

# Objektorientierte Programmierung

- Wie lassen sich *reale Sachverhalte* in einer Programmiersprache abbilden?
- Beispiel: Bilde eine Person ab
- Viele Optionen:
  - Einzelne Variablen:  
    name = "Klaus"  
    alter = 39
  - Liste/Tupel:  
    person = ["Klaus", 39] # bzw. ("Klaus", 39)
  - Dict:  
    person = {"name": "Klaus", "alter": 39}
- **Problem:** eine Person besteht nicht nur aus Eigenschaften!
  - Klaus kann laufen, atmen, essen, reden usw.

# Dictionaries sind ok



- Problem: eine Person besteht nicht nur aus Eigenschaften!
  - Klaus kann laufen, atmen, essen, reden usw.
- Lösung: Definiere eine Funktion pro Fähigkeit:

```
def laufen(person):  
    print(person["name"], "läuft")  
  
def essen(person, gericht):  
    print(person["name"], "isst", gericht)  
  
def programmieren(person, projekt):  
    print(person["name"], "programmiert", projekt)  
  
klaus = {"name": "Klaus", "alter": 39, "beruf": "Anwalt"}  
laura = {"name": "Laura", "alter": 37, "beruf": "Programmierer"}  
  
laufen(klaus)    # "Klaus läuft"  
laufen(laura)    # "Laura läuft"  
  
essen(klaus, "Pizza") # "Klaus isst Pizza"  
essen(laura, "Döner") # "Laura isst Döner"  
  
programmieren(klaus, "nichts") # "Klaus programmiert nichts"  
programmieren(laura, "die 10. Dating App") # "Laura programmiert die 10. Dating App"
```

# Problem mit Funktionen

- Vorteil: Funktionen sind flexibel
- Nachteil: Funktionen sind flexibel
- Funktionen ist es (mehr oder weniger) egal, was man ihnen als Parameter gibt
- Klar: Innerhalb der Funktion lässt sich einiges einschränken
  - Datentyp (z.b. Funktion, die nur mit Zahlen arbeitet -> int, float)
  - Eigenschaft (z.b. Funktion, die prüft, ob eine Person programmieren kann)
- Fazit: Funktionen & Dictionaries sind ok aber es wird schnell kompliziert

# Introducing: OOP

- Ein eigenes Programmierparadigma (je nach Quelle)
- Neue Konzepte: Klassen, Objekte, Attribute, Methoden
- Im Vergleich zu dict & funktion:
  - Klasse = *allgemeine Person*
  - Objekt = Klaus (Bezeichnung des Objekts)
  - Attribut = name, alter, beruf (Eigenschaften)
  - Methode = laufen, essen, programmieren (Fähigkeiten)
- Python setzt stark auf OOP
  - Imperativ  OOP
  - Funktional  OOP



float

int

list

sets

str

tuple

Always has been

Wait it's all **OBJECTS**

type()

even  
functions...

# Bestandteile des OOP: Klassen

- IHK-Sprache: „Die Klasse ist der Bauplan eines Objekts.“
- Allgemeine Beschreibung eines Sachverhalts
- Z.b. statt „Klaus“ nehmen wir „Person“ oder „Anwalt“ oder „Lebewesen“
- Es ist schwierig die ideale Klasse zu finden:
  - Klaus: Person, Anwalt, Lebewesen
  - Laura: Person, kein Anwalt, Lebewesen
  - Katze: keine Person, kein Anwalt, Lebewesen
  - Fazit: Lebewesen zu allgemein; Anwalt zu spezifisch
  - Fazit<sup>2</sup>: Menschen sind Lebewesen; Anwälte sind Personen (-> Vererbung)
- Faustregel: Besser allgemeiner als spezifischer
  - Es ist idR. Leichter zu spezifizieren als zu verallgemeinern

# Bestandteile des OOP: Objekte

- IHK-Sprache: Objekte sind Instanzen bzw. spezifische Versionen einer Klasse
- Objekte sind Instanzen: `str(x)` -> erzeuge ein str-Objekt
- Objekte sind spezifisch:
  - Klasse Person beschreibt woraus ein Objekt *bestehen wird*
  - Objekte der Klasse Person haben Werte für z.b. Name, Alter, Beruf
- Objekte besitzen Eigenschaften (Attribute)
- Objekte besitzen Methoden (Fähigkeiten)
- Mindfuck: Sogar Klassen sind Objekte
  - `type(int)` # <class 'type'> # Objekt vom Typ Typ



# Bestandteile des OOP: Attribute

- Beschreibungen der spezifischen Merkmale
- Z.b. *Name* = Klaus; *Alter* = 39; *Beruf* = Anwalt
- Werden meistens bei der Instanziierung gesetzt
- Hier besonders wichtig: Abstraktion
  - Solider Ansatz: Welche Attribute brauche ich unbedingt um 2 Objekte eindeutig voneinander unterscheiden zu können?
- Klassen können Attribute mit Defaultwerten haben
  - Die gelten dann für alle Objekte der Klasse aber können auch überschrieben werden

# Bestandteile des OOP: Methoden

- Fähigkeiten des Objekts
- Ihr kennt schon viele:
  - `list.append()`, `str.upper()`, `dict.update()` usw.
- Grundsätzlich wird unterschieden:
- Getter-Methoden:
  - Informationen eines Objekts preisgeben (evtl. verändert)
  - Z.b. „gebe Namen einer Person aus“
- Setter-Methoden:
  - Attribute des Objekts verändern
  - Z.b. „erhöhe das Alter einer Person um 1“
- Getter & Setter sind keine Pflicht!

# Warum OOP wählen?

- Modularer Aufbau des Codes:
  - Klassendefinitionen sind eigener Namespace
  - Vereinfachung der Fehlersuche
  - Vereinfachung von Tests
- Weniger Wiederholungen von ähnlichem Code:
  - Ähnliche Sachverhalte zusammenfassen in Klassen
  - Insbesondere durch Vererbung sehr effizient
- Für große Projekte:
  - Im Vorfeld festlegen, wie die Attribute & Methoden heißen
  - Dann kann man schon so programmieren, als würde die Klasse existieren
  - Einfacher für den PM zu koordinieren (z.b. Person A entwickelt Methode x)
- ABER: OOP ist niemals Pflicht und erleichtert nicht immer das Programmieren
  - viele werden euch das Gegenteil erzählen

# Übung

## 1. Modelliere das Objekt

- Attribute?
- Methoden?



## 2. Abstrahiere das Objekt zu einer Klasse

- Leitfrage: Was ist die allgemeine Kategorisierung dieses Objekts?
- Tipp: Wähle ein anderes Objekt der Klasse und vergleiche die Attribute & Methoden

# Hausaufgabe

- Ermittle 2 Objekte in deiner näheren Umgebung, die ähnlich sind
  - Alternativ: Denke dir 2 ähnliche Objekte aus
- 1. Nenne mind. 2 Attribute und deren Werte pro Objekt
- 2. Nenne mind. 2 Methoden pro Objekt
- 3. Abstrahiere die Objekte zu einer Klasse