Informatik für Schüler, Foliensatz 20
Wiederholung, Tricks und Tips

Prof. G. Kemnitz

Institut für Informatik, Technische Universität Clausthal
16. April 2009

Wiederholung Metasprache

- Was ist eine Metasprache und wozu dient sie?
- Was bedeuten die folgenden vier Grundelemente einer Metasprache?

`key`	Schlüsselwort, feststehender Name
`[...]`	optionaler Text, darf Null mal oder einmal enthalten sein
`{...}`	optionaler Text, darf beliebig oft enthalten sein
`kursiv`	Nicht-Terminalsymbol; Platzhalter für nach bestimmten Regeln zu bildende Texte
normal	Bezeichner; selbst gewählte Namen; Folge aus Buchstaben und Ziffern; keine Schlüsselworte

Syntax von Python-Anweisungen

Beschreiben Sie anhand der folgenden Syntaxdefinitionen:

- den allgemeinen Aufbau einer Funktionsdefinition

```
def Funktionsname([arg[=Wert]{,arg[=Wert]}]):
    Anweisung
    {Anweisung}
    [return Ausdruck]
```

- den allgemeinen Aufbau einer Wiederholschleife

```
for sv in sequenz:
    Anweisung
    {Anweisung}
```

- Wofür stehen die Platzhalter (Nicht-Terminalsymbole) *arg*, *Wert*, *Anweisung*, *Ausdruck* und *Sequenz*? Wie könnte man die Ersetzungsregeln formulieren?

Komandozeilenaufrufe

»man ls« zeigt u.a. die Kommandobeschreibung in der Metasprache:

```
ls [OPTION]... [FILE]...
```

- Welche Funktion hat das Kommando »man«?
- Wofür steht der Platzhalter »OPTION« und der Platzhalter »File«?
- Welche Optionen kennen Sie für das Kommando »ls«?
- Untersuchen Sie, was die Option »-a« bewirkt? (Im Manual nachschauen und ausprobieren.)
- Was tut »ls« ohne/mit Options- und File-Angabe?
- Das Zeichen »*« darf in eine File-Angabe für ein beliebiges Zeichen stehen. Wie lautet der Kommandozeilenaufruf zum Auflisten aller Datein, deren Namen
  - mit ».py« enden
  - eine »1« an einer beliebigen Stelle enthalten?

Experimente mit der Import-Anweisung

- Wozu dient die Import-Anweisung?
- Schreiben Sie eine Datei »test.txt« mit Funktionsdefinitionen und Konstantenvereinbarungen, z.B.:

```
def show(txt):
  print 'show -', txt
magic = 7
```

- Python-Interpreter im selben Verzeichnis starten und Folgendes ausprobieren:

```
show('time') # NameError...
from test import show
show('time') # jetzt geht es
magic        # NameError...
from test import magic
magic        # jetzt geht es
```

- alle Definitionen aus einem Modul (einer Programmdatei) importieren

```
from Modul import *
```

- Wofür steht der Platzhalter Module?
- Was unterscheidet diese Art der Importanweisung von der bisher verwendeten Art:

```
import Modul
```

»exec«-Anweisung

```
exec codestring
```

übersetzt den Programmtext in »codestring« und führt ihn aus.
Schreiben eines eigenen Python-Interpreters:

```
while True:
 s = raw_input('Eingabe:')
 exec s
```

- Was tut das Programm?
- Ausprobieren!
- Was pasiert bei einem Eingabefehler?
- Hinweis: Das Zeichen »Str-D«, mit dem der Python-Interpreter geschlossen wird, bedeutet »Dateiende« (EOF end of file) und löst bei der Eingabe über die Konsole einen »EOFError« aus

Aufgabe 20.1: Eingabefunktion für Zahlen

Eine Zahleneingabe besteht aus einer Ziffer, der beliebig viel Ziffern folgen dürfen:

Zahl :: *Ziffer*{*Ziffer*}
Ziffer :: 0|1|2|3|4|5|6|7|8|9[1]

Schreiben Sie eine Funktion

```
input_int(Promt) -> Zahlenwert
```

- die eine Zahlenwert anfordert
- kontrolliert, ob die Eingabe eine Zahl ist
- falls es keine Zahl ist, die Eingabeanforderung wiederholt
- wenn es eine Zahl ist, deren Wert mit

  ```
  exec 'return ' + string_mit_Zahlenwert
  ```

  zurückgibt.

[1] »::« Metazeichen für »zu ersetzen durch«; »|« Metazeichen für »entweder der rechte oder der linke Wert«

Hinweise:

- Test, ob ein Zeichen eine Ziffer ist, siehe Foliensatz 9 (Zeichenwert bestimmen und Bereichstest)
- Anweisung zur sofortigen Beendigung einer for- oder while-Schleife:

  ```
  break
  ```

- Test der Funktion zuerst durch Import im interaktiven Modus und Beispielaufrufe
- falls die interaktiven Tests erfolgreich sind, Testbeispiele zu einem Testprogramm zusammenfassen (Testanweisungen einfach mit »Pfeil zurück, Kopieren und Einfügen« in eine Programmdatei kopieren und alle Tests nochmal hintereinander ablaufen lassen)

Aufgabe 20.2: Labirint mit Schleifen

- Schreiben Sie sich ein Labirint mit Schleifen als Textdatei.
- Schreiben Sie je eine Funktion zur:
  - Erzeugung des Labirintobjekts als eine 2D-Array von Zeichen (siehe Foliensatz 13)

    `read_labirint(`*`Dateiname`*`)`
  - Anzeige des Labirintobjekts

    `show_labirint()`
  - zum Test, ob eine Position begehbar ist

    `test_Feld(x, y) -> boolean`
  - zum Kennzeichen einer benutzten Position mit einer fortlaufenden Zahl zur Visualisierung des gegangenen Weges

    `set_number(x, y)`
  - Erweiterung der Anzeigefunktion »show_labirint()«, so dass Zeichen als Doppelzeichen und Zahlen durch die zwei niederwertigen Ziffern dargestellt werden

- Erweitern Sie die Funktion »GeheSchritt()« aus Foliensatz 19 dahingehend, dass
  - der »Agent«, wenn er eine zulässige Position betritt, die fortlaufende Nummer in das Feld schreibt und
  - um nicht im Kreis zu laufen, mit Nummern gekennzeichnete Felder auch als unzulässig betrachtet
- Testen der einzelnen Funktionen wie in der Voraufgabe (zuerst interaktiv, dann mit einem Testprogramm)

Hinweis:

- eine fortlaufende Nummernvergabe verlangt eine globale Variable (vergl. Aufgabe 18.1)
- Das Hauptprogramm ist kurz und auch interaktiv testbar:
  - Einlesen des Labirints
  - Testausgabe des leeren Labirints
  - Aufruf der Funktion gehe Schritt mit den Positionsangaben für einen Labirniteingang
  - Testausgabe des Labirits mit dem Weg des »Agenten«