Einsatz der Programmiersprache Python in einem LK Informatik

Dr. Michael Savorić

Hohenstaufen-Gymnasium (HSG)

Kaiserslautern

Version 20090525

Überblick

- Beschreibung des Kurses / Einführung von Python
- Eigenschaften von Python
- Python in diesem Kurs – ausgewählte Themen:
  - Objektorientierung
  - Backtracking
  - Endliche Automaten
  - Kommunikation über die serielle Schnittstelle
  - Threads und Nebenläufigkeit
  - Socket-Programmierung
  - Projekt
- Fazit
- Literatur

Beschreibung des Kurses / Einführung von Python

- 12. Klasse
- 10 motivierte Schüler
- Mehr als ein Jahr Programmiererfahrung in Delphi

- Diskussionen mit Kollegen (Hat Delphi Zukunft?)
- Lehrer "entdeckte" Python erneut nach fast 15 Jahren

- 2 Wochen Programmieren in Python als Appetitanreger
- Freie und geheime Abstimmung: 10 zu 0 für Python

Heißt das nun: pro Python oder kontra Delphi?

Eigenschaften von Python

- Python:
  - Wenige grundlegende Befehle, viele Bibliotheken
  - Dynamisches Typsystem und Typprüfung zur Laufzeit
  - Blockbildung durch Einrücken des Quelltextes (ein Traum für Lehrer!)
  - Funktionen können 0, 1, 2, … Rückgabewerte haben
  - Besonderheiten: Listen, Langzahlarithmetik, ...
  - Programmiertechniken: imperativ, objektorientiert, funktional

- Python-Shell IDLE:
  - "Online"-Programmierung zum Kennenlernen der Sprache bzw. für kleinere Tests
  - Hilfesystem
  - Eingebauter komfortabler Editor

Objektorientierung in Python

- Im Kurs bisher nur angerissen, da in Delphi bereits umfangreich objektorientiert programmiert worden ist

- Möglichkeiten in Python:
  - Attribute und Methoden
  - Polymorphie
  - Überladen von Operatoren (==, !=, +, -, *, /, **, …) und Funktionen

  Eigene Klassen können nahtlos in Python eingebunden werden

- Beispiele:
  - Rechteck-Klasse
  - Baum-Klasse
  - Vektor-Klasse

Objektorientierung in Python (Fortsetzung)

- Auszug aus der Vektor-Klasse:

```
class Vektor: # class Vektor(object):
    def __init__(self, x = 0.0, y = 0.0, z = 0.0):
        self.x = x
        self.y = y
        self.z = z

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y \
               and self.z == other.z

    def __add__(self, other):
        v = Vektor()

        v.x = self.x+other.x
        v.y = self.y+other.y
        v.z = self.z+other.z

        return v
```

Backtracking in Python

- Beispiel: Dameproblem

  Stelle 8 Damen auf einem Schachbrett so auf, dass keine der Damen von einer anderen Dame bedroht wird.

```
D - - - - - - -
- - - - - - D -
- - - - D - - -
- - - - - - - D
- D - - - - - -
- - - D - - - -
- - - - - D - -
- - D - - - - -
```

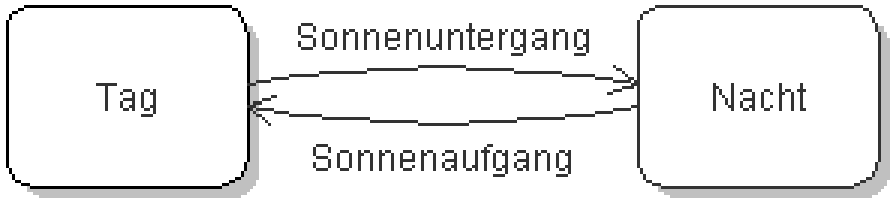
Backtracking in Python (Fortsetzung)

- Weitere Aufgabenstellungen:
  - Turmproblem
  - Labyrinth
  - Graphendurchwanderungen: alle bzw. kürzeste Wege
  - BWINF 2008 / Aufgabe 3: Alle Alpen

- Schulung des rekursiven Denkens sehr nützlich für weitere Themen im Unterricht:
  - Funktionales Programmieren in Haskell
  - Grammatiken und Sprachen

