_address]
lp_birthdays = [scb_birthday, hom_birthday]
lp_position = [scb_position, hom_position]

# STU 1
verkelle_name = "Verena Keller"
verkelle_address = "Saentisstrasse 22, 8580 Amriswil"
verkelle_birthday = "03.02.2007"
verkelle_type = "STU" # Student
verkelle_class = "1Fd"

# STU 2
heimeier_name = "Heiri Meier"
heimeier_address = "Kreisstrasse 3, 9322 Egnach"
heimeier_birthday = "11.09.2005"
heimeier_type = "STU" # Student
heimeier_class = "3Mfz"

# STU lists with infos of all students
stu_names = [verkelle_name, heimeier_name]
stu_addresses = [verkelle_address, heimeier_address]
stu_birthdays = [verkelle_birthday, heimeier_birthday]
stu_classes = [verkelle_class, heimeier_class]
```

Schlechtes Beispiel 2

```
names = ["Bert Schreiner", "Monika Hohler", "Verena Keller", "Heiri Meier"]
addresses = ["Hoehenweg 8, 8590 Romanshorn", "Seeblickstrasse 17, 9306 Freidorf", "Saentisstrasse 22, 8580 Amriswil", "Kreiss
birthdays = ["07.12.1969", "24.05.1990", "03.02.2007", "11.09.2005"]
types = ["LP", "LP", "STU", "STU"]
positions = ["HL","LB1",None,None]
classes = [None,None,"1Fd","3Mfz"]
```

- Kompakter, …
- … aber auch nicht viel besser
- Probleme:
  - Nach wie vor: versch. Infos zu einer Person in verschiedenen Variablen (jetzt Listen)
  - Extrem fehleranfällig, da mehrere hundert Personen an Schule
  - Wie umgehen mit Eigenschaften, die nicht für alle Personen gelten?
    - Position der Lehrpersonen
    - Klasse der SuS

I have a dream

- Es wäre doch toll, wenn es eine Möglichkeit gäbe, alles, was zu einer Person gehört, in eine ‘Variable’ packen zu können.

Lösung! OOP

- **Objektorientiertes Programmieren (OOP)!**
- Erstelle eine **Klasse** ‘Person’
- Diese dient dann als **Bauplan** (engl. Blueprint) dafür, wie eine ‘Person’ zu sein hat, …
- … ist selbst aber *keine* konkrete Person
- Mit Klasse kann eigenen **Datentypen** definieren
- Gibt vor, dass eine Person *haben muss*: name, address, birthday, type, position und klasse

Lösung! OOP

- Mithilfe dieses Bauplans können wir nun alle konkreten ’Personen’ erzeugen (Heiri Meier, …)
- Diese nennt man **Instanzen der Klasse** oder **Objekte** vom Typ ‘Person’
- Die Eigenschaften einer Person wie ’name’ oder ‘address’ heissen **Attribute** des Objekts
- Wir haben dann also:
  - 1 **Klasse** ’Person’
  - 645 **Objekte** vom Typ ‘Person’
  - Je 6 **Attribute**

hom (Variablennamen des Objekts)
- name = "Monika Hohler "
- address = "Seeblickstrasse …"
- birthday = "24.05.1990"
- type = "LP"
- position = "LB1"
- klasse = None

heimeier (Variablennamen des Objekts)
- name = "Heiri Meier"
- address = "Kreisstrasse 3 …"
- birthday = "11.09.2005"
- type = "STU"
- position = None
- klasse = "3Mfz"

Klassen in Python

- Erstelle **Klasse**:

```python
class Person:
    def __init__(self,_name,_address,_birthday,_type,_position,_klasse):
        self.name = _name
        self.address = _address
        self.birthday = _birthday
        self.type = _type
        self.position = _position
        self.klasse = _klasse # can't use 'class' bc is keyword
```

- Erzeuge **Objekt** vom Typ Person:

