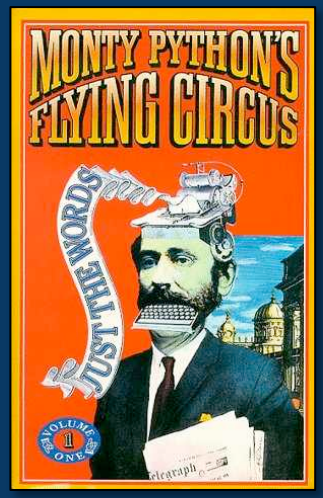
TECHNISCHE UNIVERSITÄT DRESDEN

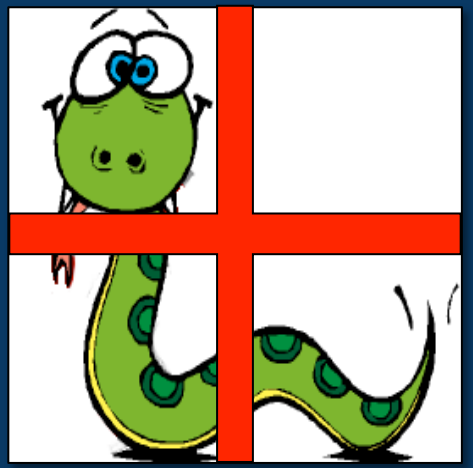
Medienzentrum – Abteilung MIT

Dipl.-Inf. Ingo Keller

Sommerkurs

Python





Sommerkurs

Python

Grundlagen

TECHNISCHE UNIVERSITÄT DRESDEN

SOMMERKURS

Referenten

- Dipl.-Ing. Carsten Knoll
  - Fakultät ET – *Institut für Regelungs- und Steuerungstheorie*
- Dipl.-Inf. Ingo Keller
  - Medienzentrum – *Medien- und Informationstechnologie*
- Dipl.-Inf. Peter Seifert
  - Medienzentrum – *Medien- und Informationstechnologie*
- Dipl.-Ing. Sebastian Voigt
  - Fakultät MW – *Institut für Verarbeitungsmaschinen und Mobile Arbeitsmaschinen*

Kursinhalte Grundlage

1. Überblick, grundlegende Sprachelemente, die wichtigsten Datentypen

2. Objektorientierung, Modulkonzept und "Perlen der Standardbibliothek"

3. Effiziente Lernmethoden

4. Arbeiten mit Dateien und Betriebsysteminteraktion

Kursinhalte Anwendungen

- Numerische Berechnungen (numpy, scipy, lineare Algebra, Interpolation,
  Statistik)
- Symbolische Berechnungen (sympy, Differenzieren, Integrieren)
- Visualisierung (Matplotlib 2D, VTK 3D)
- Webframeworks (Zope, Plone)
- Datenbank-Anbindung (relational, ZODB)
- Hardware-Ansteuerung, Schnittstellen ansprechen (ctypes, gpib, rs232)
- GUI (Qt)

Werkzeuge

- Python(x,y)
  - Softwaresuite mit allen Werkzeugen
- IDLE
  - Python Interpreter Shell
- Eclipse + PyDev
  - Integrierte Entwicklungsumgebung

PYTHON

Guido v. Rossum wollte ...

- eine neue Programmiersprache, die ...
  - Einsteigerfreundlich und leicht zu lernen ist,
  - Viele Möglichkeiten bietet ohne unübersichtlich zu werden,
  - Mehr als ein Programmierparadigma unterstützt,
  - Mit wenigen Keywords auskommt.

Everyone a programmer!

Guido v. Rossum

Python ist ...

