Programmieren mit Python

Sebastian Henneberg

17. Juni 2009

Inhaltsverzeichnis

Warum Python?

- frei verfügbar
- kostenlos
- portabel
- mächtig
- leicht
- syntaktischer Zucker
- kein kompilieren, kein linken
- automatische Speicherbereinigung

Woher kommt Python?

- **Erfinder:** Guido van Rossum
- **Jahr:** 1991
- arbeitet seit 2005 bei Google

Python soll...

- ... eine einfache, intuitive Sprache sein, die Konkurrenten in Mächtigkeit in nichts nachsteht
- ... Open Source sein, sodass jeder bei der Entwicklung helfen kann
- ... Quelltext haben, der genauso einfach zu lesen ist wie reines Englisch
- ... für tägliche Aufgaben geeignet sein und kurze Entwicklungszeiten ermöglichen

Wer inspiriert wen?

- Python wurde beeinflusst durch:
  - ABC
  - ALGOL 68
  - C
  - Haskell
  - Icon
  - Lisp
  - Modula-3
  - Perl
  - Java
- Python hat beeinflusst:
  Boo, Groovy, Ruby, Cobra, D, Dao

Python ist...

++ wiederverwendbar

++ produktivitätssteigernd

+ leicht wartbar

Python ist...

++ wiederverwendbar

++ produktivitätssteigernd

\+ leicht wartbar

\- - effizient

Python ist...

++ wiederverwendbar

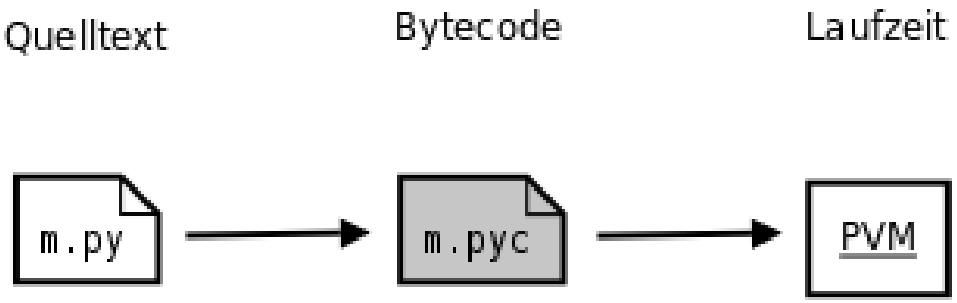
++ produktivitätssteigernd

\+ leicht wartbar

\- - effizient

+++++ Freude beim Entwickeln!

Ausführung



Wie entwickelt man?

- interaktiv mit der ’Python-Shell’

  ```
  >>>
  ```

- Skripte:
  - ```
    #!/usr/bin/python
    ''' ... '''

    #!/usr/bin/env python
    ''' ... '''
    ```
  - mit beliebigen Editor
- Programme:
  - mit Entwicklungsumgebung (Komodo, NetBeans, Eclipse mit PyDev, IDLE, ...)
  - mit beliebigen Editor
    ```

Philosophie

```
>>> import this
The Zen of Python...

>>>
```

Paradigma & Typsystem

- Python ist multiparadigmatisch
  - funtionale Skriptsprache?
  - objektorientierte Programmiersprache?
- Typsystem: duck typing
  - 'When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.'
    James Whitcomb Riley

Syntax

- minimalistisch
- sehr gut lesbar
- verzichtet aufs historische Semikolon
- Blöcke werden von Begrezungszeichen befreit (Klammern/Schlüsselwörter)

$\Rightarrow$ Programmtext muss eingerückt werden

Anwendungsbereiche

- Web-Programmierung
- XML-Verarbeitung und Web-Services
- Datenbanken
- Netzwerkprogrammierung
- GUI-Programmierung
- Parser
- Wissenschaftliches Rechnen

Kommentare

```
# this comment is just in the current line

""" this is doc-string
runs over multiple
lines :-) """

''' This module provides access to some objects used or maintained by the
interpreter and to functions that interact strongly with the interpreter.

