

# Fortgeschrittenes Programmieren in Python

Günter Dücke

**20.2. – 24.2.2017**

**Jeweils: 10:00 – 12:00 und 13:30 – 16:00**

**Klausurergebnisse**

**Klausur-Termin Freitag, 24.2.17, 13:00**

## **Inhalt:**

**Mo:** Schnelldurchgang Python Kurs-1, List comprehensions, Funktionsaufrufe, Reguläre Ausdrücke, C/C++ Bindings

**Di:** Multi-Threading/Processing, Python Science Pakete I: NumPy, SciPy und Matplotlib

**Mi:** Programm-Design, UML-Diagramme, Tests, Profiling, Object I/O (Serializing), SQL Database

**Do:** XML und JSON, Python Science Pakete II: Pandas, Network communication

**Fr:** Web Programming, Django Web-Framework, Klausur

Folien & Übungen als [pdf](#) und im WWW

**<http://www.etp.physik.uni-muenchen.de/kurs/Computing/python2>**

**! Achtung: Inhalt und Abfolge kann sich noch ändern !**

## Schlüsselqualifikation für Bachelor/Master

- Einwöchiger Blockkurs jeweils equivalent zu SQ 1+2, d.h.
  - 1. Kurs = Python für Physiker = SQ1+2 = 3 ECTS Punkte**
  - 2. Kurs = Fortgeschrittenes Programmieren in Python = SQ1+2 = 3 ECTS Punkte**aber nur einmal anrechenbar!
- Erfolgreiches Bestehen der Leistungskontrolle (= schriftlicher Kurztest am Ende) ist Kriterium für Punktevergabe
- Zusammen mit den Klausuraufgaben verteilen wir die Scheinformulare, die Sie ausfüllen und mit der Klausur abgeben.
- Nach Korrektur veröffentlichen wir die Liste (Matrikelnummer, bestanden/nicht bestanden) auf dieser Kursseite und leiten die Scheine ans Prüfungsamt weiter. Sie können Ihren Schein dann im Prüfungsamt abholen.

## Allgemeines

Python ist moderne Skript-Sprache

- klare Syntax
- einfache Struktur
- Features für objektorientiertes Programmieren
- mächtige Funktionen-Pakete (=Module) mitgeliefert

⇒ flache Lernkurve

- Einstieg wesentlich schneller als bei Fortran, C/C++, und sogar JAVA
- **Aber:** riesiger Funktionsumfang, Vielzahl von Modulen, Erweiterungen (Databases, XML, Networking, ...), Entwicklungsumgebungen, etc.

Ziele von Kurs-1: Python für Physiker:

- Python Syntax
- Grundlegende Funktionalität: Variablen, Funktionen, Klassen, I/O, Standard API

und damit

- Fähigkeit zum Erstellen kleiner Programme
- Anregungen zur Verwendung weiterführender Module

Ziele von Kurs-2: Fortgeschrittenes Programmieren in Python:

- Aktuelle IT Anwendungen mit Python
- Ausgewählte, nicht-triviale Beispiele zur Verwendung von Python
- Einsatz der grundlegenden Sprachelemente (Klassen, Exceptions, ...) in echten Anwendungen

## Literatur und Links

Zu Python gibt es ein großes Angebot an Lehrbüchern, sowie Online Kurse und Tutorials, FAQs und Diskussionsforen im Web. Hier eine kleine Auswahl:

**Learning Python** Mark Lutz, 5. Auflage, O'Reilly Series, 2013. Gute, ausführliche Einführung in Python.

**Python Essential Reference** David Beazley, 4. Auflage 2009, übersichtliche, komplette Referenz auf aktuellem Stand der Python Versionen.

**<http://www.python.org>** Offizielle Python Homepage. Unerschöpfliche Quelle für Downloads, Dokumentation, Tutorials, Links.

**Python 2.7 Quick Reference** Kompakte Online-Referenz zu Python

**How to Think Like a Computer Scientist (Python)** erhältlich als Buch und **on-line**. Gute konzeptionelle Einführung ins Programmieren.

**Äquivalente Versionen** für JAVA und C++.

**Python 2.7 Documentation** Dokumentation zu Python 2.7

**Python 2.7 Library Reference** Dokumentation der Python Standard Library

**Python Cookbook** Umfangreiche Sammlung von Rezepten zur Problemlösung in Python, allerdings eher auf fortgeschrittenem Niveau ...

**Software Carpentry** Handwerkszeug zum Programmieren für Naturwissenschaftler. Nicht nur Python sondern Rundumschlag von Shells, Programmieretechniken, XML, Spreadsheets, Databases, Web-Programming, u.v.m. Python als Glue-language, das die verschiedenen Bereiche verknüpft.

**Test-Driven Development for the Web, with Python** O'Reilly, Praktische Einführung in das Programmieren mit Tests

**Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython** O'Reilly, Praktische Einführung in pandas und Datenanalyse

**Das Python-Praxisbuch - Der große Profi-Leitfaden für Programmierer**, Addison-Wesley Verlag (vergriffen), aber als Google book verfügbar

**Computational Statistics in Python**, Gut gemachter Online Kurs zu Mathematik und Statistik mit Python

**Django**, Django Homepage mit ausführlicher Dokumentation



## Computing Kenntnisse für Physiker – eine Wunschliste

- **Alltag:** Textverarbeitung (Office, Latex), Präsentationen (PPT, TeX), WWW Nutzung, E-Mail  
⇒ *Selbststudium*
- **Datenanalyse/Statistik:** Tabellenkalkulation (Excel), SAS, ROOT ⇒ *(Selbst/Kurs)*
- **Mathematik/symbolische Algebra:** Maple, Mathematica, ... ⇒ *(Selbst/Kurs)*
- **Höhere Programmiersprache:** Fortran, C/C++, Java, ... ⇒ *Kurse*
- **Numerik:** Algorithmen (Fortran/C++) ⇒ *Vorlesung (Schein)*
- **Advanced concepts:** OO-Programmieren und -Design, GUI, Threads, Container ⇒ *Kurse*
- **Hardware Programmierung** ⇒ *(Kurs)*
- **High Level IT Anwendungen:** Databases, XML, Skript-Sprachen, Web-Programmierung, Grid, ...  
⇒ *Python Kurse*

## Python Features

- Python ist objekt-orientierte, plattform-unabhängige Programmiersprache. Entwickelt Ende der 90er Jahre.
- Python ist Interpreter/Skript Sprache, wie *Shell-scripts*, *Perl*, *Basic*, im Gegensatz zu Compiler-sprachen *C/C++*, *Fortran*, *Cobol*, ..., (JAVA irgendwo dazwischen)
- Traditionell werden Interpreter/Skript-Sprachen v.a. für Systemadministration oder Hilfs-macros (z.B. MS Excel/VB) verwendet.
- Python zunehmend als eigenständige Sprache für Vielzahl von Anwendungsbereichen.
- Insbesondere als **glue language** um unterschiedliche Bereiche zu verknüpfen, z.B. *Datennahme via Sensor in C/C++*, *Zugriff via Web-interfaces*, *Abspeichern in Datenbank* ⇒ Python ideal zur Verknüpfung

## Python vs C++

### Pros:

- Wohldefinierter, überschaubarer Sprachumfang
- Vielzahl von Hilfspaketen zu [I/O](#), [Networking](#), [Graphik](#), [Databases](#), ... in Standarddistribution integriert.
- Viele Features in Sprache integriert, die alltägliche Programmieraufgaben wesentlich erleichtern.
- Plattform unabhängig
- Flache Lernkurve, hohe Programmiereffizienz

### Cons:

- Performance–Nachteile
- Hardwarenahe Programmierung erschwert

Python und C++ nicht wirklich Konkurrenz sondern eher komplementär. In Python viele “einfache” Aufgaben sehr leicht zu lösen. Für zeit-/speicherkritische Probleme C/C++ um Längen besser. *Allerdings: Bei heutiger Computer-performance nur selten der Fall. Dann am besten heterogene Lösung*

# 1 Python Grundlagen

- Die ersten Schritte – Interaktiv, Skripte ausführen
- Python Operationen
- Grundlegende Datentypen und Definition von Variablen
- Strings und Container (Lists, Tuples, Dictionaries)
- Control-statements
- Funktionen
- Basic I/O
- **Klassen und Objekte**
- Vererbung
- Exception Handling
- ... und viele Aufgaben ...

siehe Kurs-1 **Python für Physiker**

## 2 Weitergehende und häufig verwendete Python Features

Python bietet etliche sehr nützliche weitergehende Features, die über das Standard-Repertoire gängiger Programmiersprachen hinausgehen und die in Python häufig verwendet werden. Wir behandeln hier kurz *Tools für Listen und Dicts, Generators, flexible Funktionsaufrufe, reguläre Ausdrücke*

### 2.1 Tools für Listen und Dicts

Aus einer Liste möchte man oft Elemente auswählen, die ein bestimmtes Kriterium erfüllen. Oder es soll eine Liste in eine andere Liste transformiert werden. Man könnte auch eine Kombination von Filtern und Transformieren anwenden. Hierzu stehen einerseits die Funktionen `filter` und `map` zu Verfügung. Andererseits kann man sog. list comprehensions verwenden.

- Als Beispiel soll eine Liste `[ 1, 2, 3, 4, 5 ]` dienen.
- Jedes Listenelement soll mit 10 multipliziert werden.
- Nur gerade Elemente sollen ausgewählt werden
- Nur gerade Elemente sollen ausgewählt und mit 10 multipliziert werden.

Zunächst `filter` und `map`:

```
>>> liste1 = [ 1, 2, 3, 4, 5 ]
```

```
>>> map(lambda x: x*10, listel)
[10, 20, 30, 40, 50]
>>> filter(lambda x: x % 2 == 0, listel)
[2, 4]
>>> map(lambda x: x*10, filter(lambda x: x % 2 == 0, listel))
[20, 40]
```

Aber man kann auch sog. **list comprehensions** verwenden:

```
>>> [element*10 for element in listel]
[10, 20, 30, 40, 50]
>>> [element for element in listel if element % 2 == 0]
[2, 4]
>>> [element*10 for element in listel if element % 2 == 0]
[20, 40]
```

List comprehensions haben folgenden allgemeine Form:

```
[ expr(element) for element in iterable if pred(element) ]
```

mit

- `expr(element)` ein beliebiger Ausdruck abhängig von `element`,
- `iterable` eine beliebige Sequenz und
- `pred(element)` eine Funktion, die `True` oder `False` liefert und von `element` abhängig ist.



Man kann auch mehrere Listen kombinieren:

```
[(x,y) for x in range(5) for y in range(5) ]  
[(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (
```

und jeweils auch noch *if* Bedingung einbauen:

```
[(x,y) for x in range(5) if x % 2 == 0 for y in range(5) if y % 2 == 1]  
[(0, 1), (0, 3), (2, 1), (2, 3), (4, 1), (4, 3)]
```

damit erhält man eine Kombination aller geraden Zahlen von 0 bis 4 und aller ungeraden Zahlen von 0 bis 4.

Es entspricht einer doppelten *for* Schleife:

```
result = []  
for x in range(5):  
    if x % 2 == 0:  
        for y in range(5):  
            if y % 2 == 1:  
                result.append((x,y))
```

Mit expliziten *for* Schleifen übersichtlicher und leichter verständlich, aber deutlich aufwendiger beim Schreiben und wesentlich langsamer bei der Ausführung.

Analog zu **list comprehensions** gibt es die **dict comprehensions**

```
sqdict = { i : i**2 for i in range(10) }  
print ( sqdict )  
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

## Listen zusammenführen mit zip

```
a=[ 1,2,3]
b=['a','b','c']
zip(a,b)
[(1, 'a'), (2, 'b'), (3, 'c')]
```

Ergibt kombinierte Liste von *tuples*

Kann leicht erweitert werden um 2 Listen in ein dict zu kombinieren:

```
d = { x[1] : x[0] for x in zip(a,b) }
print(d)
{'a': 1, 'c': 3, 'b': 2}
```

```
# Oder direkter ...
d = dict(zip(b,a))
```

## defaultdict

Im *collections* module gibt es nützliche Hilfs-Klassen zur Arbeit mit Listen und Dicts:

---

```
import sys 1
import urllib2 2
# open Kant's Text 3
# f=urllib2.urlopen("http://www-static.etp.physik.uni-muenchen.de/kurs/Computing/python/source/kant.txt") 4
f=urllib2.urlopen("https://goo.gl/rGqW4k") 5
# split into words 6
words=[] 7
for line in f: # iteriere ueber alle Zeilen 8
    words += line.split() # packe Words in list 9
# or more direct w/ double list-comprehension: 10
# words=[ word for line in f for word in line.split() ] 11
# 12
# count words v1 13
word_counts = {} 14
for word in words: 15
    if word in word_counts: 16
        word_counts[word] += 1 17
```

```
    else: 18
        word_counts[word] = 1 19
# Umstaendlich ... 20
# 21
# count words v2 22
word_counts = {} 23
for word in words: 24
    try: 25
        word_counts[word] += 1 26
    except: 27
        word_counts[word] = 1 28
# Auch umstaendlich ... 29
# 30
# count words v3 31
from collections import defaultdict 32
word_counts = defaultdict(int) 33
for word in words: 34
    word_counts[word] += 1 35
# defaultdict(int) initialisiert Eintraege beim Ansprechen automatisch auf int() = 0 36
# 37
```

---

<i># oder noch einfacher ...</i>	38
<b>from</b> collections <b>import</b> Counter	39
word_counts=Counter(words)	40
<i>#</i>	41
<i># Counter liefert eine Art dict zurueck mit das als Wert die Haeufigkeit enthaelt:</i>	42
word_counts["Vernunft"]	43
<i># und weitere Methoden ...</i>	44
word_counts.most_common(10) <i># die 10 haeufigsten ...</i>	45

---

## 2.2 Iterables und Generatoren (yield)

Listen, Dicts, etc, sind sogenannte *iterables*, d.h. alles über was man in einer `for ... in ...` Schleife drüberlaufen kann, also z.B.:

```
1 mylist = [x*x for x in range(3)]
2 for i in mylist:
3     print(i)
4
5 mylist
6 [0, 1, 4]
```

Bei Listen werden alle Elemente der Liste erzeugt und im Speicher abgelegt, das kann ggf. sehr viel sein.

Als Alternative gibt es *Generators*:

```
1 mygenerator = (x*x for x in range(3))
2 for i in mygenerator:
3     print(i)
4
5 mygenerator
6 <generator object <genexpr> at 0x7f63a2c05320>
```

Verwendung hier fast identisch, nur Erzeugung mit runden Klammern statt eckigen. Und es wird

Generator-Objekt angelegt, das man benutzen kann um **einmal** die Werte **nacheinander** abzurufen, es wird dabei jeweils nur das aktuelle Element angelegt, und am Ende ist das Generator Objekt fertig, d.h. man kann mit `for i in mygenerator:` nicht nochmal durchlaufen.

Man kann diese Funktionalität auch mittels sogenanter *Generator-Funktionen* und **yield** erreichen:

```
1 # a generator that yields items instead of returning a list
2 def firstnsq(n):
3     num = 0
4     while num < n:
5         yield num*num
6         num += 1
7
8 mygen = firstnsq(3)
9 for i in mygen:
10     print(i)
```

Das Key-word **yield** entspricht etwa dem **return** bei normalen Funktionen, nur der Ablauf ist anders:

- Aufruf `mygen = firstnsq(3)` führt nicht den Generator-Code aus sondern erzeugt nur Generator-Objekt
- Beim 1. Aufruf des Objekts in `for` Schleife werden Anweisungen wie in Funktion ausgeführt bis `yield` kommt, dann bricht Generator ab und liefert Wert zurück
- Bei weiteren Aufrufen wird die Schleife im Generator bis zum nächsten `yield` fortgeführt.



- Falls kein `yield` mehr kommt ist der Generator zu Ende (=fertig).

Statt mit `for ... in ...` Schleife kann man auch mit `next(...)` die einzelnen Werte abrufen:

```
1 mygen = firstnsq(3)
2 mygen
3 <generator object firstnsq at 0x7f63a2c05140>
4 next(mygen)
5 0
6 next(mygen)
7 0
8 ...
```

Mehr dazu in dieser [Erklärung](#).

## 2.3 Funktionsaufrufe

Eine Funktion in allgemeiner Form sieht folgendermassen aus:

```
def f(param1, param2='dummy', *list1, **dict1):
    """Eine Funktion die ihre Argumente ausgibt"""
    print "param1: ", param1
    print "param2: ", param2
    print "list1: ", list1
    print "dict1: ", dict1
    try:
        nachname1 = dict1.get('nachname')
        print nachname1
    except:
        pass

    return [param1, param2]

f('hello', 'world', 'mehr', 'Argumente', nachname = 'max', vorname='mueller')
```

Die Ausgabe dieses Beispielsprogramms sieht dann so aus:

```
param1:  hello
param2:  world
listel:  ('mehr', 'Argumente')
dict1:   {'nachname': 'max', 'vorname': 'mueller'}
max
```

Eine Funktion hat die Parameter:

- `param1` ohne ,default'-Wert,
- `param2='dummy'` mit einem ,default'-Wert,
- `*listel`, die eine un spezifizierte Anzahl von Parametern in einer Liste speichert
- `*dict1`, die eine un spezifizierte Anzahl von Parametern mit key,value in einem dictionary abspeichert. Die einzelnen Werte können mit z.B. `get` abgefragt werden.

## 2.4 Reguläre Ausdrücke

In Strings kann man meist einfache Teil-Strings suchen und evt. ersetzen. Hierzu kann man die einfachen String-Methoden `index`, `rindex`, `find`, `rfind`, `replace` und den Operator `in` verwenden.

Mit Hilfe von regulären Ausdrücken kann man in Strings nach komplizierten Mustern suchen und Teile des Strings ersetzen. Eine ausführliche Beschreibung mit Beispielen zu regulären Ausdrücken gibt es unter: [Regular Expressions](#)

Das Python Modul `re` stellt zahlreiche Funktionen zur Verwendung von regulären Ausdrücken zur Verfügung.

`re.search` im Vergleich zu `in`:

```
>>> import re
>>> input = 'Franz jagt im komplett verwahrlosten Taxi quer durch Bayern'
>>> re.search(r'Taxi', input)
<_sre.SRE_Match object at 0x7f9d50536e68>
>>> 'Taxi' in input
True
>>> re.search(r'Bus', input)
>>> 'Bus' in input
```

False

Ein String, der mit `r` eingeleitet wird, heisst ‚roher‘ String. In diesem müssen keine backslashes entwertet werden, d.h. man gibt in Folgendem Beispiel entweder `r'\bTaxi\b'` oder `'\\bTaxi\\b'` an.

Falls nach einzelnen Wörtern gesucht werden soll, zeigt `re.search` mit dem Extra Parameter `\b` seinen Vorteil:

```
>>> input1 = 'Franz jagt im komplett verwahrlosten Taxi quer durch Bayern'
>>> input2 = 'Der Taxibus ist zu spaet'
>>> 'Taxi' in input1, 'Taxi' in input2
(True, True)
>>> re.search(r'\bTaxi\b', input1), re.search(r'\bTaxi\b', input2)
(<_sre.SRE_Match object at 0x7f9d50536ed0>, None)
```

Zum Ersetzen benutzt man `re.sub`:

```
>>> input1 = 'Franz jagt im komplett verwahrlosten Taxi quer durch Bayern'
>>> output=re.sub(r'Taxi','Bus',input)
>>> output
'Franz jagt im komplett verwahrlosten Bus quer durch Bayern'
```

Mit dem `match` Objekt kann man auf Teile des String zurückgreifen, die zu regulären Ausdrücken passen. Mit

```
r' (\b\w+\b) \s+\1'
```

lässt sich nach einem doppelt vorkommendem Wort suchen:

```
>>> input = 'Franz jagt im komplett verwahrlosten Taxi quer quer durch Bayern'
>>> mo=re.search(r' (\b\w+\b) \s+\1',input)
>>> mo
<_sre.SRE_Match object at 0x7f9d50555300>
>>> mo.group(0)
'quer quer'
>>> mo.group(1)
'quer'
>>> mo.start()
42
>>> mo.span()
(42, 51)
>>> input[42: 51]
'quer quer'
```

`re.search` liefert nur das **erste** Vorkommen eines Such-Musters. Alle Vorkommen erhält man mit

`re.findall` oder `re.finditer`.

Ein schnelleren Zugriff auf Suchergebnisse vorallem bei grösseren Strings oder dem zeilenweisen Lesen/Suchen durch eine Datei erhält man mit `re.compile`. Der Such-Begriff wird einmal ‚kompiliert‘ und kann anschliessend wiederverwendet werden.:

```
>>> input3 = 'Franz jagt im komplett verwahrlosten Taxi quer durch Bayern'
>>> input4 = 'Franz jagt im komplett verwahrlosten Taxi quer quer durch Bayern'
>>> regdoub = re.compile(r'(\b\w+\b)\s+\1')
>>> regdoub
<_sre.SRE_Pattern object at 0xb75c71a0>
>>> regdoub.search(input3)
>>> regdoub.search(input4)
<_sre.SRE_Match object at 0xb75999a0>
>>> regdoub.sub(r'\1',input3)
'Franz jagt im komplett verwahrlosten Taxi quer durch Bayern'
>>> regdoub.sub(r'\1',input4)
'Franz jagt im komplett verwahrlosten Taxi quer durch Bayern'
```

Mit dem Python Modul `fnmatch` kann mit der Unix Dateiname-Suche Konvention in `strings` gesucht werden. Hierbei werden die von der bash-Kommandozeile bekannten Regeln verwendet:

- \* entspricht allem

`?` entspricht einem Buchstaben

`[seq]` entspricht einem Buchstaben in `seq`

`[!seq]` entspricht einem Buchstaben nicht in `seq`

Folgendes Beispiel zeigt alle Dateinamen im aktuellen Verzeichnis mit der Endung `.txt`:

```
import fnmatch
```

```
import os
```

```
for file in os.listdir('.'):
    if fnmatch.fnmatch(file, '*.txt'):
```

```
        print file
```



## 2.5 Aufgaben

- **Zufallszahlen erzeugen**

Mit Hilfe von list comprehensions erstellen Sie eine Liste von Zufallszahlen, z.B. einen Würfel-Wurf oder zwei-Würfel-Würfe gleichzeitig. Verwenden Sie hierzu `random.randrange(1, 7)`.

Lösung: [wuerfel.py](#)

- **Funktionsparameter**

Schreiben Sie eine Funktion, die nur ein Dictionary als Argument-Liste benutzt und werten Sie die Parameter in der Funktion aus.

- **Suchen in Strings**

Schreiben Sie ein Programm, das einen String und einen Substring als Eingabe nimmt. Der Substring soll im String gesucht werden und die Position und evt. mehrmaliges Vorkommen geprüft werden.

Lösung: [stringsearch.py](#)

- **Wörter zählen**

Das Programmbeispiel zum Zählen von Wörtern in `kant.txt` ist etwas schlampig gemacht, weil Satzzeichen nicht korrekt behandelt werden (z.B. `Vernunft` und `Vernunft,` werden getrennt gezählt). Wie lässt sich das beheben?

Lösung: [wordcount.py](#)

- **Strings kodieren**

Das Programm `text_encode.py` zeigt ein kurzes Beispiel zur Verschlüsselung von Text-Strings nach dem sog. **Caesar-Algorithmus**.

(a) Versuchen Sie die einzelnen Programmschritte nachzuvollziehen

(b) Ändern Sie den Algorithmus, so dass statt fester Verschiebung ein zufälliges Mapping der Characters gemacht wird (Funktion `random.shuffle(list)` bringt Elemente einer Liste in zufällige Reihenfolge)

### 3 Python und C/C++

In Python bestehen nicht alle Python-Module aus reinem Python-Code. Vielmehr rufen viele Module im Hintergrund Funktionen einer C- oder C++-Bibliothek auf. So gut wie jede Bibliothek ist potentiell von Python aus aufrufbar, wie z.B. GUI-Toolkits. Wie man aus Python C-Bibliotheken aufruft ist in der Python Dokumentation in folgenden Kapiteln beschrieben: [Extending and Embedding the Python Interpreter](#), [Python/C API Reference Manual](#). Hier sollen die zwei Module [ctypes](#) und [SWIG](#) besprochen werden, die Interface-Generatoren zur Verfügung stellen und das Arbeiten erheblich vereinfachen und automatisieren.

- ctypes
- SWIG

Verschiedene Kurzbeispiele sind auch auf folgender Seite diskutiert: [Calling C/C++ from python](#)

## 3.1 ctypes

Mit Hilfe des Standardmoduls `ctypes` können Funktionen aus dynamischen Bibliotheken ohne Erweiterungsmodule aufgerufen werden. Der Datentypen-Unterschied zwischen Python und C wird durch einige `c_*`-Datentypen überbrückt, z.B. `c_short`: `int/long`, `c_double`: `double`, `c_char_p`: `str`. Auch Pointer können verarbeitet werden:

```
1 from ctypes import *
2 i=c_int()
3 s=c_char_p('Hello World!')
4 type(s)
5 <class 'ctypes.c_char_p'>
6 s.value
7 'Hello World!'
8 i.value
9 0
10 i=c_int(42)
11 pi=pointer(i)
12 pi
13 <__main__.LP_c_long object at 0xb7890cd4>
14 pi.contents
15 c_long(42)
```

`ctypes` wird verwendet, um Funktionen aus dynamischen C-Bibliotheken zu laden und aufzurufen. Zunächst muss die Bibliothek geladen werden, um dann die C-Funktionen aufzurufen:

```
1 from ctypes import *
2 libc=cdll.LoadLibrary('libc.so.6')
3 libc
4 <CDLL 'libc.so.6', handle b78e0000 at b789caec>
5 dir(libc)
6 ['_FuncPtr', '__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattr__', '__getattribute__', '__getitem__', '__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_func_flags_', '_func_restype_', '_handle', '_name']
```

Die C-Funktionen `printf` oder `sleep` können dynmisch geladen werden:

```
1 libc.printf
2 <_FuncPtr object at 0xb77ec0fc>
3 libc.sleep
4 <_FuncPtr object at 0xb77ec094>
5 [s for s in dir(libc) if s[:2] != '__' and s[-2:] != '__']
6 ['_FuncPtr', '_func_flags_', '_func_restype_', '_handle', '_name', 'printf', 'sleep']
```

Argument- und Rückgabe-Werte sollten mit `argtypes` und `restypes` definiert werden:

```
1 from ctypes import *
2 libm=cdll.LoadLibrary('libm.so.6')
3 cos=libm.cos
4 cos
```

