# Workshop 2009
# OOP mit Python

Karl Glatz
HS Ravensburg-Weingarten

5. Dezember 2009

# Inhalt

- Syntax und Abgrenzungen
- Zugriffsrechte
- Statische Methoden
- Vererbung
- Unterschiede zu Java/C++
- Magic Methods
- Vergleiche von Objekten
- Error Handling

# Syntax: Klassen

```
class MeineKlasse(object):
    """Ein Beispielklasse"""
    Anzahl = 0

    def __init__(self, name):
        self._name = name


    def meine_methode(self):
        return 'hallo welt!' + self._name
```

Statische Klassen Variable

Konstruktor

Methode

Objekt Variable (Member)

→ Jede Methode hat als Erstes Argument self

# Syntax: Objekte

```
class MeineKlasse(object):
    """Ein Beispielklasse"""
    Anzahl = 0

    def __init__(self, name):
        self._name = name


    def meine_methode(self):
        return 'hallo welt!' + self._name
```

```
obj = MeineKlasse("Hans")
obj.meine_methode()
MeineKlasse.meine_methode(obj)
```

# Objekte und Namen

- Alles sind Objekte!

```
>>> x = 5
>>> x.__class__
<type 'int'>
```

- Auch primitive Typen sind Klassen
- Ein Objekt kann mehrere Namen besitzen
  - Referenz / Pointer
- Daher: Call-by-Reference

# Eingebaute Objekte

- Dictionaries
- Funktionen (...)

```
def function1():
    print 'Eins.'
def  function2():
    print 'Zwei.'
def  function3():
    print 'Drei.'

# switch is our dictionary of functions
switch = {
    'one': function1,
    'two': function2,
    'three': function3,
    }
print switch['one']
```

# Zugriffsrechte

- Realisiert über Konvention

```
class Auto:
    def __init__(self): # Konstruktor
        self.pub = "Public Variable"
        self._prot = "Protected OHNE Zugriffsschutz"
        self.__private = "Private Variable"
        test = 1 # Lokale Variable
```

- „_“ → Protected
  „___“ → Privat

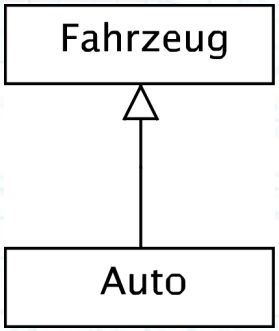
# Statische Methoden

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

    def zeigeAnzahl():
        print "Die Instanzanzahl ist", Klasse.Anzahl

    zeigeAnzahl = staticmethod(zeigeAnzahl)
```

- Methode ohne self als erstes Argument
- staticmethod(<Methode>) Aufrufen
- Alternativ: Function Decorator

# Vererbung

```
class Auto(Fahrzeug):
    typ = 2
    def __init__(self):
        Fahrzeug.__init__(self)
```




- Mehrfachvererbung möglich
  - `class Hausboot(Haus, Boot):`
- Alle Elemente können „überschrieben“ werden
- Standard: „object“ als Basis wird empfohlen

# Unterschiede zu Java/C++

- Keine abgeschlossene Struktur!!

```
obj = MeineKlasse()
obj.neues_attribut = 5
```

- Keine „echten“ Interfaces
- Alle Member sind immer „public“
- Methoden sind immer „virtuell“ [C++]
- Funktionen und Methoden sind Objekte

# Operator Überladen

- Sogenannte **MagicMethods**

```
>>> 1 + 2
3
>>> (1).__add__(2)
3
```

- __sub__ (Subtraktion, -)
- __cmp__ (Vergleich, <, >)
- __eq__ (Gleichheit, ==)
- Konstruktor: __init__(self)
- Destruktor: __del__(self)

# Beispiel: Magic Methods

```
for char in string:
    if char not in alphabet:
        raise (ValueError,
            "Char %s nicht im Alphabet  %a" %
             (char, alphabet))
```

- `string` muss hier kein String sein!
- Wichtig ist die Methode **__contains__**
- → Ermöglicht elegante Programme

# Vergleiche auf Objekte

- „==“ vs. „is“
  - „==“ vergleicht den Inhalt (__eq__)
  - „is“ vergleicht die „Signatur“

```
obj = Klasse()
obj2 = Klasse()
obj3 = obj
>>> obj == obj3
False
>>> obj is obj3
False
```

- None ist das „Null“-Objekt

# Error Handling

- Verarbeitung: **try** … **except**
- Kein Fehler: **else**
- Auslösen: **raise**

```
>>> def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print "division by zero!"
...     else:
...         print "result is", result
...     finally:
...         print "executing finally clause"
```

# Ende

Vielen Dank für die Aufmerksamkeit!

**Quellen**

http://docs.python.org/tutorial/classes.html
http://www.hs-augsburg.de/~phuewe/dva3/Python.odp (Master-Folie)