- leicht zu lernen
  - ist meist wohl strukturiert und intuitiv
  - gut lesbar

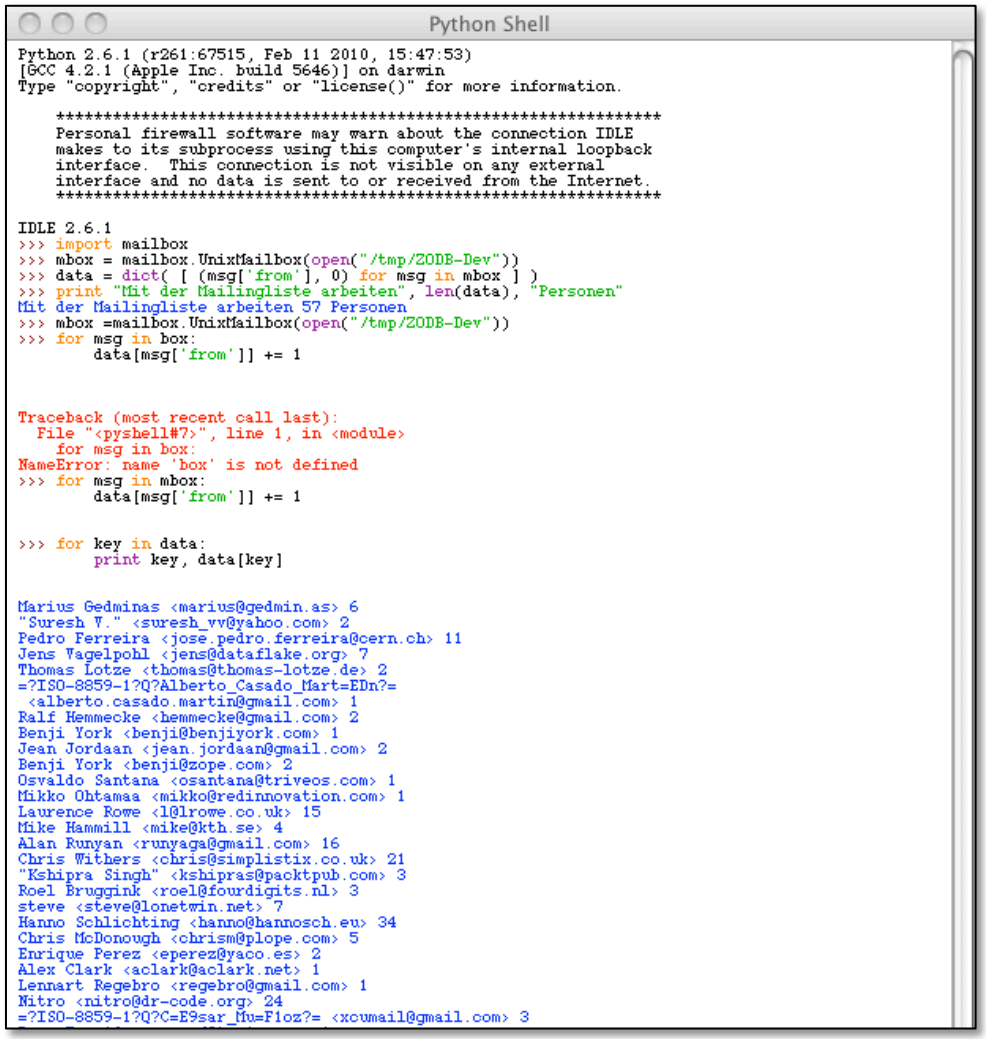
```
from zope.interface         import implements

from Products.Archetypes    import atapi

from Products.CMFCore.utils import getToolByName

from Products.ATContentTypes.content            import base
from Products.ATContentTypes.content.document   import ATDocument
from Products.ATContentTypes.content.folder     import ATFolder
from Products.ATContentTypes.content.schemata   import ATContentTypeSchema
from Products.ATContentTypes.lib.constraintypes import ConstrainTypesMixinSchema

from Products.AcTEd             import vaMessageFactory as _
from Products.AcTEd             import config

schema = ATFolder.schema.copy() + atapi.Schema(())

schema['title'].widget = atapi.StringWidget(label = _(u'Transkription'))

class AcTEd(ATFolder):
    """An Archetype for an AcTEd application"""

    schema  = schema
    default = 'global_text'
    suffix  = '_text'

    def manage_afterAdd(self, item, container = None):
        "\brief This method adds a default document to the editor folder."
        ATFolder.manage_afterAdd(self, item, container)
        temp_object = self.restrictedTraverse('portal_factory/Document/tmp_id')
        document    = temp_object.portal_factory.doCreate(temp_object, self.default)
        document.setText('')


    def _setObject(self, id, object, roles = None, user = None, set_owner = None):
        "\brief This method adds an additional document for the inserted image."
        ATFolder._setObject(self, id, object, roles, user, set_owner)

        if ( 'Image' in str(type(object)) ):
            temp_object = self.restrictedTraverse('portal_factory/Document/tmp_id')
            document    = temp_object.portal_factory.doCreate(temp_object,
                                                              id + self.suffix)


    def _delObject(self, id):
        "\brief Removes the object and the additional text document if it exists."
        ATFolder._delObject(self, id)
        textId = id + self.suffix
        if (textId in self.objectIds()) :
            ATFolder._delObject(self, textId)


    def listFolderContents(self, spec=None, contentFilter=None, suppressHiddenFiles = 0):
        "\brief Per Default only images are shown."
        if (contentFilter == None):
            contentFilter = {}
        contentFilter['portal_type'] = ['Image']
        return ATFolder.listFolderContents(self, spec, contentFilter, suppressHiddenFiles)
```

Python hat ...

- Ein Mantra (`import this`)
  - Beautiful is better than ugly
  - Explicit is better than implicit
  - Simple is better than complex
  - Flat is better than nested
  - Sparse is better than dense
  - Readability counts
  - ...

Python ist ...

