Informatik für Schüler, Foliensatz 21
Objektorientierte Programmierung

Prof. G. Kemnitz

Institut für Informatik, Technische Universität Clausthal

23. April 2009

In der Informatik werden Sachverhalte durch Daten beschrieben, mit denen etwas getan wird.
Ein typischer Sacherverhalt ist ein Text.
Was kann man mit einem Text tun (elementare Aktionen)?

- leeres Textobjekt erzeugen
- Text anhängen
- Teilzeichenkette löschen
- Text ausgeben
- ...

Der Mensch denkt in solchen elementaren Aktionen. Warum nicht auch so programmieren?

Objektorientierte Programmierung (OOP)

Objekt	Daten zur Beschreibung eines Sachverhalts
Attribut	einzelnes Beschreibungselement
Methode	Bearbeitungsfunktion für ein Objekt
Klasse	Vereinbarung der Attribute und Methoden für eine bestimmte Art von Objekten.

Entwurfstechnik:

- Definion der Datenstrukturen für den Sachverhalt
- Programmieren der Aktionen (Methoden), die für die Daten gebraucht werden
- Zusammensetzen des Zielalgorithmus aus diesen Aktionen

oder

- suche geeigneter (bereits programmierter) Klassen

Technik, um Programme übersichtlich zu strukturieren

Vorteile:

- Mehrfachnutzung: Klassen werden gern so geschrieben, dass sie für viele Sachverhalte geeignet sind
- Nachnutzung: für Python gibt es für jede nicht zu spezielle Aufgabe Module mit geeigneten Klassendefinitionen
- Abstraktion: für die Nachnutzung einer Klasse genügt die Kenntnis, der Methoden (welche gibt es, wie werden sie aufgerufen etc.). Die interne Datendarstellung und die Implementierung (Programmierung) braucht der Anwendungsprogrammierer nicht zu kennen

Klassen, Methoden und Objekte in Python

- Vereinbarung einer (einfachen) Klasse:

```
class Klassenname():
 {Attribut = Wert}
 {def Methode(self[1] {,arg}):
    Anweisung
    {Anweisung}}
```

- Vereinbarung eines Objekts

```
Objektname = Klassenname()
```

- Aufruf einer Methode

```
Objektname.Methode(Argumente_außer_self)
```

- Lesen und Schreiben von Attributen

```
a = Objektname.Attribut
Objektname.Attribut = Wert
```

[1] das Objekt selbst, steht beim Aufruf vor dem Punkt

Aufgabenstellung für die Entwicklung einer Klasse

Wir hätten gern Textobjekte mit den Methoden:

- erzeugen eines leeren Textobjekts
- Zeichenkette anhängen »append(self, txt)«
- Zeichenkette einfügen »insert(self, txt, pos)«
- Textausschnitt löschen »delete(self, von, bis)«
- letzten Bearbeitungsschritt rückgängig machen »undo(self)«
- Text anzeigen »show()«

Was für Attribute müssten solche Textobjekte haben?

- eine Zeichenkette »s« für den Text
- eine Liste »L« für die Informationen, um schrittweise den Vorzustand wieder herzustellen

Entwurf ohne »undo«-Funktion

```
class text():
# Attribute
 L = []        # undo-Liste
 s=''          # zu Beginn leere Zeichenkette
 def append(self, txt):       #  Anhaengen
  self.s += txt
 def insert(self, txt, pos): #  Einfuegen
  self.s = self.s[:pos] + txt + self.s[pos:]
 def delete(self, von, bis): #  Loeschen
  self.s = self.s[:von] + self.s[bis+1:]
 def show(self):              #  Anzeigen
  print self.s
```

- Klassendefinition im Arbeitsverzeichnis in einer Datei »Textklasse.py« speichern

Wir arbeitet man mit einer Klasse?

- in derselben Datei definieren und dann benutzen oder
- importieren und benutzen[2] (besser, da übersichtlicher)
- Python im Arbeitsverzeichnis starten; interaktiv Testen:

```
from Textklasse import text
a = text()[3]
a.append('0123456789')
a.show()
a.insert( 'aaa', 8)
a.show()
a.delete(2, 4)
a.show()
```

[2] gilt auch für Funktionen

[3] alternativ Import mit »import Textklassen« und Referenzierung mit »Textklasse.text«

Ergänzung der Undo-Methode

- einfachste Lösung: die Methoden »append«, »insert« und »delete« hängen, bevor sie »self.s« ändern, den alten Wert an die Liste »L« an
- die Funktion »undo« entnimmt, wenn die Liste nicht leer ist, den letzten Listenwert, schreibt ihn in »self.s« und löscht ihn aus der Liste

Aufgabe 21.1: Undo-Methode

- Erweitern Sie die Klassendefinition um die Funktion:

```
def undo(self):
    ...
```

- Testen Sie die Methoden wie oben im Dialogbetrieb.
- Schreiben Sie ein Testprogramm (extra Datei), das ein Textobjekt erzeugt, mehrfach Text anhängt, einfügt, löscht und dazwischen auch Operationen rückgängig macht (mindestens zehn Methodenaufrufe). Zwischen jedem Bearbeitungsschritt ist mit »print« der Methodenaufruf als Text und mit »show« der Wert des Textobjekts auszugeben.

- Beispieltestschritt:

```
x=text()
x.append('Viel Text')
print "x.append('Viel Text')"[4]
x.show()
```

[4] Ein Text in doppelten Anführungsstrichen darf einzelne Anführungsstriche enthalten und umgekehrt

Aufgabe 21.2: Verschlüsselungsmethode

- Schreiben Sie eine Methode, die für alle Zeichen mit einem Wert im Bereich von 48 bis 254 das um n Stellen zirkular verschobenen Zeichen zurückgibt. Für alle anderen Zeichen soll das Zeichen ohne Veränderung zurückgegeben werden:

  ```
  Caesar(self, n)
  ```

  (Aufgabe 9.2, nur jetzt als Methode der Klasse »text«).
- Testen Sie wie in der Voraufgabe erst interaktiv und abschließend mit einem Testprogramm (extra Programmdatei) die neue Methode mit Beispielaufrufen.

Aufgabe 21.3: Effizienzverbesserung

Die Aufbewahrung der kompletten Zeichenkettenfür jeden Bearbeitungsschritt kostet für größere Textobjekte und lange Verarbeitungsfolgen viel Speicher. Effizienter wäre es, für die Undo-Methode nur folgende Daten aufzubewahren:

- nach »append« und »insert«, den wieder zu löschen Bereich in einem Tupel ('L', a, e)
- nach »delete« die wieder einzufügenden Zeichenkette und die Einfügeposition in einem Tupel ('I', txt, pos).
- nach »Caesar« den Schlüssel in einem Tupel ('C', n)

('L', 'I', 'C' – Symbole für Undo-Aktionen, a, e – Bereich; pos – Position, n – Schlüssel).

Aufgabenstellung:

- Ändern Sie die Programmierung der Methoden entsprechend. Die Aufrufe und Funktionen der Methoden soll gleich bleiben.
- Test mit den Testprogrammen aus den Aufgaben zuvor.

Hinweis: Für Methodenaufrufe innerhalb von Methoden ist der Objektname »self«, z.B.:

```
def undo(self):
  ... # Undo-Datensatz in Tupel »t« übernehmen
 if t(0)=='C':
  self.Ceasar(self.s, t(1))
```