1 Überblick der Fachbegriffe (Terminologien) von OOP[1]

Klasse: Ein benutzerdefinierter Prototyp (Vorlage) für ein *Objekt*, dass eine Menge an Beschreibungen (Eigenschaften) definiert, die ein Objekt der Klasse charakterisieren. Diese Beschreibungen (Eigenschaften) sind *Attribute* (Klassenvariablen und Instanzvariablen) und Methoden. Beiden werden über die sogenannte Punkt-Notation angesprochen (aufgerufen).

Objekt: Eine einzigartige Instanz einer Datenstruktur, die durch seine Klasse definiert ist. Ein Objekt umfasst sowohl Klassenvariablen und Instanzvariablen als auch Methoden. In Python existiert ein Objekt erst, wenn es erzeugt wurde und nimmt dann Speicherplatz ein.

Konstruktor: Ein Definitionsbereich innerhalb der Klasse, der festlegt, wie ein Objekt bei der ersten Instanziierung (vgl. unten), welche Werte das Objekt beim Start bekommen soll. Der Konstruktor kann im übertragenen Sinne auch als Lebenselixier des Objekts begriffen werden, denn er beschreibt im übertragenen Sinne den Geburtsvorgang eines Objektes. In Python wird damit konkret Speicherplatz für ein Objekt erzeugt (bzw. frei gehalten) und mit Werten belegt. In Python wird der Konstruktor durch das Schlüsselwort *___init___* eingeleitet

Attribut: Eine Klassen- oder Instanzvariable, die Daten zugehörig zu einer Klasse oder deren Objekte enthält.

Instanzvariable: Eine Variable, die innerhalb des Konstruktors oder innerhalb einer Methode definiert ist und nur zur aktuellen Instanz einer Klasse gehört. In Python beginnen Instanzvariablen immer mit *self* als Hinweis auf den Bezug zum aktuellen Objekt, z:b. *self.geschwindigkeit = 0* setzt die Instanzvariable Geschwindigkeit auf den Wert 0.

Klassenvariable: Eine Variable, die von allen Instanzen (Objekte) einer Klasse gemeinsam genutzt wird. Klassenvariablen werden innerhalb einer Klasse definiert, aber ausserhalb von Methoden der Klasse. Klassenvariablen werden nicht so häufig gebraucht/genutzt wie Instanzvariablen.

Methode(Operation): Eine besondere Art von Funktion die in der Klassendefinition festgelegt wird. Methoden beschreiben meist das Verhalten von Objekten zur Programmlaufzeit. Damit können Objekte in ihrem Zustand verändert werden. Das geschieht durch Verändern der Attribute.

Instanziierung: Bezeichnet den Vorgang der Schaffung einer Instanz einer Klasse. Vergleichbar mit dem Initialisieren von Variablen. In Python ist z.B. *mein_kreis = Kreis()* die Instanziierung eines neues Kreisobjektes der Klasse *Kreis*.

Instanz: Ein individuelles Objekt einer bestimmten Klasse. Beispielsweise ist in Python ein durch
obj = Kreis() instanziiertes Objekt *obj*, das zu einer Klasse *Kreis* gehört, eine *Instanz* der Klasse *Kreis* (vgl. oben).

Vererbung: Die Übertragung von Charakteristika einer Klasse an eine andere Klasse, die davon abgeleitet wird (erbt).

[1] nach: http://www.tutorialspoint.com/python/python_classes_objects.htm

2 Grundkonzepte der OOP in Python[2]

2.1 Klassenerstellung (in Python 2)

Das Schlüsselwort *class* beschreibt den Anfang einer neuen Klassendefinition, gefolgt von dem Namen der Klasse und einem Doppelpunkt (vgl. Listing 1).

Konvention: Klassennamen beginnen mit einem Großbuchstaben, Attribute und Methoden mit einem Kleinbuchstaben. Im weiteren Verlauf der Namensgebung wird der sog. CamelCase angewendet:

```
class Angestellter:
    'Optionaler Klassen-Dokumentations-Text, z.B. Gemeinsame Basis-Klasse
       für alle Angestellten'
    angestAnzahl = 0
    def __init__(self, name, gehalt):
        self.name = name
        self.gehalt = gehalt
        Angestellter.angestAnzahl += 1
    def zeigeAnzahl(self):
        print "Gesamt Angestellte %d" % Angestellter.angestAnzahl
    def zeigeAngestellter(self):
        print "Name : ", self.name,  ", Gehalt: ", self.gehalt
```