```python
scb = Person("Bert Schreiner", "Hoehenweg 8, 8590 Romanshorn TG", "07.12.1969", "LP","HL",None)
```

- Zugriff auf **Attribute** von Objekt:
  - <objekt>.<attribut>
  - Beispiele:

```python
print(scb.birthday) # print birthday of scb
```

```python
scb.address = "Kirchgasse 7, 8594 Kesswil" # change address of scb
```

Klassen in Python

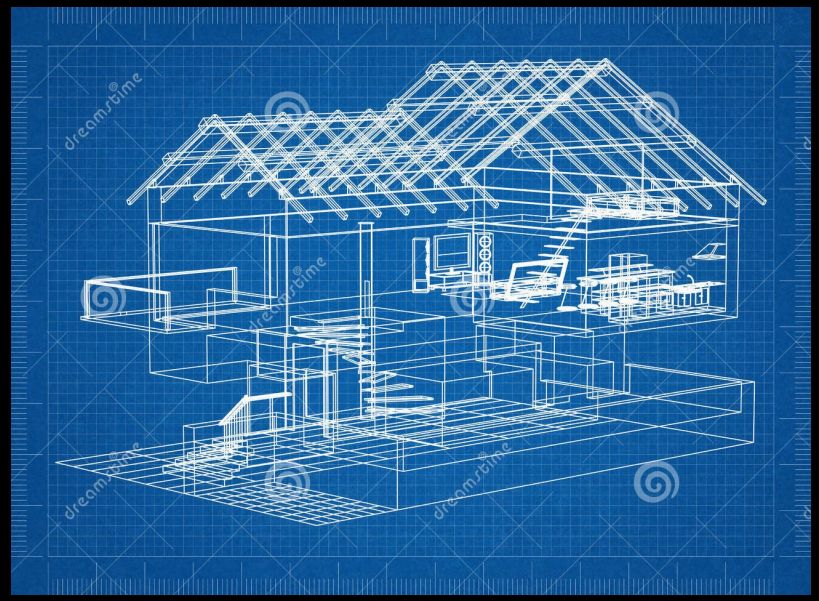
- Klassen/Objekte haben Eigenschaften, genannt **Attribute**:
  - name, address, …
- Funktion __init__(…):
  - init für *initialize*
  - heisst **Konstruktor**
  - Wird aufgerufen, wenn Objekt **erstellt** wird
  - Nutze, um Werte in Attribute zu schreiben
- self-Keyword:
  - Attribute in Klasse immer: self.<attribute>
  - self repräsentiert das Objekt

```python
class Person:
    def __init__(self,_name,_address,_birthday,_type,_position,_kla
        self.name = _name
        self.address = _address
        self.birthday = _birthday
        self.type = _type
        self.position = _position
        self.klasse = _klasse # can't use 'class' bc is keyword

