Python in der Schule

Kein Pythonkurs

Es geht in diesem Text nicht um einen Programmierkurs mit Python. Statt dessen soll an einigen ausgewählten Themenbereichen gezeigt werden, dass Python als Programmiersprache zu ihrer Behandlung geeignet ist und dabei ggf. auf einige Besonderheiten hingewiesen werden.

Gui mit wxPython

Dabei soll auch die Möglichkeit der Gui-Programmierung gezeigt werden. Dafür gäbe es mehrere Möglichkeiten, hier wird aber nur das Paket wxPython untersucht, da es mit einer hervorragenden Dokumentation[1] und einem ebenso hervorragend dokumentierten Demopaket ergänzt zur Verfügung steht.
wxPython ist bei den sitepackages zu installieren, die „docs and demos“ müssen gesondert herunter geladen und installiert werden.

Versionen

Dieses Paket ist noch nicht in einer Version für Python 3 verfügbar, so dass hier mit der immer noch aktuellen Version Python 2.7 gearbeitet wird. Unterschiede betreffen nach Aussagen der Python-Entwickler auf der für uns interessanten Ebene vor allem die Schleifenkopfdefinition mit range, die in den 2-er-Versionen eine Liste der Iterationswerte erzeugt, bei der 3-er-Version aber – und das ist sicher sinnvoller – einen Iterator.

Editor für Python selbst

Arbeiten Sie sinnvollerweise mit IDLE-Python - Shell. Daraus können Sie ein Editorfenster öffnen, dessen Inhalt Sie über den Menüpunkt ***Run*** oder die Kurztaste ***F5*** in der Auswertungsumgebung auswerten lassen können.
Um im System keine laufenden Anwendungen von Python zu hinterlassen, sollten Sie die Shell mit exit() verlassen.

Was bietet Python, was … nicht hat?

Natürlich stellt sich bei jedem Wechsel auf eine neue Programmierumgebung immer die Frage, welche Möglichkeiten man damit gewinnen kann.
Ein Vorteil gegenüber Java ist, dass auf den Hilfeseiten auch die sprachlichen Grundelemente ausführlich beschrieben werden. Der wirklich hervorragenden Klassenbibliothek von Java fehlt das leider.
Für Unterrichtszwecke wichtiger aber ist die Möglichkeit mit Shell und Editorfenster arbeiten zu können, ein deutlicher Vorteil gegenüber der Arbeit mit Java. Man kann hier eben *"einfach schnell einmal"* etwas ausprobieren.
Beim Vergleich mit DrScheme[2], das diese Möglichkeit ebenso bereitstellt, gibt es zwei Grundaspekte, die im Unterricht wichtig sind:

- Mit Python kann man funktional arbeiten, es stellt sogar das aus Scheme bekannte lambda-Konzept, Funktionen als Parameter, map usw. zur Verfügung. Aber: ist die Sprache wirklich funktional, wenn sie Endrekursion nicht erkennt? Ärgerlich!
- Python arbeitet mit infix-Notation, also heißt es ganz gewohnt 3 * 4.

1 Die Dokumentation finde ich besser als die von Python selbst.
2 → PLT Scheme is now Racket: http://racket-lang.org/ ACHTUNG: nicht ~~racket.org~~ !!!

Objektorientierung

Eine der besonderen Eigenschaften von Python findet man bei der Objektorientierung: Python bietet die Möglichkeit der Mehrfachvererbung. Wenn man diese nutzen will, lese man vorher in der Anleitung o.ä. nach, welche Regeln dabei zu beachten sind. Bekanntermaßen treten besondere Probleme bei Namen von Attributen und Methoden auf.

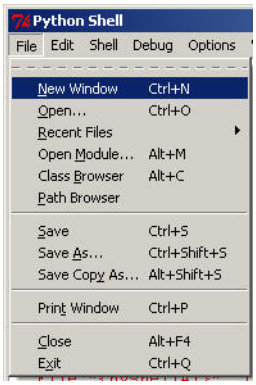
OO – dazu ein Beispiel

Betrachten wir das Konzept an Hand eines einfachen Beispiels, nämlich eines Zählers. Ein einfacher Zähler ist zunächst ein Objekt, das einen Stand ***hat*** → Attribut und zählen und den Stand anzeigen ***kann*** → Methoden. Eine Definition[1] der Klasse könnte lauten:

```
class EinfacherZaehler:
    # Konstruktor:
    def __init__(self):
        # Attribut:
        self.stand=0

    # Methoden:
    # der muss zaehlen können
    def zaehle(self):
        self.stand+=1

    # und den stand anzeigen können
    def zeigeStand(self):
        return self.stand
```



