

# Algorithmen mit Python

---

Vorbesprechung zum Proseminar  
im Sommersemester 2009

Prof. Dr. Sven Rahmann

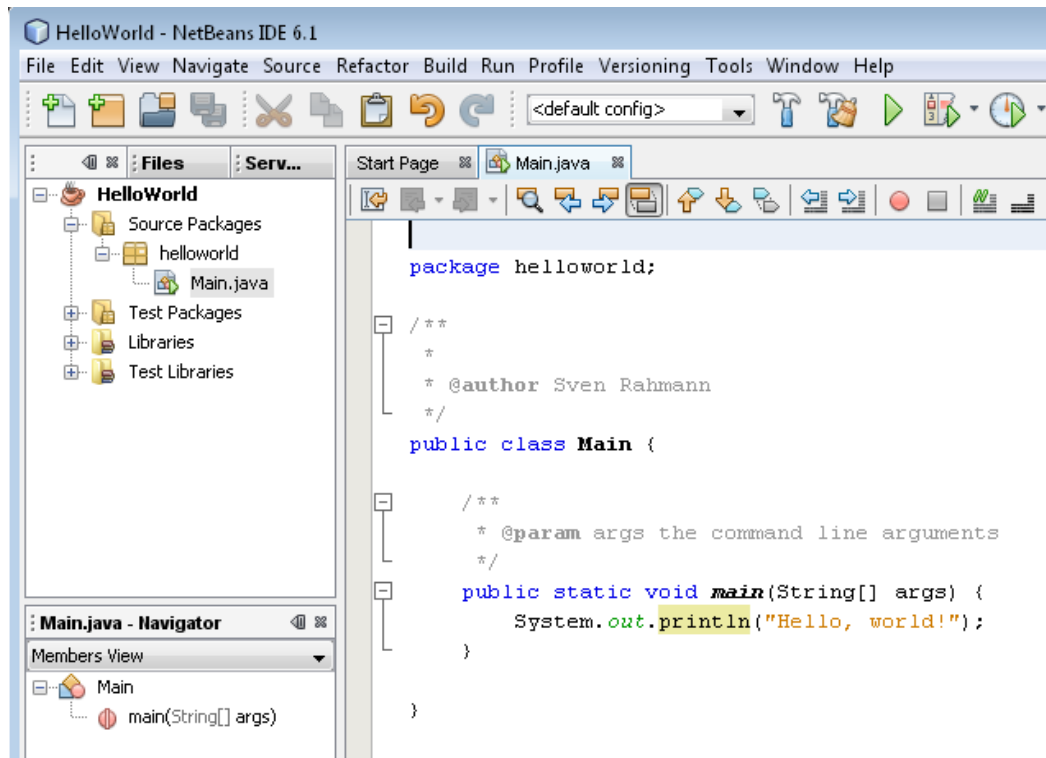


<http://www.python.org>

# Idee

Sie lernen in DAP Java und C/C++:  
80% Syntax, 20% Algorithmen-Design

Schon ein einfaches „Hello World“ braucht viel „unnützes Zeug“.  
Viele elegante Programmierkonzepte lassen sich nur mit Krücken ausdrücken.  
Vorteil: Schnelle Ausführung, viele Fehler entdeckt schon der Compiler!



# Idee

Python: interpretierte Skriptsprache.

Es gibt keinen Compiler, der sich Gedanken über Typen macht.

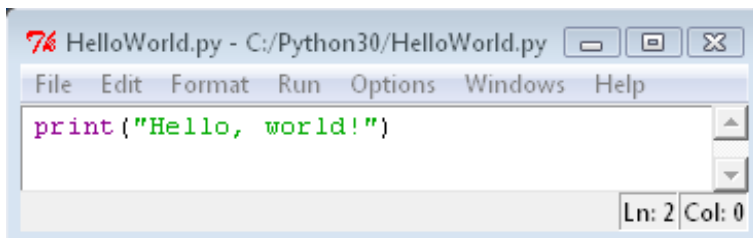
Wenn Sie etwas falsch machen, bekommen Sie eine *exception* zur Laufzeit.

