Einführung in Python Teil I

Grundlagen

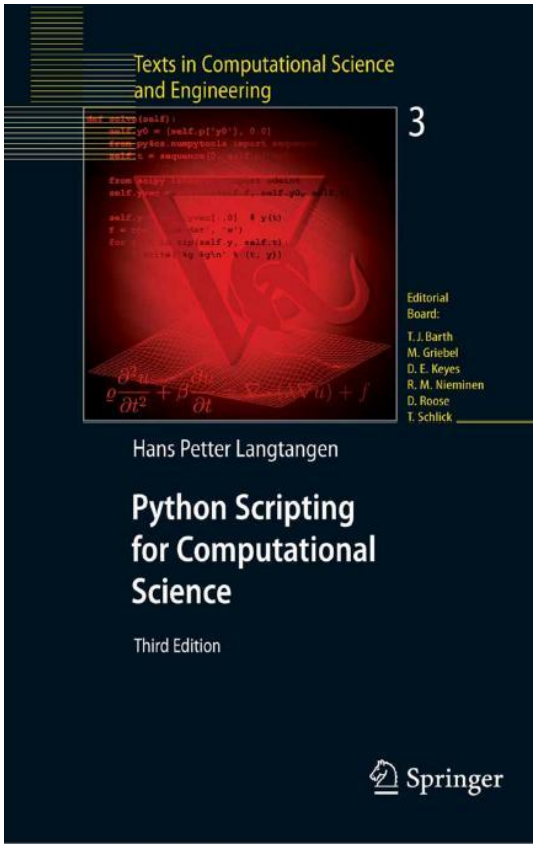
Valentin Flunkert

Institut für Theoretische Physik
Technische Universität Berlin

Do. 27.5.2010
Nichtlineare Dynamik und Kontrolle SS2010

Diese Einführung

- orientiert sich stark an [Lan05]:
  H. P. Langtangen, *Python Scripting for Computational Science*, Springer (2005)
- Buch ist aus Vorlesung hervorgegangen
  Folien zur Vorlesung im Internet
  google nach: `basic python course`



Warum Python?

Python ist eine Skriptsprache

- einfacher Syntax
- interpretiert und nicht kompiliert
  $\rightarrow$ sehr flexibel aber auch langsamer als z.B. `C`
- interaktiver Modus
- Bibliotheken für jeden erdenklichen Zweck
  (www, html, CGI, xml, Datenbank, GUI, ...)
- Batteries included
- benannt nach Monty Python's Flying Circus

Python vs. Matlab

Gemeinsamkeiten

- keine Deklaration von Variablen
- einfacher, klarer Syntax
- geeignet für Simulation und Visualisierung

Vorteile von Python

- Python ist mächtiger (vollwertige Programmiersprache)
- open source
- viele Bibliotheken für (Datenbank, CGI, GUI, ...)

Matlab hat momentan noch mehr Funktionalität für Numerik (aber Python holt schnell auf).

Interaktiver Modus

- einfacher Modus: '`python`', kommt mit Python
  besser: '`ipython`', muss extra installiert werden
- der ultimative (Taschen)Rechner
- ipython hat viele praktische Zusatzfunktionen
  - tab-Vervollständigung
  - Protokoll in Datei mitschreiben
  - Programm Geschwindigkeit analysieren
  - ...
- schnelles Testen von Code-Schnipseln
- mit '`>>> ?befehl`' Infos über Befehle und Module

Ein erstes Beispiel

- Beispiel `hw.py` aus [Lan05]

```
import sys, math          # sys und math modul laden

r = sys.argv[1]           # 1. Kommandozeilenparameter

s = math.sin(float(r))

print "Hello, World! sin(" + str(r) + ") = " + str(s)
```

- Programm ausführen:

```
python hw.py 1.4
```

Erklärung

- Module importieren

```
import sys, math
```

- Zahlen und Strings sind verschiedene Typen:

```
r = sys.argv[1]           # r ist ein string
s = math.sin(float(r))    # math.sin erwartet float
```

Ausgabe

- Gewünschte Ausgabe

```
Hello, World! sin(3.4)=-0.255541102027
```

Möglichkeiten:

- Strings Verknüpfen mit +

```
print "Hello, World! sin(" + str(r) + ") = " + str(s)
```

- Formatierung ähnlich wie in C

```
print "Hello, World! sin(%g) = %g" % (r, s)
```

- Formatierung mit Variablennamen

```
print "Hello, World! sin(%(r)g) = %(s)g" % vars()
```

Bemerkung: `"..."` und `'...'` sind gleich bedeutend;
praktisch: `print 'er sagte: "hallo"'`

Module

- Infos über Modul in der Kommandozeile mit
  'pydoc <modulname>' oder 'pydoc <modulname>.<funktion>'
- Modul importieren

```
import math
c = math.sqrt(a**2 + b**2)

import math as m                    # Abkuerzung
c = b / m.cos(alpha)

from math import *                  # alle Befehle aus math
y = sqrt(sin(x))

from math import sqrt, sin          # nur sqrt und sin

from math import sqrt as squareRoot
```

Eine Anwendung

- Aufgabe:
  - (x, y) Daten aus einer Datei einlesen (zwei Spalten)
  - Funktion auf y anwenden
  - Ergebnis in neue Datei schreiben
- Benutzung

```
python datatrans.py infilename outfilename
```

- Zwei Kommandozeilenparameter

```
infilename = sys.argv[1]
outfilename = sys.argv[2]
```

(`sys.argv[0]` ist der Name des Skripts)

Lesen und Schreiben von Dateien

- Dateien öffnen:

```
ifile = open( infilename, 'r')    # r zum Lesen
ofile = open(outfilename, 'w')    # w zum Schreiben
afile = open(appfilename, 'a')    # a zum Anhaengen
```

- Zeile für Zeile einlesen:

```
for line in ifile:
    # mach was
```

- Blöcke werden eingerückt; keine Klammern!

