Objektorientierung mit Python
SH-HILL Fachtagung 2012

Claus Albowski

Wozu eine "neue" Programmiersprache?

Sven Alisch hat bei der Einführungsveranstaltung zum ABI 2014 ...
für **Grundlegendes Niveau** vorgetragen

- modellieren einen Realitätsausschnitt mit OOM
- modellieren Beziehungen („hat ein“ / „benutzt ein“ / „ist ein“) zwischen Objekten
- modellieren mit Hilfe der UML
- erläutern die Kommunikation zwischen Objekten
- Sprachelemente sinnvoll anwenden (elementare Datentypen, Listen, Arrays)
- erläutern gegebenen Quellcode mit Fachbegriffen (Attribute, Methoden, Konstruktoren, ...)

für **Erweitertes Niveau** vorgetragen

- erkennen, nutzen und vergleichen Klassenbeziehungen (einfache Assoziation, Komposition und Aggregation)
- Sichtbarkeiten (nur Java)
- abstrakte Klassen (Hinleitung zur Polymorphie)
- Bewertung eines Modells hinsichtlich der Kohäsion und Kopplung

Grundlegendes Niveau heißt aber in vielen Fällen:

- zweistündiger Kurs
- Schülerinnen und Schüler mit keiner oder geringer Vorerfahrung mit Programmierung
- Schülerinnen und Schüler ohne ausgeprägte Grundmotivation

Idee zur Reduzierung der Probleme:

- kein Programmiersprachenwechsel in der Oberstufe
- Wahl einer Programmiersprache, mit der die Schülerinnen und Schüler ggf. schon in der Mittelstufe arbeiten können
- GUI-Programmierung, bei der die Schülerinnen und Schüler weitgehend selbstständig arbeiten können

Was bietet Python?

Python ist interaktiv

IDLE bietet die Möglichkeit, Anweisungen direkt auswerten zu lassen und das Ergebnis angezeigt zu bekommen
Der Editor ist ein struktureller Editor, er berücksichtigt also die Python-Regeln
Ein Python-Programm kann sofort ausgetestet werden

Python ist typgebunden

- Variable kennen ihren Typ
- Eine Deklaration von Typen ist nicht notwendig
- `a=10` definiert eine Variable a und weist ihr den Wert 10 zu, der damit Zahltyp ist
- `a="10"` [oder `a='10'` mit derselben Wirkung] definiert den String , also die Zeichenkette aus den beiden Zeichen 1 und 0.

Python ist prozedural

- Anweisungen werden sequenziell ausgeführt

- Mit der Anweisung
  ```
  def <Name>(<par>):
     <Anweisung 1>
     <Anweisung 2>
  ```
  können Prozeduren definiert werden

Python ist funktional

- Mit der Definition
  ```
  def <Name>(<par>):
     return <Rückgabewert>
  ```
  können Funktionen definiert werden
- Funktionsschachtelungen werden "normal" ausgewertet
- Rekursion ist möglich , allerdings erkennt Python keine endrekursiven Aufrufe, was zu unnötigen Rekursionsabbrüchen führen kann.
- Es gibt die Möglichkeit anonyme Funktionen zu definieren
  ```
  lambda x: x*x
  ```
  ist die Quadratfunktion
- ```
  apply(lambda x,y: (x+y)/2, (3,5))
  ```
  wendet die anonym definierte Mittelwertfunktion auf das Tupel (3,5) an
- Funktionen können Parameter sein

Python ist vor allem aber objektorientiert

- Ein großer Teil der Sprache ist objektorientiert definiert, dadurch folgt die Syntax in der Regel durchgehend dem gleichen Konzept.
- Das hat allerdings auch Probleme zur Folge:
  - call-by-reference
  - Methoden verändern Objekte dauerhaft
  - Listen sind Objekte
  - Zuweisungen erzeugen keine neuen Objekte

Objektorientierung mit Python

Syntax am Beispiel

Klassenkopf und Konstruktor[1] :

```
class Zaehler:
      def __init__(self):
      self.stand = 0
```

Am Ende des Klassenkopfes kommt ein Doppelpunkt. Er leitet eine neue **Strukturebene** ein, die bei Python **zwingend durch Einrückung** gekennzeichnet ist.
Der Konstruktor wird durch die beiden Unterstreichungsstriche vorn und hinten als besondere Methode gekennzeichnet. [Davon gibt es weitere spezielle Methoden.]
Das Objekt selbst wird mit `self` gekennzeichnet und muss dem Konstruktor als Parameter übergeben werden. Die Bezeichnung `self` ist willkürlich, da sie dem `this` bei Java entspricht, könnte man sie auch so benennen. Das sollte man aber vermeiden, damit andere Python-Erfahrene nicht stutzen.