Python hat

- einfache Syntax
- high-level Datentypen (Listen, Hashes/Maps)
- umfangreiche Bibliothek mit sinnvollen Klassen und Methodennamen
- Elemente aus OOP und funktionaler Programmierung

„ausführbarer Pseudocode“:

- ideal, um Algorithmen schnell zu implementieren und zu testen
- nur leichte Geschwindigkeits-Einbußen gegenüber C/Java



The image shows a screenshot of a Python IDE window titled "HelloWorld.py - C:/Python30/HelloWorld.py". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Windows", and "Help". The main text area contains the code `print("Hello, world!")`. The status bar at the bottom right shows "Ln: 2 Col: 0".

# Erster Eindruck: Fibonacci-Zahlen

Fibonacci-Folge:

- $F(0)=0$
- $F(1)=1$ ,
- $F(n) = F(n-2)+F(n-1)$  für  $n \geq 2$

Ausgabe:

```
>>>
0 0
1 1
2 1
3 2
4 3
5 5
6 8
7 13
8 21
9 34
10 55
>>> help(fib)
Help on function fib in module __main__:

fib(n)
    compute n-th Fibonacci Number F(n)

>>> |
```

```
7% fib.py - C:/Python30/fib.py
File Edit Format Run Options Windows Help

def fib(n):
    """compute n-th Fibonacci Number F(n)"""
    return fib2(n,0,1)

def fib2(n, f0, f1):
    """compute n-th Fibonacci Number F(n)
    with F(0)=f0, F(1)=f1"""
    if n==0: return f0
    return fib2(n-1, f1, f0+f1)

# short test
for n in range(11): # 0..10
    print(n, fib(n))

Ln: 2 Col: 0
```

```
>>> fib(500)
139423224561697880139724382870407283950070256587697307264108962948325571622863290691557658876222521294125
```

Zahlen in Python 3.0 können beliebig groß werden:

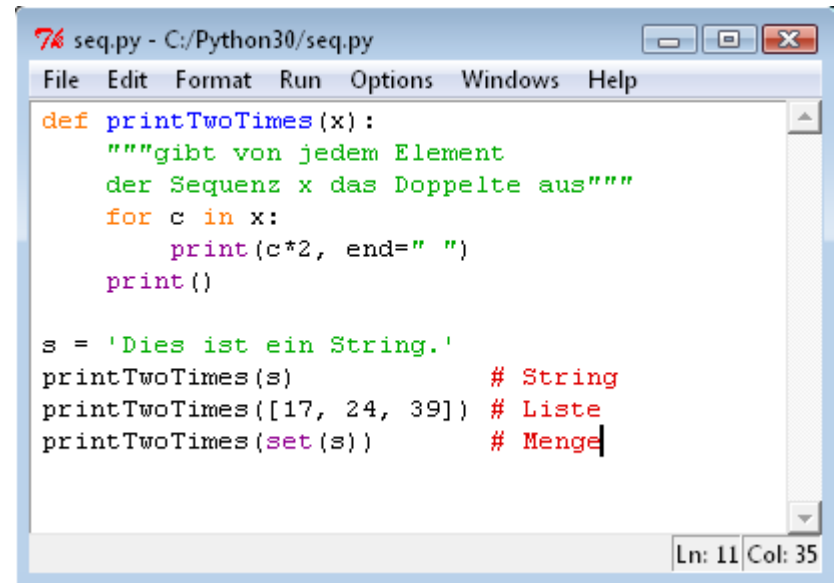
# Zweiter Eindruck: Durchmustern von Sequenzen

Quizfrage:

Was gibt das Programm rechts aus?

