Objektorientierte Programmierung (OOP) in Python

Dr. Michael Savorić

Hohenstaufen-Gymnasium (HSG)

Kaiserslautern

Version 20110821

Überblick

- Einführungsbeispiel
- Basiskonzepte der OOP
- OOP in Python
- Beispiele und Übungen
- Hinweise, Anmerkungen und Spezielles
- Zusammenfassung, Ausblick und Anhang
- Literatur

Einführungsbeispiel: Beschreibung der Aufgabe

- Warteschlange, z.B. an einer Supermarktkasse:

Kunden stellen sich in der Warteschlange hinten an und verlassen sie vorne wieder: First-In-First-Out (FIFO)

Einführungsbeispiel: Verwenden einer Python-Klasse

```
>>> w = Warteschlange()
>>> w.ausgabe()
[]
>>> w.ankommen("Albert")
>>> w.ankommen("Bernd")
>>> w.ankommen("Christoph")
>>> w.ausgabe()
['Albert', 'Bernd', 'Christoph']
>>> w.verlassen()
'Albert'
>>> w.verlassen()
'Bernd'
>>> w.verlassen()
'Christoph'
>>> w.verlassen()
>>> ...
```

Wichtige Schlüsselwörter in Python bei der OOP

- class:
  - Leitet eine Klassendefinition ein

- self:
  - Als Parameter einer Prozedur oder Funktion:

    Prozedur oder Funktion gehört zu einer Klasse bzw. zu einem Objekt
  - Vor einer Variablen:

    Variable gehört zu einer Klasse bzw. zu einem Objekt

Einführungsbeispiel: Quelltext der Python-Klasse (Auszug)

```
class Warteschlange(object):
    def __init__(self):
        self.liste = [] # oder: self.liste = list()

    def ankommen(self, objekt):
        self.liste.append(objekt)

    def verlassen(self):
        if len(self.liste) > 0:
            objekt = self.liste.pop(0)
            return objekt
        else:
            return None

    def ausgabe(self):
        return self.liste
```

Einführungsbeispiel: Quelltext der Python-Klasse

```
class Warteschlange(object):
    def __init__(self):
        self.liste = []
        self.anzahl = 0

    def ankommen(self, objekt):
        self.liste.append(objekt)
        self.anzahl = self.anzahl+1

    def verlassen(self):
        if self.anzahl > 0:
            objekt = self.liste.pop(0)
            self.anzahl = self.anzahl-1
            return objekt
        else:
            return None

    def ausgabe(self):
        return self.liste
```

Basiskonzepte der OOP (1)

- Aufteilung der zu beschreibenden Welt in Objekte

- Objekt:
  - Besitzt Eigenschaften (Attribute)
  - Kann durch Operationen (Methoden) manipuliert werden oder Informationen preisgeben

Das Objekt selbst ist zuständig für die Speicherung und Verwaltung seiner Attribute (Objektautonomie, verteilte Zuständigkeit).

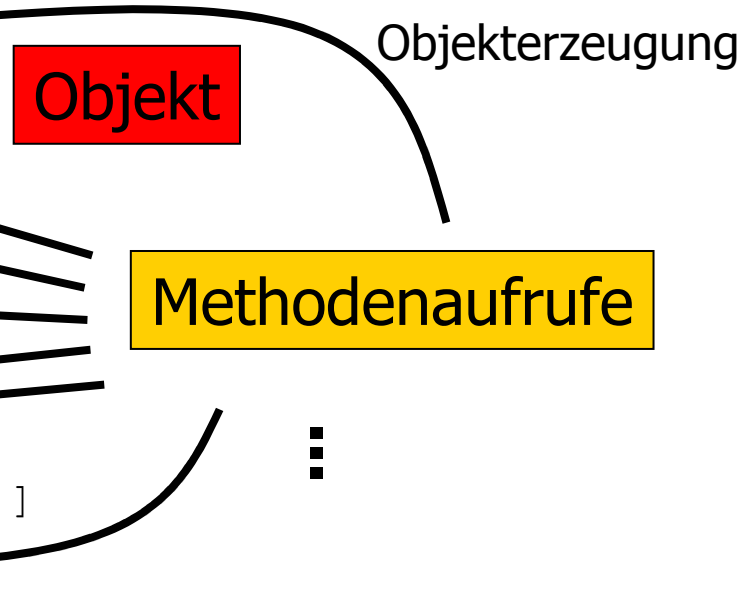
Basiskonzepte der OOP (2)

- Klasse:
  - Objekte mit gleichen Attributen und Methoden werden zu einer Klasse zusammengefasst, bzw.
  - Eine Klasse ermöglicht die Definition von Objekten mit gleichen Attributen und Methoden

Die Klasse legt die Attribute und Methoden eines Objekts dieser Klasse fest.