Nun muss ein Zählerobjekt erzeugt werden:

```
zaehler=EinfacherZaehler()
```

Botschaften (messages) schickt man dem Objekt, indem man die auch aus anderen Sprachen bekannte Schreibweise mit dem verbindenden Punkt benutzt.

```
zaehler.zeigeStand()
zaehler.zaehle()
zaehler.zeigeStand()
zaehler.zaehle()
zaehler.zeigeStand()
```

ergibt dann die Ausgaben

```
0
1
2
```

Leider lässt das System auch den direkten Zugriff auf die Attribute zu und widerspricht damit einem der wichtigsten Konzepte der OO, der Kapselung von Daten. Daher geht:

```
zaehler.stand
zaehler.stand=5
zaehler.stand
```

was zu den Ausgaben

```
2
5
```

führt.

1 Nutzen Sie IDLE Python und lassen Sie sich aus der Shell ein Editor – Fenster geben

Wirksame Kapselung

Eine wirksame Kapselung von Attributen ist dennoch möglich, wenn diese im Namen durch zwei Unterstreichungsstriche am Beginn als private gekennzeichnet werden. Mit:

```
class EinfacherZaehler:
    # Konstruktor:
    def __init__(self):
        # Attribut:
        self.__stand=0

    # Methoden:
    # der muss zaehlen können
    def zaehle(self):
        self.__stand=self.__stand+1

    # und den stand anzeigen können
    def zeigeStand(self):
        return self.__stand

zaehler=EinfacherZaehler()
print( zaehler.zeigeStand() )
zaehler.zaehle()
print( zaehler.zeigeStand() )
zaehler.zaehle()
print( zaehler.zeigeStand() )
zaehler.zaehle()
print( zaehler.zeigeStand() )
# So geht Kapselung?? --> Keine Fehlermeldung, geht aber nicht.
zaehler.__stand=5
print( zaehler.zeigeStand() )
zaehler.zaehle()
print( zaehler.zeigeStand() )
```

… erhalten wir das gewünschte Verhalten mit den Ausgaben:

```
0
1
2
3
3
4
```

Vererbung

Wir wollen nun von diesem einfachen Zähler erben und mit seiner Hilfe einen zyklischen Zähler konstruieren, also einen Zähler, der zurückgesetzt wird, wenn er seinen Maximalwert erreicht hat.

```
# Vererbung:
class ZyklischerZaehler(EinfacherZaehler):
    def __init__(self, maximalWert):
        EinfacherZaehler.__init__(self)
        self.maximalWert=maximalWert
    # zaehlen muss den Maximalwert beachten
    def zaehle(self):
        self.stand+=1
        if self.stand==self.maximalWert:
            self.stand=0
    #  Die Methode zum Anzeigen von stand erbt er.
```

Anzumerken ist, dass der Konstruktor von EinfacherZaehler explizit aufgerufen werden muss.

Die messages

```
zyklischerZaehler=ZyklischerZaehler(3)
print( zyklischerZaehler.zeigeStand() )
zyklischerZaehler.zaehle()
print( zyklischerZaehler.zeigeStand() )
zyklischerZaehler.zaehle()
print( zyklischerZaehler.zeigeStand() )
zyklischerZaehler.zaehle()
print( zyklischerZaehler.zeigeStand() )
```

ergeben dann die Ausgaben

```
0
1
2
0
```

Die direkten Zugriffe auf das Attribut stand ergeben

```
zaehler.stand
zaehler.stand=5
zaehler.stand
0
5
```

wovon die letzten Ausgabe nur um so ärgerlicher ist, da der Zaehler einen undefinierten Zustand angenommen hat.

Also auch hier gilt – als Aufgabe formuliert: Schreiben Sie die Klassendefinition mit privaten Attributen um.

Anmerkungen zu Grundkonzepten der Sprache

Einige Hinweise zu dem o.a. Programmtext sind notwendig.