Eine Funktion definieren

```
from math import sqrt, sin

def myfunc(y):
    """Berechne den Wert, wenn y>=0 ist,
    sonst gib 0 zurueck."""
    if y >= 0.0:
        return sin(y**2) * sqrt(y)
    else:
        return 0
```

docstring der Funktion kann mit pydoc gelesen werden.

Daten verarbeiten

- Input Datenformat: Zweil Spalten mit Zahlen

```
0.1     1.31231
0.4     2.60342
0.8     8.12312
```

- `(x,y)` einlesen, `y` umrechnen und in neue Datei schreiben:

```
for line in infile:
    pair = line.split()
    x = float(pair[0]); y = float(pair[1])
    fy = myfunc(y)
    ofile.write('%g\t%g\n' % (x, fy))
```

Exception handling

- Übergabe von Dateiname

```
infilename = sys.argv[1]
infile = open(infilename, 'r')
```

- Was passiert bei falschem Dateinamen? → Fehler
- Fehler abfangen:

```
try:
    infilename = sys.argv[1]
    infile = open(infilename, 'r')
except:
    # try block fehlgeschlagen
    print 'Datei existiert nicht!'
    sys.exit(1)   # programm beenden
```

Daten in Listen einlesen

- Input Datei in Liste mit Zeilen einlesen

```
lines = ifile.readlines()
```

- `x` und `y` in Listen speichern

```
x = []; y = []
for line in lines:
    xval, yval = line.split()
    x.append(float(xval))
    y.append(float(yval))
```

Mehr über Listen

- Listen initalisieren

```
>>> a = []
>>> b = [1, 2, 3, 'a', 'b', a]
>>> nums = range(10); print nums
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Listen verändern

```
>>> b.append('hallo')
>>> b.sort(); print b
[1, 2, 3, [], 'a', 'b', 'hallo']
```

- Schleifen laufen über Listen

```
for i in range(start, stop, inc):
for j in range(stop):
```

erzeugt

```
i = start, start+inc, start+2*inc, ..., stop-1
j = 0, 1, 2, ..., stop-1
```

Indexing

```
>>> X = range(15); print X
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

>>> print X[2]
2

>>> print X[-1]
14

>>> print X[11:]
[11, 12, 13, 14]

>>> print X[4:10]
[4, 5, 6, 7, 8, 9]

>>> s = 'Strings verhalten sich wie Listen'
>>> print s[1::2]
tig ehle ihweLse
```

List comprehension

- übersichtlicher Syntax um Listen zu erzeugen

```
>>> nums = range(20)

>>> squares = [ i**2 for i in nums ]
>>> print squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, ...]

>>> evenSq = [ i**2 for i in nums if i%2 == 0]
>>> print evenSq
[0, 4, 16, 36, 64, 100, 144, 196, 256, 324]
```

- ausgeschrieben

```
evenSq = []
for i in nums:
    if i%2 ==0:
        evenSq.append(i**2)
```

Dictionaries

- Dictionaries = arrays mit Text als Index
- heissen auch *hash* in anderen Sprachen
- Beispiel

```
# initialisieren
>>> a = {'femto': -15, 'nano': -9, 'mikro': -6}
>>> print a['nano']
-9

>>> a['milli'] = 'vanilli'

>>> print a.keys()
['mikro', 'nano', 'femto', 'milli']

>>> print a.values()
[-6, -9, -15, 'vanilli']
```

Mehr über Funktionen

- Funktionen haben diese Form

```
def function_name(arg1, arg2, arg3):
    """docstring ..."""
    # do something
    return something
```

- Keyword arguments

```
def Y(theta, phi, l=0, m=0):
    ...
    return ergebnis
```

aufrufen mit

```
Y(pi, 0.0, 0, 0)
Y(0.2*pi, 0.3*pi, 1, 2)

# oder besser:
Y(theta=pi, phi=0.0)
Y(theta=0.2*pi, phi=0.3*pi, l=1, m=2)
```

Funktionen wiederverwenden – neues Modul

- Datei 'meinmodul.py':

```
import math
c = 2.99E8     # Lichtgeschwindigkeit

def m_rel(m0, v):
    """Berechne die relativistische Masse"""
    gamma = 1.0 / math.sqrt( 1.0 - v**2/c**2 )
    return gamma * m0

# Wird nur ausgefuehrt wenn das Skript
# direkt ausgefuehrt wird, nicht beim import:
if __name__ == '__main__':
    print m_rel(1000, 0.5*c)
```

- Andere Datei:

```
import meinmodul as mm
m = mm.m_rel(80, 0.5*mm.c)
```

- Infos in Kommandozeile 'pydoc meinmodul'

Literatur

H. P. Langtangen: *Python Scripting for Computational Science* (Springer, 2005).