# create object of class 'Person':
scb = Person("Bert Schreiner", "Hoehenweg 8, 8590 Romanshorn TG", '
```

```python
print(scb.birthday)
print(verkelle.birthday)
```

Analogie: Häuser



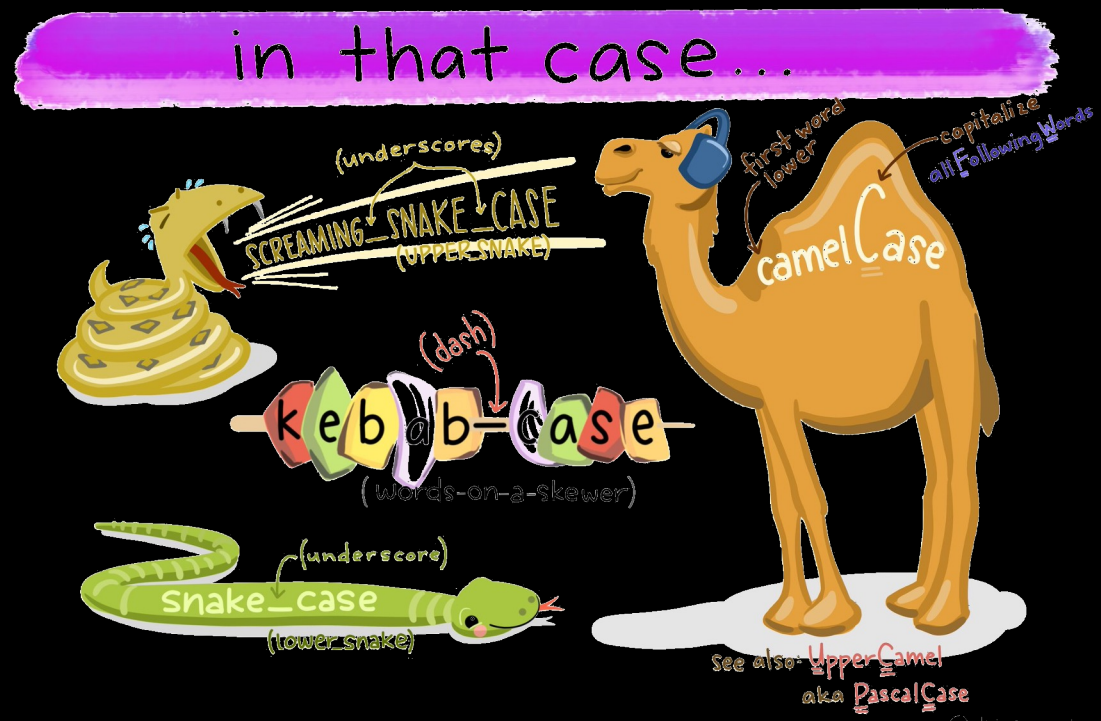
- **Klasse:** Bauplan für Haus
- **Achtung:** Nur weil es einen Plan von einem Haus gibt, gibt es noch lange kein Haus!
- **Objekte:**
  - Häuser, die mit *gleichem Bauplan* gebaut wurden:
    - house_1 = House('red')
    - house_2 = House('cyan')
    - house_3 = House('sand')
  - Werte der Attribute können variieren, z.B.
    - house_1.color ist 'red'
    - house_2.color ist 'cyan'
    - house_3.color ist 'sand'

Klassen in Python

- Style Konventionen:
  - Attribute: snake_case
  - Klassennamen im UpperCamelCase
- Verbleibendes Problem:
  - LP und STU haben beide Attribute ‘type’ und ‘klasse’
  - Gibt elegante Lösung -> später

```python
class Person:
    def __init__(self,_name,_address,_birthday,_type,_position,_klasse
        self.name = _name
        self.address = _address
        self.birthday = _birthday
        self.type = _type
        self.position = _position
        self.klasse = _klasse # can't use 'class' bc is keyword

# create object of class 'Person':
scb = Person("Bert Schreiner", "Hoehenweg 8, 8590 Romanshorn TG", "07
```



Klassen in Python

- Wie nun alle Personen-Objekte verwalten?
- Z.B. in Liste speichern
- … können diese durchgehen um z.B. alle Namen zu printen oder eine bestimmte Person zu finden.

```python
class Person:
    def __init__(self,_name,_address,_birthday,_type,_position,_klasse):
        self.name = _name
        self.address = _address
        self.birthday = _birthday
        self.type = _type
        self.position = _position
        self.klasse = _klasse # can't use 'class' bc is keyword

# instantiate objects, store in list
persons = [
    Person("Bert Schreiner", "Hoehenweg 8, 8590 Romanshorn TG", "07.12.1969", "LP","HL",None),
    Person("Monika Hohler", "Seeblickstrasse 17, 9306 Freidorf", "24.05.1990", "LP","LB1",None),
    Person("Verena Keller", "Saentisstrasse 22, 8580 Amriswil", "03.02.2007", "STU",None,"1Fd"),
    Person("Heiri Meier", "Kreisstrasse 3, 9322 Egnach", "11.09.2005", "STU",None,"3Mfz")
]