Dynamic objects:

argv -- command line arguments; argv[0] is the script pathname if known
path -- module search path; path[0] is the script directory, else ''
... '''
```

Zahlen

```
''' Ganzzahlen '''
i1 = 1234
i2 = -42
i3 = 100000000000000000000L

''' Fliesskommazahlen (double in C) '''
f1 = 1.23
f2 = .5
f3 = 2.71e-10
f4 = 4E321

''' oktale und hexadezimale Literale '''
o1 = 0177
h1 = 0x9ff
h2 = 0XFF

''' komplexe Zahlen '''
c1 = 2+5j
c2 = 2.0+5.0j
c3 = 5J
```

Operatoren

```
lambda args: expression    # anonyme Funktion
x or y                     # logisches 'oder'
x and y                    # logisches 'und'
x > y,x < y                # Vergleichoperatoren...
x >= y, x <= y
x == y
x != y, x <> y
x | y                      # bitweise 'oder'
x & y                      # bitweose 'und'
x ^ y                      # bitweise 'entweder-oder' (XOR)
~x                         # bitweise Komplement
x >> y, x << y             # Bitshift
```

Mathematische Operationen

```
x + y                    # simple Arithmetik
x - y
x * y
x / y
x // y                   # restlose Division
x % y                    # Divisionsrest (modulo)
x ** y                   # Exponentiation
```

Strings

```
s1 = ''

s2 = "spam's"

s3 = r'C:\new\text.doc'

s4 = u'foobar'

s5 = '1\tfoo\n2\tbar\n'
```

Strings

```
s1 = ''

s2 = "spam's"

s3 = r'C:\new\text.doc'

s4 = u'foobar'

s5 = '1\tfoo\n2\tbar\n'

print "I'm using \"quotes\""
print 'I\'m using "quotes"'

print """Hier sind ein 'paar' "Anfuehrungszeichen" '''"""
print '''Hier sind ein 'paar' "Anfuehrungszeichen" """'''
```

Strings

```
s1 = ''

s2 = "spam's"

s3 = r'C:\new\text.doc'

s4 = u'foobar'

s5 = '1\tfoo\n2\tbar\n'

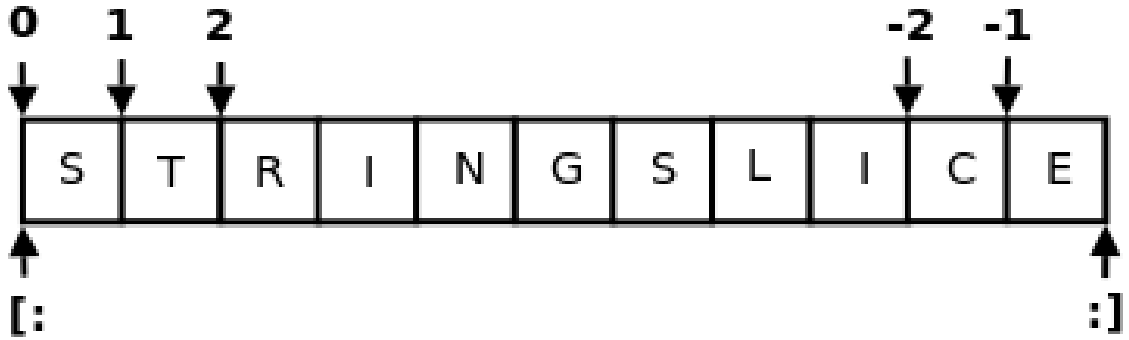
print "I'm using \"quotes\""
print 'I\'m using "quotes"'

print """Hier sind ein 'paar' "Anfuehrungszeichen" '''"""
print '''Hier sind ein 'paar' "Anfuehrungszeichen" """'''

s = "Hallo %s!" % "Publikum"
print "%d mal %d ergibt %d" % (2, 3, 6)

>>>
```

Strings und Slicing




```
s = "Hallo Welt"
s1 = s[:]
s2 = s[:5]
s3 = s[-4:]
s4 = s[0:10:1]
s5 = s[::1]
s6 = s[::-2]

>>>
```

Listen

```
L1 = []
L2 = [0,1,2,3,4]
L3 = [0,'abc',2,'def']
L4 = ['abc', ['def', 'ghi']]
L5 = [i, j, k, l]
```

```
import java.util.LinkedList;

List L2 = new LinkedList();
L2.add(0);
L2.add(1);
L2.add(2);
L2.add(3);
// ...
```

Listen