```
5 <_FuncPtr object at 0xb7712e64>
6 cos(3.14159265)
7 Traceback (most recent call last):
8   File "<stdin>", line 1, in <module>
9 ctypes.ArgumentError: argument 1: <type 'exceptions.TypeError'>: Don't know how to convert
   parameter 1
10 cos(c_double(3.14159265))
11 -1076676576
12 cos(c_double(0.0))
13 -1076662240
14 cos(c_double(0.1))
15 -1076680672
16 cos.argtypes=[c_double]
17 cos.restype=c_double
18 cos(c_double(0.0))
19 1.0
20 cos(c_double(3.14159265))
21 -1.0
```

Funktionen, die in einen Puffer schreiben, werden folgendermassen aufgerufen:

```
1 from ctypes import *
2 libc=cdll.LoadLibrary('libc.so.6')
3 gethostname=libc.gethostname
4 gethostname.argtypes=[c_char * 255, c_uint]
5 gethostname.restype=c_int
```

```
6 buf=create_string_buffer(255)
7 gethostname(buf,10)
8 -1
9 buf.value
10 'cip-ws-111'
11 gethostname(buf,30)
12 0
13 buf.value
14 'cip-ws-111'
```

## 3.2 SWIG

Um eine C oder C++-Bibliothek an Python anzubinden, verwenden wir den SWIG Interface-Generator. Diese Modul ist nicht Bestandteil der Python Standard-Installation und muss getrennt installiert werden, z.B. mit `sudo apt-get install swig` auf Ubuntu-Systemen.

### C-Klassen

Die Datei `example.c` soll in Python eingebunden werden:

```
1  /* File : example.c */
2
3  #include <time.h>
4  double My_variable = 3.0;
5
6  int fact(int n) {
7      if (n <= 1) return 1;
8      else return n*fact(n-1);
9  }
10
11 int my_mod(int x, int y) {
12     return (x%y);
13 }
14
15 char *get_time()
16 {
```



```
17     time_t ltime;  
18     time(&ltime);  
19     return ctime(&ltime);  
20 }
```

Um die Funktionen `fact`, `my_mod`, `get_time` und die Variable `My_variable` aus `example.c` zu wrappen, benötigen wir folgende Interface-Datei `example.i`:

```
1 /* example.i */  
2 %module example  
3 %{  
4 /* Put header files here or function declarations like below */  
5 extern double My_variable;  
6 extern int fact(int n);  
7 extern int my_mod(int x, int y);  
8 extern char *get_time();  
9 %}  
10  
11 extern double My_variable;  
12 extern int fact(int n);  
13 extern int my_mod(int x, int y);  
14 extern char *get_time();
```

SWIG kann für diese Interface-Datei 'glue-code' für einige Sprachen erzeugen (z.B. Perl, Java). Für Python wird SWIG folgendermassen aufgerufen und erzeugt dabei die Dateien `example.py` und

```
example_wrap.c.:
```

```
swig -python example.i
```

Nun muss `example_wrap.c.` zu einer ‚shared library‘ mit dem Namen `_example.so` kompiliert werden:

```
gcc -fPIC -c example.c example_wrap.c -I/usr/include/python2.7/  
ld -shared example.o example_wrap.o -o _example.so
```

Innerhalb von Python ist das neue Modul `example` und die entsprechenden Routinen nun folgendermassen aufrufbar:

```
1 >>> import example  
2 >>> example.fact(5)  
3 120  
4 >>> example.my_mod(7,3)  
5 1  
6 >>> example.get_time()  
7 'Sun Oct 10 19:21:11 2010\n'  
8 >>>  
9 >>> example.cvar.My_variable  
10 3.0
```

Alles, was im `%{ ... % }`-Block steht, wird unverändert in die automatisch generierte `example_wrap.c` eingefügt.

## Automatisch kompilieren

Um den gesamten Programmcode auf einmal zu kompilieren und zu einer shared library zusammenzufassen, kann folgendes Python-Programm `example_setup.py` (verwendet `distutils`) benutzt werden:

```
1 #!/usr/bin/env python
2
3 from distutils.core import setup, Extension
4
5
6 example_module = Extension('_example', sources = ['example.c', 'example_wrap.c'])
7
8 setup ( name = 'example',
9         version = '0.1',
10        author = 'John Doe',
11        description = '''An example program''',
12        ext_modules = [example_module],
13        py_modules = ['example']
14 )
```

Mit folgender Zeile erzeugt man dann das module `example`:

```
python example_setup.py build_ext --inplace
```

## C++-Klassen

C++-Klassen werden genauso wie C-Klassen eingebunden. Eine Klasse `Person` wird in Header-Datei `person.h` deklariert. Die Member-Funktionen werden in der Datei `person.cxx` deklariert. Ein Testprogramm `person_test.cxx` demonstriert die Benutzung der Klasse `Person`.

Diese C++-Programm lässt sich folgendermassen übersetzen und starten:

```
1 g++ person.cxx person_test.cxx -o person_test
2 ./person_test
```

Im SWIG-Wrapper `person.i` wird die ganze Klasse `Person` unverändert definiert. Um den Code automatisch zu kompilieren, wird die Datei `person_setup.py` verwendet.

Der SWIG glue code wird mit folgender Zeile erzeugt:

```
swig -python -c++ person.i
```

Das gesamte Modul `person` wird mit folgendem Befehl erzeugt:

```
python person_setup.py build_ext --inplace
```

Innerhalb von Python kann das Modul `Person` folgendermassen verwendet werden:

```
1 from person import Person
2 p=Person( 'Max Student' )
3 p.get_name()
4 'Max Student'
```

```
5 p.set_name( 'Maria Studentin' )
6 p.get_name()
7 'Maria Studentin'
8 p2=Person(p)
9 p2.get_name()
10 'Maria Studentin'
11 quit()
12 Person::~~Person() called
13 Person::~~Person() called
```

## 4 Threads and Multi-Processing

Python unterstützt *Multi-Threading*. Im wesentlichen heisst das, dass aus einem Python Programm heraus mehrere, “parallel laufende” Prozesse (= *Threads*) gestartet werden können.

Historie:

- In frühen Computern wurde ein Programm vom Anfang bis Ende ausgeführt, erst dann wurde das nächste Programm gestartet  $\Rightarrow$  *Batch-Processing*
- Modernere Betriebssysteme unterstützen *Multi-Tasking*, d.h. über einen Time-sharing Mechanismus wird die CPU-time auf die verschiedenen Prozesse verteilt. Auf Single-CPU Rechner läuft natürlich jeweils nur ein Prozess aber nach typisch einigen Millisekunden ist der nächste an der Reihe  $\Rightarrow$  Basis für *interaktive Multi-User Systeme*

Die einzelnen Prozesse laufen aber völlig getrennt, jedes Programm hat seinen eigenen Speicherbereich, seine eigenen Variablen, usw.

- *Multi-Threading* geht darüber hinaus. Es laufen “gleichzeitig” mehrere Versionen *eines* Programms auf einem Rechenknoten, die alle auf den *gleichen* Speicherbereich zugreifen.  
 $\Rightarrow$  Heikel, mögliche Konflikte beim Zugriff auf Speicherbereiche (Lesen Schreiben, Löschen).
- *Multi-Processing*: Programme laufen parallel, aber unabhängig, d.h. kein gemeinsamer Speicherbereich. Explizite Tools für Kommunikation nötig. Kann auch auf mehrere/viele Rechenknoten

verteilt sein.

## Neuer Trend im Programmieren: Concurrency = Multithreading

Bis vor kurzem war Multi-Threading auf spezielle Anwendungen beschränkt (GUIs, Steuerung, Server-Prozesse) und etwas für Experten.

Allerdings seit ca. 10 Jahren Zeit Trendwende bei Hardware Entwicklung:

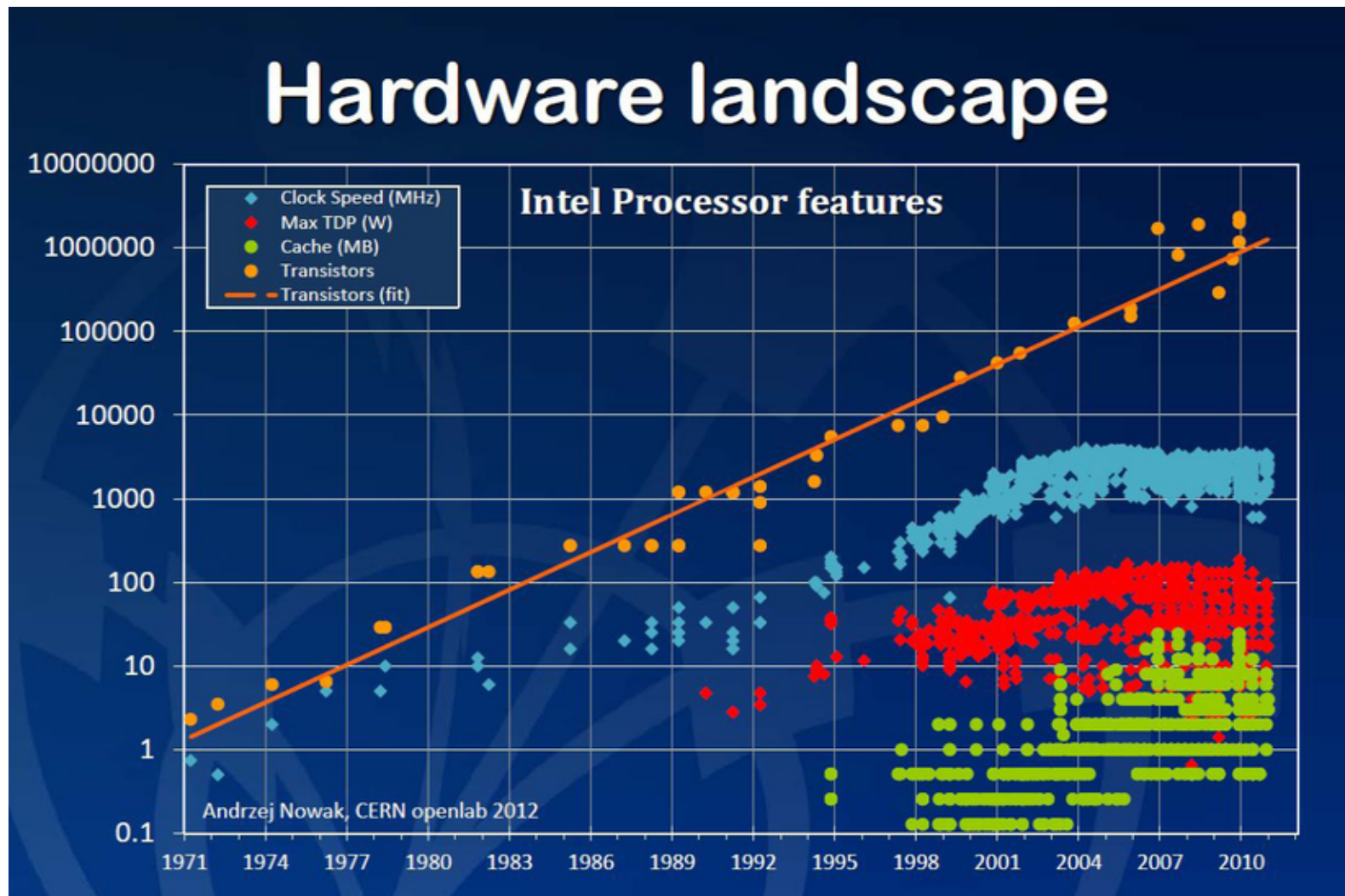
- Leistungssteigerung der Prozessoren durch steigende Taktraten und optimierte Prozessorabläufe ziemlich ausgereizt, jedenfalls stark verlangsamt.
- Stattdessen Einführung von **Multi-Core** CPUs, d.h. mehrere Prozessorkerne auf einem Chip. Zur Zeit 2-4 Cores schon gängig bei Smartphones und Laptops, 8 Cores bei Desktop Systemen und 16–80 Cores bei Servern, Trend zu mehr Cores hält an.
- Performance in Zukunft v.a. durch mehr Prozessorkerne, nur noch langsame Fortschritte bei einzelnen Prozessoren.

Verlangt zwingend **multi-threaded** Programme um Leistungssteigerungen auszunutzen.

Siehe auch Artikel von Herb Sutter: **The Free Lunch Is Over**



## Historie Transistor/CPU Entwicklung



## 4.1 Die Thread Klasse

Ableiten von Basisklasse *threading.Thread*

2 grundlegende Methoden:

- *start()* Initialisierung, **muss** vom User-Programm gerufen werden.
- *run()* Die eigentliche Methode während das Thread läuft. Wird vom Sytem gerufen, getriggert durch *start()* Ruf.

---

```
import threading 1
class BytePrinter( threading.Thread ) : 2
    """ 3
    A sample thread class 4
    """ 5
    6
    def __init__(self): 7
        """ 8
        Constructor, setting initial variables 9
        """ 10
        threading.Thread.__init__(self, name="TestThread") 11
```

---

```
def run(self):  
    """  
    overload of threading.Thread.run()  
    main control loop  
    """  
    print "%s starts" % (self.getName(),)  
    for i in range(-128,128):  
        print i  
    pass  
bp = BytePrinter()  
print "Starting BytePrinter"  
bp.start()  
print "Started BytePrinter"
```

---

Sobald `bp.start()` aufgerufen wird laufen 2 Prozesse:

- Einer fährt fort mit den Statements in Hauptprogramm nach `bp.start()`
- Der andere, das neue **bp-Thread**, startet die `bp.run()` Methode

Wann welcher Prozess zum Zuge kommt ist nicht festgelegt, sondern zufällig

Das **Thread** läuft bis

- `run()` ist beendet.
- `time.sleep(..)` oder andere Wait-Kommandos werden gerufen und unterbrechen die Ausführung

---

```
import threading 1
class BytePrinter( threading.Thread ) : 2
    """ 3
    A sample thread class 4
    """ 5
    def __init__(self): 6
        """ 7
        Constructor, setting initial variables 8
        """ 9
```

```
threading.Thread.__init__(self, name="TestThread")
10
11
def run(self):
12
    """
13
    overload of threading.Thread.run()
14
    main control loop
15
    """
16
    print "%s starts" % (self.getName(),)
17
    for i in range(-128,128):
18
        print i
19
    pass
20
bp1 = BytePrinter()
21
bp2 = BytePrinter()
22
bp3 = BytePrinter()
23
bp1.start()
24
bp2.start()
25
bp3.start()
26
```

---

Ausführungszeit für **bp-Threads** so kurz, dass i.d.R. die einzelnen Threads sequentiell laufen.

⇒ zur Demo kurzes *sleep()* in *run()* Methode.

---

```
import threading 1
import time 2
class BytePrinter( threading.Thread ) : 3
    """ 4
    A sample thread class 5
    """ 6
    def __init__(self, name="TestThread"): 7
        """ 8
        Constructor, setting initial variables 9
        """ 10
        threading.Thread.__init__(self, name=name) 11
    def run(self): 12
        """ 13
        overload of threading.thread.run() 14
        main control loop 15
        """ 16
        print "%s starts" % (self.getName(),) 17
        for i in range(-128,128): 18
            print self.getName(), i 19
            time.sleep(0.0001) # sleep 0.1 ms 20
```

<code>pass</code>	21
<code># main</code>	22
<code>bp1 = BytePrinter("Helmut")</code>	23
<code>bp2 = BytePrinter("Edi")</code>	24
<code>bp3 = BytePrinter("Angie")</code>	25
<code>bp1.start()</code>	26
<code>bp2.start()</code>	27
<code>bp3.start()</code>	28

---

Zur Unterscheidung mehrerer Threads kann man ihnen Namen geben ...

## 4.2 Synchronisation

In den bisherigen Beispielen laufen die Threads unabhängig voneinander. Komplizierter wird es wenn sie auf gemeinsame Datenbereiche zugreifen, insbesondere wenn ein Thread schreibt und ein anderes liest.

Das folgende –etwas konstruierte– Beispiel illustriert das Problem

---

```
import threading 1
import time 2
class CounterThread( threading.Thread ) : 3
    """ 4
    A sample thread class 5
    """ 6
    7
    def __init__(self, co, name="TestThread"): 8
        """ 9
        Constructor, setting initial variables 10
        """ 11
        self.co = co 12
        threading.Thread.__init__(self, name=name) 13
```



```
def run(self):
    """
    overload of threading.Thread.run()
    main control loop
    """
    print "%s starts" % (self.getName(),)
    self.co.count()
pass

class Counter(object):
    def __init__(self):
        self.num = 0
    def count(self):
        limit = self.num + 100;
        while self.num != limit:
            print self.num
            self.num += 1
            time.sleep(0.0001) # sleep 0.1 ms
        pass
# main
```

c = Counter()	34
ct1 = CounterThread(c,"ct1")	35
ct2 = CounterThread(c,"ct2")	36
ct1.start()	37
ct2.start()	38

---

Beide Threads greifen auf dasselbe *Counter-Objekt* zu, d.h. sie benutzen diesselbe **Member-variable** **self.num**.

Abhängig vom zufälligen Ablauf endet das Programm vernünftig oder geht in eine Quasi-Endlosschleife. Es ist nicht-deterministisch, obwohl keine Zufallszahlen benutzt werden ...

### Probieren Sie's aus !

Man kann im *CounterThread* Beispiel, die Variablen anders definieren/einsetzen und das spezifische Problem damit beheben, z.B. *self.num* als lokale Variable in *count()* Methode.

Das ist aber keine Lösung für den allgemeinen Fall.

Echte Abhilfe bietet in Python (u.a.) der **Locking** Mechanismus:

- `lck=threading.Lock()` Objekt wird erzeugt
- Aufruf von `lck.acquire()` bevor kritischer Bereich ausgeführt wird

- 1. Thread der `lck.acquire()` ruft läuft weiter
- Nächster Thread, der `lck.acquire()` muss warten
- bis 1. Thread `lck.release()` ruft
- usw.

Damit wird sichergestellt dass Methode `count()` nur von **einem** Thread benutzt wird, alle andren sind solange blockiert.

---

```
import threading 1
import time 2
class CounterThread( threading.Thread ) : 3
    """ 4
    A sample thread class 5
    """ 6

    def __init__(self, co, name="TestThread"): 7
        """ 8
        Constructor, setting initial variables 9
        """ 10
        """ 11
```

```
self.co = co 12
threading.Thread.__init__(self, name=name) 13
14
def run(self): 15
    """ 16
    overload of threading.Thread.run() 17
    main control loop 18
    """ 19
    print "%s starts" % (self.getName(),) 20
    self.co.count() 21
    pass 22
class Counter(object): 23
    lck = threading.Lock() 24
    def __init__(self): 25
        self.num = 0 26
    def count(self): 27
        self.lck.acquire() # get lock, waits until it's free 28
        limit = self.num + 100; 29
        while self.num != limit: 30
            print self.num 31
```

---

self.num += 1	32
time.sleep(0.0001) <i># sleep 0.1 ms</i>	33
self.lck.release() <i># release lock</i>	34
<b>pass</b>	35
<i># main</i>	36
c = Counter()	37
ct1 = CounterThread(c, "ct1")	38
ct2 = CounterThread(c, "ct2")	39
ct1.start()	40
ct2.start()	41

---

## 4.3 Python Threads und GIL

Ein fundamentales Problem bei der Verwendung von Threads in Python ist das sog. *Global Interpreter Lock (GIL)*.

- GIL ist eine Art globaler Lock des Python Interpreters, der i.W. erzwingt, dass immer nur ein Thread jeweils Python Byte-Code ausführt.
- Das verhindert die parallele Ausführung von CPU-bound Threads, u.U. laufen multi-threaded Prozesse in Python sogar langsamer als serielle Prozesse.

Beispielprogramm zur Bestimmung von Pi mit Zufallszahlen `randpi.py` braucht mit

1 Thread: 4.7 s (Aufruf `time python randpi.py 10000000 1`)

2 Threads: 6.0 s (Aufruf `time python randpi.py 5000000 2`)

- GIL ist subtiler Effekt, nicht immer offensichtlich wann Threads dadurch ausgebremst werden. Es betrifft nicht Prozesse, die IO-wait, sleep-Aufrufe, externe Funktionsaufrufe machen, in diesen Fällen gutes Scaling.

Insgesamt multi-threading Funktionalität dadurch ziemlich eingeschränkt in Python (mehr Infos in <https://wiki.python.org/moin/GlobalInterpreterLock>).

## 4.4 Python Multi-Processing

Alternative um GIL Bremse zu vermeiden ist Multi-Processing.

- Mehrere Python Prozesse laufen unabhängig voneinander
- Python Modul *multiprocessing* lässt sich ähnlich wie *threading* verwenden.
- Im Gegensatz zu *threading* kein gemeinsamer Programm/Speicherbereich, direkt konvertiertes Programm `randpiMP.py` funktioniert nicht, Ergebnis für Main-Prozess nicht sichtbar
- Explizite Kommunikation zwischen den Prozessen nötig, z.B. mit Message Queues `multiprocessing.Queue()`, wie im erweiterten Beispiel `randpiMP2.py`

Noch weitergehende Alternative ist Multi-Processing verteilt über mehrere Rechner oder ganzen Rechen-Cluster. Standard-Tool dafür ist **MPI** (*Message Parsing Interface*), das Schnittstellen für viele Programmiersprachen zur Verfügung stellt, hier ein kurzer **MPI-Überblick** aus dem Software Handwerkszeug Kurs.

## 4.5 Aufgaben

### 1. BytePrinter

- a) Testen Sie die *BytePrinter* Klassen. Lassen Sie es mehrmals hintereinander laufen um die Zufälligkeit der Abfolge zu sehen.
- b) Studieren Sie die Effekte von *sleep()*
- c) Ändern Sie die Prioritäten

### 2. Synchronisation

Gehen Sie das *CounterThread* Beispiel durch, wie kann es dazu kommen, dass ein Thread in eine Endlosschleife gerät ?

Überzeugen Sie sich von der Wirkung des *Locking* Mechanismus.

### 3. Aufzug Simulation

Nehmen Sie ein mehrstöckiges Haus an, in dem mehrere Personen arbeiten. Die Personen wechseln gelegentlich die Etagen und benutzen dazu einen 1–Personen Fahrstuhl (also noch kleiner als in Schelling 4 ...). Jede Person soll in einem eigenen Thread simuliert werden, die Arbeit zwischen der Aufzugbenutzung wird durch *sleep(some-random-time)* simuliert. Aufzugfahren ist eine separate Klasse/Methode, die Fahrzeit wird wiederum mit *sleep()* proportional zur Distanz simuliert. Mittels *Locking* kann sichergestellt werden, dass nur eine Person (=Thread) den Aufzug verwendet. Ausgabe der Aktionen auf *stdout*.



Studieren Sie den Einfluss von Zahl der Stockwerke, Zahl der Beschäftigten, u.a., auf den Durchsatz.

Einfaches Beispielprogramm: [AufzugSim.py](#) ([html src](#)).

#### 4. Multi-Threading und Multi-Processing

Testen Sie die verschiedenen Varianten des Programms zur Berechnung von Pi für Multi-Threading ([randpi.py](#)) und Multi-Processing ([randpiMP.py](#), [randpiMP2.py](#)).

## 5 Numpy, Scipy und Matplotlib

Es gibt zahlreiche wissenschaftliche Programme, Pakete und Bibliotheken, die in verschiedenen Sprachen geschrieben worden sind: Mathematica, Maple, Matlab, Root, Numerical Recipes, etc. Für große wissenschaftliche Anwendungen sind oft Ausführungsgeschwindigkeit wichtig. Es existieren zahlreiche externe Bibliotheken auf die mit einer Python API zugegriffen werden kann. Im Folgenden werden folgende Pakete besprochen:

- numpy
- scipy
- matplotlib

Zur Demonstration was damit gemacht werden kann soll ein kleines physikalisches Problem besprochen werden: Zwei Massen sind über Federn gekoppelt. Die Bewegungsgleichungen dieses Systems sollen gelöst und der Zeitverlauf der Feder-Auslenkungen graphisch dargestellt werden.

Die Lösung unter der Benutzung von `scipy` und `matplotlib` ist in [diesem Wiki](#) demonstriert.

(Source code: [two\\_springs.py](#), [two\\_springs\\_solver.py](#), [two\\_springs\\_plot\\_new.py](#))

## 5.1 Jupyter notebook

Eine tolle Bedienungsoberfläche für Python bietet das **Jupyter Notebook**, damit kann man interaktive Python Umgebung über Browser Fenster starten, die viele nützliche Features für interaktives Arbeiten bietet: history, tab completion, system interface, u.v.m.

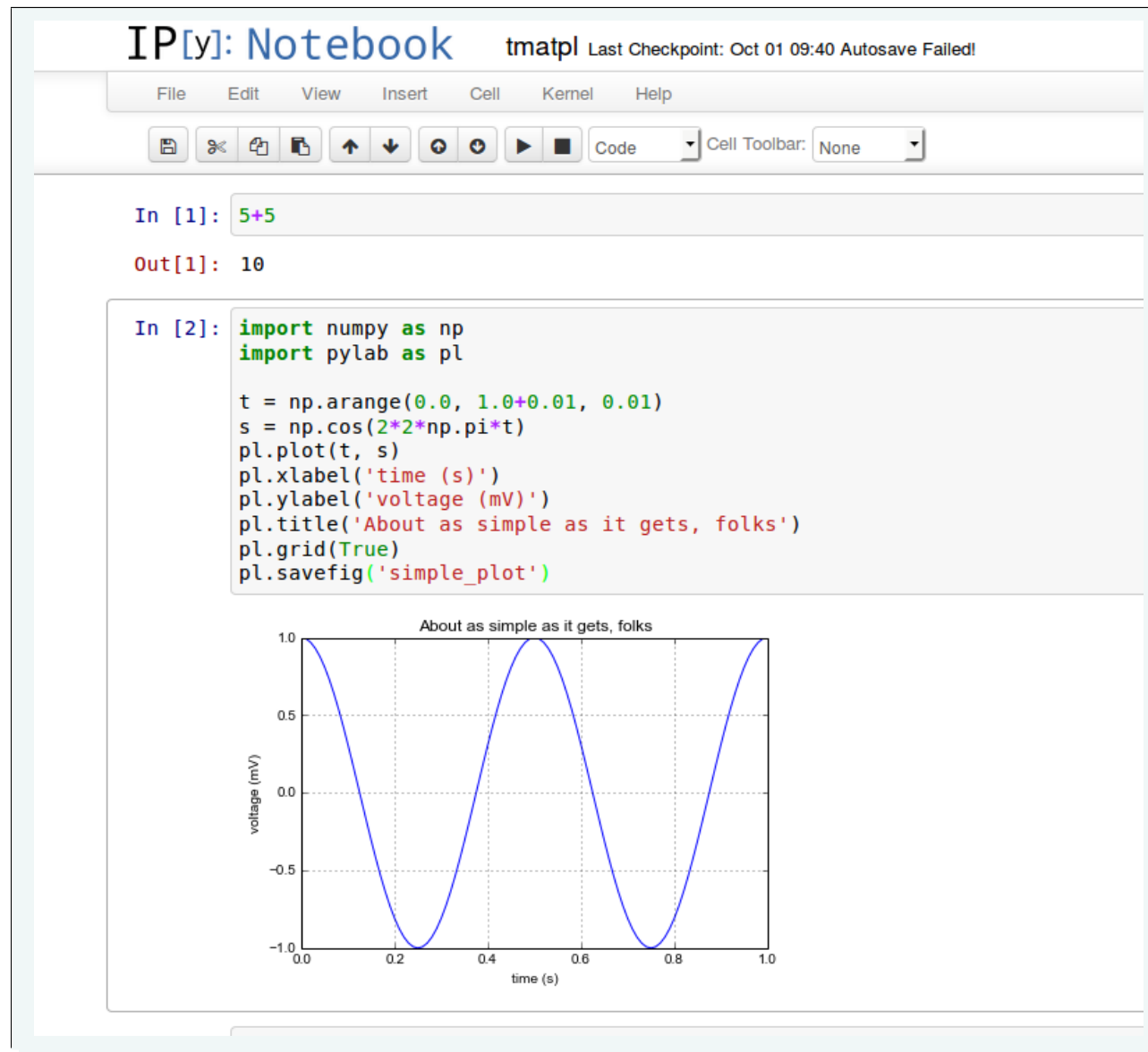
Eine besonders tolles Feature sind die eingebetteten Grafiken von Matplotlib.

- Starten mit

```
module load jupyter  
jupyter notebook --browser=firefox
```

- Beenden mit `Strg-C` im Shell Fenster bzw. `Logout` im Browser.

Weitere Infos z.B. <https://jupyter.readthedocs.io/en/latest/index.html>



## 5.2 NumPy

In der Praxis oft sehr rechenintensive numerische Operationen nötig:

- große Gleichungssysteme
- Matrizen
- Numerische Integration
- Fourier-Transformation
- Statistische Auswertungen und Berechnungen

Oft gibt es sehr ausführliche numerische Bibliotheken geschrieben in Fortran oder C/C++ (z.B. Numerical Recipes). `numpy` bietet Schnittstellen zu BLAS (Basic Linear Algebra Library) und ATLAS (Automatically Tuned Linear Algebra Software).

`numpy` bietet einen sehr effizienten Array-Datentyp `ndarray` mit Matrix-Aussehen und Funktionen aus der BLAS und ATLAS Bibliothek. Eine homogene Sammlung von `float`- und `int`-Werten wird hinter den Kulissen mit C- oder Fortran verwaltet und bietet zusätzlich die Möglichkeit *Parallelisierung oder Vektorisierung* zu nutzen anstatt einzelner Operationen in Schleife.

Folgendes Programm benötigt 15-25s

```
a = range(10000000)
b = range(10000000)
c = []
for i in range(len(a)):
    c.append(a[i] + b[i])
```

Mit `numpy` ca. 0.5s

```
import numpy as np
a = np.arange(10000000)
b = np.arange(10000000)
c = a + b
```



## Module von numpy

Interessant: `core`, `fft`, `linalg`, `lib`

```
['__config__', '_import_tools', 'add_newdocs', 'char', 'core',  
'ctypeslib', 'emath', 'fft', 'lib', 'linalg', 'ma', 'math',  
'random', 'rec', 'testing', 'version']
```

## Funktionen in `numpy.linalg` :

```
['cholesky', 'cond', 'det', 'eig', 'eigh', 'eigvals', 'eigvalsh',  
'inv', 'lstsq', 'matrix_power', 'norm', 'pinv', 'qr', 'solve',  
'svd', 'tensorinv', 'tensorsolve']
```

**Hilfetext zu** `help(numpy.linalg.qr)`

Help on function qr in module numpy.linalg.linalg:

```
qr(a, mode='full')
```

Compute QR decomposition of a matrix.

Calculate the decomposition :math:`A = Q R` where  $Q$  is orthonormal  
and  $R$  upper triangular.

.....

Der Datentyp `numpy.ndarray`, den man mit `numpy.array` erhält, ist ein homogener Datentyp, also eine Liste von `int` oder `float` mit Struktur. `ufuncs` (Universalfunktionen) operieren auf jedes Element und sind durch Fortran oder C-Implementierung sehr schnell.