Beachte:

- Es kommen keine Typdeklarationen vor!
- Es gibt aber Typen zur Laufzeit.
- Blöcke werden durch Einrücken gebildet.
- optionale benannte Argumente (end=" ")
- Operatoren können für jeden Typ individuell definiert werden.



```
7% seq.py - C:/Python30/seq.py
File Edit Format Run Options Windows Help
def printTwoTimes(x):
    """gibt von jedem Element
    der Sequenz x das Doppelte aus"""
    for c in x:
        print(c*2, end=" ")
    print()

s = 'Dies ist ein String.'
printTwoTimes(s) # String
printTwoTimes([17, 24, 39]) # Liste
printTwoTimes(set(s)) # Menge

Ln: 11 Col: 35
```

Ausgabe:

```
>>>
DD ii ee ss    ii ss tt    ee ii nn    SS tt rr ii nn gg ..
34 48 78
    ee DD gg ii SS nn ss rr tt ..
>>>
```

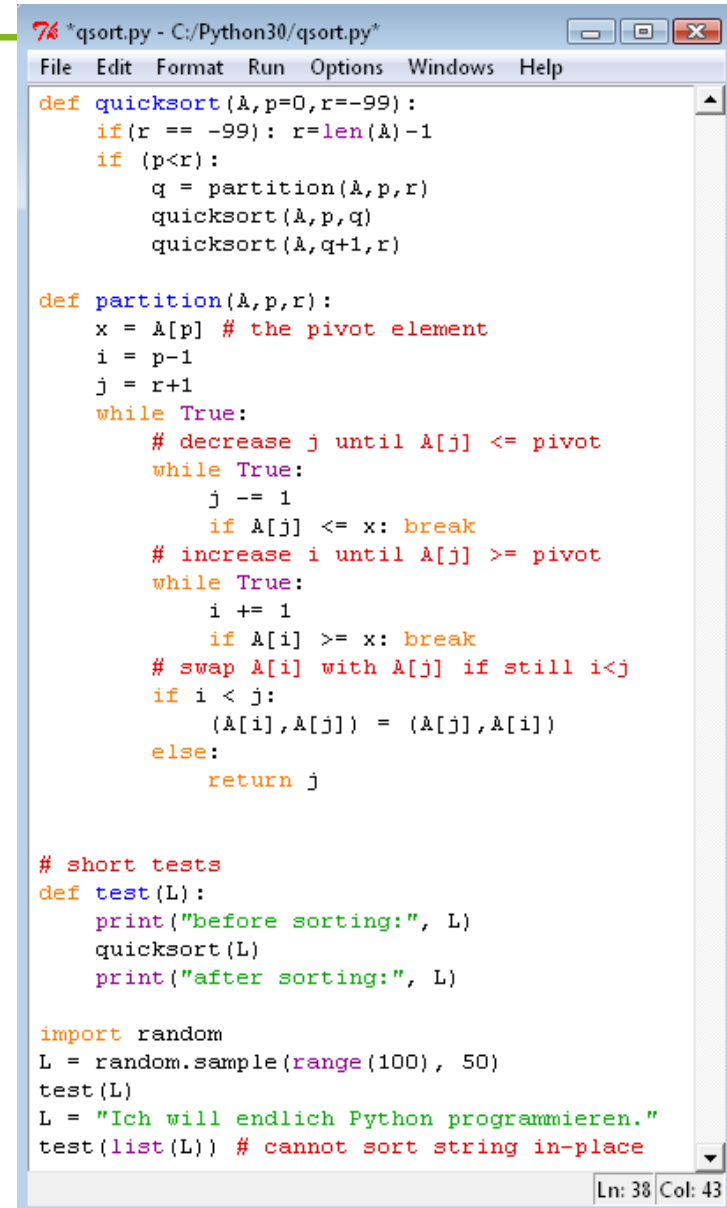
# Python = Pseudocode?

```
Quicksort(A,p,r)
    if (p < r) then
        q ← Partition(A,p,r)
        Quicksort(A,p,q)
        Quicksort(A,q+1,r)

Partition(A,p,r)
    x ← A[p]
    i ← p-1
    j ← r+1
    while (True) do
        repeat
            j ← j-1
        until (A[j] ≤ x)
        repeat
            i ← i+1
        until (A[i] ≥ x)
        if (i < j) then
            exchange A[i] ↔ A[j]
        else
            return(j)
```

Quelle:

Cormen, Leiserson, Rivest, Stein:  
Introduction to Algorithms



```
*qsort.py - C:/Python30/qsort.py*
File Edit Format Run Options Windows Help

def quicksort(A,p=0,r=-99):
    if(r == -99): r=len(A)-1
    if (p<r):
        q = partition(A,p,r)
        quicksort(A,p,q)
        quicksort(A,q+1,r)

def partition(A,p,r):
    x = A[p] # the pivot element
    i = p-1
    j = r+1
    while True:
        # decrease j until A[j] <= pivot
        while True:
            j -= 1
            if A[j] <= x: break
        # increase i until A[i] >= pivot
        while True:
            i += 1
            if A[i] >= x: break
        # swap A[i] with A[j] if still i<j
        if i < j:
            (A[i],A[j]) = (A[j],A[i])
        else:
            return j

# short tests
def test(L):
    print("before sorting:", L)
    quicksort(L)
    print("after sorting:", L)

import random
L = random.sample(range(100), 50)
test(L)
L = "Ich will endlich Python programmieren."
test(list(L)) # cannot sort string in-place

Ln: 38 Col: 43
```

# Ziele dieses Proseminars

- Python-Grundlagen lernen  
(Programmieren mit Python macht Spaß!)
- „Elegante“ Sprachkonstrukte verstehen
- Neue Algorithmen kennen lernen
- Erkennen, dass Python (fast) ausführbarer Pseudocode ist
- Algorithmen und Code präsentieren  
(funktioniert nicht mit jeder Sprache!)
- Vortragen lernen
- Ausarbeitung als Vorbereitung für Bachelor-Arbeit

# Was wird verlangt?

- Selbstständige Vorbereitung (gerne im Team!)  
Bei Fragen und Problemen stehe ich immer zur Verfügung!  
Aber: Sie bekommen den Stoff nicht häppchenweise vorgekaut.
- Gut vorbereiteter Vortrag gemäß der erlernten Präsentationstechniken  
In diesem Proseminar muss es „Code“-Folien geben!  
Der Code muss lauffähig sein und vorgeführt werden.
- Ausarbeitung in LaTeX !  
Abgabe 2 Wochen nach Vortrag
- Relativ hoher Aufwand, dafür relativ übersichtliche Algorithmen
- Belohnung:
  1. Sie beherrschen eine weitere Sprache,  
die Ihnen z.B. Rapid Prototyping erlaubt !
  2. Sie können Dokumente in LaTeX erstellen !

# Themen

## Themen: Sprachelemente vs. Algorithmen

Liste mit Literatur unter <http://www.rahmannlab.de/lehre/ps-python>

1. Philosophie von Python; Duck typing, OOP
2. Sequenzdatentypen in Python
3. Iteratoren und Generator Functions
4. Funktionale Elemente von Python
5. Heapsort und Prioritäts-Warteschlangen
6. Ordnungsstatistiken in erwarteter Linearzeit
7. Ordnungsstatistiken in garantierter Linearzeit
8. DP: Matrix-Ketten-Multiplikation; Längste gemeinsame Teilsequenz
9. DP: Optimale binäre Suchbäume
10. Huffman-Codes
11. Datenstrukturen für disjunkte Mengen (Partitionen)
12. Zahlentheoretische Algorithmen I
13. Zahlentheoretische Algorithmen II
14. Das RSA-Verschlüsselungssystem
15. Primzahltests
16. Faktorisierung von Zahlen

**Algorithmen-Themen sind noch verhandelbar!**

**Sie können selbst etwas vorschlagen.**

# TO DOs:

Sobald wie möglich:

- Python 3.0 (nicht 2.x!) installieren und Fibonacci-Code ausprobieren  
Achtung! Literatur ist meistens für 2.x  
Version 3.0 ist neu und inkompatibel zu 2.x  
Die wichtigsten Unterschiede auf: <http://www.python.org>  
Code im Seminar muss 3.0 – kompatibel sein!
- LaTeX installieren und ein Dokument „Hallo, Welt!“ erstellen
- Themenliste anschauen, Literatur scannen  
Wo liegen Ihre Interessen?  
Eigenes algorithmisches Thema?
- Link auf <http://www.rahmannlab.de/lehre/ps-python> speichern!