```
L1 = []
L2 = [0,1,2,3,4]
L3 = [0,'abc',2,'def']
L4 = ['abc', ['def', 'ghi']]
L5 = [i, j, k, l]
```

```
L2[i]
L2[2]
L3[i][j]
L2[i:j]
len(L4)
```

```
import java.util.LinkedList;

List L2 = new LinkedList();
L2.add(0);
L2.add(1);
L2.add(2);
L2.add(3);
// ...
```

```
L2.get(i);
L2.get(2);
L3.get(i).get(j);
```

```
L4.size();
```

Listen II

```
L1 + L2
L2 * 3

2 in L3
L2.append(5)
L2.insert(2, 1.5)
L2.extend([6,7,8])
L2.sort()
L2.index(4)
L2.reverse()

L2.remove(1)
del L2[k]
del L2[i:j]
L2.pop()
L2[i:j] = []

L2[i] = 1
L2[i:j] = [9,10,11]
```

Dictionaries

```
D1 = {}
D2 = {'foo': 2, 'bar' 3}
D3 = {'food': {'apple': 1, 'egg': 2}}
D2['bar']
D3['food']['apple']
D2.has_key('foo'), 'foo' in D2
D2.keys()
D2.values()
D2.copy()
D2.get(key, default)
D2.update(D1)

len(D1)
D2[key] = 42
del D2[key]
D4 = dict(zip(keys, values))
```

Listen und Dictionaries

```
D1 = {'a': 1, 'b': 2, 'c': 3}
D2 = dict(a=1, b=2, c=3)
L1 = dictionary.items()
L2 = [['a', 1], ['b', 2], ['c', 3]]
D3 = dict(L2)
D4 = dict(dict_as_list, d=4, e=5)
```

Tupel

```
t1 = ()
t2 = (0,)
t3 = (0, 'foo', 42, 1.23)
t4 = 0, 'foo', 42, 1.23
t5 = ('foo', ('bar', 'bla'))

>>> t3 == t4
True
```

Tupel

```
t1 = ()
t2 = (0,)
t3 = (0, 'foo', 42, 1.23)
t4 = 0, 'foo', 42, 1.23
t5 = ('foo', ('bar', 'bla'))

>>> t3 == t4
True

t1[i]
t5[i][j]
t3[i:j]
len(t1)
```

Tupel

```
t1 = ()
t2 = (0,)
t3 = (0, 'foo', 42, 1.23)
t4 = 0, 'foo', 42, 1.23
t5 = ('foo', ('bar', 'bla'))

>>> t3 == t4
True

t1[i]
t5[i][j]
t3[i:j]
len(t1)

t1 + t1
t3 * 2
42 in t4
```

Mengen

```
s1 = set('foobar')
s2 = s1.intersection('abc')
s3 = s2.union(('x','y'))
s4 = s3.difference({'b':1, 'y':2})
s5 = s4.symmetric_difference("xz")

>>>
```

Zusammenfassung Datentypen

Objekttyp	Kategorie	Veränderlich?
Zahlen	Numerisch	Nein
Strings	Sequenz	Nein
Listen	Sequenz	Ja
Dictionary	Abbildung	Ja
Tupel	Sequenz	Nein
Mengen	Sequenz	Ja

List Comprehensions

```
Prelude>  [y | x<-[0..9], let y=x^2]
[0,1,4,9,16,25,36,49,64,81]
```

List Comprehensions

```
Prelude>  [y | x<-[0..9], let y=x^2]
[0,1,4,9,16,25,36,49,64,81]
```

```
>>> [ x**2 for x in range(10) ]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehensions

```
Prelude>  [y | x<-[0..9], let y=x^2]
[0,1,4,9,16,25,36,49,64,81]
```

```
>>> [ x**2 for x in range(10) ]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
Prelude>  [ x+y | x<-[0..2], y<-[10,20,30] ]
[10,20,30,11,21,31,12,22,32]
```

List Comprehensions

```
Prelude>  [y | x<-[0..9], let y=x^2]
[0,1,4,9,16,25,36,49,64,81]
```

```
>>> [ x**2 for x in range(10) ]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
Prelude>  [ x+y | x<-[0..2], y<-[10,20,30] ]
[10,20,30,11,21,31,12,22,32]
```

```
>>> [x+y for x in [0,1,2] for y in [10,20,30]]
[10, 20, 30, 11, 21, 31, 12, 22, 32]
```