- Einrückungen sind bei Python zwingendes Strukturierungsmittel. Was z.B. bei Java mit den geschweiften Klammern realisiert wird, erfüllen bei Python die Einrückungen. Sie definieren Programmblöcke. Gleiche Einrückungstiefen bedeuten gleiche Programmblockebenen.
- Neue Strukturierungsebenen werden mit Doppelpunkt eingeleitet.
  Nach dem Kopf einer Klassendefinition kommt also ein Doppelpunkt, ebenso nach dem Kopf einer Funktionsdefinition, nach dem Kopf einer Schleifendefinition usw.
  Der Editor rückt dann die nächste Zeile automatische ein.
- self kennzeichnet das aktuelle Objekt, entspricht also dem this bei Java.
  Alle Methoden müssen self als ersten Parameter angeben, beim Aufruf fehlt der Parameter allerdings.
- Alle Klassen bringen eine umfangreiche Sammlung von Standardmethoden (Funktionen) mit, deren Namen alle mit zwei Unterstreichungstrichen beginnen und aufhören.
  Deren wichtigste ist der Konstruktor mit dem Namen __init__ , eine sehr sinnvolle Namensgebung, da er ja die Aufgabe der Initialisierung der Instanzen (Objekte) hat.

Vererbungsbeziehung oder Nutzerbeziehung?

Man kann dieses einfache Beispiel auch ohne Vererbung lösen, nämlich dadurch, dass sich das Zaehlerobjekt sich ein Objekt vom Typ EinfacherZaehler besorgt, seine Fähigkeiten nutzt, aber die eigentlich Funktionalität selbst bereitstellt.
[Die Klasse EinfacherZaehler bleibt wie oben.]

```
# Nutzerbeziehung:
class Zaehler():
    def __init__(self, maximalWert):
        self.einfacherZaehler=EinfacherZaehler()
        self.maximalWert=maximalWert

    # zaehlen muss den Maximalwert beachten;
    # wird er überschritten, holt sich zaehler
    # einen neuen einfacherZaehler
    def zaehle(self):
        self.einfacherZaehler.zaehle()
        if self.einfacherZaehler.zeigeStand()==self.maximalWert:
            self.einfacherZaehler=EinfacherZaehler()

    # zahler muss hier den stand selbst anzeigen können,
    # nimmt aber einfacherZaehler zu Hilfe
    def zeigeStand(self):
        return self.einfacherZaehler.zeigeStand()
```

Vorteilhaft an dieser Variante ist, dass die Variable Stand nun gekapselt ist, was allerdings für das Attribut einfacherZaehler so nicht gilt.

Relativ problemlos lässt sich der zyklische Zaehler zum Erstellen einer Uhr nutzen: Sie benötigt einen solchen zyklischen Zähler mit der Zyklenlänge 12 [oder 24] und zwei mit der Zyklenlänge 60.

Der Modellierungsaspekt ist dabei, dass eine solche Uhr jeweils ZyklischerZaehler – Objekte benutzt. Die Uhr hat dabei Controller-Funktionen. Erzeugt man eine grafische Oberfläche dazu, kann man die Anwendung des Modell-View-Controller – Konzeptes vervollständigen.

Stringbehandlung bei Kryptologie mit Python

Python bietet besondere Fähigkeiten bei der Behandlung von strings. Ein möglicher Kontext zu strings wäre die einfache Verschlüsselung von Texten mit dem Caesar zugeschriebenen Verfahren oder mit skytale.

Insbesondere in der Mittelstufe beschränkt man sich bei den Anfängen der Behandlung der Kryptologie mit Python sicherlich auf die beiden o.a. Verfahren und dabei kann man die besonderen Möglichkeiten von Python beim Umgang mit strings[1], allgemeiner auch mit Listenstrukturen nutzen – wenn man denn will.

Wenn man denn will?

Beim Einsatz besonderer "Tricks" zur Lösung stellt sich immer die Frage, welche Bedeutung die Kenntnis einer solchen Möglichkeit hat. Will man wirklich diese Tricks vermitteln oder nicht besser allgemeine Strategien zur Lösung von Problemen? Beispielhaft wird das am Beispiel skytale untersucht (siehe weiter unten).

1 Etwas unübersichtlich ist, dass einige Funktionen von strings in str – einem der vordefinierten *Sequence - Types* – zu finden sind, andere aber im Modul *string*, das extra importiert werden muss.

