



TECHNISCHE
UNIVERSITÄT
DRESDEN

FORTGESCHRITTENE PYTHON-PROGRAMMIERKONZEPTE

Folien: C. Knoll

Sprecher: S. Brückner

Dresden, WiSe 2013/14

- Funktionale Programmierung,
- Verschachtelte Funktionen
- Namensräume
- `import`-Mechanismen
- Exceptions (Fehlerbehandlung)
- PEP8, Unittests, Doku-Tools, Versionsverwaltung

- Anwendung von Funktionen auf Listen
 - Schlüsselwort `lambda` → anonyme Funktion („Wegwerf-Funktion“)

```
L = [1, 3, 5, 7, 9]
strings = map(lambda z: str(z), L)
big_numbers = filter(lambda n: n > 5, L)
```

- Kompakt zu schreiben, aber ggf. schwerer verständlich
- Alternative zu `map` und `filer`: „list comprehension“:

```
strings = [str(z) for z in L]
big_numbers = [z for z in L if z > 5]
```

- Funktionsanwendung für Matrizen und arrays:
`sympy.Matrix.applyfunc; numpy.ufunc`

- Typisch in FP: Rekursion (Funktion, die sich selbst aufruft)

```
def fakultaet(n):
    if n == 0:
        return 1
    else:
        return n*fakultaet(n-1)
```

- Funktionen: ganz normale Objekte
- können Argumente anderer Funktion sein

```
def func1(x):  
    "Ein docstring"  
    return x+1  
  
print type(func1) # <type 'function'>  
  
def doku_lesen(f):  
    print f.__doc__  
  
doku_lesen(func1) # -> "Ein docstring"
```

- Funktionen können auch innerhalb anderer Funktionen angelegt werden
- (Wichtig: Gültigkeitsbereiche der Variablen („Scopes“))

```
def fabrik(y):  
    def f(x):  
        return x+y  
  
    return f # Rueckgabe des Funktionsobjekts  
  
func1 = fabrik(10)  
func2 = fabrik(5)  
  
print func1(2), func2(2) # -> 12, 7
```

- Dekorierer: Funktionen, die auf Funktionen angewendet werden und neue Funktionen zurückgeben
- Nützlich für Vor- und Nachbehandlung von Daten
- Spezielle Syntax: mit @-Zeichen

- Namensraum: Gültigkeitsbereich von Variablen-Namen
- Jeder Name pro Namensraum höchstens einmal belegt → Eindeutigkeit
- Begriffe „namespace“ und „scope“ fast synonym
- Üblich: verschachtelte Namensräume
 - in einer normalen Funktion: 2 Ebenen
 - Funktion in Funktion: 3 Ebenen
- Suchschema: von innen nach außen
- Jedes Modul hat einen globalen Namensraum
- Ordnung halten mit Präfixen:

```
import numpy, sympy
a = numpy.arange(5)
x = sympy.Symbol('x')
```

```
print numpy.exp(a)
print sympy.exp(x)
# Funktionen heissen gleich,
# leben aber in unterschiedlichen Namensraeumen
```

- Modul = Python-Datei
- Aktuell ausgeführte Datei: „main-Modul“
- `import`: Namen (+ daran gebundene Objekte) in eigenen Namensraum importieren
- Zwei Varianten:

```
import numpy # Import des Moduls
x = numpy.arange(5)
# oder: Import einzelner Objekte
from numpy import arange, pi
x = arange(5)*pi
```

- Abkürzungen (Aliase):

```
import numpy as np
x = np.arange(5)
# oder: Import einzelner Objekte
from numpy import arange as ar
x = ar(5)
```

Weitere Abkürzungen:

```
# mehrere imports in einer Zeile (ggf. mit Alias):  
import numpy, pylab as pl, sympy
```

- Wildcard-Import (*)
- Nur zum interaktiven Ausprobieren empfohlen (keine Kontrolle):

```
from numpy import * # "namespace pollution"
```

