Teil V

IO und weitere Datentypen

Datei Ein- und Ausgabe

Dateien öffnen

- Textuelles Datei-Objekt erstellen

```
datei = open("testfile.txt")          # read
datei = open("testfile.txt", 'r')     # read
datei = open("testfile.txt", 'w')     # write
datei = open("testfile.txt", 'a')     # append
```

- Binäres Datei-Objekt erstellen

```
datei = open("testfile.txt", 'rb')    # read
datei = open("testfile.txt", 'wb')    # write
datei = open("testfile.txt", 'ab')    # append
```

- `open` Hat noch weitere Optionen (Codierung, Behandlung von newlines, . . .)

Methoden von Datei-Objekten

- Lesen

```
datei.read()        # komplett einlesen
datei.read(n)       # n Byte einlesen
datei.readline()    # eine Zeile lesen
datei.readlines()   # Liste von Zeilen
```

- Schreiben

```
datei.write("Neuer Text\n")
datei.writelines(["Erste Zeile\n", "Zweite Zeile\n"])
```

- Nicht vergessen: newlines (`\n`)
- Datei schließen

```
datei.close()
```

Textdateien Zeilenweise bearbeiten

- `for` Schleife über die Zeilen der Datei

```
datei = open("fulltext.txt")
for line in datei:      # Alt.: in datei.readlines()
    print line
# oder:
while True:
    line = f.readline()
    if not line:
        break
    print line
```

Weitere Methoden von Dateiobjekten

- `datei.tell()` – aktuelle Position ausgeben
- `datei.seek(pos)` – Zur gegebenen Position gehen
- `datei.flush()` – Ausgabepuffer leeren (Pufferung wegen Effizienz)

Standard Input, Output, und Error

- Modul `sys` stellt drei Standard Dateiobjekte bereit
  - `sys.stdin` – nur lesen
  - `sys.stdout`, `sys.stderr` – nur schreiben

```
import sys

sys.stdout.write("Text␣eingeben:␣")   # = print "..."
line = sys.stdin.readline()
# oder: line = raw_input("Text eingeben: ")

if error:
    sys.stderr.write("Error!\n")
```

- `write` Hängt nicht automatisch einen Zeilenumbruch an
- `raw_input([text])` entfernt Zeilenumbrüche
- Ein- und Ausgabe sind gepuffert

Kommandozeilenparameter

sys.argv

- `sys.argv` ist eine Liste mit Kommandozeilenparametern
- Erstes Element (`sys.argv[0]`) ist der Programmname

```
import sys
print "Programmname: %s" % (sys.argv[0])
if len(sys.argv) < 2:
    print "Keine Parameter angegeben"
    sys.exit(1)
print "Parameters:"
print ", ".join(sys.argv[1:])
```

- Bei sehr wenigen Parametern ist diese Methode praktikabel
- Schlecht bie vielen Parametern, oder wenn Parameter nur optional sein sollen,

Modul optparse

- Klasse `OptionParser` im module `optparse` vereinfacht das Parsen von Optionen

```
#!/usr/bin/python
import optparse

p = optparse.OptionParser()
# Optionen spezifizieren
p.add_option("-o", action="store", dest="opt")
p.add_option("-v","--verbose",action="store_true",
             dest="verbose",
             help="Erzeugt␣ausführliche␣Ausgabe")
# Optionen parsen
(options, args) = p.parse_args()

if options.verbose:
    print "Programm␣startet..."
print options, args
```

Modul optparse (2)

- Das Programm unterstützt jetzt
  - Standardhilfe über Parameter `-h`
  - Zwei vordefinierte Parameter
- Ausprobieren (`cmdline.py`):

```
./cmdline.py
./cmdline.py -h
./cmdline.py -o "Blubb" --verbose
./cmdline.py -v file1.txt file2.txt
```

Ausgewählte Parameter von `add_option`

- Jede Option (`"--xyz"`) kann eine Kurzform (`"-x"`) habem
- `action`: `'store'` (default), `'store_true'`, `'store_false'`, `'append'`, `'count'`, ...
- `dest` – Name der Variable zum speicher der Option
- `default` – Standardwert
- `help` – Hilfetext
- `type` – Typ der Variablen: `'string'` (default), `'int'`, `'long'`, `'float'`, `'complex'`, `'choice'`
- `choices = ['first', 'second', 'third']` – für `type='choice'`

Hilfetext mit `set_usage`

- Referenz auf den Programmnamen mit `%prog`

```
p.set_usage('''%prog [optionen] Datei(en)
Macht rein gar nichts mit den Dateien''')
```

nochmal Sequenzen: Dictionaries und Mengen

Bisherige Container zur Speicherung größerer Datenmengen

- Tupel: unveränderlich
- Liste: veränderlich
- Beide werden indiziert mit `0...n-1` (`n` Anzahl an Elementen)

