



python<sup>TM</sup>

Einführung und Grundlagen

# Agenda

- Warum Python?
- Installation
- Allgemeine Syntax
- Datentypen: Zahlen, Bool, Strings
- Listen, Tupel, List Comprehension, Slicing, Dictionaries
- Code
  - Module, Verzweigungen, Schleifen, Funktionen

# Warum Python?

- Python ist ...
  - leicht zu erlernen
  - eine höhere Programmiersprache
  - frei verfügbar
  - durch viele verschiedene Bibliotheken erweiterbar

## Paul Graham: The Python Paradox

„I didn't mean by this that Java programmers are dumb. I meant that Python programmers are smart.“

<http://www.paulgraham.com/pypar.html>

# Installation

- Windows
  - <http://www.python.org/download/>
  - MSI Paket installieren, PATH wird automatisch gesetzt
- Linux (Ubuntu)
  - aptitude install python2.6
- Tip: ipython

# Allgemeine Syntax

- Python ist casesensitiv
- Kommentare von „#“ bis Zeilenende
- Blockbildung durch Einrücktiefe

```
if x > 2:  
    x += 3    # if-Block  
print x      # if-Block beendet
```

- Umlaute in Scripts

```
# -*- coding: utf-8 -*-
```

# Datentypen: Zahlen

- Zahlen können „ganz“, „reell“ oder „komplex“ sein
  - +, -
  - \*, /
  - \*\*
  - //

```
a = 1; b = 10
a = 1.0; b = 10.0

a = b = c = 2
a, b = 1, 2.0
a, b = b, a
```

# Datentypen: Bool

- Booleans können „True“ oder „False“ sein.
- Das Ergebnis logischer Operatoren ist ein Boolean. (==, !=, >, <, >=, <=, and, or, not)

```
a = 1.0; b = 1; c = complex(1,0)
```

```
a == b and b == c
```

```
a == b == c
```

```
d = -1
```

```
a == b > d
```

# Datentypen: Strings

- Strings werden zwischen ' ', " " oder "" "" geschrieben.

```
s1 = 'a'  
s2 = "b"  
s3 = s1 + s2 # ab  
s3 = s3*4    # abababab  
  
s4 = """laengere Nachricht, ueber  
mehrere Zeilen"""
```

# Listen, Tupel

- Eine Liste wird mit eckigen Klammern umrahmt.
- Ein Tupel wird mit runden Klammern umrahmt.
- Listen darf man verändern, Tupel nicht.

# Listen, Tupel

```
x = [1, 2, 'ab', 3, 4]
'ab' in x           # True
x[0] = 11          # [11, 2, 'ab', 3, 4]
1 not in x         # True
del x[0]           # [2, 'ab', 3, 4]

# Nicht möglich
y = (1, 2, 'ab', 'c', 3, 4)
y[0] = 11
```

# List Comprehensions

- Listen existierender Listen

```
a = [2,1,3]
```

```
b = [2*x for x in a]    # b == [4,2,6]
```

```
a = [2,1,3]
```

```
b = [2*x for x in a if x < 3]
```

```
    # b == [4,2]
```

# Slicing

- Elemente von Listen und Tupel werden mit einem Index von „0“ bis „n“ belegt.

```
x = [1, 2, 3, 'a', 'b', 'c']
```

```
x[0]    # 1
```

```
x[-1]   # c
```

```
x[1:3]  = [11, 12]
```

```
x[0:0]  = [-3, -2, -1]
```

```
x[0:0]  = [[-3, -2, -1]]
```

# Dictionaries (Hash Tables)

- Dicts. bestehen aus einem Schlüssel und einem dazugehörigen Wert.
- Dicts. sind nicht sortiert!

```
a = {"gruen": 2, "rot": 1, "gelb": 2}
a["gruen"] = 5      # Wert von "gruen"
```

```
for name, num in a.items():
    print "Name: %s Num: %s" % (name, num)
```

# Code: Module

- Bibliotheken werden mit „import“ geladen.
- Mehrere Module gleichzeitig laden durch Kommas getrennt.

```
# lädt alle Funktionen aus sys in  
# Namespace "sys"  
import sys  
  
import sys, os, re
```

# Code: Verzweigung

```
z = float(raw_input("Zahl eingeben "))
z1 = z % 2

if (z1 == 0):
    print("z ist eine gerade zahl")
else:
    print("z ist keine gerade zahl")

print z1
print z
```

# Code: for-Schleifen

```
zahlen = [1,2,3,4,5,6,7,8,9,10]  
summe = 0
```

```
for i in zahlen:  
    summe = summe + i
```

```
summe = 0  
for i in range(1,11):  
    summe = summe + i
```

# Code: while-Schleifen

```
i=1
summe = 0

while i <= 10:
    summe = summe + i
    i += 1
    print "i: ", i
else:
    print "Ende, Summe: ", summe
```

# Code: Funktionen

```
def meinesum(N=10):  
    sum = 0  
    for i in range(1,N+1):  
        sum = sum + i  
    return sum
```

```
meinesum()           # 55  
meinesum(5)          # 15  
meinesum(N=20)       # 210
```

Eric Raymond

„I was generating working code nearly as fast  
as I could type.“

<http://www.linuxjournal.com/node/3882/print>