- Importiertes Modul wird „ganz normal“ ausgeführt:
(aber nur ein mal; reload → erneutes importieren)

```
# modull.py
def funktion1(x):
    return x**2 - 3*x + 5
```

```
print "abc"
Z = 123
```

— — — —

```
# main.py
import modull # -> abc
import modull # keine Ausgabe
reload(modull) # -> abc
print modull.Z + 0.4 # -> 123.4
```

- Auch Module sind Objekte:

```
print type(modull) # -> <type 'module'>
```

- Paket = Verzeichnis mit `__init__.py` und ggf. weiteren Modulen
- Verschachtelbar (`Paket.Unterpaket.Modul.Objekt`)
- `__init__.py` kann leere Datei sein

Wozu Modularisierung?

- Thematische Gliederung von Code
- Wiederverwendbarkeit, Weitergabe, Austausch

- Paket = Verzeichnis mit `__init__.py` und ggf. weiteren Modulen
- Verschachtelbar (`Paket.Unterpaket.Modul.Objekt`)
- `__init__.py` kann leere Datei sein

Wozu Modularisierung?

- Thematische Gliederung von Code
- Wiederverwendbarkeit, Weitergabe, Austausch

Importpfade:

- Python-Interpreter sucht in bestimmten Standardverzeichnissen

```
import sys
print sys.path
```

- Anpassungsmöglichkeiten:
 - PYTHONPATH-Umgebungsvariable
 - `mein_pfad.pth`-Datei (in einem der Standardverzeichnisse)
 - `sys.path.append(...)`

- Fehler beim Programmieren unvermeidlich
- Gute Fehlerbehandlung spart viel Zeit und Nerven
- Python: Exceptions
 - werden an fehlerhafter Stelle „geworfen“
 - können „(ab)gefangen“ werden

```
def F(x, y):  
    return x/y
```

```
def G(z):  
    try:  
        print F(10, z)  
    except ZeroDivisionError:  
        print "Division durch 0!"
```

```
G(5) # -> 2
```

```
G(0) # -> Division durch 0
```

- Wichtige Typen: NameError, ValueError, TypeError, AttributeError, NotImplementedError, Exception (=Basisklasse)

Empfehlungen:

- Ab und zu rückversichern ob Annahmen zu Laufzeit noch erfüllt sind

→ In eigenem Code Exceptions werfen: `raise`

```
def F(x):  
    if not isinstance(x, (float, int)):  
        meldung = "Zahl erwartet, %s bekommen" % type(x)  
        raise ValueError, meldung  
    return x**2
```

- Bequemer (⇒ niedrigere Einsatzhürde): `assert`

```
def F(x):  
    assert isinstance(x, (float, int))  
    return x**2
```

→ `AssertionError`

- Fehlende Typ-Information → nur für „Privatgebrauch“

PEP8

- **P**ython **E**nhancement **P**roposal Nr. 8
- Style-Guide für guten Code (Konventionen für Benennung und Formatierung)

`unittest`-Paket

- Automatisiertes Testen von Funktionen
- Unverzichtbar bei größeren Projekten

`sphinx`-Paket

- Automatisches Erzeugen von Dokumentation aus Code
- Erfordert gute Docstrings

Versionsverwaltungs-Werkzeug

- Unverzichtbar bei mehreren Personen, auch alleine sinnvoll
- Empfehlung: „git“ oder „mercurial“

- Funktionale Programmierung (`lambda`, `map`, `filter`, `list comprehension`)
- Verschachtelte Funktionen (Funktionen in Funktionen definieren)
- Namensräume, `import`-Mechanismen (Ordnung halten)
- Exceptions (werfen und abfangen, `assert`)
- PEP8, Unittests, Doku-Tools, Versionsverwaltung

- PEP8 (Style Guide): <http://www.python.org/dev/peps/pep-0008/>
- unittest: <http://docs.python.org/2/library/unittest.html>
- sphinx (Doku automatisch generieren): <http://sphinx-doc.org/>
- Versionsverwaltung: <http://git-scm.com/book/de/>