List Comprehensions

```
>>> freshfruit = ['  banana', '  loganberry ', 'passion fruit  ']
>>> [weapon.strip() for weapon in freshfruit]
['banana', 'loganberry', 'passion fruit']
```

Matrix transponieren:

```
>>> mat = [
...         [1, 2, 3],
...         [4, 5, 6],
...         [7, 8, 9],
...       ]
>>> print [[row[i] for row in mat] for i in [0, 1, 2]]
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

if-Anweisung

```
if_stmt ::=  "if" expression ":" suite
             ( "elif" expression ":" suite )*
             ["else" ":" suite]
```

if-Anweisung

```
if_stmt ::=  "if" expression ":" suite
             ( "elif" expression ":" suite )*
             ["else" ":" suite]
```

```
if my_bool:
    print 'Test bestanden'

if my_bool:
    print 'Test bestanden'
else:
    print 'Test nicht bestanden'

if my_bool:
    print 'Test bestanden'
elif my_bool2:
    print 'Test2 bestanden'
else:
    print 'Keinen Test bestanden'

>>>
```

while-Schleife

```
while_stmt ::=  "while" expression ":" suite
                ["else" ":" suite]
```

while-Schleife

```
while_stmt ::=  "while" expression ":" suite
                ["else" ":" suite]
```

```
while not seq_sorted:
    ''' do bubblesort '''

while x:
    ''' do something '''
    if y: break
    if z: continue
else:
    ''' do something else '''
```

for-Schleifen

```
for_stmt ::=  "for" target_list "in" expression_list ":" suite
              ["else" ":" suite]
```

for-Schleifen

```
for_stmt ::=  "for" target_list "in" expression_list ":" suite
              ["else" ":" suite]
```

```
for x in seq:
    x.doSomething()

for x in [2,3,5,7,11,13,17,19,23]:
    ''' do something '''
    if y: break
    if z: continue
else:
    ''' do something else '''

for x in range(10): print x

for x in {'foo':1,'bar':2}: print x

for x in range(len(seq)): #...
```

def-Statement

```
func_stmt ::=  "def" func_name "(" [arg1 ("," argn)*] "):" suite
               [("return" suite | "yield" suite)]
```

```
def hello()
    print 'Hello'

def hello(name)
    print 'Hello, ' + name

def fakultaet(x):
    if x > 1:
        return x * fakultaet(x - 1)
    else:
        return 1
```

Funktionen

```
def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a
```

Funktionen

```
def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a

def even(n):
    return range(0, n, 2)
```

Funktionen

```
def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a

def even(n):
    return range(0, n, 2)

def even(n):
    for i in range(0, n, 2):
        yield i
```

Funktionen

```
def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a

def even(n):
    return range(0, n, 2)

def even(n):
    for i in range(0, n, 2):
        yield i
```

```
>>> g = even(10)
>>> g
<generator object even at 0xb7cc3c5c>
>>> g.next()
0
>>> g.next()
2
''' ... '''
>>> g.next()
8
>>> g.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

lambda-Funktionen

```
f = lambda x, y, z: x + y + z
f(1, 2, 3)

key = 'mul'
print {'add': (lambda x, y: x + y),
       'mul': (lambda x, y: x * y)}[key](2, 3)

(lambda x: (lambda y: x + y))(98)(2)

def action():
    return (lambda x: x.replace('\n\n', '\n' + '-' * 20 + '\n'))
f = action()
f(text)
```

Parameter

```
def func(value)
def func(name=value)
def func(*name)
def func(**name)
```

Parameter

```
def func(value)
def func(name=value)
def func(*name)
def func(**name)

def f(a, b=2, c=3):
    print a, b, c

f(1)
f(a=1)
f(1, 4)
f(1, 4, 5)
f(1, c=6)
```

Parameter

```
def func(value)
def func(name=value)
def func(*name)
def func(**name)

def f(a, b=2, c=3):
    print a, b, c

f(1)
f(a=1)
f(1, 4)
f(1, 4, 5)
f(1, c=6)

def g(*args):
    print args

g()
g(1)
g(1, 2, 3, 4)
```