Endliche Automaten in Python

- Zustände und Zustandsübergänge, eventuell Ausgaben
- Erstes Beispiel:

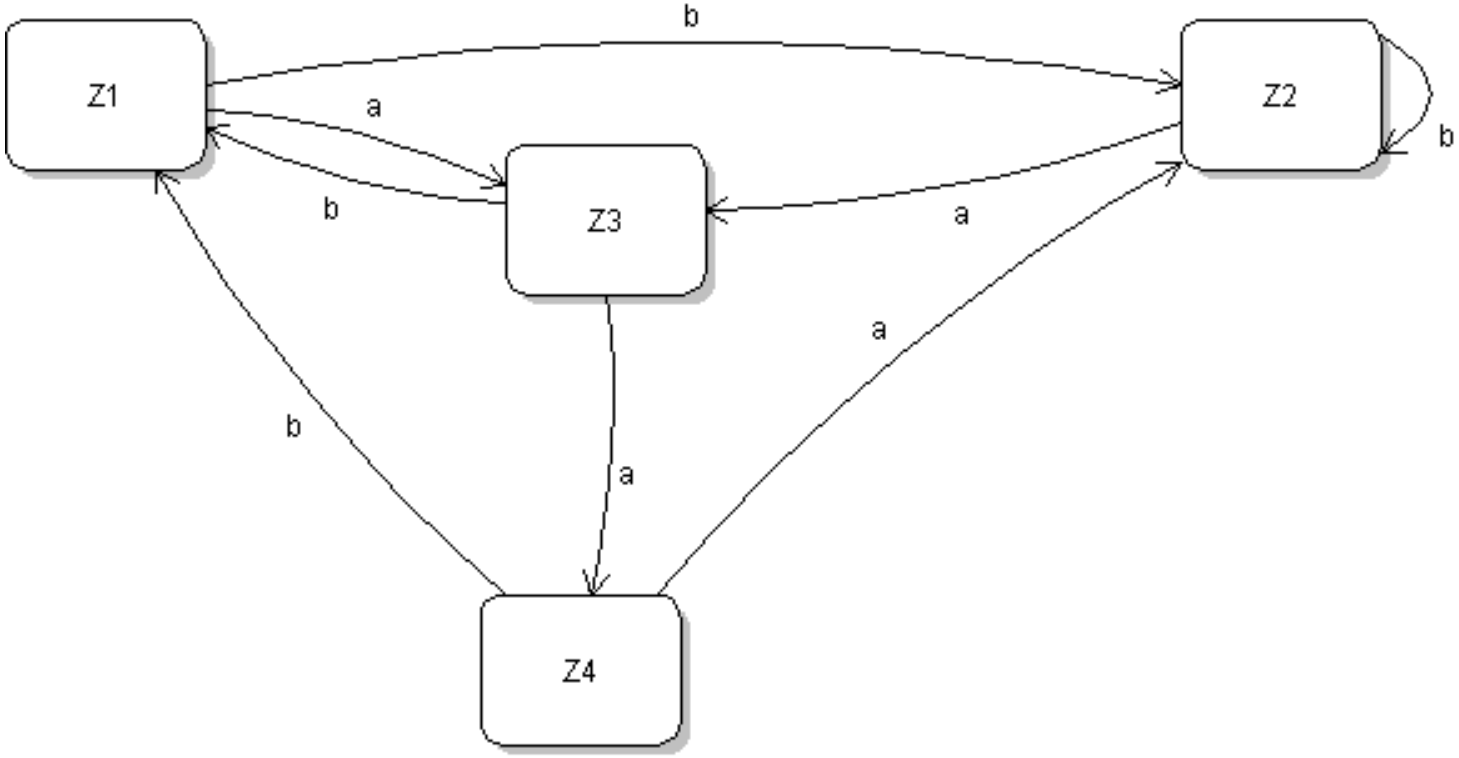


- Verschiedene Implementierungen der Übergänge:
  - Matrix
  - Listen (zentral oder dezentral für jeden Zustand)

- Durchwandern eines Automaten, diverse Ausgaben

Endliche Automaten in Python (Fortsetzung)

- Zweites Beispiel:



- Test auf Akzeptanz von Wörtern, (nicht) erreichbare Zustände, ...