- eine Interpretersprache
  - mit interaktiver Shell
  - erzeugt Python-Bytecode
  - nutzt Stackbasierte VM
    - gut dokumentiert!



Python ist ...

- eine moderne Sprache
  - Objektorientiert – kein Muss – aber konsequent
  - Skalierbar – von kleinen Skripten bis zu großen Systemen
  - OS unabhängig – Windows, *nix, OSX, BeOS, S60, u.v.m.
  - Reich an Libraries – Python Package Index (PyPI)
  - Erweiterbar – z.B. um C/C++ Code, wenns mal schnell gehen muss

Python wird genutzt von ...

- **Google Inc.**

  "Python has been an important part of Google since the beginning, and remains so as the system grows and evolves. Today dozens of Google engineers use Python, and we're looking for more people with skills in this language."

  Peter Norvig
  Director of Search Quality

- **Industrial Light & Magic**

  "Python plays a key role in our production pipeline. Without it a project the size of Star Wars: Episode II would have been very difficult to pull off. From crowd rendering to batch processing to compositing, Python binds all things together."

  Tommy Burnette
  Senior Technical Director

Python wird genutzt für ...

- Rapid Application Development (RAD)
- Skriptsammlungen
  - Paketmanagement (Gentoo)
  - Naturwissenschaft (Biopython, NumPy, PyMol)
- Anwendersysteme
  - Applikationsserver (Zope)
  - Content Management System (Plone)
  - Datenbanken (ZODB)
- Spiele
  - Server Backend Engine (EVE Online)
  - Script Engine (Civilization IV)

GRUNDLAGEN

TECHNISCHE UNIVERSITÄT DRESDEN

Python Code ausführen

- Kommandozeile
  - "Start" -> "Ausführen" -> cmd
  - C:\>python
- IDLE – Python Shell
  - "Start" -> "Python(x,y)" -> "IDLE"
- Interaktiver Modus

```
>>> <Anweisung>
<Ergebnis>
```

```
>>> type("Hello World!")
<type 'str'>
```

- Buildin Funktionen
  - `type`, `dir`, `help`

NoneType, Boolesche Werte

- None
  - Universeller False-Wert

```
>>> type(None)
<type 'NoneType'>
```

- Boolesche Werte
  - True und False

```
>>> type(True)
<type 'bool'>
```

Datentyp	False-Wert
NoneType	None
int, long	0
float	0.0
complex	0 + 0j
str	""
list	[]
tuple	()
dict	{}
set, frozenset	set(), frozenset()

TECHNISCHE UNIVERSITÄT DRESDEN

Numerische Datentypen

- Integer

```
>>> type(1)
<type 'int'>
```

- (sehr) lange Integer

```
>>> type(1L)
<type 'long'>
```

- Gleitkommazahlen

```
>>> type(1.0)
<type 'float'>
```

- Komplexe Zahlen

```
>>> type(1 + 2j)
<type 'complex'>
```

- Standardoperationen
  - Addition +
  - Subtraction -
  - Division /
  - Integerdivision //
  - Multiplikation *
  - Expotentieren **
  - Modulo %

- Build-in Funktionen
  - round, pow, etc.

```
>>> dir(__builtins__)
```

- Modul `math`

```
>>> help(math)
```

Operationen

Operation	Abkürzung
`x = x + y`	`x += y`
`x = x - y`	`x -= y`
`x = x * y`	`x *= y`
`x = x / y`	`x /= y`
`x = x % y`	`x %= y`
`x = +x`	
`x = -x`	
`x = x**y`	`x **= y`
`x = x//y`	`x //= y`

Vergleichsoperation
`x == y`
`x != y`
`x < y`
`x <= y`
`x > y`
`x >= y`

Strings

- String

```
str1 = "abc"
str2 = 'abc'
str3 = """
       abc
       """
str4 = ("abc"
        "def")
```

Escape-Sequenz	Erklärung
\a	erzeugt Signalton
\b	Backspace
\f	Seitenvorschub
\n	Linefeed
\r	Carriage Return
\t	horizonal Tab
\v	vertikal Tab
\"	Escaping "
\'	Escaping '
\\	Escaping \

Formatierung mit Strings

Syntax

```
"...%n...%m..." % (Wert1, Wert2)
```

Beispiele

```
>>> a = 'H'
>>> b = 'ello World'
>>> "%c%s" % (a,b)
'Hello World'
```

Erweiterung

```
>>> '%10.2f' % 3.1415
'      3.14'
```

Format	Erklärung
d, i	Integer mit Vorzeichen
f	Float (Dezimaldarstellung)
g, G	Float (wiss. mit Exponent)
u	Integer ohne Vorzeichen
x	Hexzahl ohne Vorzeichen
o	Oktalzahl ohne Vorzeichen
e, E	Float (Exponentendarst.)
c	Zeichen (Länge 1)
s, r	String
%	Prozentzeichen