Parameter II

```
def h(**kargs):
    print kargs

h()
h(1)
h(a=1, b=2)
```

Parameter II

```
def h(**kargs):
    print kargs

h()
h(1)
h(a=1, b=2)

def i(a, *args, **kargs):
    print a, args, kargs
```

Parameter II

```
def h(**kargs):
    print kargs

h()
h(1)
h(a=1, b=2)

def i(a, *args, **kargs):
    print a, args, kargs

def my_min(*args):
    res = args[0]
    for arg in args[1:]:
        if arg < res:
            res = arg
    return res
```

Closures

```
def closure():

    container=[0]

    def inc():
        container[0]+=1

    def get():
        return container[0]

    return inc, get

>>>
```

Closures

```
def closure():

    container=[0]

    def inc():
        container[0]+=1

    def get():
        return container[0]

    return inc, get

>>>

>>> i,g=closure()
>>> g()
0
>>> i()
>>> i()
>>> print g()
2
```

Decorator

```
def decorator(f):
    def _inner(*args, **kwargs):
        print "Decorated!"
        return f(*args, **kwargs)

    return _inner

@decorator
def some_func(a, b):
    return a + b

print some_func(1, 2)

>>>
```

Decorator

```
def decorator(f):
    def _inner(*args, **kwargs):
        print "Decorated!"
        return f(*args, **kwargs)

    return _inner

@decorator
def some_func(a, b):
    return a + b

print some_func(1, 2)
```

```
>>>
```

```
>>> print some_func(1, 2)
Decorated!
3
```

class-Statement

```
classdef    ::=  "class" classname [inheritance] ":" suite
inheritance ::=  "(" [expression_list] ")"
classname   ::=  identifier
```

class-Statement

```
classdef    ::=  "class" classname [inheritance] ":" suite
inheritance ::=  "(" [expression_list] ")"
classname   ::=  identifier
```

```
class MyClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'Hello world'

class Square:
    def __init__(self, x):
        self.x = x
    def getArea(self):
        return self.x ** 2
```

Mehrfachvererbung

```
class Greeter:
    def greet(self): return 'Hello'

class SurnameGreeter:
    def __init__(self, surname): self.setSurname(surname)
    def setSurname(self, surname): self.surname = surname
    def getSurname(self): return self.surname
    def greet(self): print 'Hello Mr/Ms ' + self.getSurname()

class NameGreeter:
    def __init__(self, name): self.setName(name)
    def setName(self, name): self.name = name
    def getName(self): return self.name
    def greet(self): print 'Hello ' + self.getName()

class Greeter2(SurnameGreeter, NameGreeter):
    def __init__(self, name, surname):
        SurnameGreeter.__init__(self, surname)
        NameGreeter.__init__(self, name)
    def greet(self):
        print 'Hello ' + self.getName() + ' ' + self.getSurname()

g = Greeter2("Sebastian", "Henneberg")
g.greet()


>>>
```

Operatorenüberladung

```
class A:
    def __init__(self, value):
        self.data = value
    def __str__(self)
        return '<class A instance, data %s>' % self.data

>>>
```

Operatorenüberladung

```
class A:
    def __init__(self, value):
        self.data = value
    def __str__(self)
        return '<class A instance, data %s>' % self.data

>>>

class B(A):
    def __add__(self, other):
        return B(self.data + other)
    def __mul__(self, other):
        return B(self.data * other)

>>>
```

Operatorenüberladung II

```
def __init__(self[, ...])
def __del__(self)
def __add__(self, other)
def __or__(self, other)
def __repr__(self)
def __str__(self)
def __call__(self[, ...])
def __hash__(self)
def __getattr__(self, name)
def __setattr__(self, name, value)
def __getitem__(self, name)
def __setitem__(self, name, value)
def __len__(self)
def __cmp__(self, other)
def __lt__(self, other)
def __eq__(self, other)
def __radd__(self, other)
def __iadd__(self, other)
def __iter__(self)
```

Gültigkeit und Sichtbarkeit

```
X = 11

class C:
    X = 22
    def f(self):
        X = 33
        self.X = 44

def g():
    global X
    X = 55

def h():
    X = 66

def i():
    print X
```

Module

my_math.py:

```
print "Hello World"

X = 23

class Square:
    def __init__(self, x):
        self.x = x
    def getArea(self):
        return self.x ** 2
```

