

# Module

- Funktionalität zusammenfassen in einer separaten .py Datei
- Importieren Bibliotheksmodul (Schon verwendet: math, time, ...)
- Beispiel (tools.py):

```
"""This module provides some helper tools.
Try it."""

counter = 42

def readfile(fname):
    "Read text file. Returns list of lines"
    fd = open(fname, 'r')
    data = fd.readlines()
    fd.close()
    return data

def do_nothing():
    "Do really nothing"
    pass
```

## Importieren

- Modul importieren

```
import tools
tools.do_nothing()
print tools.counter
```

## Importieren

- Modul importieren

```
import tools
tools.do_nothing()
print tools.counter
```

- Modul unter neuem Namen importieren

```
import tools as t
t.do_nothing()
```

## Importieren

- Modul importieren

```
import tools
tools.do_nothing()
print tools.counter
```

- Modul unter neuem Namen importieren

```
import tools as t
t.do_nothing()
```

- Einzelne Funktionen direkt importieren

```
from tools import do_nothing, readfile
from tools import counter as cntr
do_nothing()
print cntr
```

## Import beeinflussen

- Alle Symbole in den Namensraum importieren

```
from tools import *  
do_nothing()  
print counter
```

- Module können bestimmen, welche Symbole mit `from module import *` importiert werden:

```
# module tools.py  
__all__ = ['readfile', 'counter']
```

Die Funktion `do_nothing()` ist nach `import *` nicht bekannt!

## Import beeinflussen

- Alle Symbole in den Namensraum importieren

```
from tools import *  
do_nothing()  
print counter
```

- Module können bestimmen, welche Symbole mit `from module import *` importiert werden:

```
# module tools.py  
__all__ = ['readfile', 'counter']
```

Die Funktion `do_nothing()` ist nach `import *` nicht bekannt!

## Funktionalität abfragen

```
dir(tools)  
dir(math)
```

## Hilfe zu Modulen und Funktionen

- Docstrings abfragen mit `help`

```
import tools
help(tools)
help(tools.do_nothing)
```

## Hilfe zu Modulen und Funktionen

- Docstrings abfragen mit `help`

```
import tools
help(tools)
help(tools.do_nothing)
```

## Ausführbares Programm als Modul?

- `tools.py` soll sowohl Bibliothek sein, als auch selbst ausführbar

```
# tools.py
...
if __name__ == '__main__':
    print "tools.py executed"
else:
    print "tools.py imported as module"
```



## Module debuggen

- Wenn ein Modul geändert wird, werden Änderungen nur wirksam, wenn Python neu gestartet wird, oder `reload` (Python<3.0) verwendet wird.

```
reload(tools)
```

## Module debuggen

- Wenn ein Modul geändert wird, werden Änderungen nur wirksam, wenn Python neu gestartet wird, oder `reload` (Python<3.0) verwendet wird.

```
reload(tools)
```

## Pfad für Module

- Python sucht im Suchpfad und im aktuellen Verzeichnis
- Suchpfad kann erweitert werden

```
import sys
sys.path.append("folder/to/module")
import ...
```

# Pakete

- Module können gruppiert werden
- Hierbei ist die Verzeichnisstruktur wichtig

```
tools/  
  __init__.py      # Inhalt für "import tools"  
  files.py         # für "import tools.files"  
  graphics.py     # für "import tools.graphics"  
  stringtools/  
    __init__.py   # für "import tools.stringtools"  
    ... weitere Verschachtelung möglich
```

- Wenn `from tools import *` Untermodule enthalten soll, muss `tools/__init__.py` folgende Zeile enthalten:

```
__all__ = ["files", "graphics"]
```

## 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
```

# 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
```

## 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"])
```



## 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 = datei.readline()
    if not line:
        break
    print line
```

## 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 = datei.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 bei vielen Parametern, oder wenn Parameter nur optional sein sollen  
⇒ Modul `argparse`

# nochmal Sequenzen: Dictionaries und Sets

## Bisherige Container zur Speicherung größerer Datenmengen

- Tupel: unveränderlich
- Liste: veränderlich
- Beide werden indiziert mit  $0 \dots n-1$  ( $n$  Anzahl an Elementen)

# nochmal Sequenzen: Dictionaries und Sets

## Bisherige Container zur Speicherung größerer Datenmengen

- Tupel: unveränderlich
- Liste: veränderlich
- Beide werden indiziert mit  $0 \dots n-1$  ( $n$  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

# nochmal Sequenzen: Dictionaries und Sets

## Bisherige Container zur Speicherung größerer Datenmengen

- Tupel: unveränderlich
- Liste: veränderlich
- Beide werden indiziert mit  $0 \dots n-1$  ( $n$  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 bestimmtem Titel



## Dictionary

- 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["Tobias"] = "089_289_18632"  
>>> nummern["Benjamin"] = "089_289_16830"  
>>> nummern["Benjamin"]  
"089_289_18630"
```

## 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['Andreas']  
KeyError: 4  
>>> nummern.has_key('Benjamin')  
True  
>>> 'Benjamin' in nummern  
True
```

## 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['Andreas']
KeyError: 4
>>> nummern.has_key('Benjamin')
True
>>> 'Benjamin' in nummern
True
```

## Standardwerte

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

```
>>> nummern.get('Benjamin')
"089_289_18630"
>>> nummern.get('Andreas', '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

- 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, 3, 'Bla'])
s.intersection(l)     # set([2])
s.union(l)             # set([1, 2, 3, 7, 8, 'Bla'])
s.symmetric_difference(l) # set([1, 3, 7, 8, 'Bla'])
s.issubset([3,"Bla"])  # False
s.issuperset([3,"Bla"]) # True

```

- Weitere Methoden: `add`, `remove`, ...
- bestimmte Operatoren auf sets unterstützt: `-`, `^`, `|`, `&`

# 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
mkdirs(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

# 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

## Folgendes 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-Dateien

```
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(f)
        daten[f]['avgpreis'] = sum(daten[f]['preise']) \
                                / len(daten[f]['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()
```

## 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 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())
```