Kommunikation über die serielle Schnittstelle in Python

- Verbindung zweier Rechner über ein Nullmodemkabel
- RTS- und CTS-Steuerleitung der seriellen Schnittstelle als Übertragungskanäle

Sender:	Empfänger:

`import serial`
`sender = serial.Serial(0)`
`...`
`sender.setRTS(0)`
`...`
`sender.setRTS(1)`
`...`
`sender.close()`
	`import serial`
	`empfaenger = serial.Serial(0)`
	`...`
	`signal = empfaenger.getCTS()`
	`...`
	`signal = empfaenger.getCTS()`
	`...`
	`empfaenger.close()`

Threads und Nebenläufigkeit in Python

- Wozu Threads?
  - Interaktive Programme
  - Zeitsteuerungen über exakte Timer
  - Parallele Arbeiten
- Nebenläufigkeit kann zu Problemen führen:
  - Verklemmungen
  - Simultaner Zugriff auf gleiche Speicherplätze
- Threads in Python sind einfach anzuwenden!

Threads und Nebenläufigkeit in Python (Fortsetzung)

- Beispiel 1: interaktive Anwendung
  - Oszilloskop (z.B. für das Testen der seriellen Kommunikation)
- Beispiel 2: Demonstration von Nebenläufigkeitsproblemen
  - Threadtest 1
  - Threadtest 2

- Beispiel 1: DNS-Abfrage

```
import socket

text_adresse = raw_input("Text-Adresse: ")

print "\nErmittlung der IP-Adresse für den Rechner: %s" \
      % (text_adresse)

ip_adresse = socket.gethostbyname(text_adresse)

print "\nIP-Adresse = %15s\n" % (ip_adresse)
```

Socket-Programmierung in Python (Fortsetzung)

- Beispiel 2: Daytime mit UDP

```
import socket


client_socket = socket.socket(socket.AF_INET,
                              socket.SOCK_DGRAM)
daytime_server = "time.fu-berlin.de"
client_socket.sendto("", (daytime_server, 13))
data, addr = client_socket.recvfrom(100)
client_socket.close()
del client_socket
print data
```

Socket-Programmierung in Python (Fortsetzung)

- Beispiel 3: Daytime mit TCP

```
import socket


client_socket = socket.socket(socket.AF_INET,
                              socket.SOCK_STREAM)
daytime_server = "time.fu-berlin.de"
client_socket.connect((daytime_server, 13))
data = client_socket.recv(100)
client_socket.close()
del client_socket
print data
```

Socket-Programmierung in Python (Fortsetzung)

- Eigene Dienste implementiert:
  - Echo
  - Zeichenzähler
  - Dateitransfer
  - E-Mail-"Fälschen" mit SMTP
  - Bank (Kontostandsabfrage, Überweisungen, Lastschriften)
  - Spiel (Zahlenraten)

- Weitere Aufgaben:
  - Alternating-Bit-Protokoll (Quittungsbetrieb)
  - Diverse Prüfsummen (Berechnung und Test)
  - Hamming-Abstand eines Kodes

Projekt: Bundesjugendspiel-Verwaltungsprogramm

- Ergebniseingabe, -verwaltung und -ausgabe
- 4 Wochen Unterricht + 1 Projektwoche
- Datenbank + Python + CGI $\rightarrow$ verteilte Anwendung
- Datenbank- und Objektmodell in einer Facharbeit
- Die Schüler und der Lehrer werden viel dabei lernen!

Fazit

- Python ist leicht zu lernen
- Besonderheiten von Python werden gerne angenommen
- Python ist betriebssystemunabhängig und kostenlos
- Python wird gepflegt und weiterentwickelt
- Umstieg auf Python kostet etwas Zeit, es lohnt sich aber!

Literatur

- G. Lingl, Python for Kids (3. Auflage), bhv, 2008.
- Offizielle Python-Dokumente (als PDF erhältlich)
- Internet-Recherche
- Für die Zukunft:

  M. Summerfield, Programming in Python 3, Addison Wesley, 2009