Tupel

- Syntax

```
(Wert_1, ..., Wert_n)
```

- kann:
  - nicht verändert werden
  - beliebige Elemente enthalten

- Funktionen
  - `index`

- Beispiele

```
>>> t = (1,2,3)
>>> z = ('a', 'z', 1, False)
>>> t.index(2)
1
>>> z.index('a')
0
```

Listen

- Syntax

```
[Wert_1, ..., Wert_n]
```

- kann:
  - verändert werden
  - beliebige Elemente enthalten
  - sortiert werden
- Funktionen
  - `append, count, index, insert, remove, reverse, sort`

- Beispiele

```
>>> l = [1, 2, 3]
>>> m = ['a', 'z', 1, False]
>>> l.append(4)
>>> del l[0]
>>> print(l)
[2,3,4]
>>> l.reverse()
>>> print(l)
[4,3,2]
```

Sequentielle Datentypen

Operation	Erklärung
s in x	prüft, ob s in x ist
s not in x	prüft, ob s nicht in x ist
x + y	Verkettung von x und y
x * n	Verkettung, so das n Kopien von x existieren
x[n]	liefert das n-te Element von x
x[n:m]	liefert eine Teilsequenz von n bis m
x[n:m:k]	liefert eine Teilsequenz von n bis m, aber nur jedes k-te Element wird berücksichtigt
len(x)	liefert die Anzahl von Elementen
min(x)	liefert das kleinste Element
max(n)	liefert das größte Element

Dictionaries

- Syntax

```
{ Key_1: Value1,
  Key_2: Value2,
  ... }
```

```
>>>type({})
type<dict>
```

- assoziatives Array
  - Schüssel-Wert-Paare
  - Schlüssel müssen unveränderlich sein

- Beispiele

```
>>> d = {
...    "Sachsen" : "Dresden",
...    "Thüringen" : "Erfurt",
...    "Berlin" : "Berlin"
...    }
>>> e = {1:'a', 2:'b', 3:'c'}
>>> e[1]
'a'
>>> d.get("Sachsen)
'Dresden'
>>> d.get("Blub")
-> no Entry -> no Output
>>> d["Blub"]
-> KeyValueError
```

Sets

- Syntax

```
set([Element1,...,Elementn])
```

```
>>>type({})
type<set>
```

- kann:
  - jedes Element nur einmal enthalten
  - nicht sortiert werden
  - verändert werden

- `frozenset`
  - ist unveränderlich

- Beispiele

```
>>> engineers = Set(['John','Jane',
                     'Jack','Janice'])
>>> programmers = Set(['Jack','Sam',
                   'Susan','Janice'])
>>> managers = Set(['Jane','Jack',
                    'Susan','Zack'])

>>> union = engineers | programmers
>>> intersect = engineers & managers
>>> difference = managers - enineers

>>> engineers.add('Marvin')
>>> print engineers
Set(['Jane','Marvin','Janice','John',
'Jack'])
```

if-elif-else - Verzweigung

Syntax

```
if <Bedingung>:
    ...
elif <Bedingung>:
    ...
else:
    ...
```

Abkürzung

```
y = (1 if x == 'a' else 2)
```

Beispiele

```
>>> x = 1

>>> if x == 1:
...     print ("x=1")
x=1

>>> x = 4
>>> if x == 1:
...     print ("x=1")
... elif x == 3:
...     print ("x=3")
... else:
...     print ("x != 1 und x != 3")
x != 1 und x != 3
```

for - Schleife

Syntax

```
for <Variable> in <object>:
    ...
```

range Funktion

```
range(start, stop, step)
```

```
>>> range(1,10,2)
[1,3,5,7,9]
```

Beispiele

```
>>> x = ['a', 'b', 'c']
>>> count = 0
>>> for a in x:
...     count += 1
...     print(a)
...
a
b
c
>>> print(count)
3
```

while - Schleife

- Syntax

```
while <Bedingung>:
    ...
else:
    ...
```

- `break` Statement

```
while <Bedingung1>:
    if <Bedingung2>:
        break
```

- `continue` Statement

```
while <Bedingung1>:
    if <Bedingung2>:
        continue
```

- Beispiele

```
>>> x = 4
>>> while x > 1:
...     print(x)
...     x = x - 1
... else:
...     print ("x = 1")
4
3
2
x = 1
```

Funktionen

- Syntax

```
def <Name>(P1, ..., Pn):
    ...
    return <Resultat>
```

- optionale Parameter

```
def test(param = 'Hallo'):
    print (param)
```

- Beispiele

```
>>> def printSum(a,b):
...     print(a+b)
>>> printSum(1,2)
3
```

```
>>> def printMult(a,b,c,d=0):
...     return (a*b*c)+c
>>> print(mult(2,4,3))
24
>>> print(mult(a=2,c=4,b=3))
24
```