```
1
2 In [23]: import numpy as np
3 In [24]: print(typeDict)
4 {0: <type 'numpy.bool_'>,
5 ....
6  'u1': <type 'numpy.uint8'>,
7  'u2': <type 'numpy.uint16'>,
8  'u4': <type 'numpy.uint32'>,
9  'u8': <type 'numpy.uint64'>,
10 'ubyte': <type 'numpy.uint8'>,
11 'uint': <type 'numpy.uint32'>,
12 'uint0': <type 'numpy.uint32'>,
13 ....
14 In [26]: a=np.array([[1,2,3,4],[4,5,6,7],[9,10,11,12]])
15
16 In [27]: a
17 Out[27]:
18 array([[ 1,  2,  3,  4],
19        [ 4,  5,  6,  7],
20        [ 9, 10, 11, 12]])
21
22 In [28]: print a
```

```
23 → print(a)
24 [[ 1  2  3  4]
25  [ 4  5  6  7]
26  [ 9 10 11 12]]
27 In [29]: type(a)
28 Out[29]: <type 'np.ndarray'>
29 In [32]: [m for m in dir(a) if not m.startswith('__')]
30 In [35]: a.shape
31 Out[35]: (3, 4)
32 In [36]: a.reshape(4,3)
33 Out[36]:
34 array([[ 1,  2,  3],
35        [ 4,  4,  5],
36        [ 6,  7,  9],
37        [10, 11, 12]])
38 In [37]: a.reshape(2,6)
39 Out[37]:
40 array([[ 1,  2,  3,  4,  4,  5],
41        [ 6,  7,  9, 10, 11, 12]])
42 In [39]: a[2,3]
43 Out[39]: 12
44 In [40]: a.T
45 Out[40]:
46 array([[ 1,  4,  9],
47        [ 2,  5, 10],
48        [ 3,  6, 11],
```

```
49         [ 4,  7, 12]])
50 In [41]: -1*a
51 Out[41]:
52 array([[ -1,  -2,  -3,  -4],
53        [ -4,  -5,  -6,  -7],
54        [ -9, -10, -11, -12]])
55 In [42]: a+a
56 Out[42]:
57 array([[ 2,  4,  6,  8],
58        [ 8, 10, 12, 14],
59        [18, 20, 22, 24]])
60 In [43]: a-a
61 Out[43]:
62 array([[0, 0, 0, 0],
63        [0, 0, 0, 0],
64        [0, 0, 0, 0]])
65 In [44]: a*a
66 Out[44]:
67 array([[ 1,  4,  9, 16],
68        [16, 25, 36, 49],
69        [81, 100, 121, 144]])
70
71 In [45]: np.dot(a,a)
```

```
72
73 ValueError                                Traceback (most recent call last)
74
```

```
75 /tmp/examples/scipy/<ipython console> in <module>()  
76  
77 ValueError: objects are not aligned  
78  
79 In [47]: np.dot(a,a.reshape(4,3))  
80 Out[47]:  
81 array([[ 67,  75,  88],  
82        [130, 147, 175],  
83        [235, 267, 320]])  
84 In [48]: np.sqrt(a)  
85 Out[48]:  
86 array([[ 1.          ,  1.41421356,  1.73205081,  2.          ],  
87        [ 2.          ,  2.23606798,  2.44948974,  2.64575131],  
88        [ 3.          ,  3.16227766,  3.31662479,  3.46410162]])  
89 In [50]: type(np.sin)  
90 Out[50]: <type 'numpy.ufunc'>
```

## Arrays indizieren

```
1 In [1]: import numpy as np
2
3 In [2]: a=np.array([[1,2,3,4],[4,5,6,7],[9,10,11,12]])
4 array([[ 1,  2,  3,  4],
5        [ 4,  5,  6,  7],
6        [ 9, 10, 11, 12]])
7
8 In [4]: a[1,2] # Element 2. Zeile , 3. Spalte
9 Out[4]: 6
10
11 In [5]: a[1] # 2. Zeile
12 Out[5]: array([4, 5, 6, 7])
13
14 In [6]: a[:2] # 1. und 2. Zeile
15 Out[6]:
16 array([[1, 2, 3, 4],
17        [4, 5, 6, 7]])
18
19 In [7]: a[:,2] # 3. Spalte
20 Out[7]: array([ 3,  6, 11])
21
22 In [10]: a[:, -1] # Letzte Spalte
23 Out[10]: array([ 4,  7, 12])
24
25 In [12]: a[:2,1:3] # ...
```

```
26 Out[12]:  
27 array([[2, 3],  
28        [5, 6]])
```



**Werte zählen:**

```
In [51]: a=np.array(range(100))
```

```
In [25]: a
```

```
Out[25]:
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

```
In [26]: np.where(a>50)
```

```
Out[26]:
```

```
(array([51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]),)
```

```
In [27]: len(np.where(a>50))
```

```
Out[27]: 1
```

```
In [28]: len(np.where(a>50)[0])
```

```
Out[28]: 49
```

```
In [16]: a>80
```

```
Out[16]:
```

```
array([False, False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False, False,
       True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
       True,  True,  True,  True,  True,  True,  True,  True,  True,  True,  True], dtype=bool)
```

```
In [17]: a[a>80]
```

```
Out[17]:
```

```
array([81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,  
       98, 99])
```

## Daten können gespeichert werden:

---

```
import numpy as np                                1
a=np.array(np.arange(1,10).reshape(3,3))          2
print a                                           3
#                                                  4
#[[1, 2, 3],                                     5
# [4, 5, 6],                                     6
# [7, 8, 9]])                                    7
a.tofile('a.data')                               8
b=np.fromfile('a.data', dtype=int)               9
print b                                          10
#[1, 2, 3, 4, 5, 6, 7, 8, 9]                    11
```

---

## Statistische Berechnungen

Numpy liefert grosse Zahl nützlicher Funktionen mit, siehe [Numpy\\_Example\\_List](#)

### Mittelwert, Standardabweichung, ...

```
import numpy as np
T=np.array([1.3,4.5,2.8,3.9])
print T.mean(), T.std(), T.var()
```

## Korrelationen

Beispiel mit T (Temperatur), P (Druck) und  $\rho$  (Dichte):

---

```
import numpy as np                                1
T=np.array([1.3,4.5,2.8,3.9])                      2
P=np.array([2.7,8.7,4.7,8.2])                      3
print np.corrcoef([T,P])                          4
#[[ 1.          0.98062258]                        5
# [ 0.98062258  1.          ]]                    6
rho=np.array([8.5,5.2,6.9,6.5])                    7
data=np.column_stack([T,P,rho])                   8
print data                                          9
# [[ 1.3,   2.7,   8.5],                          10
# [ 4.5,   8.7,   5.2],                          11
# [ 2.8,   4.7,   6.9],                          12
# [ 3.9,   8.2,   6.5]])                          13
print np.corrcoef([T,P,rho])                      14
#[[ 1.          0.98062258 -0.97090288]            15
# [ 0.98062258  1.          -0.91538464]            16
# [-0.97090288 -0.91538464  1.          ]]          17
```



## Diagonalwerte einer Matrizze:

```
In [27]: a=np.arange(12).reshape(3,4)
```

```
In [28]: a
```

```
Out[28]:
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
In [29]: a.diagonal()
```

```
Out[29]: array([ 0,  5, 10])
```

```
In [30]: a.diagonal(offset=1)
```

```
Out[30]: array([ 1,  6, 11])
```

```
In [31]: a.diagonal(offset=-1)
```

```
Out[31]: array([4,  9])
```

```
In [32]: np.diagonal(a)
```

```
Out[32]: array([ 0,  5, 10])
```



## Standardmatrizen:

```
In [33]: np.ones(3)
Out[33]: array([ 1.,  1.,  1.])
In [34]: np.zeros(3)
Out[34]: array([ 0.,  0.,  0.])
In [35]: np.eye(3)
Out[35]:
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

## Gleichungssysteme lösen:

```
In [36]: from numpy.linalg import inv
```

```
In [37]: a=np.array([[3,1,5],[1,0,8],[2,1,4]])
```

```
In [38]: a
```

```
Out[38]:
```

```
array([[3, 1, 5],  
       [1, 0, 8],  
       [2, 1, 4]])
```

```
In [39]: inva=inv(a)
```

```
In [40]: inva
```

```
Out[40]:
```

```
array([[ 1.14285714, -0.14285714, -1.14285714],  
       [-1.71428571, -0.28571429,  2.71428571],  
       [-0.14285714,  0.14285714,  0.14285714]])
```

```
In [41]: np.dot(a, inva)
```

```
Out[41]:
```

```
array([[ 1.00000000e+00,  5.55111512e-17,  1.38777878e-16],  
       [ 0.00000000e+00,  1.00000000e+00,  0.00000000e+00],
```

```
[ -1.11022302e-16,  0.00000000e+00,  1.00000000e+00]])  
In [42]: from numpy.linalg import solve  
In [43]: a  
Out[43]:  
array([[3, 1, 5],  
       [1, 0, 8],  
       [2, 1, 4]])  
In [45]: b=np.array([6,7,8])  
In [46]: x=solve(a,b)  
In [47]: x  
Out[47]: array([-3.28571429,  9.42857143,  1.28571429])  
In [48]: np.dot(a,x)  
Out[48]: array([ 6.,  7.,  8.])
```

## Nullstellen eines Polynom mit `roots` bestimmen:

```
In [48]: import numpy as np
```

```
In [49]: a=np.array([1,-6,-13,42])
```

```
In [50]: np.roots(a)
```

```
Out[50]: array([ 7., -3.,  2.]
```

## Fourier-Transformationen `numpy.fft`

Es existieren verschiedene FFT: Standard-FFT, FFTs für reale Arrays, Hermite-FFTs.

```
In [53]: import numpy as np
```

```
In [54]: signal=np.array([-2.,8.,-6.,4.,1.,0.,3.,5.])
```

```
In [55]: fourier=np.fft.fft(signal)
```

```
In [56]: fourier
```

```
Out[56]:
```

```
array([ 13.00000000 +0.j,      3.36396103 +4.05025253j,
        2.00000000 +1.j,      -9.36396103-13.94974747j,
       -21.00000000 +0.j,      -9.36396103+13.94974747j,
        2.00000000 -1.j,      3.36396103 -4.05025253j])
```

## Lineare Algebra

Einzelwertzerlegung:  $A = U\sigma V$

```
In [67]: import numpy as np
In [68]: A=np.array([[1.,3.,5.],[2.,4.,6.]])
In [69]: U,sigma,V=np.linalg.svd(A)
In [70]: print U
-----> print(U)
[[-0.61962948 -0.78489445]
 [-0.78489445  0.61962948]]
In [71]: print sigma
-----> print(sigma)
[ 9.52551809  0.51430058]
In [72]: print V
-----> print(V)
[[-0.2298477  -0.52474482 -0.81964194]
 [ 0.88346102  0.24078249 -0.40189603]
 [ 0.40824829 -0.81649658  0.40824829]]

In [74]: Sigma = np.zeros_like(A)
```

```
In [75]: n=min(A.shape)
In [76]: sigma[:n,:n]=np.diag(Sigma)
In [77]: sigma
Out[77]:
array([[ 9.52551809,  0.          ,  0.          ],
       [ 0.          ,  0.51430058,  0.          ]])
In [78]: print np.dot(U,np.dot(Sigma,V))
-----> print(np.dot(U,np.dot(Sigma,V)))
[[ 1.  3.  5.]
 [ 2.  4.  6.]
```

## 5.3 Matplotlib (pylab)

Mit `matplotlib` können professionelle 2D-Graphen aus z.B. `scipy` und `numpy` Daten erstellt werden.

Schöne Beispiele gibt es unter: [http://scipy-cookbook.readthedocs.org/items/idx\\_matplotlib\\_simple\\_plotting.html](http://scipy-cookbook.readthedocs.org/items/idx_matplotlib_simple_plotting.html)

`matplotlib` hat folgende Eigenschaften:

- Interaktive und programmatische Benutzung
- Speichern der erzeugten Graphen in PS, EPS, SVG, PNG, ...
- interaktiver Viewer



## Einlesen und Darstellen von Daten

Folgendes Beispiel demonstriert, wie man einfache Daten aus der Datei `numbers.dat` einliest und in einem Diagramm darstellt:

---

```
#!/usr/bin/env python 1
import pylab as pl      2
if __name__ == '__main__': 3
    # read in data to list 4
    data = pl.loadtxt('numbers.dat') 5
    # create x data 6
    x = range(0,len(data)) 7
    # create empty figure 8
    pl.figure(1,figsize=(6,4)) 9
    pl.xlabel('x') 10
    pl.ylabel('numbers') 11
    # plot data with data points 12
    pl.plot(x,data,'.') 13
    pl.title('Content of numbers.dat') 14
    # save figure to PNG file 15
    pl.savefig('numbers.png',dpi=72) 16
```

```
# show canvas
```

17

```
pl.show()
```

18

---

## Eine einfache Funktion darstellen:

---

<code>#!/usr/bin/env python</code>	1
<code>"""</code>	2
<code>Example: simple line plot.</code>	3
<code>Show how to make and save a simple line plot with labels, title and grid</code>	4
<code>"""</code>	5
<code>import numpy as np</code>	6
<code>import pylab as pl</code>	7
<code>t = np.arange(0.0, 1.0+0.01, 0.01)</code>	8
<code>s = np.cos(2*2*np.pi*t)</code>	9
<code>pl.plot(t, s)</code>	10
<code>pl.xlabel('time (s)')</code>	11
<code>pl.ylabel('voltage (mV)')</code>	12
<code>pl.title('About as simple as it gets, folks')</code>	13
<code>pl.grid(True)</code>	14
<code>pl.savefig('simple_plot')</code>	15
<code>pl.show()</code>	16

---

## Ein Histogramm mit Fit-Funktion:

---

```
#!/usr/bin/env python 1
import numpy as np      2
import pylab as pl      3
mu, sigma = 100, 15     4
# 10000 Gauss-verteilte Zufallszahlen erzeugen mit Mittelwert=100 und sigma=15 5
x = mu + sigma*np.random.randn(10000) 6
# make a histogram    7
pl.hist( x ) # default binning, color, ... 8
pl.show() # show graph 9
# 10
# pl.hist returns info on entries and bins 11
n, bins, patches = pl.hist( x ) 12
# print n, bins 13
# many more arguments to control histogram 14
n, bins, patches = pl.hist(x, 50) # have 50 bins 15
# 16
mybins=np.linspace(0.,200.,101) # create array with bin edges (100+1 from 0 to 200) 17
n, bins, patches = pl.hist(x, mybins) # supply array with bin edges 18
```

```
# 19
n, bins, patches = pl.hist(x, mybins, facecolor='green', alpha=0.75) # specify color 20
# 21
n, bins, patches = pl.hist(x, mybins, normed=1, facecolor='green', alpha=0.75) # contents normalized to 1 22
# overlay reference Gauss 23
y = pl.normpdf( bins, mu, sigma) 24
l = pl.plot(bins, y, 'r--', linewidth=1) 25
# further add-ons for plot 26
pl.xlabel('Smarts') # x-axis 27
pl.ylabel('Probability') # y-axis 28
pl.title(r'$\mathrm{Histogram\ of\ IQ:}\ \mu=100,\ \sigma=15$') # title 29
pl.axis([40, 160, 0, 0.03]) # limit x-axis range 30
pl.grid(True) # overlay grid 31
pl.show() 32
```

---

Viele weitere Beispiele zu Matplotlib in Tutorial:

<http://www.labri.fr/perso/nrougier/teaching/matplotlib>

## 5.4 SciPy

`scipy` ist eine Erweiterung von `numpy` mit vielen Bibliotheken für numerische Berechnungen, wie z.B. Optimierung, Numerische Integration, Statistik usw.

Zahlreiche Beispiele gibt es z.B. in <http://scipy-cookbook.readthedocs.io/index.html>

## Lineare Regression:

---

```
#!/usr/bin/env python 1
import scipy as sp      2
import pylab as pl      3
#Linear regression example 4
# This is a very simple example of using two scipy tools 5
# for linear regression, polyfit and stats.linregress 6
#Sample data creation 7
#number of points 8
n=50 9
t=sp.linspace(-5,5,n) 10
#parameters 11
a=0.8; b=-4 12
x=sp.polyval([a,b],t) 13
#add some noise 14
xn=x+sp.randn(n) 15
#Linear regressison -polyfit - polyfit can be used other orders polys 16
(ar,br)=sp.polyfit(t,xn,1) 17
xr=sp.polyval([ar,br],t) 18
```



```
#compute the mean square error 19
err=sp.sqrt(sum((xr-xn)**2)/n) 20
print('Linear regression using polyfit') 21
print('parameters: a=%.2f b=%.2f \nregression: a=%.2f b=%.2f, ms error= %.3f' % (a,b,ar,br,err)) 22
#matplotlib plotting 23
pl.title('Linear Regression Example') 24
pl.plot(t,x,'g.--') 25
pl.plot(t,xn,'k.') 26
pl.plot(t,xr,'r.-') 27
pl.legend(['original','plus noise', 'regression']) 28
pl.show() 29
#Linear regression using stats.linregress 30
(a_s,b_s,r,tt,stderr)=sp.stats.linregress(t,xn) 31
print('Linear regression using sp.stats.linregress') 32
print('parameters: a=%.2f b=%.2f \nregression: a=%.2f b=%.2f, std error= %.3f' % (a,b,a_s,b_s,stderr)) :
```

---

## Least Square Fit:

Anpassen von beliebigen (nicht-linearen) Funktionen an gewichtete Punkte (= Mess-Punkte mit Unsicherheit).

### 1. Beispiel: Breit–Wigner Kurve an gemessene Wirkungsquerschnitte

```

/usr/bin/env python 1
import numpy as np 2
import pylab as pl 3
from scipy.optimize import curve_fit 4
def BreitWig( x, m, g, spk ): 5
    " Breit-Wigner function " 6
    mw2 = m * m 7
    gw2 = g * g 8
    eb2 = x * x 9
    return( spk * gw2*mw2 / ( ( eb2 - mw2 )**2 + mw2 * gw2 )) 10
# hadronuc cross-section 11
xv = np.array([ 88.396, 89.376, 90.234, 91.238, 92.068, 93.080, 93.912 ]) 12
yv = np.array([ 6.943, 13.183, 25.724, 40.724, 27.031, 12.273, 6.980 ]) 13
ey = np.array([ 0.087, 0.119, 0.178, 0.087, 0.159, 0.095, 0.064 ]) 14
pinit = np.array([ 90., 2., 20.]) 15
out,cov=curve_fit(BreitWig, xv, yv, pinit, ey) 16
print out 17
print cov 18
for i in range(3): 19
    print "%d %7.4f +- %7.4f " % ( i, out[i], np.sqrt(cov[i][i])) 20

```

---

l1='data'	21
pl.errorbar(xv, yv, yerr=ey, fmt='ko',label=l1)	22
l2='fit'	23
bins = np.linspace( 88., 94.5, 500)	24
pl.plot(bins,BreitWig(bins,out[0],out[1],out[2]),'r-',lw=2,label=l2)	25
pl.legend()	26
#pl.yscale('log')	27
pl.show()	28

---

## 2. Beispiel: Exponential-Funktion an Histogramm

```

# /usr/bin/env python 1
import numpy as np 2
import pylab as pl 3
from scipy.optimize import curve_fit 4
# Generate data 5
# from numpy import random, histogram, arange, sqrt, exp, nonzero 6
n = 1000; isi = np.random.exponential(0.1, size=n) 7
db = 0.01; bins = np.arange(0, 1.0, db) 8
h = np.histogram(isi, bins)[0] 9
eh = np.sqrt(h) # stat error 10
# eh = np.maximum(1., np.sqrt(h)) # stat error 11
# 12
# Function to be fit 13
# x - independent variable 14
# p0, p1 - parameters 15
fitfunc = lambda x, p0, p1 : p1 * np.exp (- x / p0 ) 16
# Initial values for fit parameters 17
pinit = np.array([ 0.8, 2. ]) 18
# Hist count less than 4 has poor estimate of the weight 19
# don't use in the fitting process 20
idx = np.nonzero(h>4) 21
out, cov = curve_fit(fitfunc, bins[idx]+0.01/2, h[idx], pinit, eh[idx]) 22
l1 = 'data' 23
# pl.errorbar(bins[idx], h[idx], yerr=eh[idx], fmt='ko', label=l1) 24
pl.errorbar(bins[:-1], h, yerr=eh, fmt='ko', label=l1) 25

```

---

<code>l2='fit'</code>	26
<code>pl.plot(bins,fitfunc(bins,out[0],out[1]),'r-',lw=2,label=l2)</code>	27
<code>pl.legend()</code>	28
<code>pl.yscale('log')</code>	29
<code>pl.show()</code>	30

---

## Das Inverse einer Martix:

```
In [4]: a=numpy.mat('[1 3 5; 2 5 1; 2 3 8]')
```

```
In [5]: a.I
```

```
Out[5]:
```

```
matrix([[ -1.48,   0.36,   0.88],  
        [ 0.56,   0.08,  -0.36],  
        [ 0.16,  -0.12,   0.04]])
```

```
In [6]: from scipy import linalg
```

```
In [7]: linalg.inv(a)
```

```
Out[7]:
```

```
array([[ -1.48,   0.36,   0.88],  
       [ 0.56,   0.08,  -0.36],  
       [ 0.16,  -0.12,   0.04]])
```

## Gleichungssysteme lösen:

```
In [15]: a=numpy.mat('[1 3 5; 2 5 1; 2 3 8]')
In [16]: b=numpy.mat('[10;8;3]')
In [17]: print linalg.solve(a ,b)
-----> print(linalg.solve(a ,b))
[[-9.28]
 [ 5.16]
 [ 0.76]]
```

## Fouriertransformierte eines Kastenpotentials

---

<i>#!/usr/bin/env python</i>	1
<b>import</b> pylab as pl	2
<b>import</b> scipy as sp	3
<b>import</b> numpy as np	4
<i># -100 bis 100 in Schritten von 0.01 -&gt; 20000 Elemente</i>	5
x = np.arange(-100,100,0.01)	6
<i># np array gefuellt mit 0, Laenge wie x</i>	7
y = np.zeros_like(x)	8
<i># y = 1 wenn -0.5 &lt; x &lt; 0.5 -&gt; box of height 1</i>	9
y [ (x > -0.5) &(x <0.5) ] = 1.0	10
<i># figure loeschen</i>	11
pl.clf()	12
<i>#</i>	13
<i># 2 subplots vertical, plot oben</i>	14
pl.subplot(2, 1, 1)	15
pl.plot(x,y)	16
pl.axis([-1,1,-0.5,2])	17
z=np.fft.fft(y)	18



---

<code>f=np.fft.fftfreq(len(y),d=0.01)</code>	19
<code># 2 subplots vertical, plot unten</code>	20
<code>pl.subplot(2, 1, 2)</code>	21
<code>pl.plot(f,np.abs(z))</code>	22
<code>pl.axis([-5,5,0,100])</code>	23
<code>pl.show()</code>	24

---

## 5.5 Aufgaben

### 1. Datenanalyse

In der Datei `rohr1.dat` finden Sie eine Liste von Messungen der Drahtposition in verschiedenen Atlas Muon-Kammer Rohren. Lesen Sie die Zahlen ein:

```
data = numpy.loadtxt('rohr1.dat')
```

(a) Bestimmen Sie Mittelwert und Standardabweichung (Hinweis: numpy-Funktionen)

(b) Tragen Sie die Werte in Histogramm ein und Plotten es.

(c) Lesen Sie analog  $(x, y)$  Koordinaten der Datei `rohr2.dat`:

```
x, y = numpy.loadtxt('rohr2.dat', unpack=True)
```

Bestimmen Sie für  $x$  und  $y$  Mittelwert und Standardabweichung sowie die Korrelation.

Erstellen Sie  $(x, y)$ -Plot.

Lösung: `read1.py` `read2.py`

### 2. Berechnung von PI

Erzeugen Sie mit dem Zufallszahlen Modul `numpy.random.random` 1 Millionen Zufalls-Punkte mit  $x, y$ -Koordinaten zwischen 0 und 1. Wie groß ist der Anteil der Punkte innerhalb eines Radius von 1 vom Ursprung  $(0,0)$  ? Wie groß ist der Fehler zum zu erwartenden Ergebnis ?

Lösung: `drop.py` `pidemo.py`

### 3. Matplotlib

Berechnen Sie die Fouriertransformierte einer Gauss- und einer Delta-Funktions-Verteilung und zeichnen Sie die entsprechenden Verteilungen.

Lösung: [fft\\_aufg.py](#)

#### 4. Collatz Vermutung

... ein Klassiker der Zahlentheorie ...

Konstruiere Zahlenfolge:

- Beginne mit irgendeiner natürlichen Zahl  $n > 0$
- Ist  $n$  gerade, so nimm als Nächstes  $n/2$
- Ist  $n$  ungerade, so nimm als Nächstes  $3n + 1$
- Wiederhole die Vorgehensweise mit der erhaltenen Zahl.

Die Collatz-Vermutung lautet: *Jede so konstruierte Zahlenfolge mündet in den Zyklus 4, 2, 1, egal, mit welcher natürlichen Zahl  $n > 0$  man beginnt.*

Plotten Sie Zahl der Iterationen bis zur 1 gegen die Start-Zahl.

Lösung: [Collatz.py](#)

## 6 Objektorientiertes Programmieren – Allgemeines

Zitat: **Eine objektorientierte Programmiersprache ist weder eine notwendige noch eine hinreichende Bedingung für objektorientiertes Programmieren**

Objektorientiertes Programmieren heisst

- Denken, Modellieren, Designen, Implementieren in Objekten
- Objekte kommunizieren über definierte Interfaces direkt miteinander
- anstatt sequentiellm Aufruf aus Hauptprogramm
- OO Sprache wie Java, C++ oder Python äusserst hilfreich für Umsetzung aber Grund-Idee auch in C oder Pascal möglich.
- Prozedurales Programmieren oder Spaghetti-Code auch in C++, Java, Python möglich (*“Ich programmiere Fortran, egal in welcher Sprache”*)

## 6.1 UML - Unified Modelling Language

*“The Unified Modelling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a systems blueprints, including conceptual things like business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.”*

Grady Booch, Ivar Jacobsen, Jim Rumbaugh

(OMG Unified Modelling Language Specification, Version 1.3, March 2000)

## 6.2 Wozu UML ?

“Standardsprache” um

- Software–Systeme darzustellen (*anders als im source-code editor*)
- Abhängigkeiten und Beziehungen auszudrücken
- Abläufe zu visualisieren
- Anforderungen zu erarbeiten
- Weitergehende Abstrahierung

Für Physiker nichts Neues:

- **Mathematik** zur abstrakten, formalen Beschreibung der Natur
- Feynman-Diagramme für Teilchenphysik–Reaktionen.

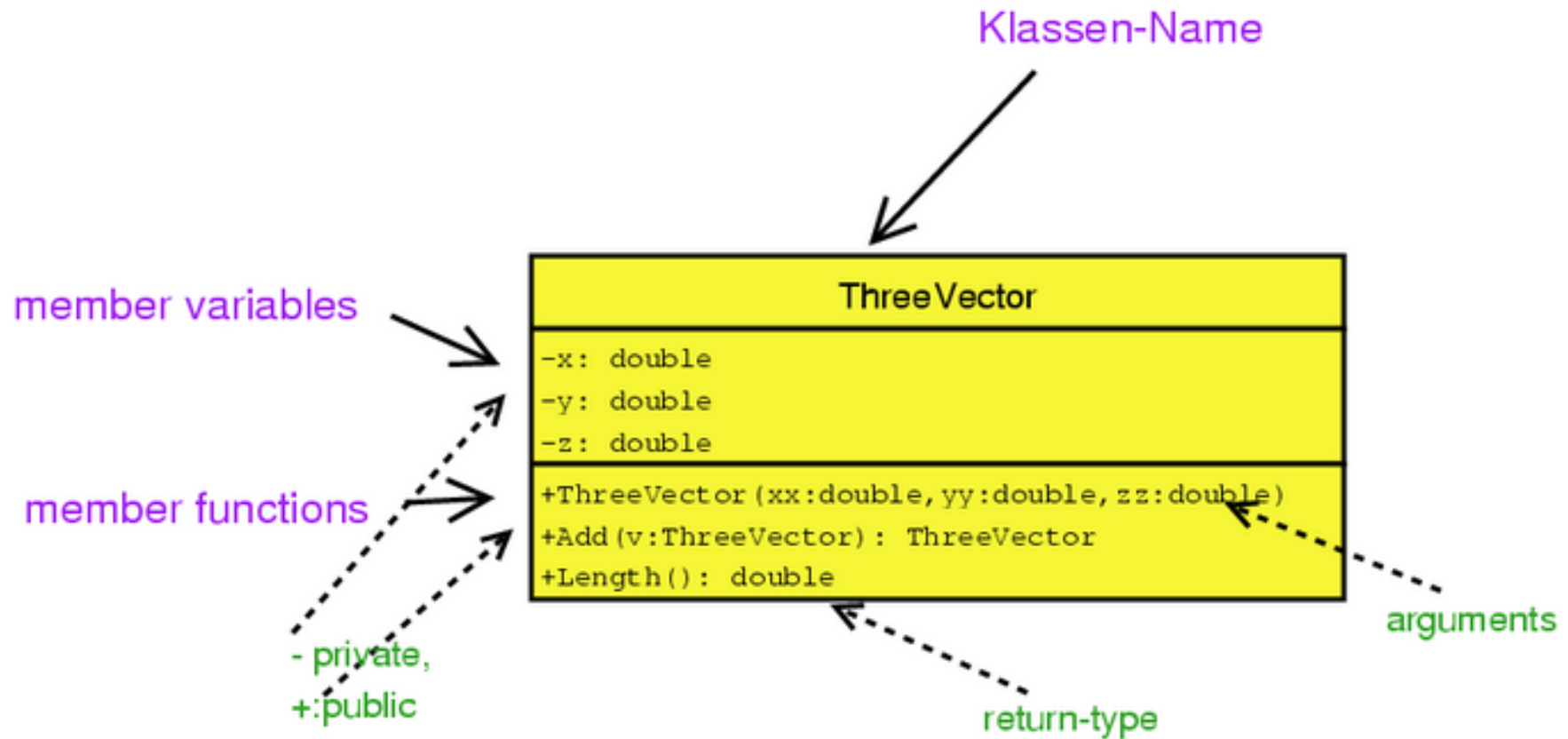
## 6.3 Klassen in UML

Klassen beschreiben Objekte:

- Status: Werte der Member-Variablen
- Konstruktor: Erzeugen des Objektes
- Interface: Signatur der member functions
- Verhalten: Implementierung der Member functions

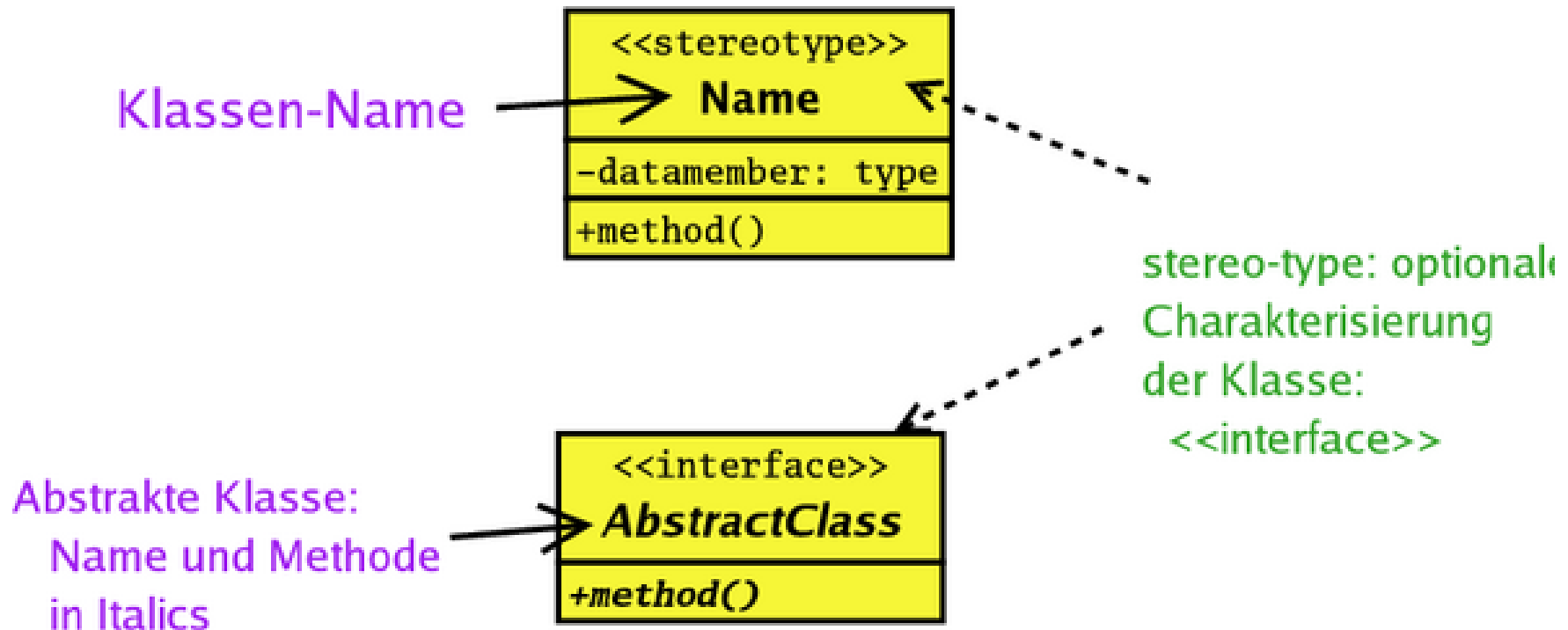
Klassen bzw. Objekte haben **Beziehungen** zu anderen Klassen/Objekten

## UML Klasse



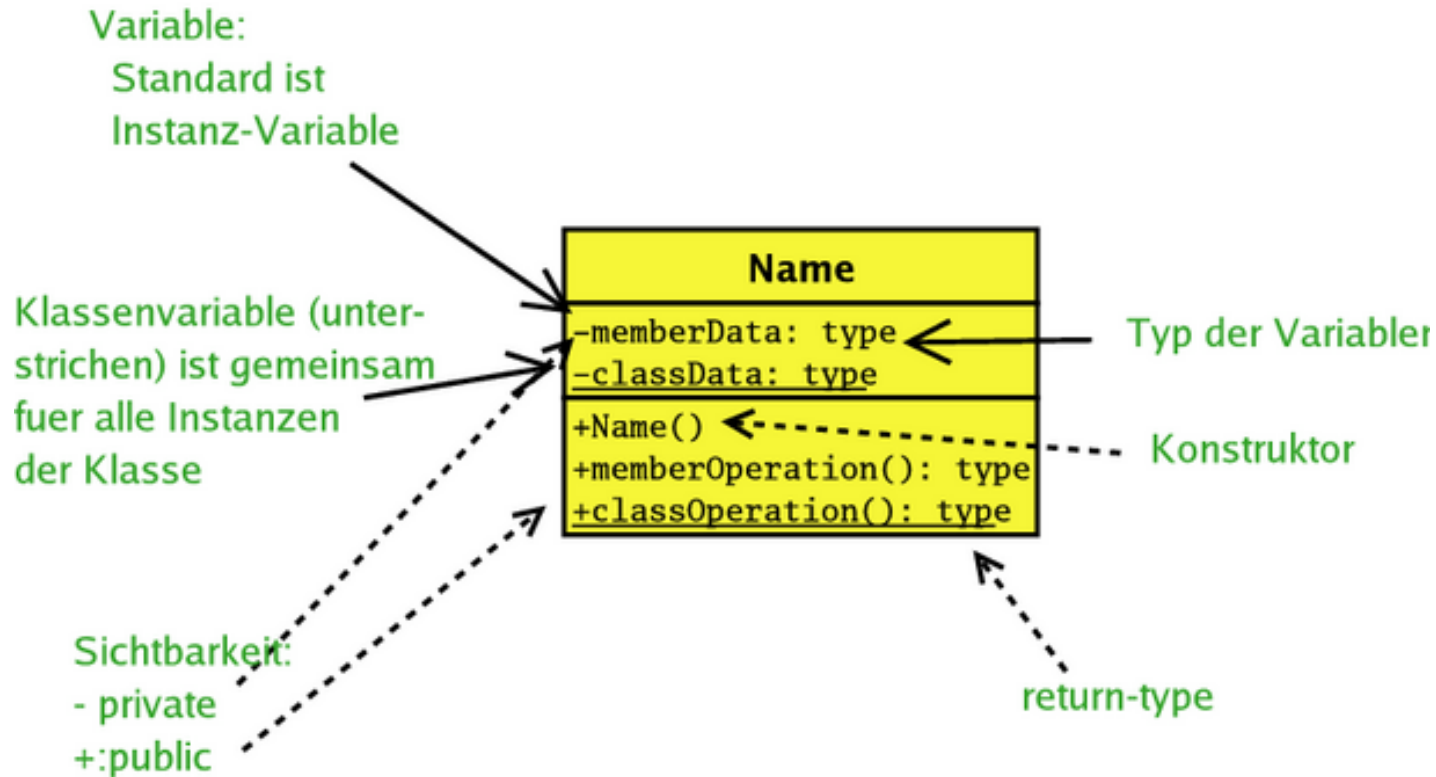


## Name der Klasse



## Klassen–Attribute und –Operationen

Attribute sind die Instanz–Variablen (oder Klassen–Variablen falls *static* deklariert).



Syntax: `visibility name : type`

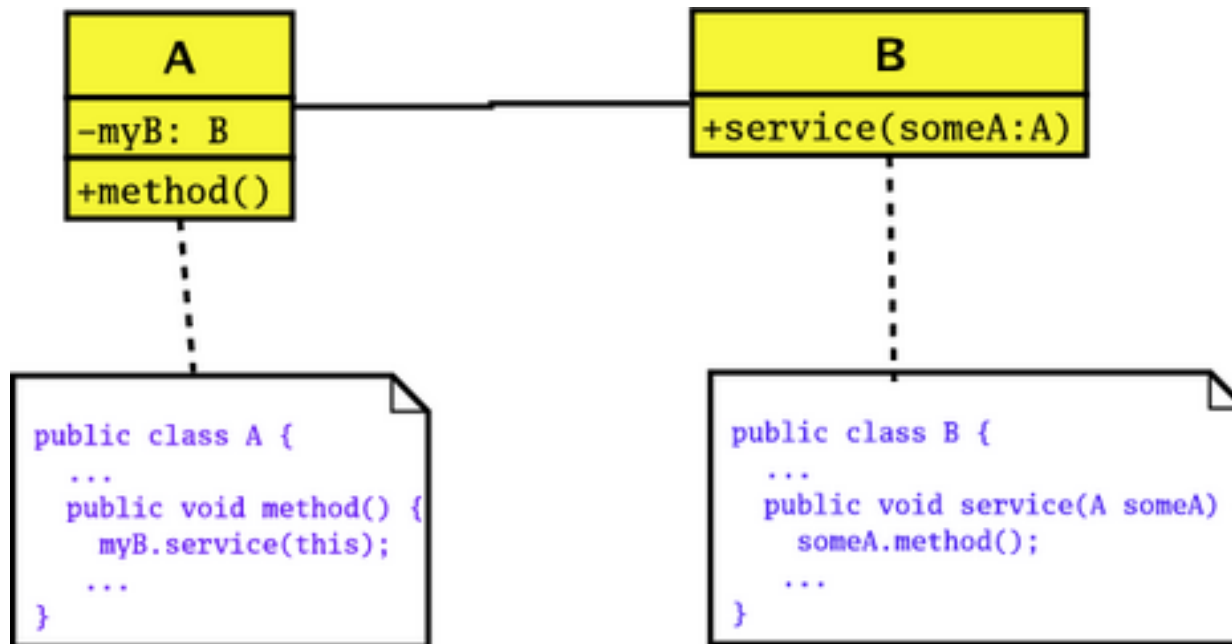
## 6.4 Beziehungen zwischen Klassen

Klassen bzw. Objekte können auf vielerlei Arten voneinander abhängen:

- **Association:** Nicht näher spezifizierte (i.d.R. lose) Kopplung von Klassen, nur in eine Richtung oder bi-direktional
- **Aggregation bzw. Composition:** Eine Klasse enthält ein (oder mehrere) Objekte einer anderen Klasse.
- **Inheritance**

## Binary Association

Beide Klassen kennen einander ...

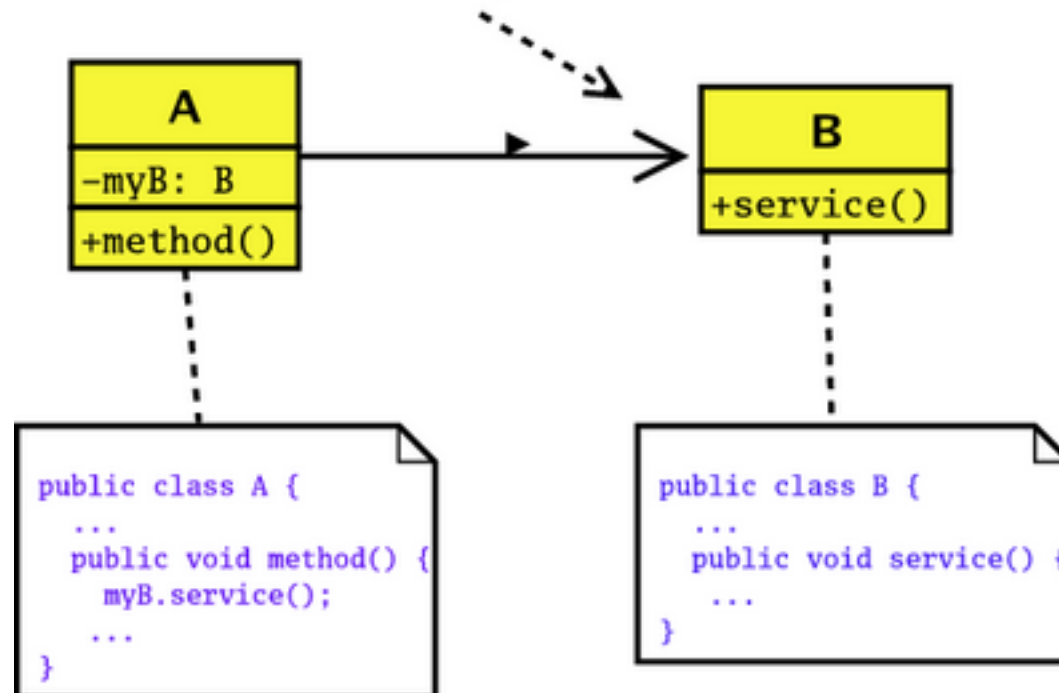


... als Referenz oder Funktionsargument oder in Container oder ...

## Gängiger ist die Unary Association:

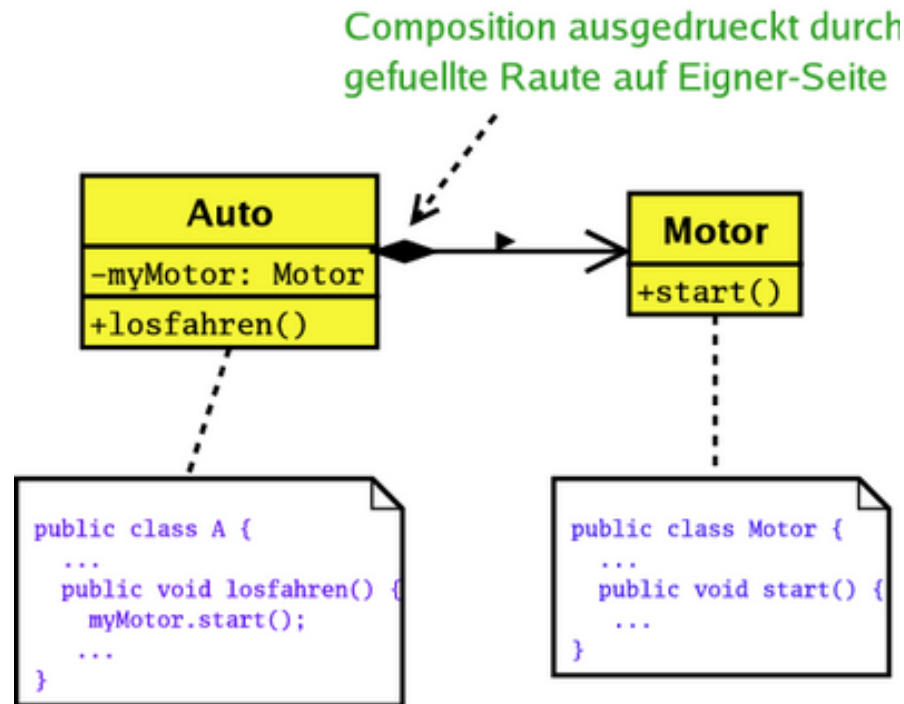
A kennt B aber B weiss nichts von A

Pfeil gibt Richtung der  
Abhaengigkeit an:  
A haengt von B ab



## Aggregation/Composition:

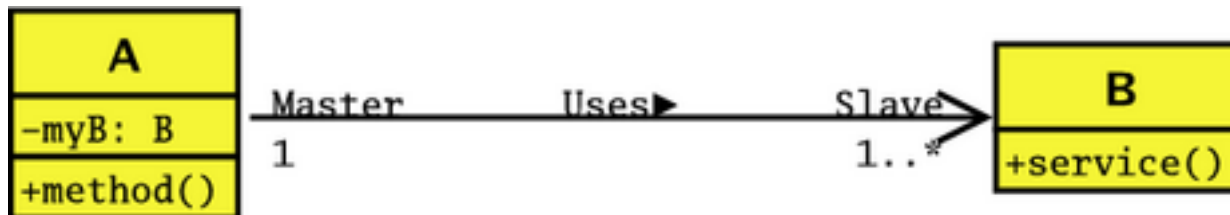
**A** enthält **B**: (*whole-part relation*), z.B. Auto enthält Motor



Subtiler Unterschied zwischen Aggregation und Composition, letztere kontrolliert *lifetime* des enthaltenen Objekts ⇒ siehe Literatur

## Details der Association:

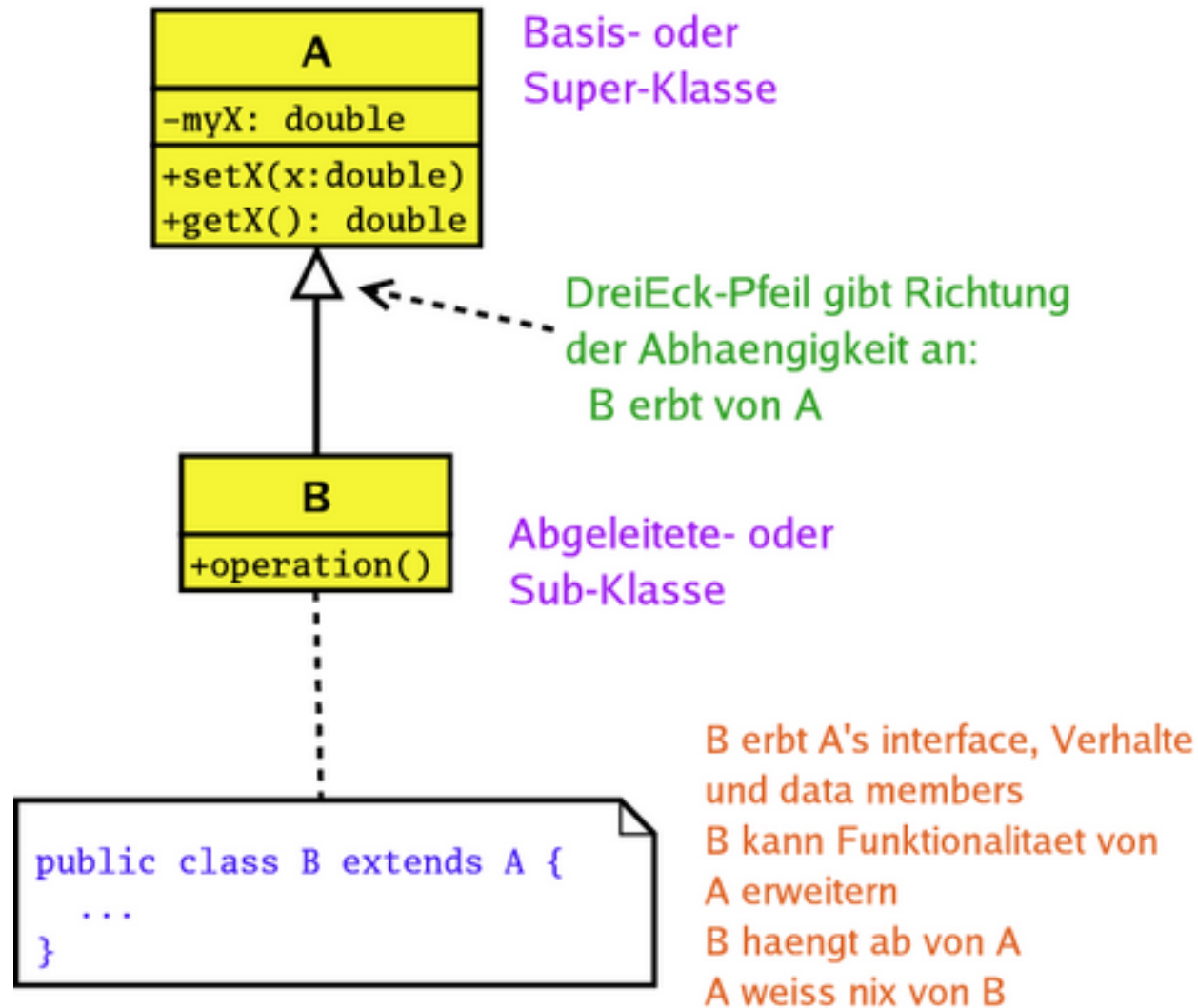
- Association kann Name haben (*uses*)
- An den Enden können Namen für die **Rollen** der beiden Klassen stehen (*Master, Slave*) ...
- ... und die Multiplizität, d.h. die Zahl der beteiligten Objekte stehen



- Richtig gelesen ergibt das zusammen ein Kompakt-Drehbuch:  
*One .. Master .. uses .. one or more .. Slaves*

## Vererbung (Inheritance):





## Zusammenfassung:

Mit UML Klassendiagrammen kann man Eigenschaften von Klassen, d.h. *Name, member-variablen, interfaces und Methoden* kompakt grafisch darstellen.

Je nach Kontext kann man eine präzise Auflistung aller member-variablen und Methoden mit Argumenten ausgeben oder nur eine unvollständige Liste der relevanten Elemente.

Man kann verschiedene Arten von Beziehungen zwischen Klassen mit UML ausdrücken:

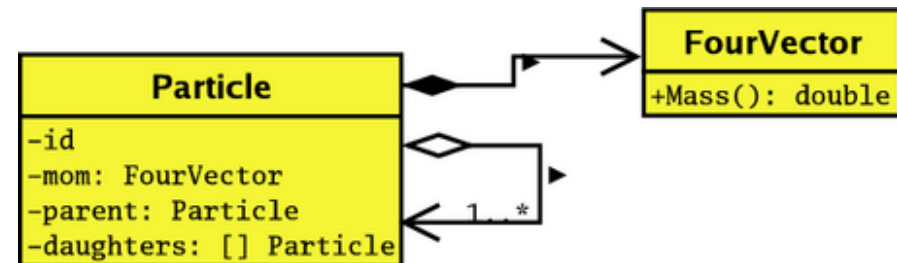
*Association, aggregation, composition, inheritance*

Die Verbindungen können vage Skizzen sein oder präzise Information über *Namen, Rollen, Multiplicities, ...* enthalten.

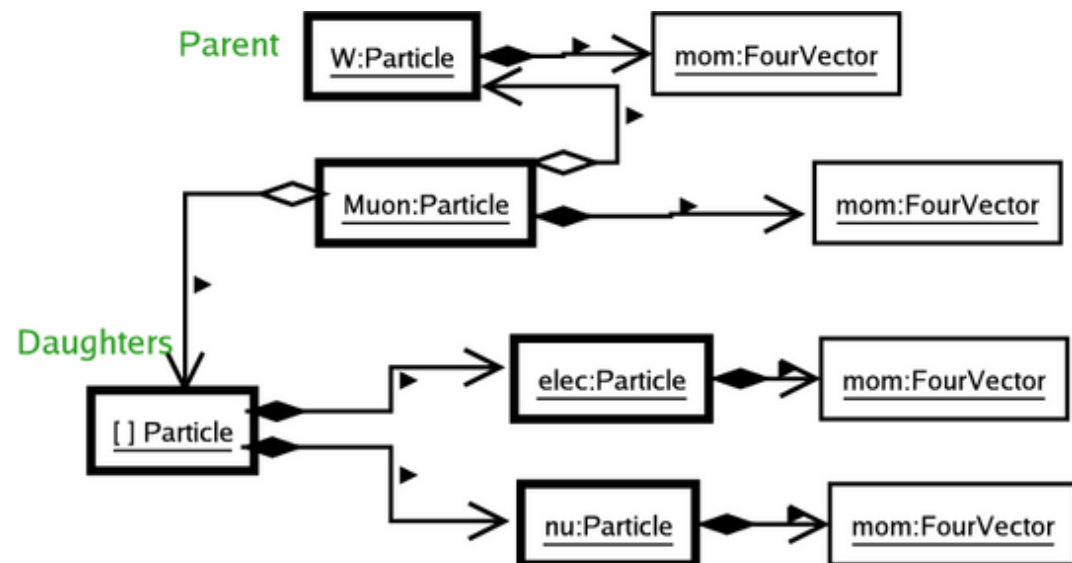
Klassendiagramme beschreiben das **statische Design** der Programm Struktur.

## 6.5 Objekt-Diagramme

Klassendiagramme beschreiben das **statische Design** der Programm Struktur. Fix, keine Änderung beim Programmablauf

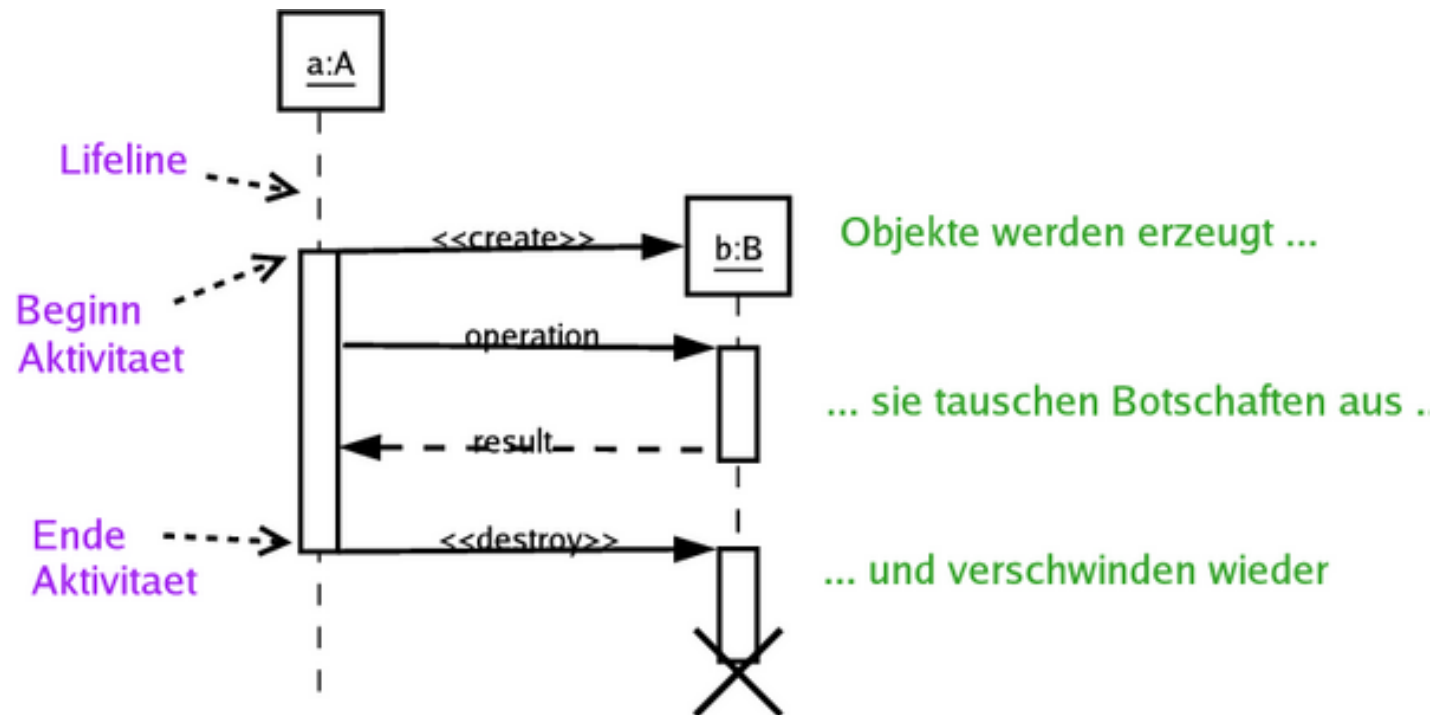


Analog **Objektdiagramme**: Beziehung zwischen den Objekten. Nur gültig für bestimmten Zeitpunkt zur Programm-Laufzeit !

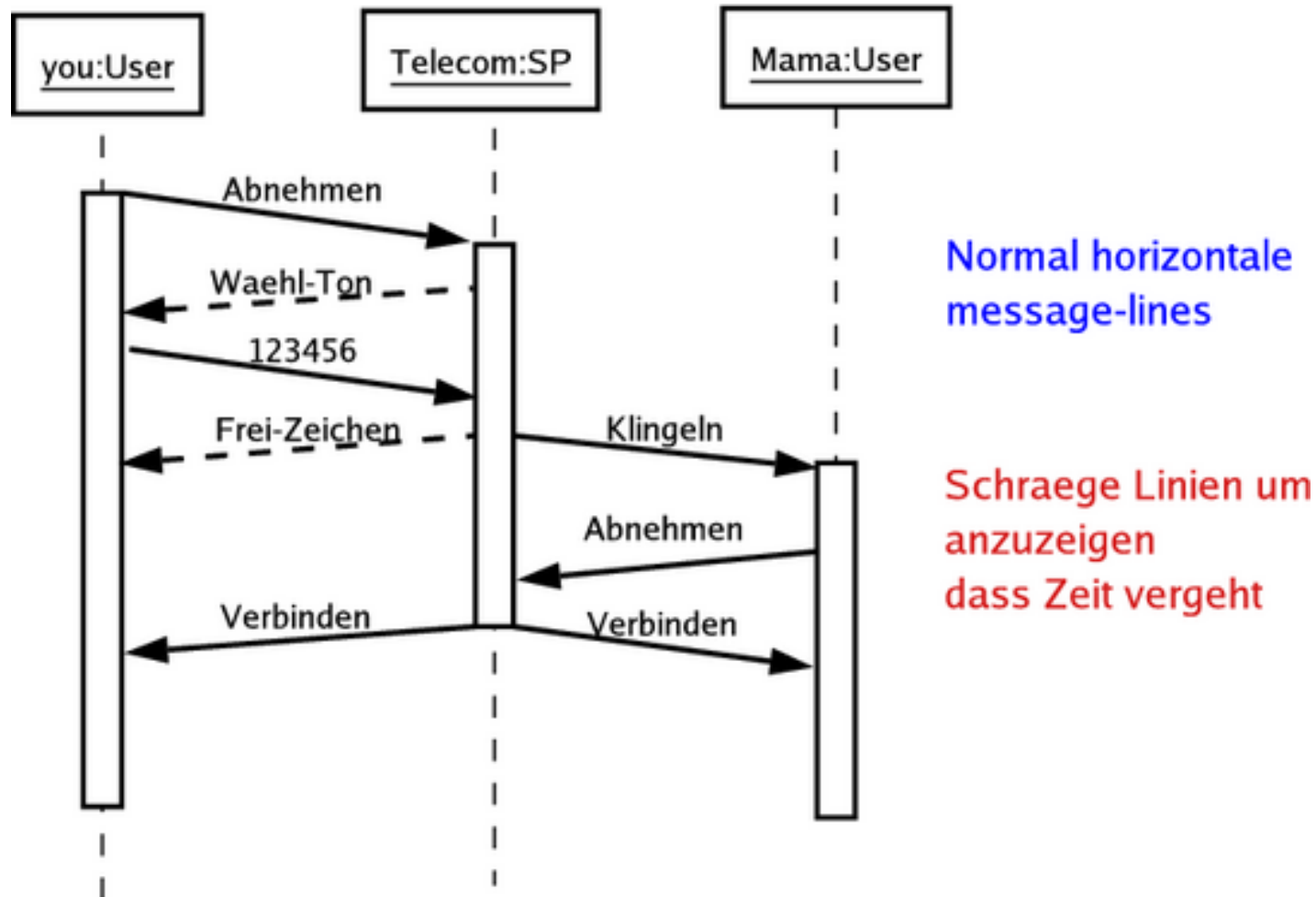


## 6.6 Sequenz-Diagramme

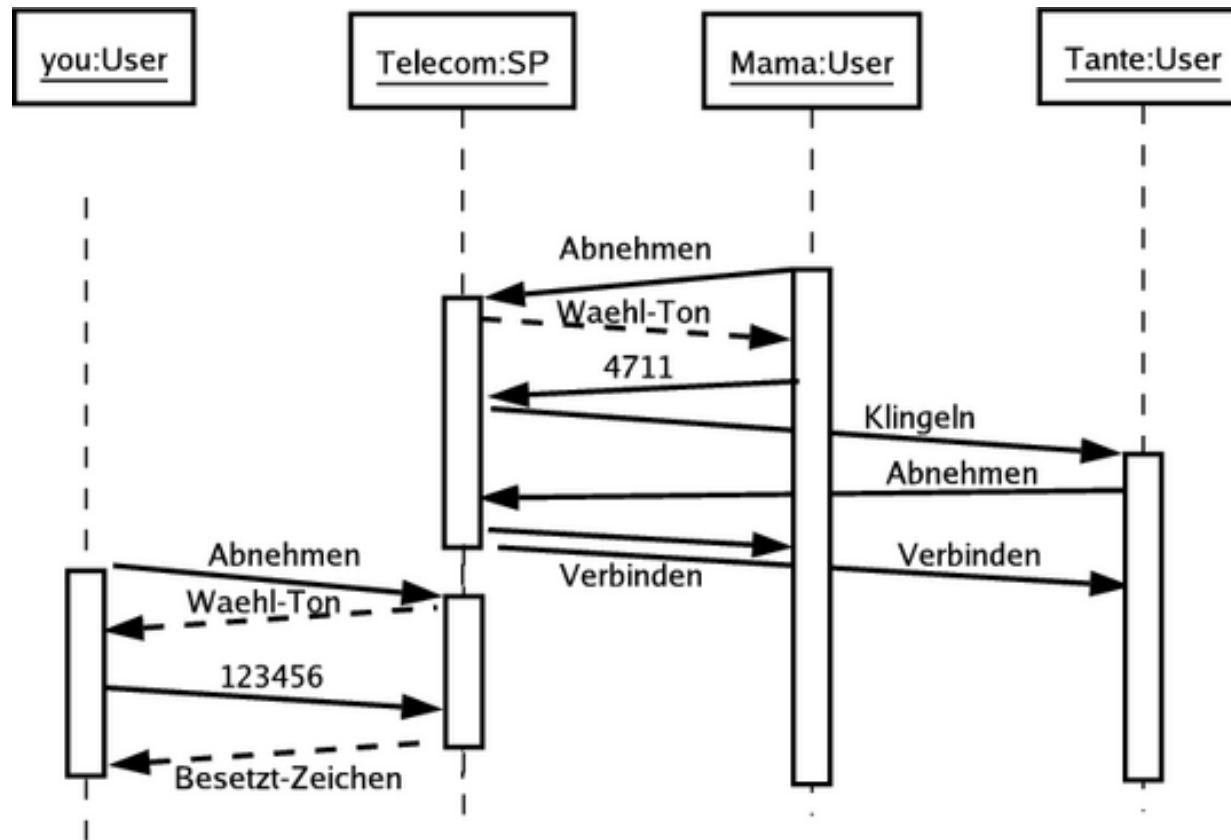
Neben der Klassen bzw. Objektstruktur und den Beziehungen zwischen ihnen ist die **Dynamik**, d.h. der Ablauf des Programms in einem vorgegebenen Szenario wichtig für das Design



## Konkretes Beispiel – Mama anrufen



## Meistens ist besetzt...



## 6.7 UML Summary

- **Class-Diagrams** zeigen die statischen Eigenschaften von Klassen und die (möglichen) Verbindungen zwischen ihnen.
- **Object-Diagrams** zeigen die konkreten Eigenschaften und Verbindungen für Objekte als **Momentaufnahme** für ein bestimmtes Szenario.
- **Sequence-Diagrams** zeigen den zeitlichen Ablauf, die Kommunikation der beteiligten Objekte für ein bestimmtes Szenario.

Das ist noch nicht alles: Darüberhinaus gibt's *use-case-Diagrams, Collaboration-Diagrams, Package-Diagram, ...*

siehe z.B.: [UML Quick Reference](#)

Für grosse OO-Projekte mit professionellen Teams spielt **formales UML Design** wichtige Rolle: 30 – 60 % des Aufwands.

- Umfassende Schulung nötig
- Mächtige kommerzielle Tools für UML Design: *Rational Rose, Together, ...*
  - Design in allen Details in UML Diagrammen  $\Rightarrow$  automatische Generierung von Source-Code daraus
  - analog **re-engineering**: Tools analysieren existierenden Code und erstellen UML Diagramme

Aber auch im kleineren Massstab (*Projekte in Physik*) sehr nützlich:

- UML Diagramme als einfache Skizzen per Hand oder mit **Umbrello** Programm (*Linux-tool*)
- Gute Basis für Diskussionen (*cf Source-Code*) und zur Dokumentation
- Präzises Erfassen der Anforderungen am besten über Diskussion von **use-cases**  $\Rightarrow$  **Sequence-Diagrams**