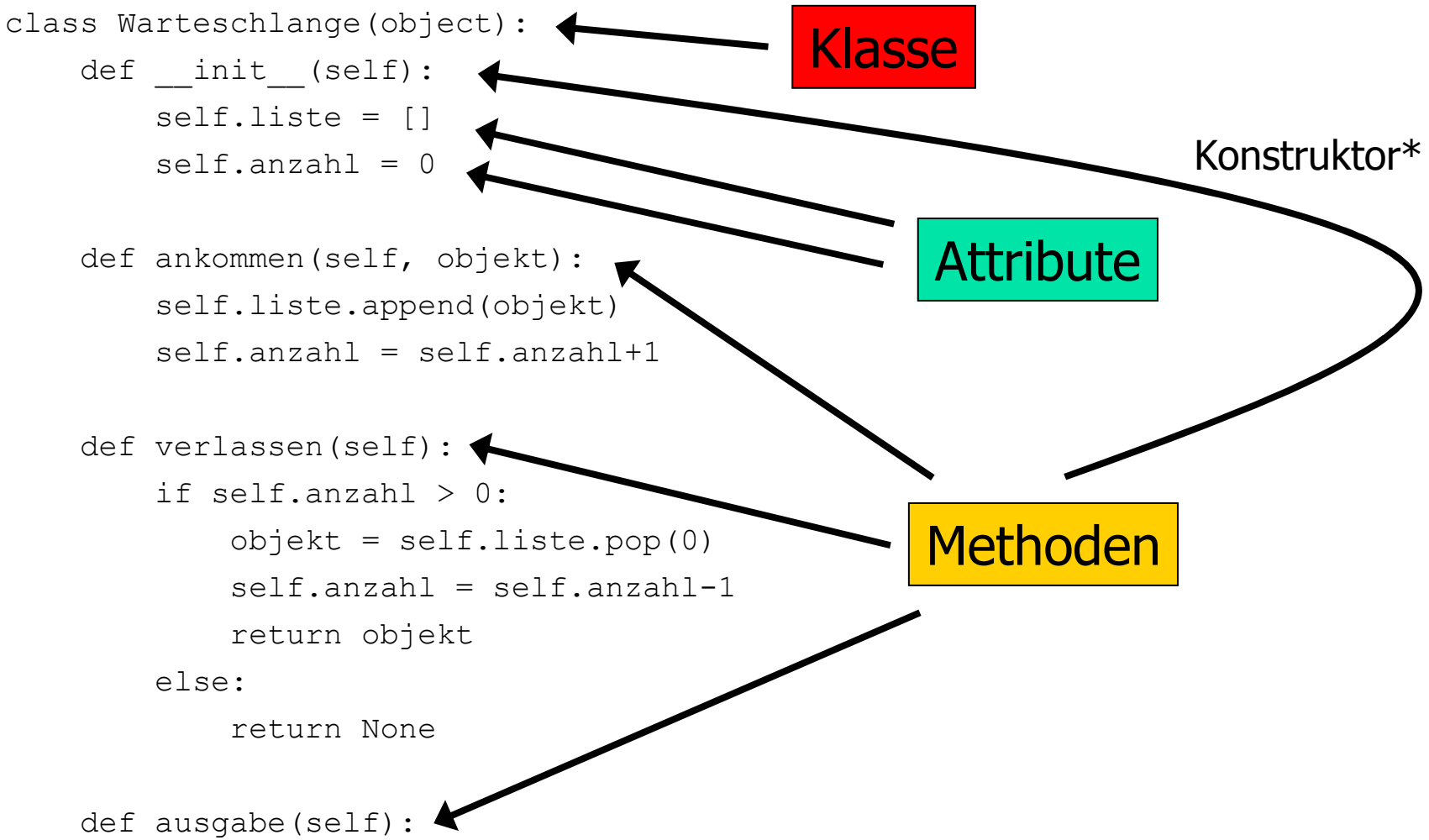
Einführungsbeispiel: Verwenden der Python-Klasse (Rückbl.)

```
>>> w = Warteschlange()
>>> w.ausgabe()
[]
>>> w.ankommen("Albert")
>>> w.ankommen("Bernd")
>>> w.ankommen("Christoph")
>>> w.ausgabe()
['Albert', 'Bernd', 'Christoph']
>>> w.verlassen()
'Albert'
>>> w.verlassen()
'Bernd'
>>> w.verlassen()
'Christoph'
>>> w.verlassen()
>>> ...
```



Einführungsbeispiel: Quelltext der Python-Klasse (Rückblick)

```
class Warteschlange(object):
    def __init__(self):
        self.liste = []
        self.anzahl = 0

    def ankommen(self, objekt):
        self.liste.append(objekt)
        self.anzahl = self.anzahl+1

    def verlassen(self):
        if self.anzahl > 0:
            objekt = self.liste.pop(0)
            self.anzahl = self.anzahl-1
            return objekt
        else:
            return None

    def ausgabe(self):
        return self.liste
```



Übungen – Teil 1

- Klasse Warteschlange ausprobieren:
  - Warteschlangen-Objekt erzeugen
  - Diverse Methoden des Warteschlangen-Objekts verwenden
  - Prüfen, ob man auf die Attribute eines Warteschlangen-Objekts von außerhalb des Objekts (lesend / schreibend) zugreifen kann – ist das sinnvoll?

- Klasse Warteschlange erweitern und testen:
  - Warteschlangen-Datei unter anderem Namen abspeichern und verändern
  - Neue Methoden definieren (eventuell sind zusätzliche Attribute notwendig!): anzahl_aktuell(...), leer(...), anzahl_insgesamt(...)

Basiskonzepte der OOP (3)

- Klassen können untereinander in Beziehung stehen:
  - Klasse A beinhaltet ein Objekt der Klasse B als Attribut
  - Klasse A verwaltet Objekte der Klasse B
  - Klasse A ist eine Erweiterung oder Spezialisierung der Klasse B (Vererbung)
  - ...

