



Programmierung I

Einführung in Python, Beyond the Basics

G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de



Höhere Datenstrukturen



- Eines der Features, das Python so mächtig macht (VHLL)
- Zwei große wichtige Klassen von höheren DS
 - Sequenzen und Dictionaries
- Sequenzen repräsentieren geordnete Mengen von Objekte
 - Sie werden mit natürlichen Zahlen indiziert
 - Unterklassen:
 - Strings
 - Tupel
 - Listen
 - Veränderbare (*mutable*) und nicht-veränderbare (*immutable*)
- Dictionary:
 - "assoziative Datenstruktur"



Strings



- Strings sind Zeichen-Sequenzen

```
a = "Python ist toll"
```

- Zugriff auf die Zeichen erfolgt durch den **Index-Operator** []

```
b = a[3] # b = 'h'
```

- Teilstrings erhält man mit dem **Slice-Operator** [i:j]

```
c = a[3:5] # b = "hon"
```

- Strings lassen sich mit dem **+-Operator** **konkatenerieren**

```
d = a + " sagt der Professor"
```



- Strings sind in Python nicht veränderbar, wenn sie einmal festgelegt wurden

```
# Folgendes ist in Python nicht möglich  
a = "Qython ist toll"  
a[0] = 'P'
```

- Strings sind eine (von mehreren) *immutable sequence* Klassen
- Operationen auf Strings

Operationen auf Strings	
<code>s[i]</code>	Ergibt Element <i>i</i> der Sequenz <i>s</i>
<code>s[i:j]</code>	Ergibt einen Teilbereich (<i>slice</i>)
<code>len(s)</code>	Ergibt die Anzahl der Elemente in <i>s</i>
<code>min(s)</code>	Ergibt Minimum
<code>max(s)</code>	Ergibt Maximum



Listen / Arrays



- Folge **beliebiger** und **inhomogener** Werte
- Beispiele (Listenlitterale):

```
studenten = [ "Meier", "Mueller", "Schmidt" ]  
l = [ 1, 2, 3 ]  
l = [ "null", "acht", 15 ]
```

- Elementenummerierung: von 0 bis Anzahl-1 !
- Zugriff mit []

```
student_des_monats = studenten[2]  
studenten[0] = "Becker"
```

- Mit `append()` werden neue Elemente am Ende der Liste hinzugefügt

```
studenten.append("Bach")
```



Operationen auf Listen



Operationen auf Listen

<code>s[i]</code>	Ergibt Element i der Sequenz s
<code>s[i:j]</code>	Ergibt einen Teilbereich (<i>slice</i>)
<code>len(s)</code>	Ergibt die Anzahl der Elemente in s
<code>min(s)</code>	Ergibt Minimum
<code>max(s)</code>	Ergibt Maximum
<code>s.append(x)</code>	Fügt neues Element x an das Ende der Liste
<code>s.extend(l)</code>	Fügt eine neue Liste l an das Ende von s
<code>s.count(x)</code>	Zählt das Vorkommen von x in s
<code>s.index(x)</code>	Liefert kleinsten Index i mit <code>s[i] == x</code>
<code>s.insert(i,x)</code>	Fügt x am Index i ein
<code>s.remove(x)</code>	Liefert Element i und entfernt es aus der Liste
<code>s.reverse()</code>	Invertiert die Reihenfolge der Elemente
<code>s.sort([cmpfunc])</code>	Sortiert die Elemente



- Erzeugen einer Liste von Zahlen mit range :

- Syntax: `range([start], stop[, step])`

- Beispiel:

```
x = range(0,100)    # 0, ..., 99
x = range(10)      # 0, ..., 9
for i in range(0,N):
    ...
```

- Bemerkung: es gibt keinen speziellen Datentyp Array!

