Python by Example

Thorsten Hillebrand

HAW Hamburg

09. 06. 2008

Motivation

- Wer setzt Python ein?
  - Als Scriptsprache
    - OpenOffice.org
    - Blender
    - Maya
    - Gimp
  - realisierte Softwareprojekte
    - Zope Application Server
    - NASA
    - Google setzt Python umfassend ein
    - YouTube ist nahezu vollständig in Python geschrieben
    - Industrial Light and Magic
    - offizieller BitTorrent-Client
    - CCP Games
    - EVE-Online
- Editoren und IDEs
  - PyDev http://wiki.python.org/moin/PyDev
  - eric http://www.die-offenbachs.de/eric/index.html
  - Boa Construktor http://boa-constructor.sourceforge.net

Überblick

- Erscheinungsjahr 1990
- Guido van Rossum (Python Software Foundation)
- Aktuelle Version: 2.5.2 (22. Februar 2008)
- Typisierung: stark, dynamisch (Duck Typing)
- Plattformunabhängig (Bytecode)
- Python License
- Garbage Collection
- Alles ist ein Objekt

```
values = ([number for number in range(1,1000) if number % 5 == 0 \
            or number % 3 == 0])
print reduce((lambda x,y: x+y), values)

def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        return fib(n-1) + fib(n-2)

print reduce(lambda x,y: x*y, range(1,6))
```

Besondere Merkmale

- Sehr kompakte Programmtexte
- Einrückung kennzeichnet Blöcke
- Objektorientiert, imperativ, funktional
- Keine Deklaration
- Minimalistisch
- Ganze Zahlen beliebiger Länge

```
x = y = z = 5              # 5 5 5
x, y, z = 1, 2, 3          # 1 2 3
print x < y < z            # True

a, b = 23, 42              # 23 42
a, b = b, a                # swap
print a, b                 # 42 23
print "The answer is %d" % a   # The answer is 42

print reduce(lambda x,y: x*y, range(1,99))
#9426890448883247745626185743057242473809693764078951663494238
#8777294707070023223798882976159207729119823605850588608460426
#9412647567360000000000000000000000
```

Python vs. Java

- import - import
- self - this
- None - null
- dynamisch - statisch
- kompakt - gesprächig
- Einrückung kennzeichnet Blöcke - Klammern kennenzeichnen Blöcke

Sequenzen

- Folge mehrerer Objekte
- Listen sind veränderbar
- Strings und Tupel sind nicht änderbar
  - Keine Zuweisung s[i] = ...
  - Kein Anhängen und Löschen von Elementen
  - Funktionen wie upper liefern einen neuen String zurück
- Slicing

```
a = [1, "spam", 9.0, 42]          # [1, 'spam', 9.0, 42]
print 42 in a                     # True
print "spam" not in a             # False
print a[1]                        # spam
print a[0:2]                      # [1, 'spam']
print a[2:]                       # [9.0, 42]
print a[:]                        # [1, 'spam', 9.0, 42]
print a[-1]                       # 42
print min(a), max(a), len(a)      # 1 spam 4

t = (2,4,6)
print t                           # (2, 4, 6)
x, y, z = t
print x, y, z                     # 2 4 6
```

Strings

- kurze Zeichenketten “ oder ’
- lange Zeichenketten ”” ”” ”” oder ’ ’ ’
- Unicode u’ oder u"
- Raw r“ oder r’

```
> print 'NI'*3
NININI
> print 'sp\nam'
sp
am
> print r'sp\nam'
sp\nam
> s = """Hello
. world"""
> print s
Hello
world
```

Listen

- Items beliebiger Datentypen
- List als Stack benutzen

```
stack = [3,4,5]
stack.append(6)              # [3, 4, 5, 6]
print stack.pop()            # 6
print stack                  # [3, 4, 5]
```

- List als Queue benutzen

```
queue = ["Eric", "John"]
queue.append("Terry")        # ['Eric', 'John', 'Terry']
print queue.pop(0)           # Eric
print queue                  # ['John', 'Terry']
```

- List Comprehension

```
> [str(round(355./113, i)) for i in range(1, 7)]
['3.1', '3.14', '3.142', '3.1416', '3.14159', '3.141593']
> [i for i in range(50) if i % 7 == 0]
[0, 7, 14, 21, 28, 35, 42, 49]
```

Dict

- Folge von Wertepaaren
- Schneller Zugriff über keys

```
knights = {'gallahad' : 'the pure', 'robin' : 'the brave'}

print 'gallahad' in knights       # True
print 'lancelot' not in knights   # True

knights['gallahad'] = 'The Pure'
print knights['gallahad'] # The Pure
print knights.keys()      # ['gallahad', 'robin']
print knights.values()    # ['The Pure', 'the brave']
print knights.items()     # [('gallahad', 'The Pure'), ('robin', 'the brave')]

knights2 = {'lancelot' : 'not-quite-so-brave'}
knights.update(knights2)

print 'lancelot' in knights     # True

for k, v in knights.items():
    print k, v
#gallahad The Pure
#robin the brave
#lancelot not-quite-so-brave...
```