  Klassenabhängigkeiten und Klassenhierarchien bilden sich

- Einführungsbeispiel:
  - Klasse Warteschlange enthält ein Objekt der Klasse list und ein Objekt der Klasse int als Attribute
  - Klasse Warteschlange verwaltet Objekte anderer Klassen

Basiskonzepte der OOP (4)

- Geheimnisprinzip:
  - Auf die Attribute eines Objekts sollte von außen nicht direkt zugegriffen werden können (Datenkapselung)
  - Die interne Struktur eines Objekts kann unter Beibehaltung der Schnittstellen jederzeit geändert werden (Transparenz)

- Einführungsbeispiel:
  - Datenkapselung ist nicht berücksichtigt worden
  - Transparenz jedoch ist gewährleistet

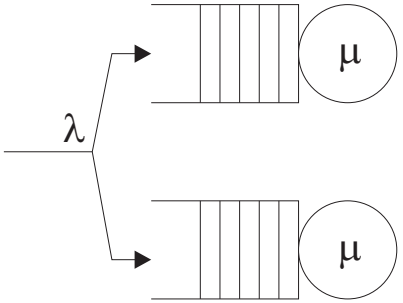
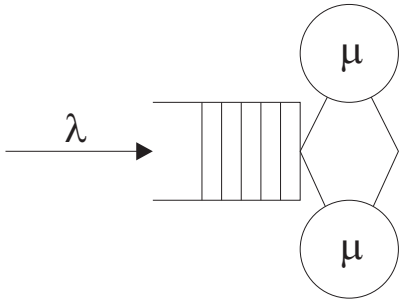
Einführungsbeispiel: Verbesserung (Datenkapselung)

```
import copy

class Warteschlange(object):
    ...

    def ausgabe(self):
        return copy.deepcopy(self.liste)
```

Es wird eine echte Kopie (= tiefe Kopie) der Liste self.liste zurückgeliefert und keine Referenz (= flache Kopie)!

Einführungsbeispiel: Anwendung der Python-Klasse (1)

- Simulation: Kundenverhalten wird stochastisch modelliert
- Verschiedene Strategien:

1. Strategie	2. Strategie
 "Supermarkt"	 "Post"

- Ergebnis: Die "Post"-Strategie ist deutlich überlegen!

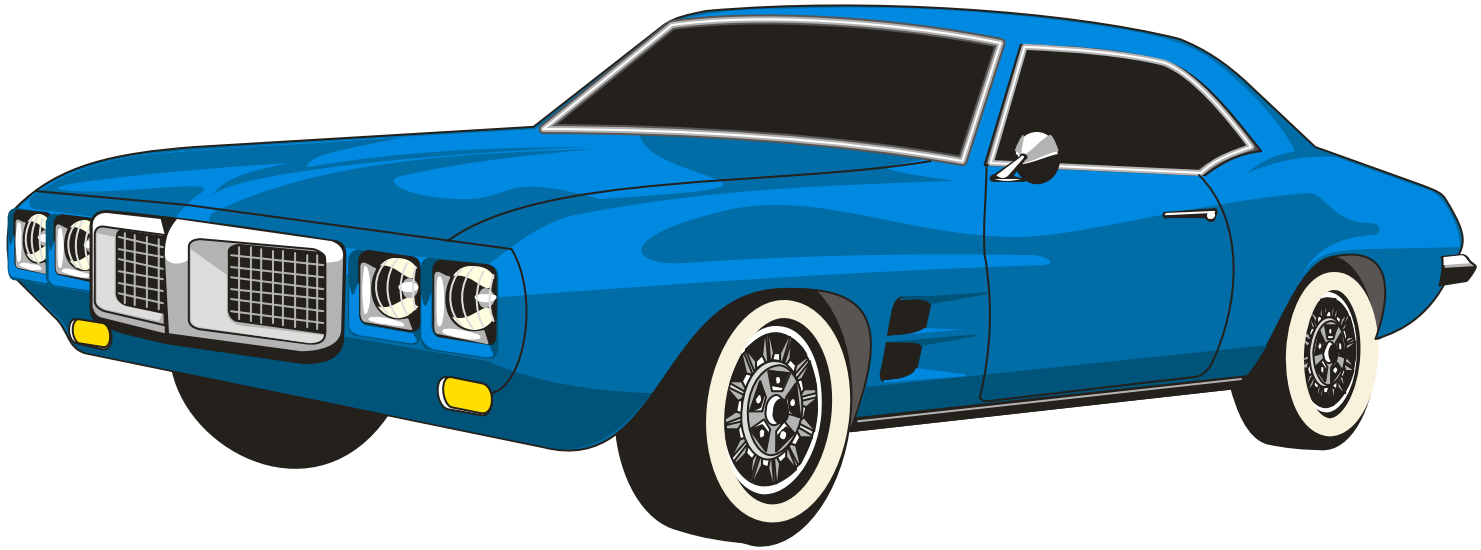
Einführungsbeispiel: Anwendung der Python-Klasse (2)

- Verteilte Anwendung: Auktion

- Bieter (Clients) schicken Gebote an das Auktionshaus (Server)

- Gebote sollen im Auktionshaus in der Reihenfolge ihres Eintreffens verarbeitet werden, d.h.:
  - Eventuell neues Höchstgebot bestimmen und veröffentlichen
  - Immer: den aktuellen Bieter informieren über seinen Gebotsstatus (Höchstbieter: ja / nein) und das aktuelle Höchstgebot
  - Bei neuem Höchstgebot: zusätzlich den Bieter des alten Höchstgebots über das aktuelle Höchstgebot informieren