# go throigh all objects and print ...
for person in persons:
    print(person.name)
```

Methoden

- Haben gesehen: Klassen haben **Attribute**
- Speichern darin **Eigenschaften**, die ein Objekt ausmachen …
- Für Person z.B. den Namen, das Geburtsdatum, …
- Klasse kann neben Eigenschaften auch **Verhaltensweisen** haben …
- in Form von **Klassenmethoden** (kurz. Methoden)
- Sind **Funktionen**, die *zur Klasse gehören*
- Methoden für Klasse ‘Person’:
  - Ideen?
  - get_info(): Stellt wichtigste Infos zu Person in String zusammen
  - get_age(): Berechnet Alter aufgrund Geburtsdatum und heutigem Datum
  - change_klasse(): erhöht z.B. Klassenbezeichnung für neues Semester (1Fd -> 2Fd)

Methoden

- Implementiere wie normale **Funktionen** (def …) aber *in* Klasse
- Als **erstes Argument** muss self übergeben:
  - Gibt an, dass Klassenmethode ist
  - Gibt auch noch andere Arten von Methoden (später)
- Kann auch **weitere Argumente** übergenen (siehe change_klasse)

```python
class Person:
    def __init__(self,_name,_address,_birthday,_type,_position,_klasse):
        self.name = _name
        self.address = _address
        self.birthday = _birthday
        self.type = _type
        self.position = _position
        self.klasse = _klasse # can't use 'class' bc is keyword

    def get_info(self):
        return f"Name: {self.name}, Adresse: {self.address}, Geburtstag: {self.birthday}, Typ:

    def get_age(self):
        # TODO: determine age using datetime module
        return None

    def change_klasse(self,x=1):
        # TODO: change class by x, default value is x=1
        return None

verkelle = Person("Verena Keller", "Saentisstrasse 22, 8580 Amriswil", "03.02.2007", "STU",None
heimeier = Person("Heiri Meier", "Kreisstrasse 3, 9322 Egnach", "11.09.2005", "STU",None,"3Mfz"
verkelle.change_klasse() # Verena
heimeier.change_klasse(-1) # Heiri has to repeat a year
```

Klasse (Objektorientierung)

Unter einer **Klasse** (auch *Objekttyp* genannt) versteht man in der objektorientierten Programmierung ein abstraktes Modell bzw. einen *Bauplan* für eine Reihe von ähnlichen Objekten.

Die Klasse dient als Bauplan für die Abbildung von realen Objekten in Softwareobjekte und beschreibt Attribute (Eigenschaften) und Methoden (Verhaltensweisen) der Objekte. Verallgemeinernd könnte man auch sagen, dass eine Klasse dem Datentyp eines Objekts entspricht.

TurtleGraphics & OOP

- Haben OOP bereits angetroffen
- TurtleGraphics!
- Konstruktor: Turtle()
- Klassenmethoden:
  - setPenColor(…)
  - forward(…)
- Können auch **mehrere Turtles** erzeugen

```
1 from gturtle import *
2
3 hans = Turtle()
4
5 hans.setPenColor("red")
6 hans.forward(100)
```

```
 1 from gturtle import *
 2
 3 tf = TurtleFrame() # needed to display multiple turtles on same screen
 4
 5 jack = Turtle(tf) # instantiate first turtle object
 6 rose = Turtle(tf) # instantiate second turtle object
 7
 8 jack.setPenColor("red")
 9 jack.setPenColor("green")
10
11 rose.setPos(-200,0)
12 jack.forward(100)
13 rose.forward(100)
14 jack.left(90)
15 rose.right(90)
16 jack.forward(100)
17 rose.forward(100)
```

Aufträge

- Siehe Wiki
- Aufträge 1 & 2