Module

my_math.py:

```
print "Hello World"

X = 23

class Square:
    def __init__(self, x):
        self.x = x
    def getArea(self):
        return self.x ** 2

import my_math
square = my_math.Square(5)
square.getArea()
my_math.X
```

Module

my_math.py:

```
print "Hello World"

X = 23

class Square:
    def __init__(self, x):
        self.x = x
    def getArea(self):
        return self.x ** 2

import my_math
square = my_math.Square(5)
square.getArea()
my_math.X

from my_math import Square, X as a
Square(5)
square.getArea()

>>>
```

Module II

Prüft ob ein Modul importiert wurde oder direkt ausgeführt wird:

```
''' ..... '''

if __name__ == "__main__":
    print "I was not imported"
else:
    print "I was imported"
```

$\Rightarrow$ nützlich um Unit-Tests in Module einzubetten

Module II

Prüft ob ein Modul importiert wurde oder direkt ausgeführt wird:

```
''' ..... '''

if __name__ == "__main__":
    print "I was not imported"
else:
    print "I was imported"
```

$\Rightarrow$ nützlich um Unit-Tests in Module einzubetten

- **__init__.py** ist nötig, damit ein Verzeichnis als Modul erkannt wird
- __init__.py wird beim einbinden von Modulen, die sich tiefer im Modulbaum befinden, ausgeführt

```
>>>
```

Methode austauschen

```
class Class:
    def method(self):
        print 'Hey a method'

instance = Class()
instance.method()

def new_method(self):
    print 'New method wins!'

Class.method = new_method
instance.method()

>>>
```

benutzdefinierte Funktion aufrufen

```
def key_1_pressed():
    print 'Key 1 Pressed'

def key_2_pressed():
    print 'Key 2 Pressed'

def key_3_pressed():
    print 'Key 3 Pressed'

def unknown_key_pressed():
    print 'Unknown Key Pressed'
```

benutzdefinierte Funktion aufrufen

```
def key_1_pressed():
    print 'Key 1 Pressed'

def key_2_pressed():
    print 'Key 2 Pressed'

def key_3_pressed():
    print 'Key 3 Pressed'

def unknown_key_pressed():
    print 'Unknown Key Pressed'
```

```
keycode = 2

if keycode == 1:
    key_1_pressed()
elif keycode == 2:
    key_2_pressed()
elif number == 3:
    key_3_pressed()
else:
    unknown_key_pressed()
# prints 'Key 2 Pressed'
```

benutzdefinierte Funktion aufrufen

```
def key_1_pressed():
    print 'Key 1 Pressed'

def key_2_pressed():
    print 'Key 2 Pressed'

def key_3_pressed():
    print 'Key 3 Pressed'

def unknown_key_pressed():
    print 'Unknown Key Pressed'
```

```
keycode = 2

if keycode == 1:
    key_1_pressed()
elif keycode == 2:
    key_2_pressed()
elif number == 3:
    key_3_pressed()
else:
    unknown_key_pressed()
# prints 'Key 2 Pressed'
```

```
functions = {1: key_1_pressed, 2: key_2_pressed, 3: key_3_pressed}
functions.get(keycode, unknown_key_pressed)()
# prints 'Key 2 Pressed'
```

Dateien

```
>>> f = open('/tmp/workfile', 'r+')
>>> f.write('0123456789abcdef')
>>> f.seek(5)      # Go to the 6th byte in the file
>>> f.read(1)
'5'
>>> f.seek(-3, 2) # Go to the 3rd byte before the end
>>> f.read(1)
'd'
>>> f.close()
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: I/O operation on closed file
```

Literatur

Für den Einstieg:

- **Learning Python** *Mark Lutz & David Ascher*
  O'Reilly, 2. Auflage Dezember 2003
- **Einführung in Python** *deutsche Übersetzung*
  O'Reilly, 2. Auflage August 2007
- **Dive Into Python** *Mark Pilgrim*
  Apress, 2. Auflage August 2004
- **Python** *Peter Kaiser, Johannes Ernesti*
  Galileo Press, Januar 2008

Weiterführende Literatur sortiert nach Schwerpunkt und Landessprache findet man unter:
`http://wiki.python.org/moin/PythonBooks`

Vielen Dank
für Ihre Aufmerksamkeit.