Beispiel: Ein reales Objekt



Attribute und Attributwerte	Modell: Pontiac Firebird, Farbe: blau, Tachostand: 42000, PS: 340, VMax: 220, V: 0, Licht: aus, …
Methoden	beschleunigen, bremsen, Licht einschalten, Licht ausschalten, hupen, blinken, …

Beispiel: das reale Objekt als abstraktes Objekt

traumauto	
Attribute und Attributwerte	modell = “Pontiac Firebird“ farbe = “blau“ ps = 340 vmax = 220 v = 0
Methoden	ermittle_modell() setze_farbe(farbe) ermittle_farbe() setze_geschwindigkeit(v) aendere_geschwindigkeit(delta_v) ermittle_geschwindigkeit()

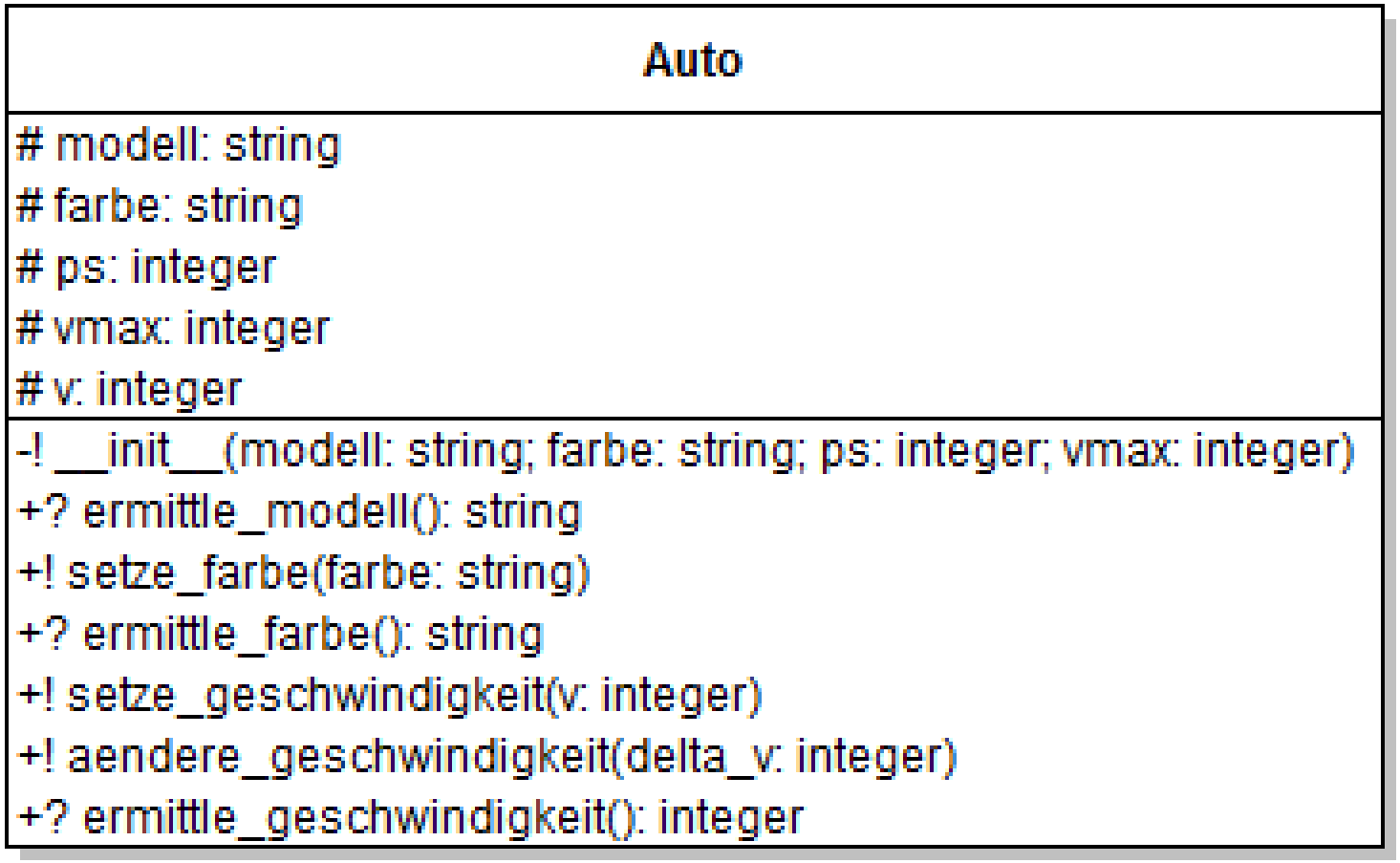
Beispiel: Verwenden des abstrakten Objekts

```
>>> traumauto = Auto(“Pontiac Firebird“, “blau“, 340,
220)
>>> traumauto.ermittle_modell()
'Pontiac Firebird'
>>> traumauto.setze_farbe(“gruen“)
>>> traumauto.ermittle_farbe()
'gruen'
>>> traumauto.setze_geschwindigkeit(200)
>>> traumauto.ermittle_geschwindigkeit()
200
>>> ...
```

Beispiel: allgemeine Klasse Auto (Pseudocode)

Klasse Auto	
Attribute	modell: string farbe: string ps: integer vmax: integer v: integer
Methoden	proc __init__(modell: string; farbe: string; ps: integer; vmax: integer) func ermittle_modell(): string proc setze_farbe(farbe: string) func ermittle_farbe(): string proc setze_geschwindigkeit(v: integer) proc aendere_geschwindigkeit(delta_v: integer) func ermittle_geschwindigkeit(): integer