## 6.8 Software Engineering

Zentraler Zweig der Informatik befasst sich mit *Software Engineering*. Ursprünglicher Ansatz:

- Methoden und Vorgehensweisen aus anderen technischen Bereichen auf Software-Entwicklung übertragen: Bewährte Ingenieursverfahren aus Grossanlagenbau, Bauwirtschaft, Fahrzeugbau, ...
  - Präzise, detaillierte Spezifikation
  - Sorgfältiges Design
  - Unterteilung des Projekts soweit wie möglich in unabhängige Standardkomponenten
- Viele wichtige Fortschritte in Software Engineering in letzten Jahrzehnten
- Aber das Ziel Software Projekte analog zu andren Technik-Bereichen mit *Engineering* Methoden hinreichend kontrollieren zu können wurde nicht erreicht:  
*Nur ca 20 % der Software-Projekte sind erfolgreich im engeren Sinne, d.h. Kostenplan, Termine, Funktionalität weitgehend oder voll erfüllt* (Quelle: iX Magazin 2/06)
- Software zu dynamisch und lebendig; Software im Einsatz bewegt sich in hochkomplexem Zustandsraum; Auswirkungen nicht in Einzelheiten zu erfassen

Art und Weise des Programmierens extrem abhängig von Grösse des Projekts (*beachte: Faustregel sind 10 Zeilen Programmcode pro Tag !*)

**O(50-500) Zeilen** Kurse, Mini-Projekte. I.d.R. ohne formalen Design-Prozess lösbar: kurz Nachdenken, dann Programmieren. Programm von Einzelperson voll überschaubar. Kommentierter, strukturierter Code auch von anderen weiterzubenutzen (*...aber meist wird's komplett neu gemacht*).

**O(500-10k) Zeilen** Von Einzelperson oder Klein-Gruppe zu bewältigen. Weiterentwicklung und Weitergabe kritisch ohne formales Design.

**O(100k-1M) Zeilen** Entwickler-team nötig. Aufteilung in Unterprojekte. Ausführliches Design entscheidend, u.U. Hauptteil der Zeit.

**>1M Zeilen** scheitern häufig ...

**Aufwand wächst stark nicht-linear mit Grösse des Projektes !**

## 6.9 Probleme bei grossen Software Projekten

### Software ist **starr**, d.h.

- Abhängigkeiten schwer zu überschauen
- Unvorhersehbare Nebeneffekte treten auf bei jeder Änderung im Ablauf oder Austausch von Modulen
- Aufwand für Modifikationen kaum abzuschätzen
- Management Vorgabe *“Don’t touch a working system”*

### Software ist **fragil**, d.h.

- Kleine Änderungen haben grosse Nebenwirkungen
- Viele, schwer zu überschauende Stellen müssen angepasst werden
- Gefahr einiges zu übersehen  $\Rightarrow$  Bug
- Wenn  $P(bug|change) \approx 1$  kann das System nicht mehr weiterentwickelt werden.

## Software ist nicht wiederverwendbar, d.h.

- Code zur Lösung eines Problems schon vorhanden
  - Enthält aber Abhängigkeiten auf viele andere, für Problem irrelevante Dinge
  - einige kleinere Anpassungen nötig
- 2 Möglichkeiten:
  - Man übernimmt Verantwortung für den code und die nötigen Anpassungen
  - Man macht unabhängige *cut& paste* Kopie des benötigten codes.
- Idealerweise möchte man den code aber einfach nur benutzen und nicht Verantwortung für Maintenance übernehmen.

## Dependency Management

- In einem grossen Software Projekt sind eine Vielzahl von Abhängigkeiten zwischen den einzelnen Komponenten unvermeidbar
- Entscheidend ist die Kontrolle der Abhängigkeiten der einzelnen Komponenten
  - OO Methoden um Abhängigkeiten zu überwachen, z.B. UML Diagramme
  - OO “Tricks” um Abhängigkeiten zu minimieren (*abstrakte Klassen, Interfaces*)

## 6.10 Software Engineering Modelle

Vielfalt von Methoden im Einsatz, kein bestimmtes Verfahren konnte sich allgemein durchsetzen.

**Code&Fix** Einfach drauflos, nimm den Stier bei den Hörnern, Trial and Error. Nicht wirklich ein Modell, aber de-facto Standard in vielen Bereichen.

- Vorteil: Sehr flexible, schnelle Entwicklung
- Nachteil: Schwer zu pflegen und weiterzuentwickeln über längere Perioden

Ok, für kleine, überschaubare Projekte (sowohl Code-Umfang (1 kloc) als auch beteiligte Personen (1-3) als auch Lebensdauer (Monate–Jahr)

**Wasserfall Modell** lineare Design – Implementierung Sequenz

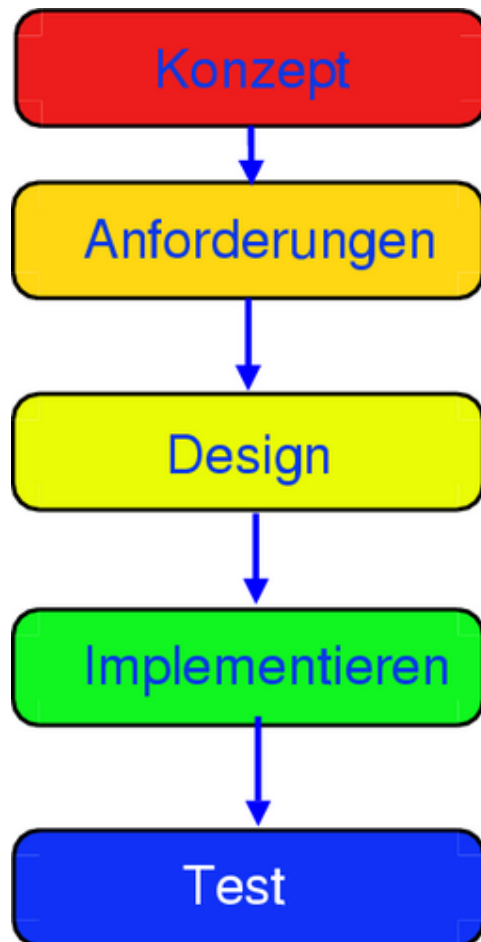
**RUP** Rational Unified Process, Design mit UML, iterativ, d.h. Zyklen von Design – Implementierung. “Schwergewichtiges Modell”, viel Aufwand und Erfahrung nötig, lange Design-Phasen

**Agile Methoden** XP, RAD, Scrum, ... Konträrer Ansatz, kurze Zyklen, schnelle Implementierung, Änderungen nicht Ausnahme sondern Regel, inkrementelles Umsetzen der Spezifikationen nach Prioritäten, lauffähige Zwischenprodukte.

Erst detaillierte Test-suite definieren, dann Erstellen des eigentlichen Programms. Test-Suite integraler Bestandteil der Software, wird bei jeder Änderung ausgeführt. ⇒ *Sehr nützlicher Ansatz auch für kleine Projekte und Teams ! (Nightly builds)*

**noch viele weitere Varianten, Alternativen ...**

## Wasserfall Modell



**Einzelner Durchlauf !**



**Wasserfall Modell** entwickelt von US Militär in den 60ern für grosse Programmierprojekte, analog zu Vorgehen in klassischen Ingenieur-Disziplinen.

Entscheidende Voraussetzungen:

- Detailliertes Verständnis aller Probleme im Vorhinein
- Stabile Umgebung, keine neuen Anforderungen während der Nutzung

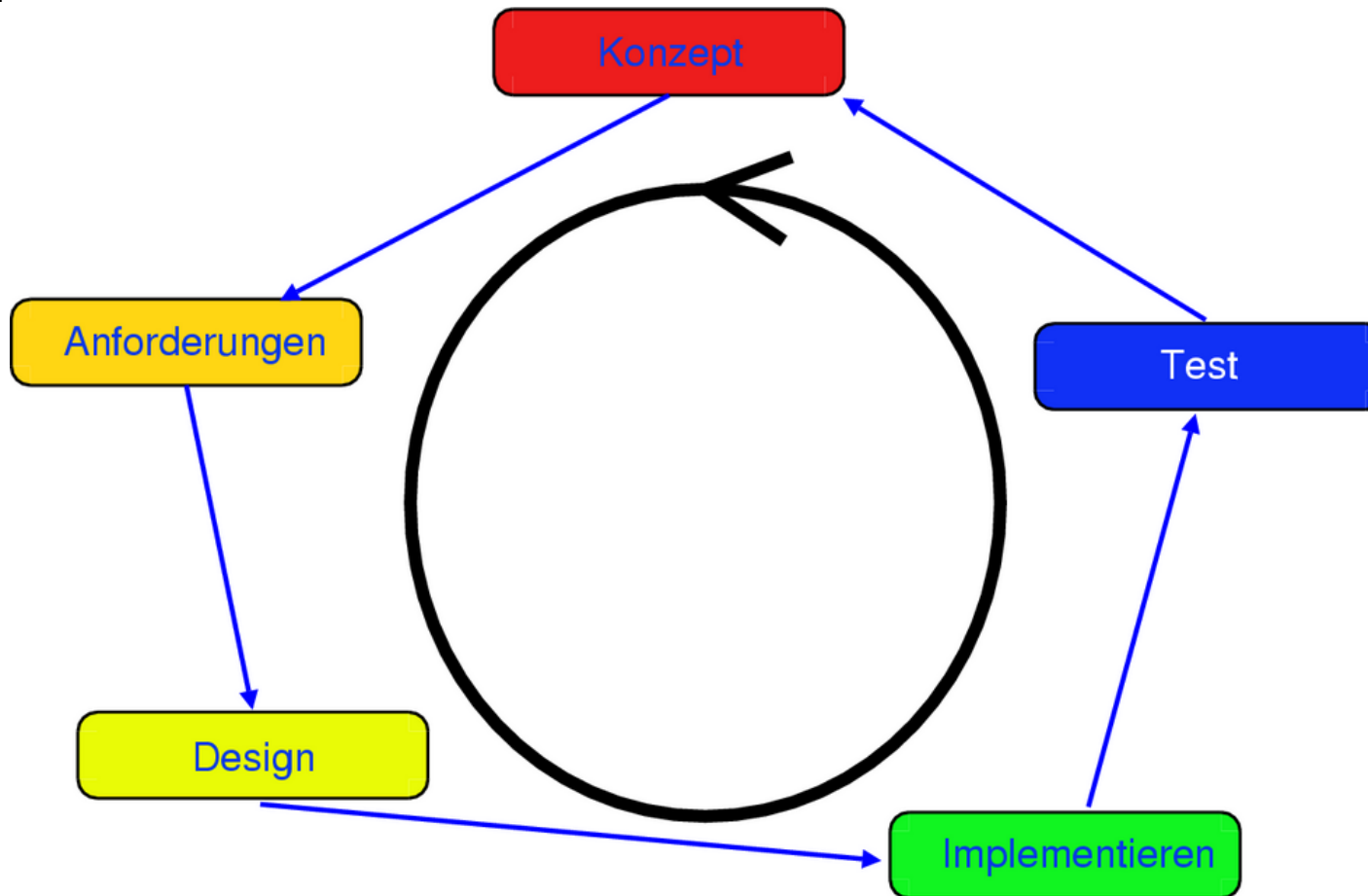
Änderungen später sind sehr schwierig, genauso Wiederverwendung und Erweiterung.

**Software-Entwicklung**  $\neq$  **Brückenbau** :

- Nutzeranforderungen ändern sich
- Umgebung, Tools, Services ändern sich
- ...

## UML Modell

Einerseits:



**Iterativ, zyklisches Durchlaufen !**

## UML Modell cont.

und andererseits:

### Objektorientiertes Programmieren

- Objekte als **Black Boxes**: *Komplexität* innen versteckt, einfache Bedienung von aussen.
  - *interface* zur Kommunikation
  - *Zustand*: Objekt enthält Daten
  - *Verhalten*: Methoden und Algorithmen
- Höhere Stufen der Abstraktion

## XP – Extreme Programming

Beinhaltet 2 wesentliche Konzepte

- Tests sind zentraler Bestandteil der Entwicklung: **“write tests first”** . D.h. bevor Klassen und Funktionen konkret implementiert werden, entwickelt man erst die Tests, die diese Klassen und Funktionen erfüllen müssen. ⇒ Tests als Detailspezifikation.

Diese Tests sind Teil des Entwicklungspakets und werden bei jeder Änderung und Neu-Übersetzung ausgeführt (*nightly builds*) , dadurch werden Fehler und Probleme bei Weiterentwicklung schnell erkannt.

- **Pair programming:** Entwicklung nicht als Einzelkämpfer sondern als 2er Team: Einer schreibt den Code, der andere denkt darüber nach, entwirft Tests, behält die grossen Ziele im Auge, etc. Die Rollen werden abgewechselt.

Zumindest das Einbeziehen von Tests in die Entwicklung lässt sich auch bei kleinen Projekten ohne weiteres anwenden, erfordert allerdings entsprechende Disziplin.

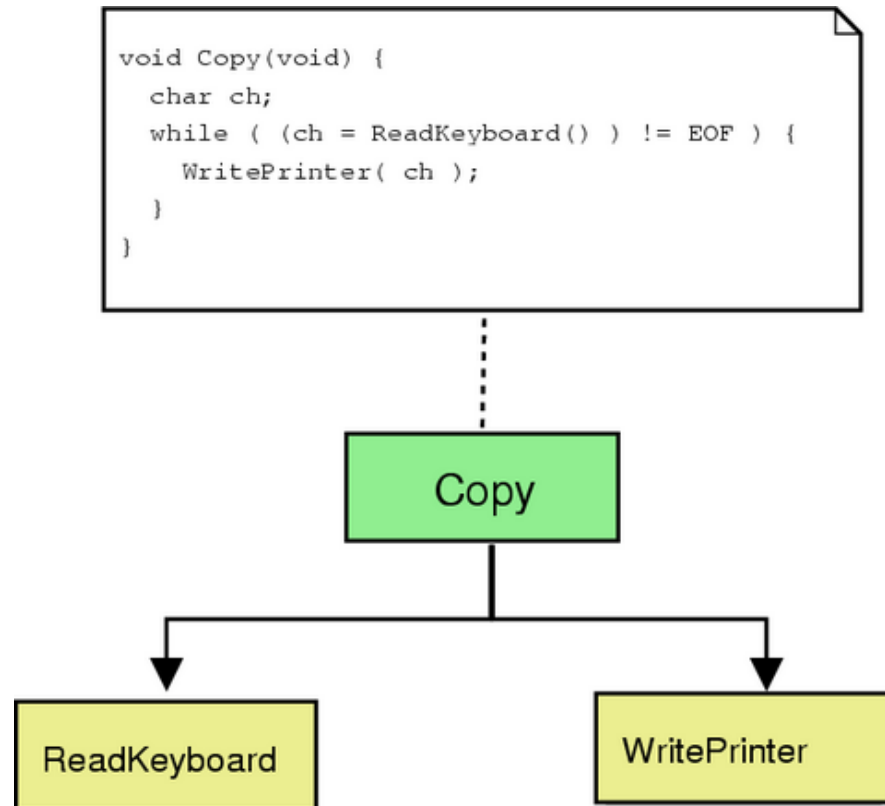
Beispiel ThreeVector Klasse:

Für systematisches Einbeziehen ist die Verwendung eines Test-Frameworks sinnvoll, z.B. <http://www.junit.org/>. *Nützliche Testklassen/funktionen, Integration in Build-Prozess.*

## 6.11 Code&Fix – ein Beispiel

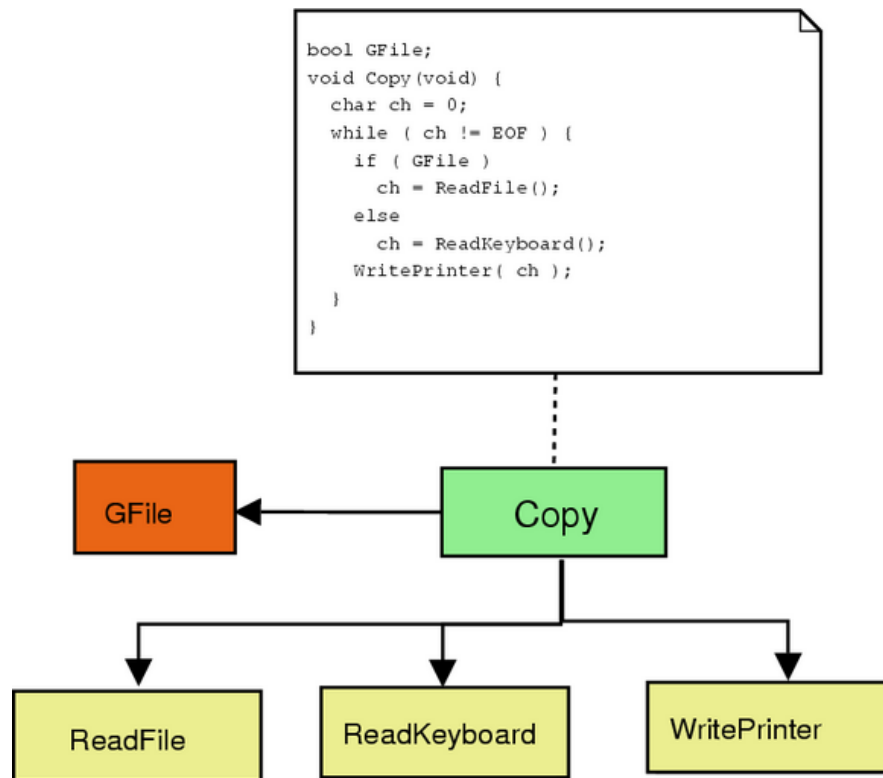
Eine simple *C-Funktion* `Copy()`, liest vom Keyboard und schreibt auf einen Printer.

### Copy – Version 1



## Copy – Version 2

Es wäre schön auch von Files lesen zu können. **Aber:** Es soll **backward-compatible** sein, vorhandene Applikationen müssen nichts ändern.

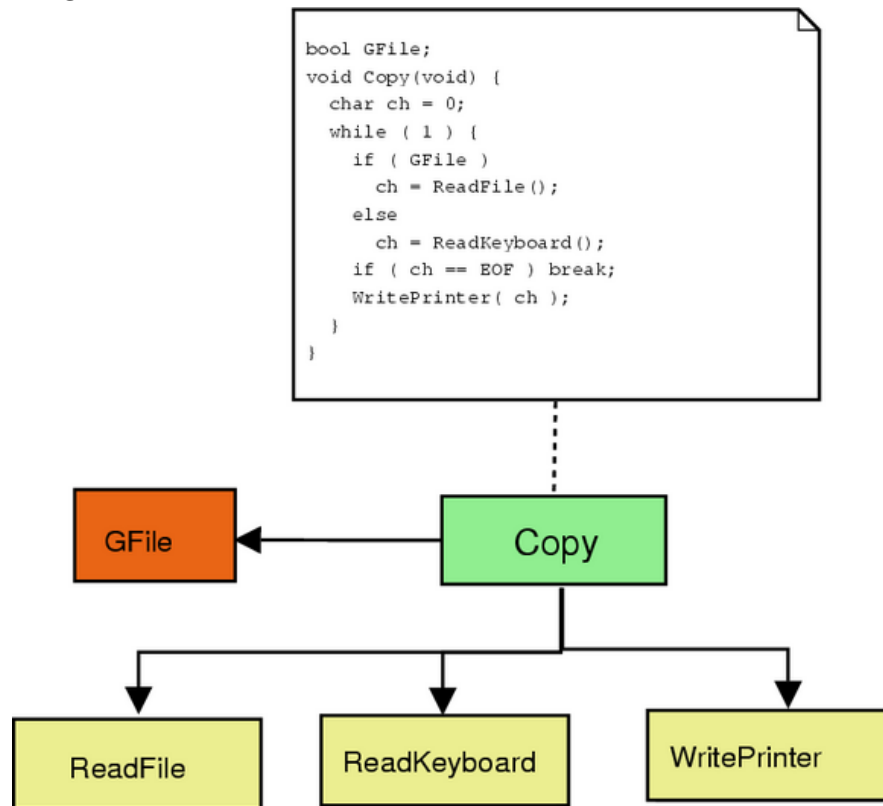


**Globale Variable als Flag**

## Copy – Version 3

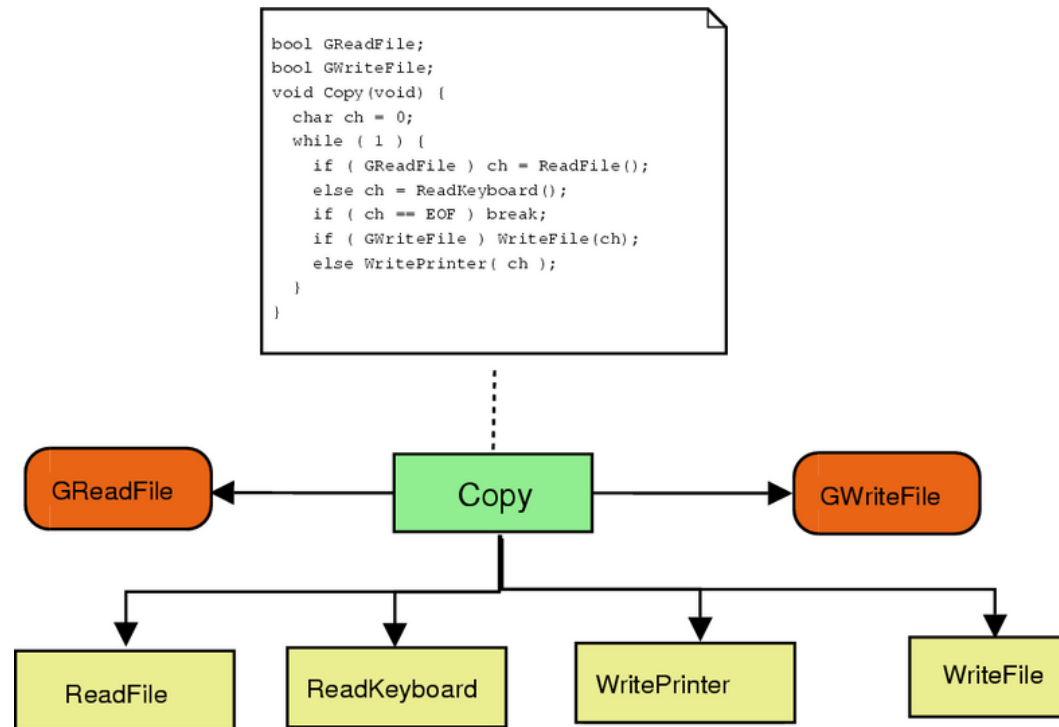
Oh je, da hat sich ein Fehler eingeschlichen: Es soll kein EOF ausgegeben werden !

Bug-fix:



## Copy – Version 4

Ausgabe in Files wäre ja auch noch ganz nett. Natürlich **backward-compatible**, also noch eine globale Variable !



Die Funktion wird immer grösser und komplexer, die Verwendung komplizierter ...

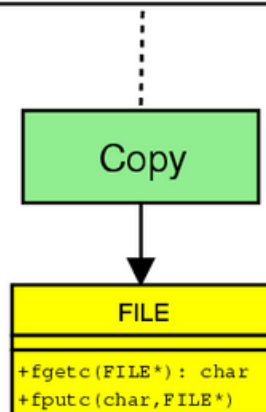


## Copy – Version 5

Zeit für ein ordentliches Re-design. Eine erfahrene C Programmiererin zeigt uns wie man's richtig macht:

```
#include <stdio.h>

void Copy(File *in, File *out) {
    char ch ;
    while ( (ch = fgetc( in )) != EOF ) {
        fputc( ch, out );
    }
}
```



OO-like in C: `FILE` wie Klasse für generische Byte-Streams.

**ABER:** Alle Stellen in denen `Copy()` verwendet wird müssen entsprechend angepasst werden !

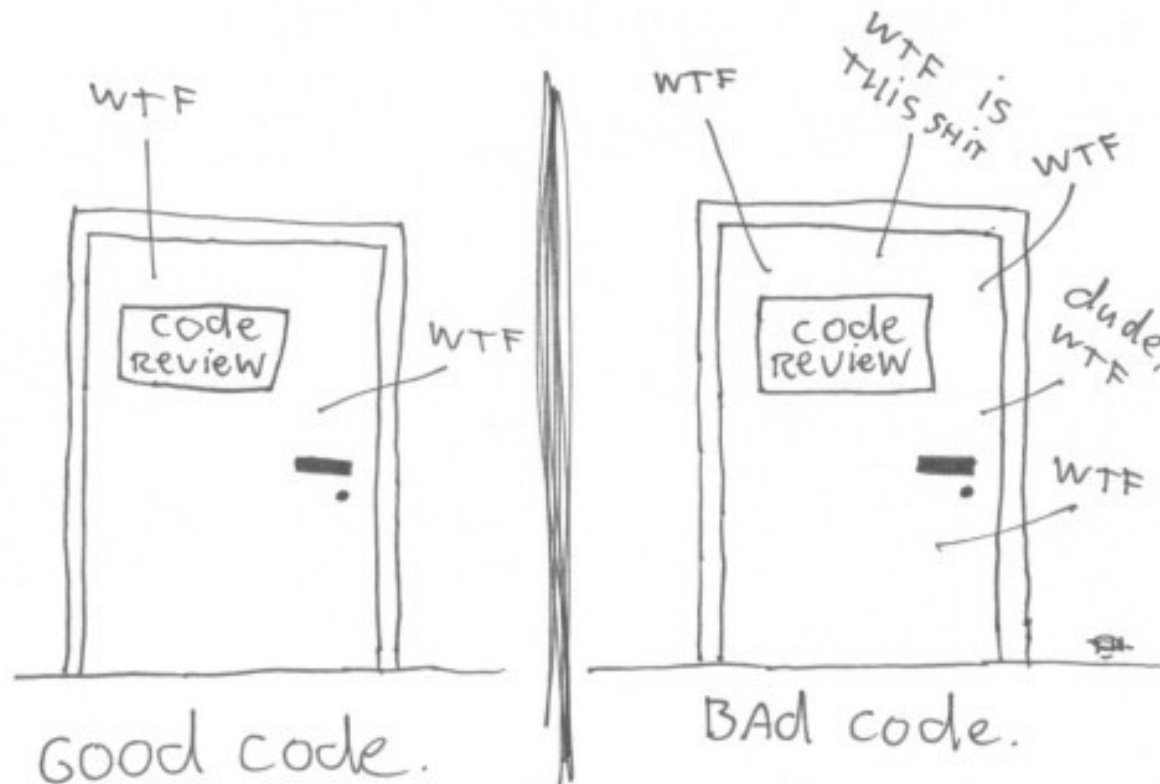
Typisches Beispiel wie Code altert und quasi *vergammelt*.

- einfache, klare Funktion zu Beginn
- mehr und mehr Features werden dazugepackt
- backward-compatibility erzwingt obskure Konstruktionen

OO-Analyse und -Design:

- Flexible, leicht erweiterbare Systeme
- Klar-definierte *responsibility* einer Klasse/Funktion.

The ONLY valid measurement  
of code quality: WTFs/minute



(c) 2008 Focus Shift/OSNews/Thom Holwerda - <http://www.osnews.com/comics>

## 6.12 Aufgaben

### 1. **umbrello** Programm

Unter Linux gibt es das Open–Source Programm **umbrello** zum Zeichnen von UML–Diagrammen.

( *Applications*  $\Rightarrow$  *Programming*  $\Rightarrow$  *Umbrello* )

Sie können damit Diagramme für neue Klassen erstellen und über den Source–Code Generator das Programmgerüst ausgeben lassen.

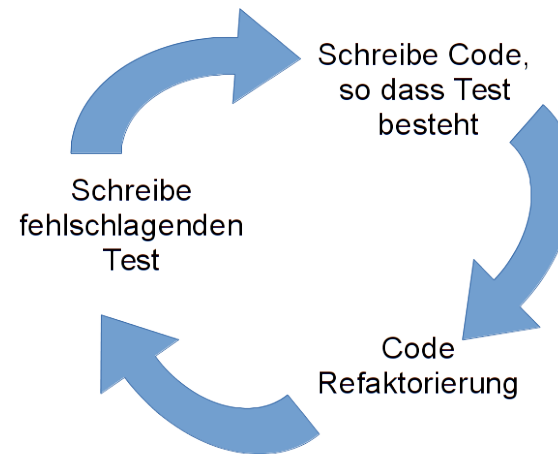
Oder man kann existierenden Source–Code importieren und UML Klassendiagramme dazu erzeugen. Probieren Sie es aus für unsere Beispiele zu *ThreeVector* und *LorentzVector*

### 2. **Fahrkartenautomat**

Analysieren Sie die Funktion eines **DB–Fahrkartenautomates**, d.h entwerfen Sie typische *use-cases*, definieren Sie *Komponenten* und *Operationen*, erstellen Sie Sequenz–Diagramme. (Am besten als Teamarbeit von 2-3 Personen).

## 7 Entwicklung mit Tests

Test-Driven Development (TDD) ist ein Programmierstil, bei dem zunächst ein Test geschrieben wird, der fehlschlägt, anschließend wird der eigentliche Programmcode implementiert, mit dem der Test dann funktioniert. Der somit entwickelte Programmcode dient dann als Grundlage für einen neuen erweiterten Test der neue Funktionalität prüft und anschließend mit dem wiederum erweiterten Programmcode funktioniert.



Die Programmentwicklung ist somit ein iterativer Prozess, der einen zwingt, über das Programmverhalten **vor** der Implementierung nachzudenken und zusätzlich verschiedene Fälle und Verzweigungen im Programm überprüft.

Allgemein kann der Befehl `assert` verwendet werden, um Bedingungen im Programm zu testen und gegebenenfalls eine exception auslösen:

```
1 In [1]: assert 42==42
2
3 In [2]: assert 2==1
4
5 AssertionError                                Traceback (most recent call last)
6 <ipython-input-2-f8e8f6bf7e85> in <module>()
7 ----> 1 assert 2==1
8
9 AssertionError:
```

Konkret werden sog. Unit Tests bzw. in Python das Modul `unittest` verwendet, um Programmcode zu überprüfen. Die am meisten gebrauchten Methoden sind:

- `assert`: Basis assert, das eigene assertions erlaubt
- `assertEqual(a, b)`: überprüfe, ob a und b gleich sind
- `assertNotEqual(a, b)`: überprüfe, ob a und b nicht gleich sind
- `assertIn(a, b)`: überprüfe, ob a in b ist
- `assertNotIn(a, b)`: überprüfe, ob a nicht in b ist
- `assertFalse(a)`: überprüfe, ob der Wert von a False ist

- `assertTrue(a)`: überprüfe, ob der Wert von `a` `True` ist
- `assertIsInstance(a, TYPE)`: überprüfe, ob `a` vom Typ "TYPE" ist
- `assertRaises(ERROR, a, args)`: überprüfe, ob `a` mit dem Werte `args` ausgerufen einen `ERROR` Exception erzeugt

Als Test-Umgebung kann entweder direkt das Modul `unittest` bzw. das Paket `nose` verwendet werden:

```
1 nosetests beispiel_unit_test.py
```

## 7.1 TDD Beispiel

Als Beispiel zur TDD soll ein sehr einfacher Taschenrechner dienen, der in der `add` Methode die Addition zweier Zahlen ausführt und das Ergebnis zurückgibt. Zunächst wird ein leeres Projekt angelegt:

```
1 mkdir mytest
2 cd mytest
3 mkdir app
4 mkdir test
5 touch app/__init__.py
6 touch test/__init__.py
```

Im Verzeichnis `test` wird die Datei `test_rechner.py` mit folgendem Inhalt erzeugt:

```
1 import unittest
2
3 class TddBeispiel(unittest.TestCase):
4
5     def test_rechner_add_method_gibt_richtiges_ergebnis(self):
6         rech = Rechner()
7         res = rech.add(2,2)
8         self.assertEqual(4, res)
9
10 if __name__ == '__main__':
11     unittest.main()
```

Ein Test hat folgende Struktur:

- Import des Moduls `unittest`.
- Anlegen einer von `unittest.TestCase` abgeleiteten Klasse, die alle Tests beinhaltet.
- Alle Methoden dieser Klasse implementieren verschiedene Tests und müssen mit `test_` beginnen.
- Die letzten beiden Zeilen ermöglichen es, den Test mit dem Standard `unittest` Modul und dem Befehl `python test_rechner.py` auszuführen.

Das Programm `nosetests` ermöglicht das automatische Ausführen aller existierenden Tests:



```
1 > nosetests
2 E
3 =====
4 ERROR: test_rechner_add_method_gibt_richtiges_ergebnis (test.test_rechner.TddBeispiel)
5 -----
6 Traceback (most recent call last):
7   File "/home/j/Johannes.Elmsheuser/cip_home/python/mytest/test/test_rechner.py", line 6, in
8     test_rechner_add_method_gibt_richtiges_ergebnis
9     rech = Rechner()
10 NameError: global name 'Rechner' is not defined
11 -----
12 Ran 1 test in 0.025s
13
14 FAILED (errors=1)
```

**Fehler war provoziert** ... es fehlt noch die tatsächliche Implementierung von `Rechner`. Es wird also die Datei `app/rechner.py` erzeugt:

```
1 class Rechner(object):
2
3     def add(self, x, y):
4         pass
```

und der Test erweitert:

```
1 import unittest
2 from app.rechner import Rechner
3
4 class TddBeispiel(unittest.TestCase):
5
6     def test_rechner_add_method_gibt_richtiges_ergebnis(self):
7         rech = Rechner()
8         res = rech.add(2,2)
9         self.assertEqual(4, res)
10
11 if __name__ == '__main__':
12     unittest.main()
```

Ein erneutes Ausführen der Tests ergibt:

```
1 > nosetests
2 F
3 =====
4 FAIL: test_rechner_add_method_gibt_richtiges_ergebnis (test.test_rechner.TddBeispiel)
5 -----
6 Traceback (most recent call last):
7   File "/home/j/Johannes.Elmsheuser/cip_home/python/mytest/test/test_rechner.py", line 9, in
8     test_rechner_add_method_gibt_richtiges_ergebnis
9     self.assertEqual(4, res)
10 AssertionError: 4 != None
```

```
11 |  
12 | Ran 1 test in 0.020s  
13 |  
14 | FAILED (failures=1)
```

Der Test zeigt an, daß die Methode `add` noch kein richtiges Ergebnis liefert. Dies kann folgendermaßen korrigiert werden:

```
1 | class Rechner(object):  
2 |  
3 |     def add(self, x, y):  
4 |         return x+y
```

```
1 | > nosetests  
2 | .  
3 |  
4 | Ran 1 test in 0.019s  
5 |  
6 | OK
```

Der Test funktioniert jetzt, allerdings wird nur der Fall getestet, der tatsächlich zunächst interessiert - was passiert allerdings, falls andere Typen als Zahlen verwendet werden, da Python das Addieren von z.B. Strings oder Listen mit der gleichen Syntax erlaubt ? Um diese Fälle zu testen, wird der Test erweitert:

```
1 import unittest
2 from app.rechner import Rechner
3
4 class TddBeispiel(unittest.TestCase):
5
6     def setUp(self):
7         self.rech = Rechner()
8
9     def test_rechner_add_method_gibt_richtiges_ergebnis(self):
10         res = self.rech.add(2,2)
11         self.assertEqual(4, res)
12
13     def test_rechner_gibt_fehler_wenn_beide_args_nicht_zahlen(self):
14         self.assertRaises(ValueError, self.rech.add, 'zwei', 'drei')
15
16 if __name__ == '__main__':
17     unittest.main()
```

Der neue Test überprüft, ob eine `ValueError` exception ausgelöst wurde. Zusätzlich wurde die Methode `setUp` verwendet, die zur Initialisierung der Tests verwendet werden kann. Das Test-Ergebnis sieht zunächst folgendermaßen aus, da noch kein `ValueError` im eigentlichen code ausgelöst wird:

```
1 > nosetests
2 .F
3 =====
```

```
4 FAIL: test_rechner_gibt_fehler_wenn_beide_args_nicht_zahlen (test.test_rechner.TddBeispiel)
5
6 Traceback (most recent call last):
7   File "/home/j/Johannes.Elmsheuser/cip_home/python/mytest/test/test_rechner.py", line 14, in
8     test_rechner_gibt_fehler_wenn_beide_args_nicht_zahlen
9     self.assertRaises(ValueError, self.rech.add, 'zwei', 'drei')
10   AssertionError: ValueError not raised
11
12 Ran 2 tests in 0.015s
13
14 FAILED (failures=1)
```

Der code muss also folgendermassen verbessert werden:

```
1 class Rechner(object):
2
3     def add(self, x, y):
4         number_typen = (int, long, float, complex)
5
6         if isinstance(x, number_typen) and isinstance(y, number_typen):
7             return x + y
8         else:
9             raise ValueError
```

Um alle Kombinationsmöglichkeiten zu überprüfen, werden noch weitere Tests hinzugefügt:

```
1 import unittest
2 from app.rechner import Rechner
3
4 class TddBeispiel(unittest.TestCase):
5
6     def setUp(self):
7         self.rech = Rechner()
8
9     def test_rechner_add_method_gibt_richtiges_ergebnis(self):
10         res = self.rech.add(2,2)
11         self.assertEqual(4, res)
12
13     def test_rechner_gibt_fehler_wenn_beide_args_nicht_zahlen(self):
14         self.assertRaises(ValueError, self.rech.add, 'zwei', 'drei')
15
16     def test_rechner_gibt_fehler_wenn_x_arg_keine_zahl(self):
17         self.assertRaises(ValueError, self.rech.add, 'zwei', 3)
18
19     def test_rechner_gibt_fehler_wenn_y_arg_keine_zahl(self):
20         self.assertRaises(ValueError, self.rech.add, 2, 'drei')
21
22 if __name__ == '__main__':
23     unittest.main()
```

Alle 4 Tests funktionieren nun erfolgreich:

```
1 > nosetests
2 ....
3
4 Ran 4 tests in 0.020s
5
6 OK
```

## 7.2 Mit PDB debuggen

Manchmal schlägt ein Test fehl, und es ist nicht sofort ersichtlich, wo der Fehler liegt. In diesem Fall ist die einfachste Methode einige `print` Befehle in den code einzufügen, um aktuelle Variableninhalte anzuzeigen. Als Beispiel soll folgender “falsche” code dienen, in dem die Werte subtrahiert anstatt addiert werden und zusätzlich noch einige `print` Ausgaben gemacht werden:

```
1 class Rechner(object):
2
3     def add(self, x, y):
4         number_typen = (int, long, float, complex)
5
6         if isinstance(x, number_typen) and isinstance(y, number_typen):
7             print 'X = %s' %x
8             print 'Y = %s' %y
9             res = x - y
10            print 'Res = %s' %res
11            return res
12        else:
13            raise ValueError
```

Das Ergebnis des Test zeigt auch die zusätzlichen Bildschirmausgaben an:

```
1 > nosetests
2 F...
```



```
3 =====
4 FAIL: test_rechner_add_method_gibt_richtiges_ergebnis (test.test_rechner.TddBeispiel)
5 -----
6 Traceback (most recent call last):
7   File "/home/j/Johannes.Elmsheuser/cip_home/python/mytest/test/test_rechner.py", line 11, in
8     test_rechner_add_method_gibt_richtiges_ergebnis
9     self.assertEqual(4, res)
10 AssertionError: 4 != 0
11 ----- >> begin captured stdout << -----
12 X = 2
13 Y = 2
14 Res = 0
15 ----- >> end captured stdout << -----
16 -----
17 -----
18 Ran 4 tests in 0.021s
19
20 FAILED (failures=1)
```

Bei fortgeschrittenem Programmcode sollte man einen Debugger, wie z.B. `pdb` verwenden. Diesen Debugger kann man einfach mit einem `import pdb` ins das Programm einbinden und den Programmablauf mit einem sog. “breakpoint” unterbrechen, wie in folgendem Beispiel gezeigt:

```
1 class Rechner(object):
2
```

```
3 def add(self, x, y):
4     number_typen = (int, long, float, complex)
5
6     if isinstance(x, number_typen) and isinstance(y, number_typen):
7         import pdb; pdb.set_trace()
8         return x + y
9     else:
10        raise ValueError
```

Mit dem zusätzlichen Parameter `nosetests -s` erhält man nun den interaktiven `pdb` Prompt:

```
1 > nosetests -s
2 > /home/j/Johannes.Elmsheuser/cip_home/python/mytest/app/rechner.py(8)add()
3 -> return x + y
4 (Pdb) list
5      3         def add(self, x, y):
6      4             number_typen = (int, long, float, complex)
7      5
8      6             if isinstance(x, number_typen) and isinstance(y, number_typen):
9      7                 import pdb; pdb.set_trace()
10     8 ->             return x + y
11     9             else:
12    10                 raise ValueError
13 [EOF]
14 (Pdb)
```

Man kann jetzt interaktiv z.B. die Werte von `x` und `y` anzeigen lassen bzw. z.B. schrittweise das Programm weiter ausführen. Eine nützliche Befehle von `pdb` sind:

- `n`: einen Schritt weiter zur nächsten Programmzeile
- `list`: 5 Programmzeilen vor und nach dem aktuellen “breakpoint” anzeigen
- `args`: Aktuelle Variablen anzeigen.
- `continue`: Das Programm bis zum Ende ausführen
- `jump <line number>`: Bis zur Zeile `<line number>` springen
- `quit/exit`: `pdb` beenden

## 7.3 Aufgaben

- **Primzahlen**

Schreiben Sie einen Test für den Algorithmus zur Primzahlenberechnung aus dem ersten Python-Kurs (Siehe Aufgabe 6 (a): [Python-Kurs](#), [prim\\_simple\\_lsg.py](#))

## 8 Profiling von Programmen

“Premature optimization is the root of all evil (or at least most of it) in programming.”

[http://en.wikiquote.org/wiki/Donald\\_Knuth/](http://en.wikiquote.org/wiki/Donald_Knuth/)

Man sollte normalerweise nicht im ersten Ansatz ein Programm im Hinblick auf die höchste Effizienz und Schnelligkeit programmieren, da hierdurch der Code üblicherweise schnell unübersichtlich wird und schwierig zu debuggen ist.

D.h. man sollte erstmal schauen, dass man ein funktionierendes Programm hat, das die Spezifikationen (*=Tests!*) erfüllt und erst danach bei Tests unter Real-Bedingungen die Performance untersuchen und ggf. optimieren. Dazu gibt es spezielle **profiling** Methoden, die eine systematische Optimierung bzw. Suche nach *Bottlenecks* erleichtern.

### 8.1 Einschub: Decorators

Ein decorator wird als Software Design Muster verwendet. Decorators verändern dynamisch die Funktionalität einer Funktion, Methode oder Klasse ohne dass eine Sub-Klasse verwendet oder der Programmcode verändert werden muss. In Python kann eine decorator-Funktion mit `@mydecorator` vor einer Funktionsdefinition eingefügt werden. Ein Beispiel:

```
1 def logger(func):  
2     def inner(*args, **kwargs): #1
```

```
3         print "Arguments were: %s, %s" % (args, kwargs)
4         return func(*args, **kwargs) #2
5     return inner
6
7 @logger
8 def foo1(x, y=1):
9     return x * y
10
11 @logger
12 def foo2():
13     return 2
14
15 foo1(5, 4)
16 foo1(1)
17 foo2()
```

Siehe auch folgende ausführliche [Beschreibung](#) von decorators.

## 8.2 Einfacher Timer

Timer sind einfache und flexible Methoden, um die Programmausführzeit zu messen. Folgendes Beispiel demonstriert einen einfachen Timer mit Hilfe eines decorators:

```
1 import time
2
```

```
3 def timefunc(f):
4     def f_timer(*args, **kwargs):
5         start = time.time()
6         erg = f(*args, **kwargs)
7         ende = time.time()
8         print f.__name__, 'brauchte', ende - start, 's'
9         return erg
10    return f_timer
11
12 def get_number():
13     for x in xrange(50000000):
14         yield x
15
16 @timefunc
17 def teure_funktion():
18     for x in get_number():
19         i = x ^ x ^ x
20     return 'Ergebnis'
21
22 #
23 res = teure_funktion()
```

Timer erfordern schrittweise Funktionen auf höherer Ebene zu testen und dann anschließend tiefer in einzelne Funktionen einzusteigen, um die einzelne Schwachstelle zu identifizieren, bevor diese behoben werden kann.

Sehr nützlich ist zusätzlich das `timeit` Module, um kleine Programmschnipsel zu testen.

## 8.3 Eingebauter Profiler

Das eingebaute `profile` Modul zeichnet alle Funktionsaufrufe und deren Ausführzeit auf. Ein einfaches interaktives Beispiel demonstriert die Funktionsweise:

```
1 import cProfile
2 import re
3 cProfile.run('re.compile("foo|bar")')
```

Obiges Beispiel erzeugt folgendes Ergebnis:

```
1 cProfile.run('re.compile("foo|bar")')
2      194 function calls (189 primitive calls) in 0.001 seconds
3
4      Ordered by: standard name
5
6      ncalls  tottime  percall  cumtime  percall filename:lineno(function)
7           1    0.000    0.000    0.001    0.001 <string>:1(<module>)
8           1    0.000    0.000    0.001    0.001 re.py:188(compile)
9           1    0.000    0.000    0.001    0.001 re.py:226(_compile)
10          1    0.000    0.000    0.000    0.000 sre_compile.py:178(_compile_charset)
```

Der profiler kann auch auf der Kommandozeile für ein vollständiges Programm angeschaltet werden:



```
python -m cProfile myscript.py
```

bzw. die Ausgabe in eine Datei umgeleitet werden, die dann mit Tool gprof2dot graphisch aufbereitet werden kann:

```
python -m cProfile -o myscript.profile myscript.py
```

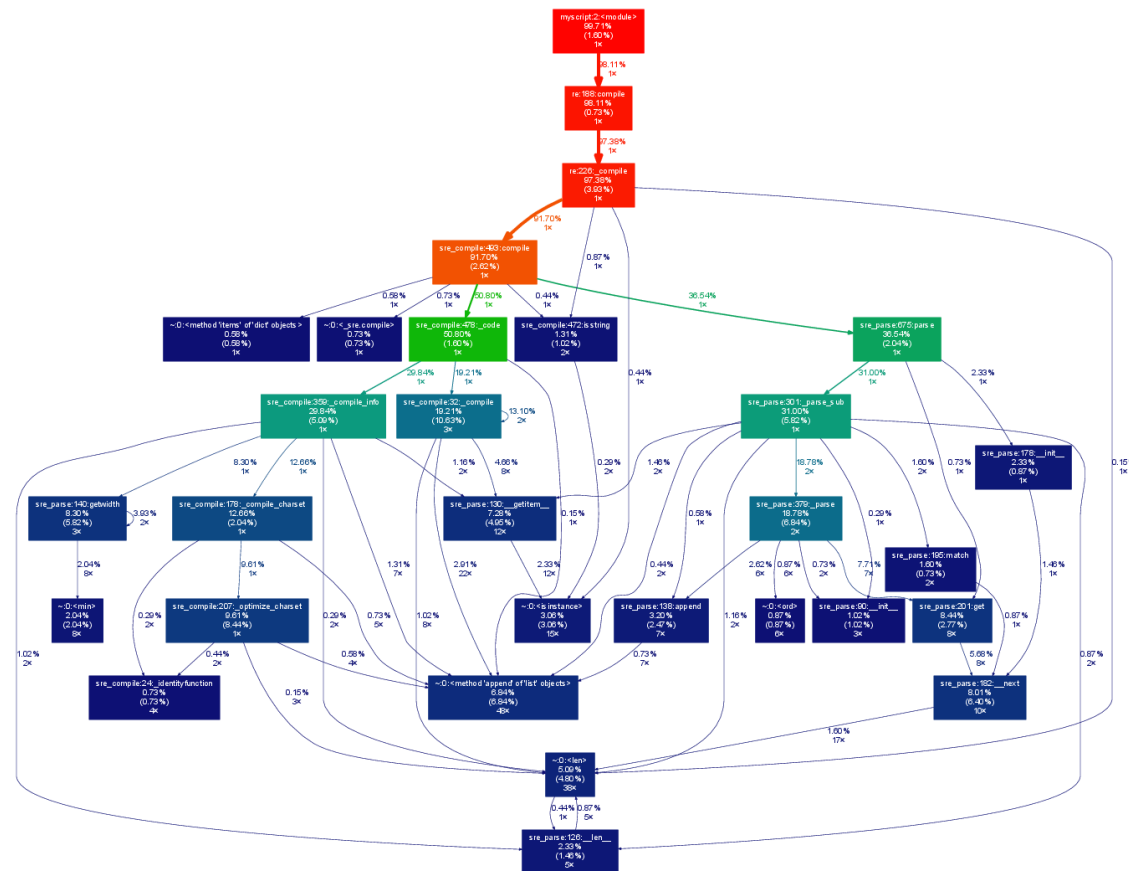
Zur Visualisierung lädt man das Skript gprof2dot.py mit folgendem Befehl herunter:

```
git clone https://github.com/jrfonseca/gprof2dot
```

und erzeugt ein Ablaufdiagramm in der Datei myscript.pdf mit folgendem Befehl:

```
python gprof2dot/gprof2dot.py -f pstats myscript.profile | dot -Tpdf -o myscript.pdf
```

Ein Beispieldiagramm sieht folgendermaßen aus:



Ein etwas ausführlicheres Beispiel, da profiling mit Hilfe eines decorators einschaltet und eine formatierte Ausgabe erzeugt, ist in folgendem Programmbeispiel gegeben:

```
1 import cProfile
```

```
2
3 def do_cprofile(func):
4     def profiled_func(*args, **kwargs):
5         profile = cProfile.Profile()
6         try:
7             profile.enable()
8             res = func(*args, **kwargs)
9             profile.disable()
10            return res
11        finally:
12            profile.print_stats()
13    return profiled_func
14
15 def get_number():
16     for x in xrange(5000000):
17         yield x
18
19 @do_cprofile
20 def teure_funktion():
21     for x in get_number():
22         i = x ^ x ^ x
23     return 'Ergebnis'
24
25 # profiling
26 result = teure_funktion()
```

Der profiler erzeugt folgende Ausgabe:

```

1      5000003 function calls in 6.832 seconds
2
3      Ordered by: standard name
4
5      ncalls  tottime  percall  cumtime  percall  filename:lineno(function)
6      5000001    3.346    0.000    3.346    0.000  prof3-de.py:15(get_number)
7           1    3.486    3.486    6.832    6.832  prof3-de.py:19(teure_funktion)
8           1    0.000    0.000    0.000    0.000  {method 'disable' of '_lsprof.Profiler' objects}

```

Man erkennt, welche Funktion langsam ist, aber nicht den eigentlichen Grund.

## 8.4 Line Profiler

Der sog. **line\_profiler** ermöglicht das zeilenweise Profiling eines Programms und ist damit ein sehr mächtiges aber auch langsames Hilfsmittel.

Zunächst muss das Paket `line_profiler` lokal installiert werden:

```

1  mkdir -p $HOME/local/
2  export PYTHONPATH=$PYTHONPATH:$HOME/local
3  easy_install --install-dir $HOME/local/ line_profiler
4  export PATH=$PATH:$HOME/local

```

Bei jeder Verwendung einer neuen shell müssen folgende Umgebungsvariablen neu gesetzt werden:

```
1 export PYTHONPATH=$PYTHONPATH:$HOME/local
2 export PATH=$PATH:$HOME/local
```

Nun soll folgendes script mit dem line\_profiler ausgeführt werden:

```
1 import numpy as np
2 import numpy.random as rdn
3
4 # fuer line_profiler Kommentierung herausnehmen
5 # @profile
6 def test():
7     a = rdn.randn(100000)
8     b = rdn.randn(100000)
9     c = a * b
10
11 test()
```

Die zu untersuchende Funktion wird mit dem decorator `@profile` versehen und anschließende folgender Befehl ausgeführt:

```
1 kernprof -l myscript.py
```

kernprof erzeugt eine Instanz von LineProfiler und kann dann mit dem decorator `@profile` aufgerufen

werden. Per default wird eine lprof-Datei erzeugt, die anschließend mit folgendem Befehl angezeigt werden kann:

```
1 python -m line_profiler myscript.py.lprof
```

Alternativ kann auch direkt beim kernprof-Aufruf die zusätzliche Option `-v` verwendet werden, um direkt folgendes Ergebnis zu erhalten:

```
1 Wrote profile results to myscript2.py.lprof
2 Timer unit: 1e-06 s
3
4 Total time: 0.080464 s
5 File: myscript2.py
6 Function: test at line 5
7
8 Line #      Hits      Time    Per Hit   % Time  Line Contents
9 =====
10      5                @profile
11      6                def test():
12      7             1         2         2.0        0.0         n = 1000000
13      8             1    39044    39044.0       48.5         a = rdn.randn(n)
14      9             1    38967    38967.0       48.4         b = rdn.randn(n)
15     10             1     2451     2451.0        3.0         c = a * b
```

## 8.5 IPython Profiling

Innerhalb von IPython kann auch der profiler bzw. der line\_profiler verwendet werden. Mit dem Magic-Kommando `%prun` kann der profiler für eine vorher geladene Funktion ausgeführt werden:

```

1 In [1]: from myscript import test
2
3 In [2]: %prun test()
4         5 function calls in 0.083 seconds
5
6 Ordered by: internal time
7
8      ncalls  tottime  percall  cumtime  percall filename:lineno(function)
9           2    0.079    0.039    0.079    0.039 {method 'randn' of 'mtrand.RandomState' objects}
10          1    0.003    0.003    0.082    0.082 myscript2.py:7(myfunc)
11          1    0.001    0.001    0.083    0.083 <string>:1(<module>)
12          1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}

```

Der line\_profiler kann folgendermaßen verwendet werden:

```

1 In [1]: %load_ext line_profiler
2
3 In [2]: from myscript2 import test
4
5 In [3]: %lprun -f test test()
6 Timer unit: 1e-06 s
7
8 Total time: 0.086143 s

```

```
9 File: myscript2.py
10 Function: test at line 7
11
12 Line #      Hits      Time  Per Hit   % Time  Line Contents
13 =====
14      7                      def test():
15      8          1          4      4.0      0.0      n = 1000000
16      9          1     44333  44333.0    51.5      a = rdn.randn(n)
17     10          1     39055  39055.0    45.3      b = rdn.randn(n)
18     11          1      2751   2751.0     3.2      c = a * b
```

## 8.6 Weitere Links

Folgende Blogs zeigen weitere Beispiele, wie man Programmcode Profiling durchführt:

- [Optimizing code](#)
- [A guide to analyzing Python performance](#)
- [Profiling Python Like a Boss](#)



## 8.7 Aufgaben

- **Primzahlen**

Profiling Sie den Algorithmus zur Primzahlenberechnung aus dem ersten Python-Kurs. Finden Sie mit dem Timer, cProfile und line\_profiler die Zeilen im Programmcode, die am meisten Zeit verbrauchen (Siehe Aufgabe 6 (b): [Python-Kurs](#), [prim\\_all\\_lsg.py](#)).

## 9 Object I/O (Serializing), Database Access

- Object I/O (Serializing)
- Database Access

## 9.1 Serialisieren mit `Pickle`

Fast jedes Programm besitzt eine Form von persistentem Speicher entweder in Form von Dateien oder Datenbanken. Mit dem `file`-Typ können sämtliche Dateioperationen ausgeführt werden, allerdings gibt es komfortablere Methoden, um persistente Daten in verschiedenen Formen zu speichern.

Um ein Python-Objekt zu speichern und z.B. auf einen anderen Rechner zu übertragen, muß es zunächst in Stringform vorliegen. Dabei handelt es sich um eine spezielle Stringform, die eine Rekonstruktion des Python-Objekts erlaubt.

Ein naiver `str` und `eval` Versuch:

```
1 obj={'one': [1, 'I', 'eins'], 'two': [2, 'II', 'zwei']}
2 obj_as_str=str(obj)
3 type(obj_as_str)
4 obj_as_str
5 obj2=eval(obj_as_str)
6 type(obj2)
7 obj2['two']
8 hex(id(obj)),hex(id(obj2))
```

Mit `str(obj)` wird ein String erzeugt, der mit `eval(obj_as_str)` ausgewertet wird und in ein neues Objekt `obj2` erzeugt wird.

Diese Methode funktioniert allerdings nicht bei rekursiven Datenstrukturen oder eigenen Datentypen:

```
1 class MyType(object):  
2     message='Hallo'  
3     num = 123  
4     def hello(self):  
5         return self.message+' '+str(self.num)  
6  
7 obj=MyType()  
8 obj.hello()  
9 str(obj)  
10 eval(str(obj)) # —> syntax error
```

Es wird also eine allgemeine Methode zum Serialisieren bzw. Deserialisieren benötigt.

Mit dem Modul `pickle` oder `cPickle` und deren Funktionen `dumps` und `loads` bzw. `dump` und `load` können Python-Objekte serialisiert und de-serialisiert werden.

```
1 class MyType(object):
2     message='Hallo'
3     num = 123
4     def hello(self):
5         return self.message+' '+str(self.num)
6
7 obj=MyType()
8 from cPickle import dumps, loads
9 obj_as_str=dumps(obj)
10 obj2=loads(obj_as_str)
11 obj2
12 # <__main__.MyType object at 0xb7cba78c>
13 type(obj2)
14 # <class '__main__.MyType'>
15 obj2.hello()
16 # 'Hallo 123!'
```

Man kann Objekte auch direkt in eine Datei schreiben:

```
1 from cPickle import dump, load
2 out = open('myobj.pickle', 'wb')
3 dump(obj, out)
4 out.close()
5 inp=open('myobj.pickle', 'rb')
6 obj3=load(inp)
7 inp.close()
8 obj3.hello()
9 # 'Hallo 123!'
10 dumps(obj3)
11 # 'ccopy_reg\n_reconstructor\np1\n(c__main__\nMyType\np2\nnc__builtin__\nobject\np3\nNtRp4\nn.'
```

In der letzten Zeile erkennen wir, wie das Python Objekt serialisiert wird.

Allerdings wird nicht die Klassen-Deklaration mit abgespeichert. Diese muss immer im Programm erst bekannt gemacht werden.

Das funktioniert automatisch wenn die Klassendeklaration in externer Datei im momentanen Verzeichnis steht, d.h. man erzeugt eine Datei `mytype.py` mit dem Inhalt:

```
1 class MyType(object):  
2     message='Hallo'  
3     num = 123  
4     def hello(self):  
5         return self.message+ ' '+str(self.num)
```

Speichert ein Objekt ab:

```
1 from cPickle import dump, load  
2 from mytype import MyType  
3 obj=MyType()  
4 obj2=MyType()  
5 obj2.message='Guten Tag'  
6 obj2.hello()  
7 # 'Guten Tag!'  
8 f=open('myobj.pickle','wb')  
9 dump(obj,f)  
10 dump(obj2,f)  
11 f.close()
```

und in einer neuen Python session:

```
1 from cPickle import dump, load
```

```
2 f=open( 'myobj.pickle', 'rb' )
3 obj=load( f )
4 obj2=load( f )
5 obj.hello()
6 # 'Hallo!'
7 obj2.hello()
8 # 'Guten Tag!'
```



## 9.2 DB-API 2.0 SQL-Anbindungen

Relationale Datenbanken werden in der Regel mit SQL angesteuert.

- Relationale Datenbank: Sammlung von Tabellen in denen Datensätze abgespeichert mit eindeutigen Schlüssel.
- SQL: Structured Query Language, Datenbanksprache zur Definition, Abfrage und Manipulation von Daten in relationalen Datenbanken
- Bekannte Datenbanken: SQLite, PostgreSQL, MySQL, Oracle,...
- Mit DB-API 2.0 können verschiedene DBs mit gleichem look-and-feel angesprochen werden

Code Fragment, wie man sich mit einer DB verbindet, eine Tabelle erzeugt, schreibt und wieder liest:

```
1 # nur zur illustration , code laeuft nicht
2 import dbapi
3
4 conn = dbapi.connect(dbname='dbase', host='db.example.com',
5                      user='dbuser', password='pw')
6 curs = conn.cursor()
7
8 curs.execute(''CREATE TABLE atable ( A INT, B VARCHAR(5) )'')
9 curs.execute(''INSERT INTO atable VALUES (%s, %s)'' , (42, 'ABCDE'))
10 curs.execute(''INSERT INTO atable VALUES (%s, %s)'' , (4711, 'ZZZZZ'))
11
12 conn.commit()
13 conn.close()
14
15 curs.execute(''SELECT * FROM atable ORDER BY A'')
16 tpl = curs.fetchone()
17
18 while tpl is not None:
19     print "A", tpl[0], "B", tpl[1]
20     tpl = curs.fetchone()
```

## 9.3 SQLite mit sqlite3

SQLite ist im Gegensatz zu anderen SQL-Datenbanksystemen kein Server sondern eine in **C** geschriebene Bibliothek. Es werden SQL-Abfragen und Statements unterstützt. Dabei wird eine normale index-sequentielle Datei benutzt. SQLite hat geringe Anforderungen an CPU und Speicher.

Ein Schema festlegen:

```
1 elmsheus@laptop:~/python/kurs09$ sqlite3 /tmp/blogdb.sqlite3
2 SQLite version 3.6.10
3 Enter ".help" for instructions
4 Enter SQL statements terminated with a ";"
5 sqlite> CREATE table comments (
6 id INTEGER PRIMARY KEY AUTOINCREMENT,
7 subject TEXT,
8 author TEXT,
9 text TEXT
10 );
11 sqlite> .quit
```

Oder per Datei einlesen:

```
1 elmsheus@laptop:~/python/kurs09$ rm /tmp/blogdb.sqlite3
2 elmsheus@laptop:~/python/kurs09$ sqlite3 /tmp/blogdb.sqlite3 < blogdb.schema
```

Nun kann man per `INSERT` SQL-Statements Tabellen manuell füllen oder per `SELECT` Tabellen abfragen - hierzu verwenden wir das Python Modul `sqlite3`.

Wichtige Befehle:

- `.help`
- `.schema`: Zeigt Datenbankschema an
- `.dump`: Zeigt Schema und Daten einer Datenbank an (Backup)

Werte in Datenbank einfügen:

```
1 sqlite> INSERT INTO comments VALUES(  
2 1,  
3 'The Python Blog',  
4 'Max Mustermann',  
5 'How are you ?'  
6 );  
7 sqlite> INSERT INTO comments VALUES(  
8 2,  
9 'The Perl Blog',  
10 'Marie Mustermann',  
11 'How are you ?'  
12 );
```

Werte abfragen:

```
1 sqlite> SELECT subject FROM comments ORDER by author;  
2 The Perl Blog  
3 The Python Blog  
4 sqlite> SELECT subject ,text FROM comments ORDER by author;  
5 The Perl Blog|How are you ?  
6 The Python Blog|How are you ?
```

Verbinden der SQLite Datenbank in Python mit `sqlite3` Modul (nicht autocommit-Modus):

```
1 import sqlite3
2
3 conn = sqlite3.connect( '/tmp/blogdb.sqlite3', isolation_level='DEFERRED' )
4 conn
5 # <sqlite3.Connection object at 0x892a4a0>
```

Wichtige Methoden:

- `close`: Schließt Datenbank-Verbindung. Ist autocommit ausgeschaltet, werden Transaktionen ohne `commit` nicht ausgeführt.
- `commit`: schließt offene Transaktionen ab und speichert diese in der Datenbank
- `rollback`: Transaktion abbrechen.
- `cursor`: `Cursor`-Objekt zur Datenbankabfrage, Daten eintragen, verändern, löschen.

