http://www.icp.uni-stuttgart.de

Physik auf dem Computer
Wiederholung: Python, NumPy, Matplotlib
Objekte und Klassen

Olaf Lenz **Axel Arnold**

Institut für Computerphysik
Universität Stuttgart

Sommersemester 2013

http://www.icp.uni-stuttgart.de

Teil 1: Wiederholung Python

Python

- Wiederholung der wichtigsten Eckpunkte
- interpretierte Programmiersprache
- aktuelle Versionen 3.3.1 bzw. 2.7.4

Hilfe zu Python

- Homepage: `http://python.org`
- „A Byte of Python“: `http://abop-german.berlios.de`
- „Dive into Python“: `http://diveintopython.net`

Python starten

```
#!/usr/bin/python

print "Hello World"
```

- Skript mit `python helloworld.py` starten
- oder Skript ausführbar machen (`chmod a+x helloworld.py`)
- OS erkennt an „ #! “ (*shebang*) in erster Zeile, welcher Interpreter benutzt werden soll
- Python interaktiv mit `ipython`
  (Tab-Vervollständigung, History, . . .)

Syntax

```
#!/usr/bin/python

# Eine sinnlose Bedingung
if 1 != 1:
    pass
else:
    print "Hello World"
```

- Umlaute vermeiden oder Encoding-Cookie einfügen
- Kommentare mit „#“
- Einrückung spielt eine Rolle!
- Alle gleich eingerückten Zeilen gehören zum selben Block
- Ein „:“ am Ende der vorigen Zeile beginnt den neuen Block
- Spart viele Codezeilen!
- `pass` macht nix

Zahlen, Zeichenketten, Vergleiche, Variablen

```
>>> -12345
-12345
>>> 13.8E-24
1.38e-23
```

```
>>> "Hello World"
'Hello World'
>>> 'Hello World'
'Hello World'
>>> """Hello
... World"""
'Hello\nWorld'
```

```
>>> 1 == 2 or 1 != 2
True
>>> 1 < 2 and not 1 > 2
True
>>> '1' == 1
False
```

```
>>> number1 = 1
>>> number2 = number1 + 5
>>> number1, number2
(1, 6)
>>> s = "Keine Zahl"
>>> s
'Keine Zahl'
```

Typen

```
>>> print 13 + "a"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
>>> x = 13
>>> x
13
>>> x = "a"
>>> x
'a'
```

```
>>> x = 13
>>> type(x)
<type 'int'>
>>> x = 13.
>>> type(x)
<type 'float'>
>>> x = 2
>>> x += 0.1
>>> type(x)
<type 'float'>
```

Arithmetische Ausdrücke, Bedingungen, Schleifen

```
>>> ((1.+2.)*3./4.)**5.
57.6650390625
>>> '1'+'2'
'12'
>>> 1/2, 1%2
(0, 1)
>>> 1./2
0.5
```

```
if a < 3:
   print "a < 3"
elif a > 7:
   print "a > 7"
else:
   print "3 < a < 7"
```

```
i = 1
while i < 10:
    if i % 7 == 0: break
    if i % 3 == 0: continue
    print i
    i += 1
# Ausgabe: 1 2 4 5 6
```

```
>>> for s in ["Axel", "Olaf"]:
...    print "Hello", s
...
Hello Axel
Hello Olaf
```

http://www.icp.uni-stuttgart.de

Funktionen

```
def max2(a, b):
    if a > b: return a
    else: return b
```

```
pi = 3.14159
def circle_area(r):
    return pi*r**2
```

```
def max(*seq):
    maxv = seq[0]
    for v in seq:
        if v > maxv:
            maxv = v
    return maxv
mymax = max(1, 22, 5, 8, 5)
```

```
>>> def lj(r, epsilon = 1.0, sigma = 1.0):
...     return 4*epsilon*( (sigma/r)**6 - (sigma/r)**12 )
>>> lj(2**(1./6.))
1.0
>>> lj(2**(1./6.), 1, 1)
1.0
>>> lj(epsilon=1.0, sigma=1.0, r=2.0)
0.0615234375
```

Dokumentation, Funktionen als Werte, Rekursion

```
def print_f(f, seq):
    """Berechnet Funktion f fuer alle Elemente von seq
       und gibt sie aus."""
    for v in seq:
        print v, f(v)

def myfunc(x):
    return x*x

print_f(myfunc, range(10))
```

```
def fib(a):
    if a <= 1: return 1
    else: return fib(a-2) + fib(a-1)
```

Listen

```
>>> mylist = ["Olaf", "Axel", "Floh"]
>>> mylist[0]
'Olaf'
>>> mylist + ["Dominic"]
['Olaf', 'Axel', 'Floh', 'Dominic']
>>> mylist[1:3]
['Axel', 'Floh']
>>> mylist.append('Dominic')
>>> mylist
['Olaf', 'Axel', 'Floh', 'Dominic']
>>> del mylist[-1]
>>> mylist
['Olaf', 'Axel', 'Floh']
>>> len(mylist)
3
```