Beispiel: allgemeine Klasse Auto (UML-Diagramm)



Erstellt mit: violet (Java-Programm)

Symbole im UML-Diagramm

Symbol:	Bedeutung:
+	"public": unbeschränkter öffentlicher Zugriff auf ein Attribut oder eine Methode
#	"protected": geschützter Zugriff auf ein Attribut oder eine Methode von der Klasse oder von abgeleiteten Klassen
-	"private": nur die Klasse hat Zugriff auf ein Attribut oder eine Methode
?	eine Methode ist eine Funktion
!	eine Methode ist eine Prozedur

Übungen – Teil 2

- Auto-Klasse:
  - Schreiben Sie eine Auto-Klasse in Python, die die im Pseudocode bzw. im UML-Diagramm beschriebenen Attribute besitzt und die aufgeführten Methoden entsprechend ihrer Beschreibung anbietet
  - Testen Sie Ihre Auto-Klasse

Beispiel: Mögliche Anwendungen einer Auto-Klasse

- Rennsimulation bzw. Rennspiel
- Fuhrpark einer Autovermietung verwalten
- Modellierung von Verkehrsgeschehen (Staus, Unfälle, ...)
- Autos eines Autohauses
- ...

Je nach Anwendung sind unterschiedliche Auto-Klassen notwendig!

OOP in Python: Grundsätzliches

- Bei der Objekterzeugung Aufruf eines Konstruktors*:

  __init__(...)

- Bei der Objektfreigabe Aufruf eines Destruktor (nur bei Bedarf*):

  __del__(...)

- Innerhalb eines Objekts kann durch die Variable

  self

  auf das Objekt verwiesen werden

- Datenkapselung von privaten Attributen durch __ vor dem Attributnamen in einer Methode, z.B.:

  self.__modell = "Pontiac Firebird"

OOP in Python: Auto-Klasse (Auszug) als Beispiel

```
class Auto(object):
    def __init__(self, modell, farbe, ps, vmax):
        self.__modell = modell
        self.__farbe = farbe
        self.__ps = ps
        self.__vmax = vmax

        self.__v = 0

    def ermittle_modell(self):
        return self.__modell
    ...
```

OOP in Python: Verwenden der Auto-Klasse

- Objekt erzeugen:

  traumauto = Auto(“Pontiac Firebird“, “blau“, 340, 220)

- Objekt verwenden, z.B.:

  traumauto.ermittle_modell()

  traumauto.aendere_geschwindigkeit(200)

- Objekt explizit freigeben:

  del traumauto

OOP in Python: Details zur Datenkapselung (Auto-Klasse)

- Was nicht geht (Datenkapselung!):

  traumauto.__modell

- Was aber geht:

  traumauto._Auto__modell

- Folgerung:

  Die Datenkapselung ist in Python durch einen "Trick" zu umgehen (das sollte man den Schülern erst später mitteilen!)

OOP in Python: Details zur Datenkapselung (allgemein)

- Attribute ohne Unterstrich zu Beginn des Namens sind öffentlich (public, UML: +)

- Attribute mit einem Unterstrich zu Beginn des Namens sind zwar öffentlich, aber dennoch als schützenswert (protected, UML: #) markiert

- Attribute mit zwei Unterstrichen zu Beginn des Namens sind als privat (private, UML: -) markiert und sind von außerhalb eines Objekts nicht zugreifbar (außer über den erwähnten "Trick")

Vereinfachende Vereinbarungen für diesen Kurs

- Verzicht auf private Attribute in einer Klasse
- Verzicht auf die ermittle_...(...)-Methoden
- Verzicht auf die setze_...(...)-Methoden, falls sinnvoll

OOP in Python: neue Auto-Klasse (Auszug) als Beispiel

```
class Auto(object):
    def __init__(self, modell, farbe, ps, vmax):
        self.modell = modell
        self.farbe = farbe
        self.ps = ps
        self.vmax = vmax

        self.v = 0

    def ermittle_modell(self):
        return self.modell
    ...
```

vgl. Folie 27

Vererbung

- Bestehende Klassen können erweitert oder spezialisiert werden

- Eine neue Klasse baut auf einer oder mehreren bestehenden Klassen auf und definiert lediglich die Änderungen bzw. Erweiterungen