Caesar

Die kürzeste Variante bietet die Funktion (Methode) translate() [1], die bei string-Objekten die buchstabenweise Ersetzung ermöglicht. Etwas aufwändiger, aber immer noch sehr einfach gelöst, ist die Version mit einem Übersetzungsalphabet. Nach dem Import des Paketes **`string`** ...

```
import string
```

… kann man mit ...

```
Gross=string.ascii_uppercase
```

… eine Variable für alle Großbuchstaben bekommen. Das Erzeugen der Übersetzungstabelle (als String) erledigt die folgende Funktion, die mit "slices" arbeitet, also die Möglichkeit nutzt, Daten bei sequenziellen Datentypen auszuschneiden.

```
def UebersetzungsTabelle(position):
    if position not in range(26): position=0
    return Gross[position:26]+Gross[0:position]
```

Ein Beispieltext:

```
Text="Alle meine Entchen schwimmen auf dem See, Koepfchen in das Wasser,
Schwaenzchen in die Hoeh."
```

Übersetzungsfunktion

```
def chiffriere(text, buchstabe):
    buchstabe=buchstabe[0].upper() # nur den ersten Buchstaben akzeptieren
    text=text.upper()
    tabelle=UebersetzungsTabelle(Gross.find(buchstabe))
    ergebnis=""
    for i in range(len(text)):
        if Gross.find(text[i])!=-1:                  # kein Großbuchstabe
            ergebnis+=tabelle[Gross.find(text[i])]
    return ergebnis
```

Weitere Anmerkungen zu Grundkonzepten der Sprache

Zunächst einmal zu Datentypen:

1. Python kennt keinen Datentyp Zeichen.
   Ein einzelnes Zeichen ist einfach ein string mit der Länge 1.
   Will man auf dieses Zeichen zugreifen, z.B. weil man zu einem "u" den entsprechenden ASCII-Code bestimmen will, kann dies also auch so übergeben.
   Aus einem string heraus – beispielsweise nach wort = "Schluessel", erhält man den Buchstaben über indizierten Zugriff: wort[i].
2. Mit **`in`** kann in Schleifen aus jedem Sammlungstyp der Wert der Schleifenvariablen bezogen werden.
   Das o.a. Programm könnte also folgendermaßen modifiziert werden:

```
for buchstabe in text:
    if Gross.find(buchstabe)!=-1:
        ergebnis+=tabelle[Gross.find(buchstabe)]
```

1 *Aus der Hilfe:* ***string*.translate(s, table[, deletechars])**
Delete all characters from s that are in deletechars (if present), and then translate the characters using table, which must be a 256-character string giving the translation for each character value, indexed by its ordinal. If table is None, then only the character deletion step is performed.

while

for ist nicht die einzige Möglichkeit zur Steuerung von Schleifen. Ein einfaches Beispiel mit while zur Untersuchung einer Zahl auf die Primzahleigenschaft

```
def istPrim(test, primzahlen = [2,3,5,7,11,13]):
    if test in primzahlen:
        return True
    while True:                                # den Abbruch erledigt das return
        for teiler in primzahlen:
            if (test % teiler) ==0:
                # print( "Teiler ist : " + str(teiler) )
                return False
        if test<primzahlen[len(primzahlen)-1]*primzahlen[len(primzahlen)-1]:
            return True
        else:
            primzahlen+=[naechstePrimzahl(primzahlen)]

def naechstePrimzahl(primzahlen):
    neu = primzahlen[len(primzahlen)-1] + 2
    while not istPrim(neu, primzahlen):
        neu+=2
    return neu
```

skytale

Die folgende Lösung für die Verschlüsselung eines Textes mit skytale nutzt so stark die Möglichkeit von slices[1], dass man das Algorithmische nicht mehr erkennt.

```
def chiffriere(text, step):
    ergebnis=''
    for i in range(step):
        ergebnis+=text[i:len(text):step]
    return ergebnis
```

Die Entschlüsselung erfolgt durch:

```
def dechiffriere(text,step):
    return chiffriere(text,len(text)/step)
```

Der Aufruf

```
chiffriere('1234567890abcdefghijklmnopqrstuvwxyz',2)
```

liefert:

```
            13579acegikmoqsuwy24680bdfhjlnprtvxz
```

1 Aus der Hilfe: s[i:j:k] → slice of s from i to j with step k.