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 konkatenieren

```
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	
`s[i]`	**Ergibt Element i der Sequenz s**
`s[i:j]`	**Ergibt einen Teilbereich (*slice*)**
`len(s)`	**Ergibt die Anzahl der Elemente in s**
`min(s)`	**Ergibt Minimum**
`max(s)`	**Ergibt Maximum**

Listen / Arrays

- Folge beliebiger und inhomogener Werte
- Beispiele (Listenliterale):

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

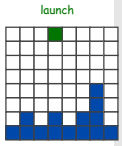
Mehrdimensionale Listen / Arrays

- 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

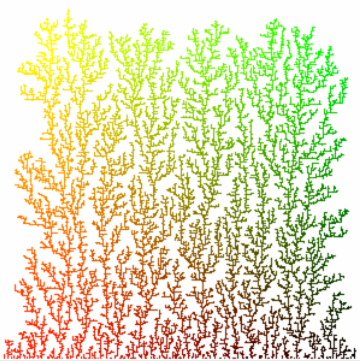
- Modell für diffusionsbegrenztes Wachstum
- Grundlage ist die Brown'sche Molekularbewegung
- Beispiele:
  - Anlagerung von Rußteilchen an Wänden und Kaminen
  - Bildung von Fellzeichnungen bei Zebra, Tiger, Leopard,...
  - Korallenwachstum
- Idee: Monte Carlo simulation.
  1. Erzeuge einen Partikel 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



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:

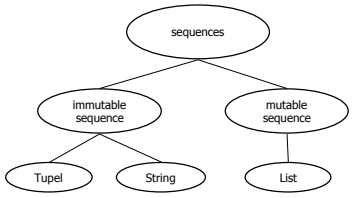
```
x, y = y,x
```

  - 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"]
```


Operationen auf Dictionaries

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