- Bestehende Klassen können so wiederverwendet werden

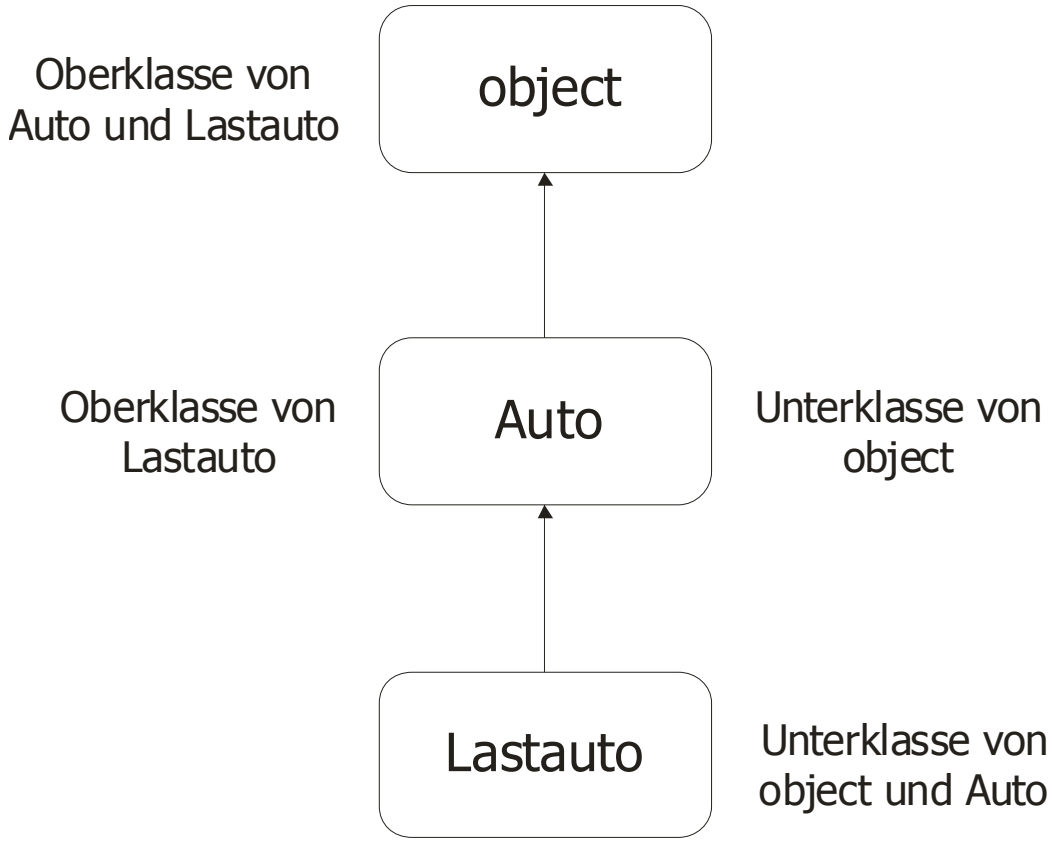
OOP in Python: Vererbungs-Beispiel (1)

- Die Auto-Klasse soll erweitert werden, so dass auch die Zuladung eines Autos berücksichtigt wird

- Es wird eine neue Klasse Lastauto definiert, die:
  - Sämtliche Attribute und Methoden der Klasse Auto übernimmt
  - Zwei neue Attribute mit den Namen zuladungmax und zuladung besitzt
  - Zwei neue Methoden setze_zuladung(...) und aendere_zuladung(...) erhält

Die Klasse Auto heißt Oberklasse der Klasse Lastauto, die Klasse Lastauto heißt Unterklasse der Klasse Auto

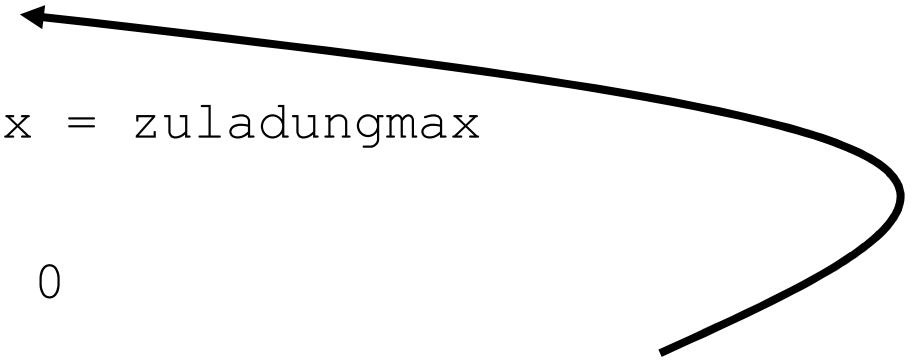
OOP in Python: Vererbungs-Beispiel (2)



Die höchste Oberklasse in einer Klassenhierarchie heißt Basisklasse

OOP in Python: Vererbungs-Beispiel (3)

```
from auto import Auto

class Lastauto(Auto):
    def __init__(self, modell, farbe, ps, vmax,
                 zuladungmax):
        super().__init__(modell, farbe, ps, vmax)

        self.zuladungmax = zuladungmax

        self.zuladung = 0

    ...
```



Aufruf des Oberklasse-Konstruktors*

Vererbung: Hinweise (1)

- In der Unterklasse unverändert übernommene Methoden der Oberklasse werden nicht neu definiert!

- In der Unterklasse werden nur Methoden definiert, die:
  - Aus der Oberklasse stammen und verändert werden müssen
  - In der Unterklasse neu hinzugekommen sind

Vererbung: Hinweise (2)

- Auf private Attribute oder Methoden einer Oberklasse können Unterklassen nicht zugreifen

- Attribute oder Methoden einer Oberklasse, auf die eine Unterklasse zugreifen soll und die nicht öffentlich sind, sind als geschützt zu deklarieren

- Kurs-Vereinbarung:

  Alle Attribute und Methoden der Klassen sind öffentlich

Übungen – Teil 3

- Klasse Konto erzeugen und testen:
  - Attribute: kontonummer, kontobesitzer, kontostand
  - Methoden: gutschrift(...), lastschrift(...), stand(...), ...