Vorteile

- Tupel: sehr effizient, da unveränderlich
- Liste: höher Overhead, aber immer noch effizient
- Optimal, wenn eine Folge von Werten gespeichert werden soll

Beschränkungen

- keine Mengen darstellbar
- Keine Bezeichner für Objekte möglich
  z.B. suche Buch mit bestimmter ISBN oder bestimmten Titel

Dictionaries

- Bildet einen Schlüssel auf einen Wert ab
- Werte können beliebigen Typ haben
- Schlüssel nur unveränderliche Objekte (int, float, string, tuple,...)
- Kann als primitive Datenbank verwendet werden
- Erstellung mit geschweiften Klammern
- optional in den Klammern: Folge von `key:value` Paaren

```
>>> nummern = {}
>>> nummern["Martin"] = "089␣289␣18636"
>>> nummern["Benjamin"] = "089␣289␣16830"
>>> nummern["Martin"]
"089␣289␣18636"
```

Methoden von Dictionaries

```
>>> d = {7:'A', 'Muh': 'Maeh', (4, 5, 6):[6, 5, 4]}
>>> d.keys()
[(4, 5, 6), 'Muh', 7]
>>> d.values()
[[6, 5, 4], 'Maeh', 'A']
>>> d.items()
[((4, 5, 6), [6, 5, 4]), ('Muh', 'Maeh'), (7, 'A')]
>>> del d['Muh']
>>> for (key, value) in d.items():
>>>     print key, value
>>> for i in d.items():
>>>     print i[0], i[1]
>>> for key in d.keys():
>>>     print key, d[key]
```

Fehlerhafte Zugriffe

- Tupel und Listen: gesamter Indexbereich mit Werten belegt
- Bei Dictionaries gibt es keinen Indexbereich
- Was passiert, wenn ein unbekannter Schlüssel gesucht wird?

```
>>> nummern['Tobias']
KeyError: 4
>>> nummern.has_key('Martin')
True
>>> 'Martin' in nummern
True
```

Standardwerte

- Methode `get` liefert `None` bei nicht existierendem Schlüssel
- Optionaler zweiter Parameter für `get` definiert Standartwert

```
>>> nummern.get('Martin')
Martin
>>> nummern.get('Tobias', 'keine␣Nummer␣vorhanden')
'keine␣Nummer␣vorhanden'
```

set

- Repräsentiert eine Menge
- Jedes Element ist maximal ein Mal in der Menge
- `set` ist veränderbar, Elemente müssen aber unveränderlich sein
- Kann initialisiert werden mit beliebiger Sequenz

```
set("Hallo")                    # set(['a', 'B', 'l'])
s=set([3,1,2,3,"Bla",2])        # set([1, 2, 3, 'Bla'])
l = [2,8,7]
s.difference(l)                 # set([1, 'Hallo'])
s.intersection(l)               # set([2, 3])
s.union(l)                      # set([1, 2, 3, 4, 'Bla'])
s.symmetric_difference(l)       # set([1, 4, 'Bla'])
s.issubset([3,"Bla"])           # False
s.issuperset([3,"Bla"])         # True
```

- Weiter Methoden: `add`, `remove`, ...

Interaktion mit dem Betriebssystem

Modul os

- Schnittstelle zu Funktionalitäten des Betriebssystems
- Variablen:

```
import os
os.environ # Dictionary mit Umgebungsvariablen
os.linesep # Zeilenseparator; \r\n win, \n linux
os.name    # Name des OS-spez. Moduls (posix, dos,...)
```

- Some general purpose methods

```
os.getcwd()     # current working directory
os.chdir(path)  # wechseln ins Verzeichnis path
os.getgroups()  # Gruppen (UNIX)
os.uname()      # Systeminformation (UNIX)
os.times()      # (usr, sys, c_usr, c_sys, wct)
...
```

OS Methoden für Dateien

```
chmod(path, mode)          # chmod
chmod("tmp.py", 0664)      # Modus erfordert vier Zeichen
listdir(path)              # ls
mkdir(path)                # mkdir
makedirs(path)             # mkdir -p
rename(src, dst)           # mv
remove(path)               # rm
rmdir(path)                # rmdir
removedirs(path)           # rm -rf
stat(path)                 # stats (atime, ...)
```

Beispiel: Aktienkurse

Aufgabe:

- In mehreren csv-Dateien liegen Kursdaten vor
- Eine Datei pro Firma (Dateiname=Firmenname)
- Spalte mit den Kursen wird ermittelt
- Methode zum Einlesen der Daten in interne Datenstruktur
- Grafische Darstellung der Daten
- Ermittlung von Mittelwerten

Folgenes wird geübt:

- Kontrollstrukturen
- Funktionen
- Module
- Dictionaries
- Dateizugriff
- Stringmanipulation mit `s.split(chr)`
- Reduce-Operationen (`min`, `max`, `sum`)

Format der csv-Dateien (finance.yahoo.com)

```
Date,Open,High,Low,Close,Volume,Adj Close
2010-11-01,66.68,71.43,65.55,68.10,90700,68.10
2010-10-01,63.15,70.10,59.55,66.00,304000,66.00
2010-09-01,50.53,64.65,50.38,63.30,120700,63.30
2010-08-02,53.80,55.85,47.65,48.20,177600,48.20
2010-07-01,51.45,56.89,50.05,54.05,186700,54.05
2010-06-01,49.01,55.10,47.21,50.55,415500,50.55
2010-05-03,50.90,52.27,42.22,49.19,1485400,49.19
2010-04-01,47.58,53.01,46.60,50.94,1098400,50.94
2010-03-01,41.95,47.57,41.80,47.01,788800,47.01
2010-02-01,46.47,48.74,40.58,41.81,647900,41.81
2010-01-04,53.62,54.38,45.00,45.81,553800,45.81
2009-12-01,52.51,54.63,50.74,53.30,326000,53.30
2009-11-02,48.27,54.10,46.03,50.93,406200,50.93
2009-10-01,50.01,56.57,47.00,48.23,536200,48.23
...
```

Nötige Module

- `os` für `listdir`, `pylab` zum plotten

```
import os
from pylab import *
```

Ermitteln aller Firmen

- Im Verzeichnis liegen beliebig viele csv-Datein

```
def liesFirmen(dir):
  firmenliste = []
  dateien = os.listdir(dir)
  for dateiname in dateien:
    if dateiname[-4:] == ".csv":
      firmenliste.append(dateiname[:-4])
  return firmenliste
```

Liste mit Kurswerten einer Firma einlesen

- Datei zum lesen öffnen
- Datei zeilenweise durchgehen
- Elemente der letzten Spalte extrahieren

```
def liesKurse(firma):
  datei = open(firma+'.csv', 'r')
  datei.readline() # Kopfzeile
  preise = []
  for zeile in datei:
    spalten = zeile.split(',')
    preis = spalten[-1]
    preise.append(float(preis))
  datei.close()
  preise.reverse()
  return preise
```

Automatisch alles einlesen

- Firmennamen herausfinden
- Dateien Einlesen
- Speichern der Kurswerte in einem Dictionary
- Berechnen von Mittelwerten

```
def init():
  daten = {}
  firmen = liesFirmen(".")
  for f in firmen:
    daten[f] = {}
    daten[f]['preise'] = liesKurse(firma)
    daten[f]['avgpreis'] = sum(daten[f]['preise']) \
                           / len(daten[firma]['preise'])
    daten[f]['minpreis'] = min(daten[f]['preise'])
    daten[f]['maxpreis'] = max(daten[f]['preise'])
  return daten
```

Plotten der Daten

- Details zum Plotten gibt es später

```
def zeichneKurse(daten, firmenliste):
  for firma in firmenliste:
    plot(daten[firma]['preise'])
  show()
```

- Verwendung als Programm und Modul

```
if __name__ == '__main__':
  daten = init()
  maxlen = max([len(key) for key in daten.keys()])
  for firma in daten.keys():
    name = firma+" "*(15-len(firma))
    min = daten[firma]['minpreis']
    max = daten[firma]['maxpreis']
    avg = daten[firma]['avgpreis']
    print "%s: Preis(min/max/avg): %6g, %6g, %6g" \
          %(name, min, max, avg)
```

- Ausführung als Programm

```
>python aktienkurse.py
Deutsche_Bank (min/max/avg):   25.56, 153.55, 80.5186
Microsoft     (min/max/avg):   15.66,   42.8, 24.5753
Daimler       (min/max/avg):   22.62, 110.15, 47.6528
```

- Ausführung als Modul

```
>>> from aktienkurse import *
>>> daten = init()
>>> zeichneKurse(daten, daten.keys())
```

Ausblick

- 22. Nov.: reguläre Ausdrücke und Exceptions
- 29. Nov.: Objektorientierte Programmierung
- 06. Dez.: Visualisierung mit Turtle und TKinter
- 13. Dez.: Simulation von Partikelsystemen
- 20. Dez.: Datenstrukturen: Bäume, Stacks und Queues
- 10. Jan.: Wissenschaftliches Rechnen mit NumPy und SciPy
- 17. Jan.: Wärmeleitungsgleichung
- 24. Jan.: Lineare Gleichungssysteme
- 31. Jan.: Mehrgitterverfahren
- 07. Feb.: Klausurtraining
- 01. Mär.: Klausur (ca. 11:30-13:00)