TECHNISCHE UNIVERSITÄT DRESDEN

PYTHONKURS FÜR INGENIEUR:INNEN

Objektorientierte Programmierung in Python

Folien: C. Statz, D. Pataky, C. Knoll

Dresden, WiSe 2020/21

Programmierparadigmen

- Wunsch: Wiederverwenbarkeit schon implementierter Funktionalität
- Copy & Paste?
  - Häufige Quelle von Fehlern
    (nicht alles notwendige geändert)
  - Spätere Änderungen an vielen Stellen notwendig → Aufwand

→ Problem lässt sich mit verschienden Paradigmen angehen

- Was ist ein Programmierparadigma?
  - Satz von Regeln für formale und strukturelle Code-Gestaltung
- Wozu dient es?
  - Unterstützung bei Erstellung von gutem Code
  - Bestimmte Herangehensweise nahelegen/priorisieren
- Kein Dogma!
  - Anwendung abhängig vom konkreten Problem (u. Geschmack)
  - Anders als z.B. Java erzwingt Python kein bestimmtes Paradigma
  - Kombinationen möglich

Bisher genutzte Paradigmen

Prozedurale Programmierung

```
def quadrat_zahl(x):
    return x**2
quadrat_zahl(7) # -> 49
```

Modulare Programmierung

```
from ipydex import IPS, activate_ips_on_exception
from numpy import array
import sympy as sp
```

Objektoriente Programmierung (OOP)

- Beschreibung eines komplexen Systems als Zusammenspiel von Objekten
- Objekte bestehen aus Daten („**Attribute**“) und zugehörigen Funktionen („**Methoden**“)
- Objekte sind **Instanzen** einer **Klasse**
  d.h. ein Objekt ist die konkrete Variable, die Klasse ist der Datentyp

→ Objektorientierung ist umfangreiches Konzept

- Genug Details für ein ganzes Semester

- Hier: wichtigste Begriffe und Prinzipien klären

Beispiel für ein Objekt

Ein Ball mit Attributen und Methoden

- Der Ball hat z.B. die Eigenschaften (Attribute)
  - Radius
  - Material
  - ...
  - (Substantive)
- Der Ball stellt auch gewisse Aktionen (Methoden) bereit
  - Ball werfen (x,y,z-Ebene)
  - Ball rollen (x,y-Ebene)
  - ...
  - (Verben)

Der Ball ist eine spezielle Kugel

Listing: ex01_kugel.py

```
class Kugel():
    """Eine Klasse ist allgemeiner Bauplan eines Objekts"""

    def __init__(self, radius, mittelpunkt=(0, 0, 0)):
        """
        Initialisierungs-Methode. Wird bei Objekt-Erstellung automatisch ausgeführt.
        Enspricht ungefähr dem 'Konstruktor' in anderen Programmiersprachen
        """
        self.radius = radius # z.B. in cm
        self.mittelpunkt = mittelpunkt # als (x, y, z)-Tupel

    def berechne_volumen(self):
        r = self.radius
        return (4/3)*pi*(r**3)
```

Instanziierung

Auch andere Bälle sind Kugeln:

```
# Definition der Klasse
class Kugel():
    def __init__(self, radius, mittelpunkt=(0,0,0)):
        ...
# Instanziierung der Klasse
# (Argumente werden an __init___ uebergeben)
fussball = Kugel(radius=21)
tennisball = Kugel(3)
flummi = Kugel(1)

print("Flummi-Radius:", flummi.radius) # Zugriff auf Attribut
V = tennisball.berechne_volumen() # Zugriff auf Methode
```

- Klasse `Kugel` ist nur der Bauplan
- Instanziierung: Erstellung von konkreten Objekten nach dem Bauplan
- Eigener Speicherbereich für jedes Objekt

→ Attribut-Werte sind unabhängig voneinander
(jede `Kugel` -Instanz hat eigenen Radius, Mittelpunkt, ...)

- Jedes Objekt hat eindeutige Speicher-Adresse (auslesbar mit `id(...)`)

Vererbung

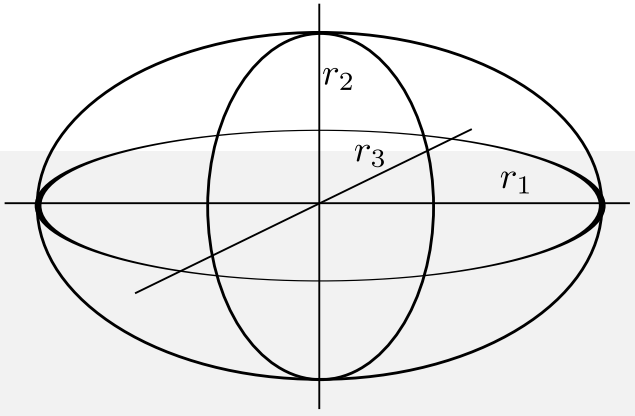
- Schaffung einer neuen Klasse basierend auf einer bestehenden
- Nur begrenzte Analogie zur biologischen Vererbung
- Typischer Fall: Vererbung vom Abstrakten zum Speziellen
- Darstellung: `Basisklasse` $\leftarrow$ `Kindklasse` („$\leftarrow$“ $\hat{=}$ „erbt von“)
- Bsp: `Nahrungsmittel` $\leftarrow$ `Brot` $\leftarrow$ `Schwarzbrot`
- Kindklasse hat alle Attribute/Methoden der Basisklasse
  - Können überschrieben werden
- Zusätzliche Attribute/Methoden in Kindklasse möglich

Wozu ist Vererbung gut?