- Klasse Dispokonto erzeugen und testen:
  - Wie Klasse Konto, nur mit einem zusätzlichen Attribut dispo
  - Manche Methoden der Klasse Konto müssen angepasst werden

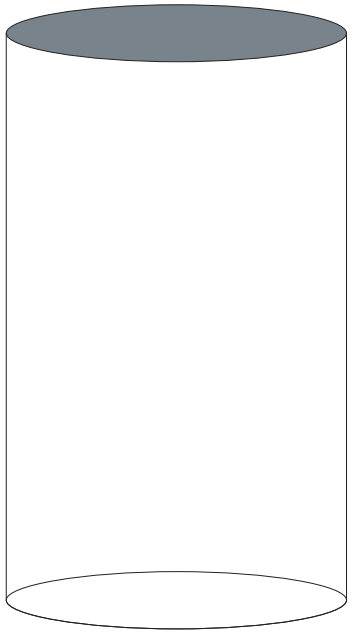
Objektorientierte Modellierung (OOM)

- Realität:
  - Einige Objekte können vieles gemeinsam haben, aber nicht alles
  - Einige Objekte können aus anderen Objekten zusammengesetzt sein
  - ...

- Modellierung:
  - Gemeinsamkeiten von Objekten erkennen und in einer Klassenhierarchie zusammenfassen
  - Verwendung bereits bestehender (und getesteter!) Klassen, falls möglich und sinnvoll
  - ...

OOM- Beispiel: Klasse Zylinder

- Parameter eines Zylinders:
  - Radius, Höhe



- Bestandteile eines Zylinders:
  - "Boden", "Decke"
  - Mantel

- Klasse Zylinder:
  - Sollte intern Objekte passender anderer Klassen verwenden

Übungen – Teil 4

- Klasse Zylinder:
  - Schreiben Sie eine Klasse Zylinder, die intern Objekte der Klassen Kreis und Rechteck verwendet
  - Versuchen Sie, so viele Methoden wie möglich der Klassen Kreis und Rechteck in Ihrer Zylinderklasse zu verwenden
  - Testen Sie Ihre Klasse Zylinder

Übungen – Teil 5

- Klasse Roboter:
  - Ein Roboter soll durch eine Welt wandern, bis er ein bestimmtes Feld in dieser Welt erreicht hat
  - Ein Hauptprogramm in der Datei roboter_anwendung.py (siehe nächste Folie) ist für diese Wanderung bereits vorhanden
  - Es ist in der Datei roboter.py (siehe übernächste Folie) eine zum Hauptprogramm passende Roboter-Klasse zu programmieren
  - Testen Sie Ihre Roboter-Klasse

Roboter-Hauptprogramm (roboter_anwendung.py)

```
from roboter import *
import random

roboter = Roboter(1, 1) # x = 1, y = 1, Blickrichtung = oben
print()
print(roboter.position()+" "+roboter.blickrichtung())
print()
while roboter.x != welt_x or roboter.y != welt_y:
    wert = random.uniform(0, 1)
    if wert <= 0.25: # 25 % Wahrscheinlichkeit für Drehen
        roboter.drehen() # Rechtsdrehung um 90 Grad
        print("Drehen.")
    else:
        roboter.laufen() # Einen Schritt in Blickrichtung gehen
        print("Laufen.")
    print(roboter.position()+" "+roboter.blickrichtung())

print()
print("Der Roboter hat insgesamt "+str(roboter.schritte)+\
      " Schritte und "+str(roboter.drehungen)+\
      " Drehungen ausgeführt.")
print()
```

Roboter-Klasse (roboter.py)

```
# Größe der Roboter-Welt:
welt_x = 10
welt_y = 10

# Klasse Roboter:
class Roboter(object):
    ...
```

Zusammenfassung und Ausblick

- OOP in Python ist einfach zu realisieren

- Bewährte Vorgehensweise in der OOP:
  - Welt in Objekte einteilen
  - Gemeinsamkeiten und Abhängigkeiten der Objekte ermitteln (OOM)
  - Klassenhierarchien erstellen und Klassen definieren
  - Bereits bestehende Klassen konsequent verwenden

- Weitere Möglichkeiten – zum Teil Python-spezifisch:
  - Siehe Anhang

Anhang

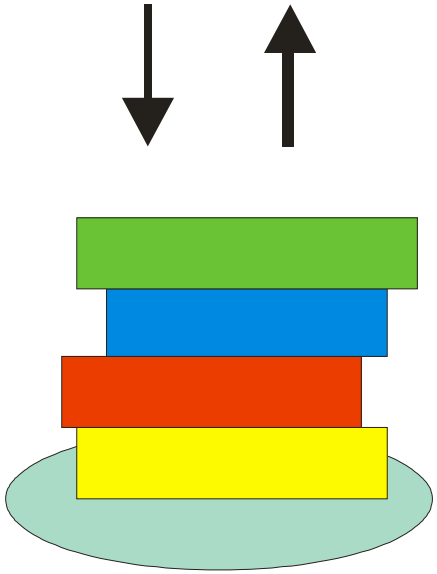
Weitere Übungen – Teil 1

- Bestehende Python-Klasse, z.B. list, an eigene Bedürfnisse anpassen

- Klasse Endliche_Warteschlange erzeugen und testen:
  - Klasse Warteschlange als Vorlage nehmen, umbenennen und anpassen
  - Neues Attribut anzahl_plaetze bestimmt die Anzahl der Plätze in der Warteschlange (sinnvolle Erweiterungsstelle überlegen!)
  - Bei einer vollen Warteschlange werden ankommende Objekte abgewiesen (Rückgabewert über den Erfolg / Misserfolg sinnvoll!)