- Array wäre Liste mit fester Größe



Beispiel: Mischen einer Liste

- Aufgabe: zufällige Permutation von (0,...,N-1) erzeugen

```
import sys
import random
N = int( sys.argv[1] )
a = range( 0, N )
for i in range( 0, N ):
    r = random.randint(0, i)
    a[r], a[i] = a[i], a[r]
print a
```

- Beispiel



Mehrdimensionale Listen / Arrays



- Listen können als Elemente auch selbst wieder Listen enthalten

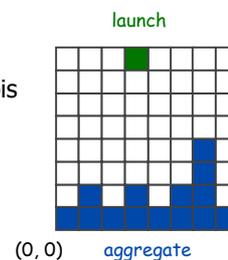
```
a = [1, "Dave", 3.14, ["Mark", 7, 9, [100, 101]], 10]
a[1] # Ergibt "Dave".
a[3][2] # Ergibt 9.
a[3][3][1] # Ergibt 101.
```



Beispiel: Diffusion Limited Aggregation



- Modell für diffusionsbegrenztetes Wachstum
- Grundlage ist die Brown'sche Molekularbewegung
- Beispiele:
 - Anlagerung von Rußteilchen an Wänden und Kaminen
 - Bildung von Fellzeichnungen bei Zebra, Tiger, Leopard,...
 - Korallenwachstum
- Idee: Monte Carlo simulation.
 1. Erzeuge einen Partikel an der launch site.
 2. Der Partikel wandert zufällig durch das 2-D Gitter bis
 - er einen anderen Partikel berührt ⇒ dann wird er dort angelagert
 - er das Gitter wieder verlässt
 3. Gehe wieder zu 1.

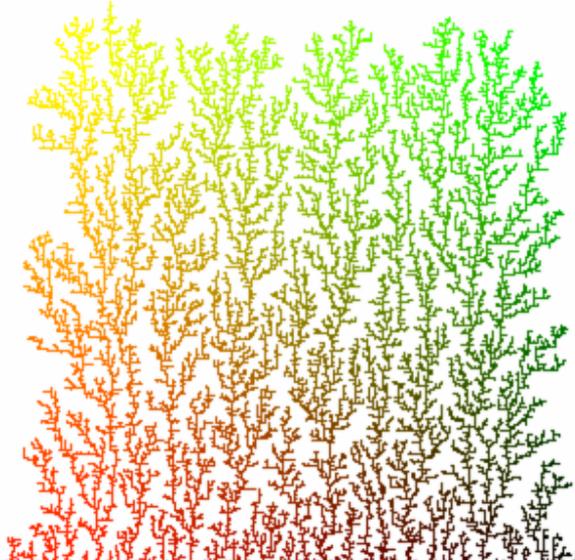


```
import Image
import random
N = 200
launch = N - 10;
dla = []
for i in range( 0, N ):
    b = []
    for j in range( 0, N ):
        b.append(0)
    dla.append(b)
for i in range(0, N):
    dla[i][0] = 1
im = Image.new("RGB", (N, N), (256, 256, 256) )
```

Erzeugen der 2D-Liste (Array)

Die unterste Reihe initialisieren

Prof. Dr. G. Zachmann Informatik 1 - WS 05/06 Einführung in Python, Teil 2 11



Prof. Dr. G. Zachmann Informatik 1 - WS 05/06 Einführung in Python, Teil 2 13



Tupel



- Tupel sind wie Listen eine geordnete Menge beliebiger Objekte
- Tupel sind wie Strings nicht veränderbar (*immutable*)
- Beispiel:

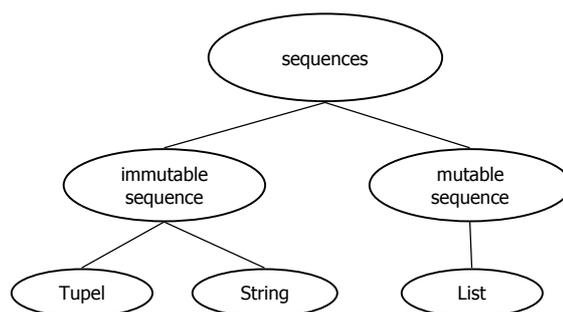
```
t1 = (12, "17", 42)
t2 = (t1, )          # Tupel mit 1 Elem
```
- Operationen: wie für Listen, ohne die verändernden Operationen
- Häufige Verwendung:
 - Swap:

```
x, y = y, x
```
 - Rückgabe mehrere Funktionswerte:

```
return (x, y, z)    # z.B. ein Punkt
```
 - Dort wo in C ein `struct` verwendet worden wäre



Hierarchie von Sequenz-Typen





Dictionary



- Ein **Dictionary** ist ein **assoziatives Array**, bei dem Objekte mit Schlüsseln (**Keys**) indiziert werden (anstatt Integer)

```
student = {  
    "Name"           : "Meier",  
    "Matrikelnummer" : "123456",  
    "Klausurnote"    : 4  
}
```

- Zugriff auf die Elemente erfolgt über den Index-Operator

```
a = student["Name"]  
student["Klausurnote"] = 1
```

- Wird oft (leider) **Hash** oder **Map** genannt



Beispiel: "Chaos Game" mit Dictionary



```
import Image  
import random  
import sys  
  
im = Image.new("RGB", (512, 512), (256, 256, 256) )  
N = int( sys.argv[1] )  
  
vertex = { "r" : (0.0,0.0),      # dictionary of coords of vertices  
           "g" : (512.0,0.0),  
           "b" : (256.0,443.4)  
         }  
x0, y0 = 0.0, 0.0           # start at "red" vertex  
  
for i in range( 0, N ):  
    r = random.random()  
    if r < 0.333:  
        x1, y1 = vertex["r"]  
    elif r < 0.6667:  
        x1, y1 = vertex["g"]  
    else:  
        x1, y1 = vertex["b"]  
  
    x0 = ( x0 + x1 ) / 2.0  
    y0 = ( y0 + y1 ) / 2.0  
    im.putpixel ( (int(x), int(y)), (int(x), int(y), 0) )  
  
im.show()
```

alter
Code

- Enthaltensein wird mit `has_key()` getestet

```

if student.has_key( "Name" )
    name = student["Name"]
else
    name = "Mustermann"

```

- Eine Schlüssel-Liste erhält man mit `keys()`

```

l = student.keys()
# liefert
# ["Name", "Matrikelnummer", "Klausurnote"]

```

- Mit `del` entfernt man Elemente

```

del student["Klausurnote"]

```

Prof. Dr. G. Zachmann Informatik 1 - WS 05/06 Einführung in Python, Teil 2 18

Operationen auf Dictionaries

Operationen auf Dictionaries	
<code>len(m)</code>	Ergibt die Anzahl der Elemente in <code>m</code>
<code>m[k]</code>	Ergibt Element von <code>m</code> mit Schlüssel <code>k</code>
<code>m[k] = x</code>	Setzt <code>m[k]</code> auf <code>x</code>
<code>del m[k]</code>	Entfernt <code>m[k]</code> aus <code>m</code>
<code>m.clear()</code>	Entfernt alle Elemente von <code>m</code>
<code>m.copy()</code>	Macht eine Kopie von <code>m</code>
<code>m.has_key(k)</code>	Ergibt <code>1</code> , falls <code>m</code> einen Schlüssel <code>k</code> enthält, sonst <code>0</code>
<code>m.items()</code>	Ergibt eine Liste von Schlüssel-Wert-Paaren
<code>m.keys()</code>	Ergibt eine Liste aller Schlüssel in <code>m</code>
<code>m.values()</code>	Ergibt eine Liste aller Objekte in <code>m</code>

Prof. Dr. G. Zachmann Informatik 1 - WS 05/06 Einführung in Python, Teil 2 19