Skriptsprachen: Python
Klassen und Objekte

Jan Krüger, Alexander Sczyrba

Technische Fakultät
Universität Bielefeld

12. Februar 2019

Python

–

Klassen und Objekte

Übersicht

- Was ist noch mal OOP?
- Klassen
  - Methoden
  - Konstruktor / Destruktor / Attribute
  - Sichtbarkeit
  - Statische Attribute und Methoden
- Vererbung
- Magic Members

Was ist noch mal OOP?

- Datenkapselung
- Methoden/Funktionen
- Sichtbarkeit
- Vererbung (Wiederverwendbarkeit)

Klassen

- Objekte werden durch Klassen definiert

```
class bar(object):
    pass
```

- `pass` ist eine `noop` build-in Funktion
  - Platzhalter fuer *Prototyping*
  - *fertige* Programme enthalten keine `pass` Anweisung

Methoden

- Methoden ähneln Funktionen und werden genauso definiert
  - `self` ist **immer** erstes Argument
  - Definition innerhalb des `class` Blocks

```
class bar(object):
      def noop(self):
         pass
      def jump(self,von, nach):
         pass
```

- Objekt von einer Klasse erzeugen : `obj = bar()`
- `obj` ist eine Referenz auf eine Objektinstanz
- Referenz explizit löschen : `del(obj)`

Kon- und Destruktoren, Attribute

- Konstruktor wird bei der Objektinstanziierung aufgerufen
- Konstruktor werden wie Methoden definiert : `__init__(self)`
- beliebig viele Argumente (`self` ist erstes Argument)
- Destruktor hat keine Argumente (ausser `self`) : `__del__(self)` und wird vom GC aufgerufen **bevor** das Objekt gelöscht wird.
- Attribute i.d.R. im Konstruktor definiert

```
class foo(object):
      def __init__(self):
          self.a = 1
          self.text = "Hello␣World"
```

Sichtbarkeit

- bisher alle Methoden/Attribute public
- nicht immer sinnvoll
- Einschränkung der Sichtbarkeit durch Namensgebung

`name`	public	
`_name`	protected	**A**chtung!
`__name`	private	

- Zugriff auf private Member $\rightarrow$ AttributeError

statische Attribute und Methoden

- statische Attribute werden in der Klassen definiert
- Zugriff erfolgt mittels `KLASSENNAME.ATTRIBUTENAMEN`
- statische Methoden haben **k**ein `self` Argument
- Definition mittels `staticmethod`

```
class Konto(object):
    Anzahl = 0

    def zeigeAnzahl():
        print Konto.Anzahl

    zeigeAnzahl = staticmethod(zeigeAnzahl)

    ...
```

Aufgabe

Erstellt einen **Stack** ohne dabei auf die von Python zur Verfügung gestellten Datentypen/Objekte zurückzugreifen. Der **Stack** soll die Funktionen **push** und **pop** zur Verfügung stellen. (Nutzt dabei das Gelernte aus den Kapiteln Module bzw. Klassen und Objekte).

Vererbung

- bisher nur : Kapselung von Daten und Methoden

Vererbung

- bisher nur : Kapselung von Daten und Methoden
- jetzt : Vererbung
- jede Klasse muss **object** erweitern

```
class bar (object):
        ...
```

- foo erbt von bar und erweitert um ...

```
class foo (bar):
        ...
```

Vererbung (2)

- definiert die Erweiterende Klasse `__init__`, dann sollte `__init__` der Oberklasse aufgerufen werden.

```
class foo (bar):
      def __init__(self):
          bar.__init__(self)
          ...
```

  Vorsicht vor undefinierten Zustaenden!
- in Python *Mehrfachvererbung* moeglich
- Reihenfolge von links nach rechts

Aufgabe

Schreibe eine Klasse **ExtStack**, welche die Klasse um die Methoden **empty** und **size** erweitert.

Magic Member

- spezifizieren Eigenschaften einer Klasse
- Attribute und Methoden
- starten und enden immer mit zwei `__`
- eine Auswahl:

`__init__(self[,...])`	Konstruktor
`__del__(self)`	Destruktor
`__str__`	String Repräsentation
`__[eq|ne]__(self,other)`	(Nicht-)Gleichheit
`__[lt|le|gt|ge]__(self,other)`	Vergleichbarkeit
`__cmp__(self,other)`	Vergleichbarkeit
`...`	

- $\rightarrow$ Python Dokumentation