Kontrollstrukturen

```
zahl = 42
counter = 0
while True:
    geraten = int(raw_input('Zahl: '))
    counter += 1
    if zahl == geraten:
        print 'Richtig nach %d Versuchen.' % counter
        break
    elif geraten < zahl:
        print 'Die Zahl ist etwas hoeher.'
        continue
    else:
        print 'Die Zahl ist etwas niedriger.'
        continue
    print 'Diese Stelle wird nie erreicht'
else:
    print "Schleife vollstaendig durchlaufen"
print "Programmende."
```

```
Zahl: 1
Die Zahl ist etwas hoeher.
Zahl: 50
Die Zahl ist etwas niedriger.
Zahl: 42
Richtig nach 3 Versuchen.
Programmende.
```

Schleifen mit Bedingungen

- break
- continue
- else

```
x = 2
while x <= 1000000:
    x = x*2
else:
    print "Kleinste Zweierpotenz, "
    print "die groesser als 1000000 ist: ", x
#Kleinste Zweierpotenz,
#die groesser als 1000000 ist:  1048576
```

Iteration

- range([start,] stop [,step])

```
print range(6)                  # [0, 1, 2, 3, 4, 5]
print range(0, 10, 2)           # [0, 2, 4, 6, 8]
print range(-10, 10, 2)         # [-10, -8, -6, -4, -2, 0, 2, 4, 6, 8]
```

- for-each Schleife

```
lst = [6,3,9,15,2,5,27,0] + range(13,50,7)
numbers = lst[:]
numbers.sort()
for item in reversed(numbers):
    print item,
# 48 41 34 27 27 20 15 13 9 6 5 3 2 0
```

- Dateien

```
for line in open('file.txt'):
    print line,
#Python
# Batteries included
```

Ausnahmen

```
while True:
    try:
        num = int(raw_input("Zahl: "))
    except ValueError:
        print "Das war keine Zahl"
    except:
        print "Anderer Fehler"
    else:
        print "Eingegebene Zahl: ", num
        break
    finally:
        print "wichtiger Code wird ausgefuehrt"

print num
```

```
Zahl: abc
Das war keine Zahl
wichtiger Code wird ausgefuehrt
Zahl: x
Das war keine Zahl
wichtiger Code wird ausgefuehrt
Zahl: 42
Eingegebene Zahl:  42
wichtiger Code wird ausgefuehrt
42
```

Funktionen

```
def doit(x=10, y=10, z=10):
    sum = x + y + z
    return sum, x, y, z

t = doit()
print t                         # (30, 10, 10, 10)
print type(t)                   # <type 'tuple'>
print doit(1, 2, 3)             # (6, 1, 2, 3)
print doit(z=3, y=2, x=1)       # (6, 1, 2, 3)
sum, x, y, z = doit(1, z=3)
print sum                       # 14
```

```
def keys(*args, **kwargs):
    for i in args:
        print i
    for k,v in kwargs.items():
        print k,v

keys(1, 2, "test", x=20, y=30, python="holy grail")
```

```
1
2
test
y 30
x 20
python holy grail
```

Batteries included

- Große Standardbibliothek
- Funktionale Programmierung
  - `map(function, iterable ...)`

```
import math
a = [9, 16, 25, 36]
print map(math.sqrt, a)                    # [3.0, 4.0, 5.0, 6.0]
print map(lambda x: x*2, a)                # [18, 32, 50, 72]
print map(math.pow, a, [2 for i in a]) # [81.0, 256.0, 625.0, 1296.0]
```

  - `filter(function, iterable)`

```
a = range(10)
print filter(lambda x: x % 2, a)          # [1, 3, 5, 7, 9]
```

  - `reduce(function, iterable [,init])`

```
def add(x,y):
    return x+y
a = range(10)
print reduce(add, a)                       # 45
print reduce(lambda x,y: x+y, a)           # 45
```

OO

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def say_hello(self):
        print "Hello World"

    def __add__(self, other):
        return Point(self.x+other.x, self.y+other.y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __str__(self):
        return "%d %d" % (self.x, self.y)

if __name__ == "__main__":
    p1 = Point(5, 10)
    p2 = Point(5, 10)
    p1.say_hello()          # Hello World
    print p1 == p2          # True
    p2 = Point(100, 200)
    print p1 == p2          # False
    print p1 + p2           # 105 210
```

OO - Vererbung

```
import threading, time
class Counter(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.counter = 0

    def run(self):
        while self.counter < 10:
            self.counter += 1
            time.sleep(1)
            print self.counter

c = Counter()
c.start()
print "Thread gestartet"
c.join()
print "Thread Ende"
```

```
Thread gestartet
1
.
9
10
Thread Ende
```

Fazit

- leicht zu lernen
- minimale Syntax
- große Standardbibliothek
- gute Verbreitung
- Gargabe Collection
- Open Source
- Einrückung sichert die Wartbarkeit
- Funktionen mit Defaultwerte

Literatur und Links

- Literatur
  - Python gepackt, mitp Verlag, ISBN: 3-8266-1512-3
  - Programming Python, O'Reilly Verlag, ISBN 0-596-00925-9
  - Dive Into Python
    - kostenlos unter http://www.diveintopython.org
  - Think Python
    - kostenlos unter http://www.greenteapress.com/thinkpython
- Links
  - http://docs.python.org
  - http://www.python.org
  - http://de.wikipedia.org/wiki/Python_(Programmiersprache)
  - http://www.python-forum.de
  - http://projecteuler.net
  - http://www.pythonchallenge.com

Ende

Vielen Dank für die Aufmerksamkeit!