Weiteres Übungsbeispiel: Beschreibung der Aufgabe

- Stapel, z.B. ein Bücherstapel:



Bücher werden auf den Stapel gelegt und von ihm genommen: Last-In-First-Out (LIFO)

Weitere Übungen – Teil 2

- Klasse Stapel erzeugen und testen:
  - Klasse Warteschlange als Vorlage nehmen, umbenennen und anpassen
  - Interne Verwaltung des Attributs liste kann frei gewählt werden, sollte aber zur Methode ausgabe(...) passen
  - Fachlicher Hinweis: Methoden ankommen(...) bzw. verlassen(...) heißen beim Stapel normalerweise push(...) bzw. pop(...)

- Klasse Endlicher_Stapel erzeugen und testen:
  - Klasse Endliche_Warteschlange oder Klasse Stapel als Vorlage nehmen (was ist sinnvoller?), umbenennen und anpassen
  - Attribut anzahl_plaetze und Rückgabewert beim Ankommen von Objekten

Erweiterte Stapel- und Warteschlangen-Klassen

- Methode look(...): Das nächste Objekt zurückliefern, ohne es vom Stapel oder aus der Warteschlange zu entfernen

- Anwendung des erweiterten Stapels: Türme von Hanoi
  - 3 Stapel
  - Objekte in den Stapeln sind Objekte der noch zu schreibenden Klasse Scheibe (Attribut: durchmesser)
  - Interaktion mit dem Benutzer:
    - Aktuellen Spielstand ausgeben
    - Auf Eingabe des Benutzers warten
    - Eingabe des Benutzers verarbeiten

OOP in Python: Besondere Methoden (1)

- Vergleichsmethoden:

Methode	Verwendung	Beschreibung
__lt__(self, objekt)	a < b	liefert True, wenn a < b, sonst False
__le__(self, objekt)	a <= b	liefert True, wenn a <= b, sonst False
__eq__(self, objekt)	a == b	liefert True, wenn a == b, sonst False
__ne__(self, objekt)	a != b	liefert True, wenn a != b, sonst False
__ge__(self, objekt)	a >= b	liefert True, wenn a >= b, sonst False
__gt__(self, objekt)	a > b	liefert True, wenn a > b, sonst False

OOP in Python: Besondere Methoden (2)

- Mathematische Operationen (Auszug):

Methode	Verwendung	Beschreibung
__add__(self, objekt)	a + b	addiert a und b
__sub__(self, objekt)	a - b	subtrahiert b von a
__mul__(self, objekt)	a * b	multipliziert a und b
__truediv__(self, objekt)	a / b	dividiert a durch b
__neg__(self)	-a	negiert a
__float__(self)	float(a)	wandelt a in einen Fliesskommawert

Hinweise

- Innerhalb vieler der hier nur auszugsweise vorgestellten besonderen Methoden müssen neue Objekte erzeugt, verändert und zurückgeliefert werden

- Es sind dann aber elegante und mächtige Rechnungen möglich, z.B.:

```
>>> a = Bruch(1, 2)
>>> b = Bruch(2, 3)
>>> c = -3*a+5*b
>>> c
Bruch(+11 / +6)
>>>
```

Beispiel 1: erweiterte Kontoklasse (Auszug)

- Testen auf Gleichheit bei Konto-Objekten: Konto-Objekte sollen gleich sein, wenn ihre Kontostände den gleichen Wert aufweisen

```
class Konto(object):
    ...

    def __eq__(self, konto):
        if self.kontostand == konto.kontostand:
            return True
        else:
            return False
```

Beispiel 2: erweiterte Autoklasse (Auszug)

- Testen auf Gleichheit bei Auto-Objekten: Auto-Objekte sollen gleich sein, wenn sie das selbe Auto beschreiben

```
class Auto(object):
    ...

    def __eq__(self, auto):
        if id(self) == id(auto):
            return True
        else:
            return False
```

Weitere Übungen – Teil 3

- Klasse Bruch:
  - Attribute: zaehler, nenner
  - Methoden: Vergleichsoperationen, mathematische Operationen, erweitern(...), ggt(...), kuerzen(...), kehrwert(...), echter_bruch(...), gemischte_schreibweise(...), ...

- Klasse Vektor:
  - Attribute: x, y, z
  - Methoden: betrag(...), __eq__(...), __ne__(...), __add__(...), __sub__(...), __mul__(...), __div__(...), ...
  - Funktionen für Vektor-Objekte (diese sind außerhalb der Klasse zu definieren!): winkel(...), skalarprodukt(...), ...

Anmerkungen

- Der eigentliche Konstruktor in Python heißt __new__(...). Dieser wird jedoch nur selten selbst programmiert. Er wird, falls er nicht selbst programmiert worden ist, automatisch erzeugt und vor __init__(...) aufgerufen.

  Die __init__(...)-Methode dient genau genommen nur der Initialisierung des Objekts.

- Der Destruktor __del__(...) wird so gut wie nie selbst programmiert. Er wird auch nicht – wie in anderen Sprachen üblich – in jedem Fall bei der Freigabe eines Objekts aufgerufen.

Literatur

- Eigene Unterlagen (GK und LK)

- Internet:
  - http://www.python.org/
  - http://www.hsg-kl.de/faecher/inf/python/oop/index.php
  - ...

- M. Summerfield, Programming in Python 3, Addison-Wesley, 2009, ISBN-13: 978-0-13-712929-4, 44.99 $