Listing 1: Eine einfache Klassendefinition

- Der 'Optionale Klassen-Dokumentations-Text' kann später mit *Klassenname.__doc__* benutzt werden.
- Die Variable *angestAnzahl* ist eine *Klassenvariable,* deren Wert in allen Instanzen der Klasse geteilt wird. Darauf kann zugegriffen werden mit *Angestellter.angestAnzahl* von innerhalb oder ausserhalb der Klasse.
- Die erste Methode *__init__()* ist eine spezielle Methode, die auch Klassen-Konstruktor oder Initialisierungsmethode genannt wird. Diese wird aufgerufen, wenn Python eine neue Instanz dieser Klasse erstellt.
- Die anderen Methoden werden wie normale Funktionen deklariert mit der Ausnahme, dass das erste Argument (Parameter) der Methode das Schlüsselwort *self* ist. Es wird beim Aufruf der Methode *nicht* verwendet, da Python es selbständig in der Liste der Parameter ergänzt)

Erschaffung von Instanzen einer Klasse

Um eine Instanz einer Klasse zu erzeugen, ruft man die Klasse mit Hilfe des Klassennamens auf und übergibt die notwendigen Argumente für den Konstruktor, die in der *__init__* Methode definiert wurden:

[2] ebd.

```
# Hier wird ein erstes Objekt der Angestellter-Klasse erschaffen
# die Klassenvariablen (Attribute) name und gehalt werden
# mit den übergebenen Werten des Konstruktors belegt.
angest1 = Angestellter("Zara", 2000)
# Hier wird ein zweites Objekt der Angestellter-Klasse erschaffen
angest2 = Angestellter("Manni", 5000)
# Fortsetzung im nächsten Listing
```

Zugriff auf Attribute

Die Attribute und Methoden eines Objektes werden mit Hilfe des Punkt-Operators (dot operator) angesprochen. Klassenvariablen benutzen den Klassennamen statt den Objektnamen:

```
# Fortsetzung des obigen Listings
angest1.zeigeAngestellter()      # beachte: kein self in der Klammer beim Aufruf!
angest2.zeigeAngestellter()      # aber in der Definition!
print "Gesamt Angestellte %d" % Angestellter.angestAnzahl
```

Nun alles zusammen nochmal mit Klassendefinition und Objekterzeugung:

```
class Angestellter:
    'Optionaler Klassen-Dokumentations-Text, z.B. Gemeinsame Basis-Klasse für alle Angestellten'
    angestAnzahl = 0
    def __init__(self, name, gehalt):
        self.name = name
        self.gehalt = gehalt
        Angestellter.angestAnzahl += 1
    def zeigeAnzahl(self):
      print "Gesamt Angestellte %d" % Angestellter.angestAnzahl
    def zeigeAngestellter(self):
        print "Name : ", self.name,  ", Gehalt: ", self.gehalt

# ENDE der Klassendefinition

# ANFANG Hauptprogramm (main)

# Hier wird ein erstes Objekt der Angestellter-Klasse erschaffen
angest1 = Angestellter("Zara", 2000)
# Hier wird ein zweites Objekt der Angestellter-Klasse erschaffen
angest2 = Angestellter("Manni", 5000)
angest1.zeigeAngestellter()     # beachte: kein self in der Klammer beim Aufruf!
angest2.zeigeAngestellter()
print "Gesamt Angestellte %d" % Angestellter.angestAnzahl
```

liefert folgende Ausgabe:

```
Name : Zara , Gehalt: 2000
Name : Manni , Gehalt: 5000
Gesamt Angestellte 2
```

Die Attribute können im Hauptprogramm oder auch in der Konsole direkt angesprochen und geändert werden, wenn sie, wie bei Python üblich, alle *public* (öffentlich zugänglich) sind (vgl. Geheimnisprinzip, Datenkapselung):

```
angest1.gehalt = 1500
angest1.gehalt = 1700
```

Es ist allerdings grundsätzlich so, dass dafür in einer Klasse sogenannte *getter* und *setter*- Methoden (nach get - holen und set - setzen) definiert werden sollten. Über diese lassen sich dann definiert die Attribute des Objektes setzen oder zurückgeben. Das hat insbesondere den Vorteil, dass an dieser zentralen Stelle eine Prüflogik für unzulässige oder inkonsistente Werte implementiert werden kann. Hier ein Beispiel für die Erweiterung der o.a. Klasse, um das Attribut *gehalt* zu setzen (bzw. zu ändern):

```
def setGehalt(self,gehalt):      # beachte die Angabe von self als einen Parameter!
    self.gehalt = gehalt
```

Aufgaben (handschriftlich auf einem neuen Blatt oder unten)

1. Ergänzen Sie die o.a. Klassendefinition um mindestens zwei weitere sinnvolle Attribute und zwei weitere sinnvolle Methoden.
2. Falls ein neuer Konstruktor notwendig ist, so schreiben Sie diesen auf.
3. Implementieren Sie Ihre Lösungen.