Flache und tiefe Kopien

```
>>> list1 = [ 1, 2, 3 ]
>>> list2 = list1
>>> del list1[-1]
>>> list2
[1, 2]
```

```
>>> list1 = [ 1, 2, 3 ]
>>> list2 = list1[:]
>>> del list1[-1]
>>> list2
[1, 2, 3]
```

- Eine Liste wird nicht *kopiert*
- Eine Variable speichert nur eine *Referenz* auf die Liste
- Gilt für alle Typen, außer „*basic types*“:
  `int`, `float`, `bool`, `string`, `tupel`

Tupel


```
>>> kaufen = ("Muesli", "Kaese", "Milch")
>>> print kaufen[1]
Kaese
>>> for f in kaufen[:2]: print f
Muesli
Kaese
>>> kaufen[1] = "Camembert"
TypeError: 'tuple' object does not support item assignment
>>> print k + k
('Muesli', 'Kaese', 'Milch', 'Muesli', 'Kaese', 'Milch')
```

- Ähnlich wie Listen, aber nicht veränderbar
- Wird vor allem intern verwendet:

```
>>> def f(): return 1, 2, 3
>>> f()
(1, 2, 3)
```

Assoziative Listen: Dictionaries

```
>>> de_en = { "Milch": "milk", "Mehl": "flour" }
>>> de_en
{'Mehl': 'flour', 'Milch': 'milk'}
>>> de_en["Oel"]="oil"
>>> for de in de_en: print de, "=>", de_en[de]
Mehl => flour
Milch => milk
Oel => oil
>>> for de, en in de_en.iteritems():
...     print de, "=>", en
Mehl => flour
Milch => milk
Oel => oil
>>> if "Mehl" in de_en:
...    print "Kann \"Mehl\" uebersetzen"
Kann "Mehl" uebersetzen
```

Module

```
import random
from math import sqrt, log, cos, pi
def boxmueller():
    """
liefert normalverteilte Zufallszahlen
nach dem Box-Mueller-Verfahren
    """
    r1, r2 = random.random(), random.random()
    return sqrt(-2*log(r1))*cos(2*pi*r2)
```

Neu: Erzeugen und Verwenden von Objekten

```
>>> mylist = list()
>>> mylist.append(2)
>>> mylist.append(3)
>>> mylist.append(1)
>>> mylist.sort()
>>> mylist
[1, 2, 3]
```

- `list` ist eine *Klasse*
- `mylist` ist ein *Objekt* (a.k.a. *Instanz* der Klasse)
- *Methoden* (hier z.B. `mylist.append()`) können aufgerufen werden

Die meisten Typen in Python sind Klassen!

```
>>> mydict = dict()
>>> mydict['Mehl'] = 'flour'
>>> mydict['Milch'] = 'milk'
>>> mydict.keys()
['Mehl', 'Milch']
>>> "Hallo Welt!".find("Welt")
6
```

Formatierte Ausgabe

```
>>> x, y = 1, 2
>>> 'x={} y={}'.format(x, y)
'x=1 y=2'
>>> '{0}{1}{0}'.format('abra', 'cad')
'abracadabra'
>>> '{:<30}'.format('left aligned')
'left aligned                  '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
'           centered           '
>>> '{:*^30}'.format('centered')  # use '*' as a fill char
'***********centered***********'
```

- `str.format()` ersetzt `%`!

Ein-/Ausgabe: Dateien in Python

```
eingabe = open("ein.txt")
ausgabe = open("aus.txt", "w")
nr = 0
ausgabe.write("Datei {} mit Zeilennummern\n".format(eingabe.nam
for zeile in eingabe:
    nr += 1
    ausgabe.write("{}: {}".format(nr, zeile))
ausgabe.close()
```

Teil 2: Wiederholung NumPy und Matplotlib

Numerik mit Python – numpy

- `numpy` ist ein Modul für effiziente numerische Rechnungen
- Baut auf *n*-dimensionalem Feld-Datentyp `numpy.array` auf
  - Feste Größe und Form wie ein Tupel
  - Alle Elemente vom selben (einfachen) Datentyp
  - Aber sehr schneller Zugriff
  - Viele Transformationen
- Bietet
  - mathematische Grundoperationen
  - Sortieren, Auswahl von Spalten, Zeilen usw.
  - Eigenwerte, -vektoren, Diagonalisierung
  - diskrete Fouriertransformation
  - statistische Auswertung
  - Zufallsgeneratoren
- Hilfe unter `http://docs.scipy.org`

http://www.icp.uni-stuttgart.de

array – eindimensionale Arrays

```
>>> import numpy as np
>>> print np.array([1.0, 2, 3])
array([ 1.,  2.,  3.])
>>> print np.ones(5)
array([ 1.,  1.,  1.,  1.,  1.])
>>> print np.arange(2.2, 3, 0.2, dtype=float)
array([ 2.2,  2.4,  2.6,  2.8])
```