- Gemeinsame Nutzung von Struktur und Code (Attribute und Methoden)
  $\rightarrow$ Reduziert Implementierungsaufwand
- Dokumentation der Ähnlichkeiten zwischen Klassen

Vererbung in Python

Die Kugel ist ein spezieller Ellipsoid




```
class Ellipsoid():
    def __init__(self, r1, r2, r3):
        ...

class Kugel(Ellipsoid):
    def __init__(self, radius):
        # Kugel ist ein Ellipsoid mit nur einem Radius
        Ellipsoid.__init__(self, radius, radius, radius)
```

- Klasse `Kugel` wird hier von Klasse `Ellipsoid` abgeleitet
- Attribute und Methoden werden vererbt
- Konstruktor wird überschrieben, sodass er mit einem Radius auskommt.

Mehrere Ebenen

```
class GeometrischesObjekt():
    ...
class Quader(GeometrischesObjekt):
    ...
class Ellipsoid(GeometrischesObjekt):
    ...
class Kugel(Ellipsoid):
    ...
tennisball = Kugel(3)
```

- Klasse `GeometrischesObjekt` ist hier Basisklasse (auch „Oberklasse“) aller weiteren Klassen
- Verdeutlichung mittels der python-Funktionen `isinstance(...)` und `issubclass(...)`

```
isinstance(tennisball, Kugel) # True
isinstance(tennisball, GeometrischesObjekt) # True
isinstance(tennisball, Quader) # False
issubclass(Kugel, Ellipsoid) # True
issubclass(Kugel, Quader) # False
```

Python-Besonderheit (1): `self`

- Eine Methode ist eine zu einem Objekt gehörende Funktion
- Bei Ausführung einer Methode muss bekannt sein, zu welchem konkreten Objekt (d.h. zu welcher Instanz) diese Methode gehört

→ Übergabe der Instanz als implizites erstes Argument

- Übliche Bezeichnung (Konvention): `self`
- Mit anderen Worten: `self` ist Platzhalter für die konkrete Instanz zu einem Zeitpunkt wo diese noch nicht existiert

Listing: ex02_self.py

```
# Hinweis: die Funktion id(...) gibt für jedes Objekt eine eindeutige
# Identitäts-Nummer zurück. Gleiche Nummer heißt: selbes Objekt

class KlasseA():
  def m1(self):
      print(id(self))

  def m2(x): # !! self vergessen
      print(x)

a = KlasseA() # Instanz erzeugen

a.m1() # kein explizites Argument (aber implizit wird a übergeben)
print(id(a)) # Übereinstimmung

a.m2() # kein Fehler (entspricht print(a), weil x die Rolle von self übernimmt)
a.m2(23) # Fehler: Zu viele Argumente übergeben (a (implizit) und 23 (explizit))
```

Besonderheit (2): alles ist ein Objekt

- Keine „primitiven Datentypen“ (wie in Java oder C++)
- Alles ist Objekt:
  - Zahlen (Instanzen von `int`, `float`, ...)
  - Zeichenketten (Instanzen von `str`, `bytes`, ...)
  - Auch Funktionen und Klassen sind Objekte

```
class KlasseA():
    pass

def funktion1():
    pass

type(KlasseA) # -> classobj
type(funktion1) # -> function
```

- Schlüsselworte sind keine Objekte!

```
type(while) # Syntax-Fehler
type(def) # Syntax-Fehler
type(class) # Syntax-Fehler
```

Zusammenfassung

Vorgestellte Begriffe

- Klasse
- Instanz (= Objekt)
- Attribut
- Methode
- Konstruktor
- Basisklasse
- Vererbung

Vorgestellte Python-Konstrukte

- `class`, `isinstance(...)`, `issubclass(...)`, `self`, `id(...)`, `type(...)`

Weitere Themen mit OOP-Bezug:

- Kapselung
- Polymorphie
- Ducktyping
- Mehrfachvererbung
- statische Methoden
- Meta-Klassen-Programmierung
- Operatorüberladung und „Magische Methoden“

Links zur Objektorientierten Programmierung

- Offizielle Python-Doku zu Klassen, Instanzen, `self` etc.
- Demystifizierung von `self` sowie `__init__` vs. `__new__`
- Einführender Blog-Beitrag