1 Es gibt auch eine vordefinierte Destruktormethode, die entsprechend `__del__` heißt.
```

Leider wird bei Python nicht nur gefordert, dass Zugriffe auf **Instanzvariable**[1] immer mit Punktnotation mit `self` gekennzeichnet werden, sondern auch die Zugriffe auf die **Instanzmethoden**. Erschwerend kommt noch hinzu, dass bei der Definition der Instanzmethoden `self` immer erster Parameter sein muss, so dass die Zahl der Parameter in der Definition immer um mindestens 1 größer ist als beim Aufruf und bei diesem eben mit Punktnotation vor dem Funktionsnamen stehen muss.
Das macht das Programmieren manchmal etwas unhandlich [ich vergesse es auch immer wieder einmal].

Mindestens um 1 größer?

Python kennt kein overload von Methoden. Allerdings kann man Methodenaufrufe [das gilt auch für Funktionen/Prozeduren außerhalb von Klassen] mit unterschiedlicher Signatur dadurch erzielen, dass man einige der Parameter vordefiniert. Die Funktion

```
def Flaechenberechnung(laenge, breite = 20):
      return laenge * breite
```

kann daher mit einem Parameter aufgerufen werden und nimmt dann für die Breite den vordefinierten Wert oder mit zwei Parametern und in dem Fall wird der vordefinierte Wert überschrieben. Es ist sogar ein Aufruf

```
      Flaechenberechnung(breite = 50, laenge = 70)
```

mit richtig zugeordneten Parametern möglich.

Kapselung

Kapselung gibt es in zwei Varianten: Ein Unterstreichungsstrich vor dem Attributnamen kennzeichnet "privat nach Vereinbarung", zwei Unterstreichungsstrichen vor dem Namen bedeuten eine private Variable, auf die man [wie bei Java] prinzipiell[2] nicht einmal von einer erbenden Klasse aus zugreifen kann.

Vererbung

Vererbung geht zunächst ohne große Überraschungen durch die Angabe der vererbenden Klasse hinter dem Klassennamen in Klammern , allerdings lässt Python Mehrfachvererbung zu. Man kann daher statt der bei Java üblichen Implementation von Interfaces [gibt es bei Python nicht] erben. Ob das gut ist, will ich hier nicht diskutieren.
Der Aufruf des Konstruktors der vererbenden Klasse ist zwingend notwendig! Es gibt also nicht wie bei Java einen impliziten Aufruf. Anders als bei anderen Methoden erfolgt der Aufruf hier mit dem Klassennamen vor `__init__` und `self` als erstem Parameter.

---

1 Es gibt auch Klassenvariable, ihre Definition erfolgt einfach im Kopf ohne die self-Kennzeichnung. Sie müssen auch bei internen Aufrufen jeweils mit dem Namen der Klasse vor dem Variablennamen gekennzeichnet werden.
Klassenmethoden im Sinne von static-Methoden bei Java kann man mit der Kennzeichnung @staticmethod in der Zeile vor der Methodendefinition kennzeichnen. Zugriffe auch hier immer mit dem Namen der Klasse vor dem Methodennamen.

2 Leider kann man das "austricksen" und dennoch von außen auf das "gekapselte" Attribut zugreifen.

OO Grafik mit wxPython

Zu Python gibt es mehrere Grafik-Toolkits .
Eine Erläuterung findet man im Python-Buch von Galileo-Computing [ISBN 978-3-8362-1110-9] , das man auch als eBook herunterladen kann. [http://www.galileo-computing.de]
Dort sind **Tkinter** , **PyGtk** und **PyQt** etwas ausführlicher beschrieben
Ich arbeite wegen der guten Erfahrungen im Unterricht ausschließlich mit wxPython, warum wird hoffentlich deutlich werden.

Windows

- Herunterladen von http://wxpython.org/
- Passende Version zur eigenen Python-Version verwenden [Python 3 geht nicht!]
- Installation möglichst in den Ordner
- <Python>/Lib/sitepackages
- **Unbedingt** auch die *"wxPython docs, demos und tools"* herunterladen und sinnvollerweise auch in das o.a. Verzeichnis installieren !

Linux-Distributionen installieren beide in der Regel selbst in die richtigen Verzeichnisse[1].

Demo-Code nutzen

Der besondere "Mehrwert" steckt in den Demos. Sie ermöglichen es, dass Schülerinnen und Schüler nach nur wenigen Hilfen sich schon in der Mittelstufe mit copy and paste die für ihre Anwendung nötigen Teile aus den jeweiligen Beispielen entnehmen und dann zu ihrer Anwendung passend bearbeiten.
Eine der notwendigen Hilfen ist das Bereitstellen einer Anwendung, also einer Klasse, die von der Application-Klasse App erbt und das Bereitstellen der wenigen Programmzeilen, die zu ihrem Start notwendig sind. Diese sind natürlich nicht in den Demos enthalten, da sie Teile der Demo selbst sind.
Dabei lernen die Schülerinnen und Schüler, dass zunächst ein Frame-Objekt für das Fenster selbst benötigt wird und ein Panel-Objekt für die Inhaltsfläche des Fensters. Es ist belanglos, ob die Definition der Inhalte des Panel-Objektes innerhalb der definierten Frame-Klasse erfolgt oder ob dafür eine eigene Klasse geschrieben wird.
Der Code der Application-Klasse könnte in der einfachsten Variante so aussehen:

```
class MyApp(wx.App):
      def OnInit(self):
            fenster = MyFrame(None, -1, "Test")
            fenster.Show(True)
            return True
```

Und ihr Aufruf [geschützt für den Fall, dass dies nicht das Hauptprogramm ist; günstig für Tests!]:

```
if __name__ == '__main__':
      app = MyApp(redirect=False)
      app.MainLoop()
```

Die Definition der Frame-Klasse, die hier den Namen MyFrame trägt, wäre dann noch zu realisieren: Copy and paste!

1 Dafür muss man unter linux ggf. IDLE nachinstallieren.