- `np.array` erzeugt ein Array (Feld) aus einer Liste
- `np.arange` entspricht `range` für beliebige Datentypen
- `np.zeros/ones` erzeugen 1er/0er-Arrays
- `dtype` setzt den Datentyp *aller* Elemente explizit
- ansonsten der einfachste für alle Elemente passende Typ

Mehrdimensionale Arrays


```
>>> print np.array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3],
       [4, 5, 6]])
>>> print np.array([[[1,2,3], [4,5,6]], [[7,8,9], [0,1,2]]])
array([[[1, 2, 3],
        [4, 5, 6]],

       [[7, 8, 9],
        [0, 1, 2]]])
>>> print np.zeros((2, 2))
array([[ 0.,  0.],
       [ 0.,  0.]])
```

- Mehrdimensionale Arrays entsprechen verschachtelten Listen
- Alle Zeilen müssen die gleiche Länge haben
- `np.zeros/ones`: Größe als Tupel von Dimensionen

Elementzugriff und Subarrays

```
>>> a = np.array([[1,2,3,4,5,6], [7,8,9,0,1,2]])
>>> print a.shape, a[1,2], a[1]
(2, 6) 9 [7 8 9 0 1 2]
>>> print a[0,1::2]
array([2, 4, 6])
>>> print a[1:,1:]
array([[8, 9, 0, 1, 2]])
>>> print a[0, np.array([1,2,5])]
array([2, 3, 6])
```

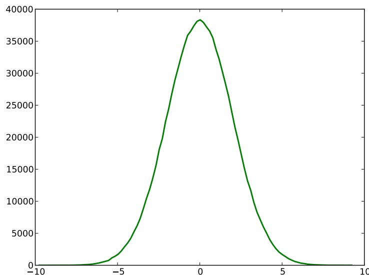
- `[]` indiziert Elemente und Zeilen usw.
- Auch Bereiche wie bei Listen
- `a.shape` ist die aktuelle Form (Länge der Dimensionen)
- `int`-Arrays, um beliebige Elemente zu selektieren

Methoden

```
>>> a = np.array([[1,2], [3,4]])
>>> a = np.concatenate((a, [[5,6]]), axis=0)
>>> print a.transpose()
[[1 3 5]
 [2 4 6]]
>>> i = np.array([[1,0],[0,1]]) # Einheitsmatrix
>>> print a*i                   # punktweises Produkt
[[1 0]
 [0 4]]
>>> print np.dot(a,i)           # echtes Matrixprodukt
[[1 2]
 [3 4]]
```

Plots mit matplotlib

```
import matplotlib.pyplot as pyplot
...
x = np.linspace(0, 2*np.pi, 1000)
y = np.sin(x)
pyplot.plot(x, y, "g", linewidth=2)
pyplot.show()
```

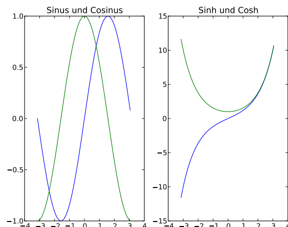


Mehrfache Graphen

```
pyplot.figure()

pyplot.subplot(121, title="Sinus und Cosinus")
pyplot.plot(x, np.sin(x))
pyplot.plot(x, np.cos(x))

pyplot.subplot(122, title="Sinh und Cosh")
pyplot.plot(x, np.sinh(x), x, np.cosh(x))
```



Teil 3: Objekte und Klassen in Python

Klassen selbst definieren

```
# define a new class
class Circle:
    """A class that describes a circle."""
    # define a method
    def create(self, r):
        self.radius = r
    def area(self):
        return 3.14159*self.radius**2

circle = Circle()   # create an instance
circle.create(1.0)  # call a method
print circle.area()
```

- Eine Methode erhält das Objekt selbst als erstes Argument (`self`)
- Ein Objekt kann Variablen speichern (`self.radius`)

Konstruktor und Destruktor

```
>>> class Circle:
...     def __init__(self, r): self.radius = r
...     def area(self): return 3.14159*self.radius**2
...     def __del__(self): print 'Kaputt!'
>>> circle = Circle(1.0)
>>> print circle.area()
3.14159
>>> del(circle)
Kaputt!
```

- Die Methode `__init__` ist der *Konstruktor*
- Definiert die Argumente beim Erstellen der Klasse
- Die Methode `__del__` ist der *Destruktor*
- Erlaubt Operationen, bevor die Klasse gelöscht wird (z.B. Schließen einer Datei)