3 Klassendefinitionen in eigener Datei und deren Verwendung

Normalerweise werden die Klassendefinitionen jeweils in eine eigene Datei geschrieben und dann in das Hauptprogramm (Main) importiert.

Angestellter.py

```
1  class Angestellter:
2     'Optionaler Klassen-Dokumentations-Text, z.B. Gemeinsame Basis-Klasse für alle Angestellten'
3     angestAnzahl = 0
4     def __init__(self, name, gehalt):
5        self.name = name
6        self.gehalt = gehalt
7        Angestellter.angestAnzahl += 1
8     def zeigeAnzahl(self):
9       print "Gesamt Angestellte %d" % Angestellter.angestAnzahl
10    def zeigeAngestellter(self):
11       print "Name : ", self.name,  ", Gehalt: ", self.gehalt
12    def setGehalt(self,gehalt):  # beachte die Angabe von self in der Parameterliste!
13       self.gehalt = gehalt
14
15 # ENDE der Klassendefinition
16
```

Main.py

```
1  from Angestellter import *
2  # ANFANG Hauptprogramm (main)
3
4  # Hier wird ein erstes Objekt der Angestellter-Klasse erschaffen
5  angest1 = Angestellter("Zara", 2000)
6  # Hier wird ein zweites Objekt der Angestellter-Klasse erschaffen
7  angest2 = Angestellter("Manni", 5000)
8
9  angest1.zeigeAngestellter()    # beachte: kein self in der Klammer beim Aufruf!
10 angest2.zeigeAngestellter()
11
12 angest2.setGehalt(4430)
13 angest2.zeigeAngestellter()
14 print "Gesamt Angestellte %d" % Angestellter.angestAnzahl
```

Für den Import gibt es grundsätzlich zwei Möglichkeiten:

> from Klassenname import ∗

(s.o) Dann wird bei der Erzeugung des Objektes nur der Klassenname angegeben (vgl. Listing Main.py, Zeilen 5, 7 und für die Klassenvariable Zeile 14)

Die andere Möglichkeit:

> import Klassenname

führt dazu, dass bei der Verwendung des Konstruktors sowie der Klassen- und Instanzvariablen der Klassenname explizit mit angegeben werden muss (vgl. Listing Main2.py, Zeilen 5, 7, 14):

Main2.py

```
1  import Angestellter
2  # ANFANG Hauptprogramm (main)
3
4  # Hier wird ein erstes Objekt der Angestellter-Klasse erschaffen
5  angest1 = Angestellter.Angestellter("Zara", 2000)
6  # Hier wird ein zweites Objekt der Angestellter-Klasse erschaffen
7  angest2 = Angestellter.Angestellter("Manni", 5000)
8
9  angest1.zeigeAngestellter()    # beachte: kein self in der Klammer beim Aufruf!
10 angest2.zeigeAngestellter()
11
12 angest2.setGehalt(4430)
13 angest2.zeigeAngestellter()
14 print "Gesamt Angestellte %d" % Angestellter.Angestellter.angestAnzahl
15
```

Die bevorzugte Methode für die OOP sollte somit die erste Möglichkeit sein, da der Klassenname nicht mehr explizit wiederholt werden muss. Es sind allerdings Situationen denkbar, in denen die explizite Angabe des Klassennamens sinnvoll sein kann. Beispielsweise:

```
import math
x = math.pi / 2
y = math.sin(x)
print y
print math.pi
```

Dann ist der Bezug zur Klasse explizit im Programmtext erkennbar. Ebenfalls die Verwendung der Konstanten der Klassen (hier pi).

Der Vorteil, jede Klasse in einer eigenen Datei zu definieren, besteht drin, dass arbeitsteilig an einem Projekt gearbeitet werden kann, das nach OOP-Prinzipien implementiert werden soll. Nach Festlegung eines Klassendiagramms und der Definition der Attribute und Methoden kann jeder Mitarbeiter am Projekt seine eigenen Klassen bearbeiten. Dabei hilft ein Mechanismus von Python bei der Implementierung:

Steht beispielsweise noch nicht fest wie Attribute oder Methoden implementiert werden, so kann mit dem Schlüsselwort *pass* (ausweichen, lasse aus) in Python dieser Abschnitt gekennzeichnet werden. Dann kann trotzdem die Klasse implementiert werden, ohne dass jetzt schon eine genaue Implementierung festgelegt wird bzw. ohne dass eine Funktionalität vorhanden ist. Somit lässt sich trotzdem das Projekt komplett durchlaufen, nur das an den entsprechenden Stellen die Funktionalität fehlt.

Vertiefung

- Ein ergänzendes Tutorial zur Einführung in Klassen in Python 2 (!) findet sich u.a. auf http://www.python-kurs.eu/klassen.php
- Ergänzung zur Modularisierung (import): http://www.python-kurs.eu/modularisierung.php