```
1 curs = conn.cursor()
```

Wichtige `cursor` Methoden:

- `close`, `execute`, `executemany`, `fetchone`, `fetchall`

Daten in Datenbank eintragen:

```
1 curs.execute('''INSERT INTO comments VALUES (? , ?, ?, ?)''', (None, 'a subject', 'an author', 'a
    text'))
2 # <sqlite3.Cursor object at 0x892b650>
3 curs.rowcount
4 # 1
5 curs.execute('''INSERT INTO comments VALUES (? , ?, ?, ?)''', (None, 'another subject', 'another
    author', 'another text'))
6 # <sqlite3.Cursor object at 0x892b650>
7 curs.rowcount
8 # 1
9 conn.commit()
```

Aus Sicherheitsgründen (SQL injection vulnerability) sollte immer mit Platzhaltern '?' gearbeitet werden.

```
1 curs.execute('''SELECT * FROM comments ORDER BY id ''')
2 # <sqlite3.Cursor object at 0x892b650>
3 curs.rowcount
4 # -1
5 curs.fetchone()
6 # (1, u'The Python Blog', u'Max Mustermann', u'How are you ?')
7 curs.fetchone()
8 # (2, u'The Perl Blog', u'Marie Mustermann', u'How are you ?')
9 curs.fetchone()
10 # (3, u'a subject', u'an author', u'a text')
11 curs.fetchone()
```

```
12 # (4, u'another subject', u'another author', u'another text')
13 curs.execute(''SELECT * FROM comments ORDER BY id'')
14 # <sqlite3.Cursor object at 0x892b650>
15 result=curs.fetchone()
16 while result is not None:
17     print result
18     result = curs.fetchone()
19
20 # (1, u'The Python Blog', u'Max Mustermann', u'How are you ?')
21 # (2, u'The Perl Blog', u'Marie Mustermann', u'How are you ?')
22 # (3, u'a subject', u'an author', u'a text')
23 # (4, u'another subject', u'another author', u'another text')
```

Man kann auch mit `fetchall` gleich alle Einträge in Liste einlesen, aber zumindest bei grossen Datensätzen ist es aus Speichergründen besser `fetchone` oder `fetchmany` zu verwenden.

Datensätze verändern oder löschen:

```
1 curs.execute(''UPDATE comments SET author=? WHERE id=?'', ('me',2))
2 # <sqlite3.Cursor object at 0x892b650>
3 curs.rowcount
4 # 1
5 conn.commit()
6 curs.execute(''SELECT * FROM comments WHERE id=?'', (2,))
7 # <sqlite3.Cursor object at 0x892b650>
8 curs.fetchone()
```



```
9 # (2, u'The Perl Blog', u'me', u'How are you ?')
10 curs.execute(''DELETE FROM comments WHERE id<'', (3,))
11 # <sqlite3.Cursor object at 0x892b650>
12 curs.rowcount
13 # 2
14 conn.rollback()
15 curs.execute(''SELECT count(*) FROM comments'')
16 # <sqlite3.Cursor object at 0x892b650>
17 curs.fetchone()
18 # (4,)
19 curs.execute(''SELECT * FROM comments ORDER BY id'')
20 # <sqlite3.Cursor object at 0x892b650>
21 curs.fetchall()
22 # [(1, u'The Python Blog', u'Max Mustermann', u'How are you ?'),
23 #  (2, u'The Perl Blog', u'me', u'How are you ?'),
24 #  (3, u'a subject', u'an author', u'a text'),
25 #  (4, u'another subject', u'another author', u'another text')]
26 conn.close()
```

Soweit nur absolute Basics für Database Nutzung. **Relationale Datenbanken** bieten viele weitere Features, insbesondere zum Verknüpfen und Cross-Referenzieren verschiedener Tabellen. Gute Einführung in **Software Carpentry: Databases and SQL**.

## 9.4 Aufgaben

1. Sie haben folgende Daten vorliegen:

```
Max Mueller 123498765 1.3
Lise Meier 123498764 2.0
Heinz Schmitt 123498761 1.0
Maria Kunz 123498762 3.0
Karl Hinz 12349879 3.3
Anna Muster 12349878 1.7
```

Speichern Sie diese Daten in eine ASCII-Datei, definieren Sie einen geeigneten Datentyp, lesen Sie die Daten aus der ASCII-Datei ein, und speichern diese Daten im `pickle`-Format ab. Lesen sie diese Daten anschliessend wieder aus der `pickle`-Datei ein und überprüfen Sie die Konsistenz der Daten.

Lösung: `test_pickle.py`

2. Erzeugen Sie eine Tabelle in `SQLite3` mit den vorhergehenden Daten und lesen Sie diese mit der `cursor` Funktion aus.

Lösung: `studenten.schema`

Lösung: `test_sqlite3.py`

3. Beliebtes Beispiel für Datenanalyse ist die Passagierliste der Titanic, `titanic.db` ist sqlite3 Variante

des Datensatzes. Laden Sie den Datensatz herunter und öffnen Sie es mit sqlite3 oder Python. Vergleichen Sie die Überlebensraten getrennt für Klassen und Geschlecht.

Hinweis: Man kann es direkt mit sqlite3 und passenden SQL Kommandos machen

```
sqlite3 titanic.db
...
.tables
.schema titanic
...
select survived,class,sex from titanic;
...
```

Weitere Anweisungen z.B. in [SQLite Tutorial](#)

Oder in Python Daten einlesen und mit Python Operationen bestimmen.

Lösung: [titanic.sqlite](#)

Lösung: [titanic.py](#)

4. Machen Sie das SQLite/DB Tutorial auf [Software Carpentry: Databases and SQL](#).  
Hier die SQLite Datei mit den Tabellen: [survey.sqlite](#)

## 10 XML und JSON

- Einführung und HTML
- XML Basics
- XML Prozessieren mit SAX
- JSON – Lightweight Alternative
- ... und Aufgaben ...


## 10.1 HTML

HTML - Hyper Text Markup Language, die Basis für Webseiten

14.10.2009 19:42 [Login](#) | [Mobil](#) | [RSS](#) | [Schlagzeilen](#) | [Startseite](#) | [Wetter](#) [sued-café](#) | [jetzt.de](#) | [SZ-Magazin](#)

---

# sueddeutsche.de

**Bernard Madoff**  
Der Betrüger als  
Partyschreck 


---

[Home](#) | [E-Paper](#) | [Immobilienmarkt](#) | [Stellenmarkt](#) | [Motormarkt](#) | [Anzeigen](#) | [SZ-Shop](#) | [Abo & Service](#) | [Tickets](#)

[Politik](#) | [Wirtschaft](#) | [Geld](#) | [Kultur](#) | [Sport](#) | [Leben](#) | [Karriere](#) | **NEU** [München](#) | [Bayern](#) | [Panorama](#) | [Auto](#) | [Digital](#) | [Wissen](#) | [Fitness](#) | [Reise](#)

[Bundestagswahl](#) | [Tag in Kürze](#) | [Video](#) | [Bilder](#) | [Kaufdown](#)  [Finden](#)

---




Neue Identität der "Mona Lisa"

## Frauentausch in Öl

Gelüftetes Lächelgeheimnis: Seit längerem äußern Forscher Zweifel daran, dass auf Leonardo da Vincis Gemälde "La Gioconda" die Gattin eines Florentiner Kaufmanns zu sehen ist. Der Historiker Roberto Zapperi hat nun herausgefunden, wer es tatsächlich sein könnte. *Interview: Kia Vahland* [mehr ...](#)

[Meinung](#) | [Kolumne](#) | [Forum](#)



Thomas Hummel  
**Komm und spür es!**

## FLUGELFLITZER

Kolumne: Deutscher Alltag [Die DDR lebt](#)  
[Medizin und Wahnsinn \(100\)](#) [Moorkekse an Holzkohle-Mus](#)  
[weitere Kolumnen](#)

[SZ unverbindlich testen](#) - [hier kostenlose Leseprobe](#)

[IQ-Test: Wie klug sind Sie?](#)

... ist eine Art XML:

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2 <html>
3 <head>
4     <title>Nachrichten aus Politik , Kultur , Wirtschaft und Sport – sueddeutsche.de</title>
5     <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
6     ....
7     </head>
8 <body class="sueddeutsche">
9 <div class="sueddeutsche"><a name="top"></a>
10 <div id="wrapper">
11 <!-- lbox --><div id="metanav">
12     <p><span>Willkommen bei Sueddeutsche.de! Es ist jetzt </span>14.10.2009 <span>um </span>
        >19:42</p>
13
14     <ul class="nav">
15         <li class="login"><a href="#" onclick="ns onclick (this ,'', 'onclick_navigation.lbox.
            login ','clickin '); return false;">Login</a></li>
16         <li class="mobil"><a href="http://www.sueddeutsche.de/service/app/187/81106/" onclick="
            ns onclick (this ,'', 'onclick_navigation.lbox.mobil ','clickin '); return false;" rel="
            nofollow">Mobil</a></li>
17     </ul>
18     ...
19 </div>
20 ... etc

```

Das Layout wird nicht direkt beim Erstellen des Dokuments festgelegt (wie in Standard Office Programmen) sondern man spezifiziert (=markiert) mit sog. **tags** bestimmte Kategorien *Überschrift, Tabelle, Liste, ...* die in HTML vordefiniert sind.

Interpretation und Darstellung ist dann Sache des Web-Browsers (bzw. Einstellungen)



## 10.2 XML Basics

XML - eXtensible Markup Language ist dasselbe Konzept: Der Inhalt wird mit **tags** klassifiziert

Nur sind die tags in XML nicht vorgegeben sondern frei definierbar (=eXtensible).

- Universeller Standard für Datenaustausch zwischen verschiedenen Anwendungen (Application Neutrality):

*Erzeugen mit Java und Weiterverarbeiten mit Python, ...*

Unabhängig von Plattform (Unix, Windows, Mac, ...)

- Erweiterbar ohne dass vorhandene Anwendungen angepasst werden müssen
- Direkt lesbar und mit einfachem Texteditor zu bearbeiten (mehr oder weniger ...)

## Erstes einfaches Beispiel:

```
1 <?xml version="1.0" encoding="ISO-8859-1"? >
2 <Class>
3   <Student name="Mueller" prename="Max">
4     <ID>123498765</ID>
5     <Grade>2.0</Grade>
6   </Student>
7   <Student name="Kunz" prename="Maria">
8     <ID>123498762</ID>
9     <Grade>3.0</Grade>
10  </Student>
11 </Class>
12 <Student name="Hinz" prename="Karl">
13   <ID>12349879</ID>
14   <Grade>3.3</Grade>
15 </Student>
16 </Class>
```

Ganz ähnlich wie HTML, nur kann man eben Datenfelder bzw. Tags selbst bestimmen.

## 10.3 XML Struktur im Detail

- XML Dokument beginnt mit XML declaration:

```
<?xml version="1.0" encoding="ISO-8859-1"? >
```

Spezifiziert die XML Version und ggf. das sog. Character Encoding

- Der Textbereich von `<Class>` bis zu `</Class>` ist ein *XML Element*. Elemente müssen open bzw close Tag haben ( `<Class>` bzw `</Class>` )
- Elemente können weitere Elemente enthalten, wie im Beispiel `Student` das `Grade` Element.
- Ein Element kann ausserdem noch Attribute haben, wie bei `Student` noch das Attribut `(name="Paul")`
- Reihenfolge der Sub-Elemente spielt keine Rolle ...
- Flexibel erweiterbar, weitere Elemente können eingefügt werden, z.B. `BachelorGrade`, existierende Anwendungen laufen weiterhin  
(im Gegensatz zu sonstigen Formaten wie ASCII Text, CSV, Pickle, SQL, ...)

## Character Data und Markup

- Markup tags: beginnt mit `<` oder `&` und endet mit `>` oder `;`
- Character Data: Jeder Character String dazwischen, darf nicht direkt Start eines Markup Tags beinhalten.
- XML Text - Markup und Character Data zusammen bilden den XML Text und zusammen mit Declaration Header das XML Dokument.

Der Markup ist eine Art Schema für die Interpretation der Character Data

- Kommentare können als `<!--comment-->` eingefügt werden

## Element Tag

- Der Element Tag beinhaltet bzw definiert seinen Typ und Struktur
- Das Start Tag kann Attribute enthalten, die Reihenfolge spielt keine Rolle: `<Value a="12" b="16">` ist das gleiche wie `<Value b="16" a="12" >`
- Attribut Werte müssen in Quotes stehen: `attribute="value"`
- Das End Tag muss exakt dem Namen des Start Tags entsprechen

## Document Types und Schemas

- Document Typ ist eine Art Datendefinition für ein XML Dokument
- Spezifiziert die erlaubte Struktur – ähnlich wie eine Klasse und ihre Membervariablen die Datenstruktur vorgeben
- ... wird auch *Schema* genannt
- Eine Möglichkeit dies festzulegen ist die **DTD**: Document Type Definition – es gibt aber noch etliche Alternativen

## DTDs

- Kann im Dokument als Vorspann integriert sein oder auch separat als `.dtd` File
- DTDs ermöglichen eine Art Typdefinition für Dokumente, die die korrekte Struktur festlegt
- Damit kann ein XML Dokument validiert werden, wird von manchen XML Parseern automatisch gemacht.
- wir beschränken uns auf simple Beispiele ...

## Integrierte DTD:

```
1 <?xml version="1.0"?>
2 <!DOCTYPE product [
3   <!ELEMENT name (#PCDATA)>
4   <!ELEMENT price (#PCDATA)>
5   <!ELEMENT product (name, price)>
6 ]>
7 <product>
8   <name>Bean Crusher< /name>
9   <price>3.95< /price>
10  </product>
```



## 10.4 SAX Parser

SAX ist ein weitverbreiteter XML Parser, Versionen für Perl, Python, Java, ... teilt SAX verarbeitet Dokumente als *streams*, d.h. das Dokument wird abschnittsweise verarbeitet.

Alternativer gängiger Parser ist DOM (Document Object Model), der ganzes Dokument liest und intern im Memory bereitstellt ⇒ gut geeignet für komplexe baum-artige XML Strukturen.

Verwendung:

- *Handler Klasse* definieren
- Vorgegebene Methoden der Klasse implementieren: `startElement`, `endElement`, ...
- Diese werden beim "Document-Parsing" aufgerufen, ähnlich wie call-back Funktionen in TkInter-GUI
- Die Handler Klassen wird SAX übergeben vor dem Document-Parsing

## XML Beispiel (article.xml):

```
1 <?xml version="1.0"?>
2 <webArticle category="news" subcategory="technical">
3   <header title="NASA Builds Warp Drive"
4     length="3k"
5     author="Joe Reporter"
6     distribution="all"/>
7   <body>Seattle , WA – Today an anonymous individual
8     announced that NASA has completed building a
9     Warp Drive and has parked a ship which uses
10    the drive in his back yard. This individual
11    claims that although he hasn't been contacted by
12    NASA concerning the parked space vessel, he assumes
13    that he will be launching it later this week to
14    mount an exhibition to the Andromeda Galaxy.
15  </body>
16 </webArticle>
```

Handler-KlasseXML (simplehandler.py):

---

```
#!/usr/bin/env python 1
# 2
from xml.sax.handler import ContentHandler 3
class ArticleHandler(ContentHandler): 4
    """A handler to deal with articles in XML""" 5
    def startElement(self, name, attrs): 6
        print "Start element:",name 7
    def endElement(self, name): 8
        print "End element:",name 9
def main(): 10
    import sys 11
    12
    from xml.sax import make_parser 13
    14
    ch = ArticleHandler() 15
    saxparser = make_parser() 16
    17
    saxparser.setContentHandler(ch) 18
```

---

saxparser.parse(sys.argv[1])	19
if __name__ == '__main__':	20
main()	21
# call as	22
# python simplehandler.py article.xml	23

---

## Etwas mehr XML processing ...

- Handler erweitern und tag-spezifisches Prozessieren einbauen
- zusätzliche Methode `characters()` zum Prozessieren der eigentlichen Daten.

---

```
from xml.sax.handler import ContentHandler 1
class ArticleHandler(ContentHandler):      2
    """A handler to deal with articles in XML""" 3
    inArticle = 0                          4
    inBody    = 0                          5
    isMatch    = 0                         6
    title      = ""                        7
    body       = ""                        8
    def startElement(self, name, attrs):    9
        if name == "webArticle":          10
            subcat = attrs.get("subcategory", "") 11
            if subcat.find("tech") > -1:      12
                self.inArticle = 1           13
                self.isMatch = 1            14
            elif self.inArticle:             15
```

```
if name == "header": 16
    self.title = attrs.get("title", "") 17
if name == "body": 18
    self.inBody = 1 19
def characters(self, characters): 20
    MAXLEN=800 21
    if self.inBody: 22
        if len(self.body) < MAXLEN: 23
            self.body += characters 24
        if len(self.body) > MAXLEN: 25
            self.body = self.body[:MAXLEN-2] + "... " 26
            self.inBody = 0 27
    def endElement(self, name): 28
        if name == "body": 29
            self.inBody = 0 30
def main(): 31
    import sys 32
    33
    from xml.sax import make_parser 34
    35
```

---

```
ch = ArticleHandler()                                     36
saxparser = make_parser()                                 37
                                                         38
saxparser.setContentHandler(ch)                           39
saxparser.parse(sys.argv[1])                               40
if ch.isMatch:                                             41
    print "News Item!"                                     42
    print "Title:", ch.title                               43
    print "Body:", ch.body                                 44
if __name__ == '__main__':                                 45
    main()                                                  46
# call as                                                  47
# python texthandler.py article.xml                       48
```

---

## 10.5 JSON – Lightweight Alternative

Zunehmend populär für Datenaustausch wurde in den letzten Jahren JSON (JavaScript Object Notation). JSON Dokumente sind im Prinzip JavaScript Anweisungen, die Syntax ist sehr ähnlich zu Python und etwas kompakter als XML. Die Daten werden als Python Dicts bzw Listen/Dicts von Dicts behandelt.

Beispiel (aus [Wikipedia](#)):

```
1 {  
2   "Herausgeber": "Xema",  
3   "Nummer": "1234–5678–9012–3456",  
4   "Deckung": 2e+6,  
5   "Waehrung": "EURO",  
6   "Inhaber": {  
7     "Name": "Mustermann",  
8     "Vorname": "Max",  
9     "maennlich": true,  
10    "Hobbys": [ "Reiten", "Golfen", "Lesen" ],  
11    "Alter": 42,  
12    "Kinder": [],  
13    "Partner": null  
14  }  
15 }
```



In Python verarbeiten:

```
1 import json
2 f=open( 'ex.json' )
3 a=json.load( f )
4 a
5 # {u'Inhaber': {u'Hobbys': [u'Reiten', u'Golfen', u'Lesen'], u'Name': u'Mustermann', u'Kinder':
6   #   [], u'Vorname': u'Max',
7   # u'maennlich': True, u'Partner': None, u'Alter': 42}, u'Waehrung': u'EURO', u'Herausgeber': u'
8   #   Xema',
9   # u'Nummer': u'1234-5678-9012-3456', u'Deckung': 2000000.0}
10 a[ 'Nummer' ]
11 # u'1234-5678-9012-3456'
12 a[ 'Inhaber' ][ 'Alter' ]
13 # 42
```

## 10.6 Daten I/O – Summary

Viele Möglichkleiten und Technologien für Daten I/O.

Keine allgemeine Lösung , Auswahl gemäss Problem und Anforderungen.

- ASCII Text Ausgabe:  
**Pro:** Einfach, portabel.  
**Con:** Keine Struktur, keine komplexen Daten.
- Binäres Format wie Pickle:  
**Pro:** Schnell, effizient, Speicherung komplexer Objektstrukturen.  
**Con:** Python-spezifisch
- (Relationale) Datenbank (SQLite, Postgres, Oracle, ...):  
**Pro:** Speicherung beliebiger komplexer Daten, Verknüpfungen zwischen Tabellen.  
**Con:** Sperrige Syntax, kein direkter IO von Objekten.
- Strukturierte Ausgabe mit XML oder JSON:  
**Pro:** komplexe Strukturen möglich, les- (und editier-) bares Format, weit verbreiteter Standard für Datenaustausch  
**Con:** ineffizient, speicher-intensiv, kein direkter IO von Objekten

## 10.7 Aufgaben

### 1. SAX

Testen Sie die vorgestellten Beispiele zu XML Prozessieren mit SAX.

### 2. Real-world Beispiel zu XML – 1

Datenaustausch via XML ist heute Quasi-Standard, viele Anwendungen bieten zumindest optional Ausgabe in XML Format an. Dadurch einfaches weiterverarbeiten möglich.

Praktisches Beispiel aus unserem Bereich ist SGE Batch System am LRZ Linux Cluster. Das Batch System verwaltet "Jobs" auf einem Rechnercluster und liefert Informationen zum Status der laufenden Jobs wie z.B. Job-Id, verbrauchte CPU Zeit, Start-Zeit, Memory Bedarf, u.v.m. Diese Infos sind wichtig für Anwender, aber auch für Monitoring Systeme um die Auslastung des Clusters zu Überwachen.

Eine Beispiel-Ausgabe enthält die xml-Datei [SGE-qstat.xml](#), und ein simpler SAX Parser ist [SGE-qstat-parse.py](#).

- (a) Finden Sie Job-id, User-name, Startzeit und verbrauchte CPU Zeit der Jobs
- (b) Wichtige Monitoring Grösse ist der Vergleich verbrauchte CPU Zeit zu abgelaufener Wallclock-time. Niedrige Werte sind Anzeichen für I/O lastige Jobs oder Probleme mit Netzwerk oder Storage Systemen. Bestimmen Sie CPU/Wallclock-time Verhältnis der Jobs und tragen Sie es in ein Histogramm ein (matplotlib)

(Lösungsbeispiel: [SGEqstatparse2.py](#)).

### 3. Real-world Beispiel zu XML – 2

Weiteres Beispiel sind GPS Geräte, die für Navigation oder Freizeitsport verwendet werden. Die Aufzeichnung der “Tracks” erfolgt oft im sogenannten “gpx” Format, das ist in XML definiert und erlaubt leichten Austausch der aufgezeichneten Tracks bzw. die Weiterverarbeitung der Daten.

Eine Beispiel-Track von MTB Tour in den Alpen enthält die Datei [GPS\\_example.gpx](#), mit Infos zu Position, Datum/Uhrzeit, Höhe und Herzfrequenz. Ein simples Parsing Skript ist [GPX-simple-parse.py](#).

(a) Machen Sie weiteren Plot zum Verlauf der Herzfrequenz ...

(b) ... und zur Geschwindigkeit. Dazu müssen Sie Abstand der einzelnen Punkte und die Zeitdifferenz berechnen. Hilfsfunktionen dafür sind in [gpxutil.py](#).

(Lösungsbeispiel: [GPX-parse.py](#). Und natürlich gibt es schon viele Programme zum Anzeigen von GPX Dateien, z.B. <http://www.atlsoft.de/gpx/>. )

### 4. JSON Beispiel

Der Inhalt von Tabellen bzw Spreadsheets kann bequem in JSON Format umgewandelt werden und dann in Python als Liste von dicts eingelesen werden, jede Zeile ist dict-Element einer List. Die Spalten-Header sind die **keys** des dicts.

Die Datei [wlcg.json](#) wurde von <https://wlcg-rebus.cern.ch/apps/pledges/resources/> heruntergeladen und enthält die *Pledges* (=Verpflichtungen) einer Vielzahl von Rechenzentren, die Ressourcen für die Datenanalyse bei LHC bereitstellen.

Lesen Sie die Datei in python über json als Liste von Dicts ein, geben Sie die Disk Kapazität der Sites aus Germany für ATLAS aus und bestimmen Sie die Summe.

(Lösungsbeispiel: [wlcg\\_json.py](#)).

## 11 Pandas – Datenanalyse mit Python

Pandas ist ein Set von Python Modulen für Datenanalyse, d.h.

- IO Tools für Text-Files, CSV Dateien, Web-Sever, u.v.m.
- Tools zur Manipulation bzw. Interpretation der Daten, insbesondere Time-Series (Zeitreihen)
- Bequeme Visualisierung
- Viele mächtige Features zur Manipulation, Kombination, Gruppierung, Auswahl der Daten, hier nur einige grundlegende (aber interessante) Beispiele
- Einige wenige geschickt platzierte Anweisungen in Pandas ermöglichen komplexes *data-mining*, das mit üblichem Programmieren nur mit großem Aufwand zu erreichen wäre.

## 11.1 Datenformate

Grundlage von Pandas sind spezielle Array Typen, die auf numpy Arrays aufbauen.

- **Series:** ein-dimensionales Array, kann verschiedene Datentypen enthalten.
- **DataFrame:** zwei-dimensionales Array, Tabelle mit Spalten verschiedener Datentypen, sehr flexibel verwendbar, ideal zum Prozessieren von Spread-Sheets.
- **Panel:** drei-dimensionales Array ...

Pandas Setup:

```
1 In [1]: import pandas as pd
2 In [2]: import numpy as np
3 In [3]: import matplotlib.pyplot as plt
```



## Pandas–Series:

```
1 In [4]: obj=pd.Series([4,3,5,-6])
2
3 In [5]: obj
4 Out[5]:
5 0      4
6 1      3
7 2      5
8 3     -6
9 dtype: int64
```

Series nicht einfach Array sondern hat assoziierten Index (=Data Labels).

```
1 In [7]: obj.values
2 Out[7]: array([ 4,  3,  5, -6])
3 In [8]: obj.index
4 Out[8]: Int64Index([0, 1, 2, 3], dtype='int64')
```

Index kann auch explizit angegeben werden:

```
1 In [9]: obj=pd.Series([4,3,5,-6],index=['a','b','c','d'])
2
3 In [10]: obj
4 Out[10]:
5 a      4
```

```
6 | b      3
7 | c      5
8 | d     -6
9 | dtype: int64
10 |
11 | In [11]: obj.index
12 | Out[11]: Index([u'a', u'b', u'c', u'd'], dtype='object')
```

Übliche Numpy Array Operations funktionieren:

```
1 | In [12]: obj[obj>3]
2 | Out[12]:
3 | a      4
4 | c      5
5 | dtype: int64
6 |
7 | In [13]: obj**2
8 | Out[13]:
9 | a     16
10 | b      9
11 | c     25
12 | d     36
13 | dtype: int64
14 |
15 | In [14]: np.exp(obj)
16 | Out[14]:
```

```
17 | a      54.598150
18 | b      20.085537
19 | c     148.413159
20 | d       0.002479
21 | dtype: float64
```

## Pandas–DataFrame:

DataFrames haben sowohl Zeilen- als auch Spalten-Index.

Erzeugung direkt aus Dict mit Listen:

```
1 dates = pd.date_range('20130101', periods=6)
2 df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
3
4 In [19]: df
5 Out[19]:
6           A          B          C          D
7 2013-01-01 -0.281995 -0.020310 -0.471895 -0.660280
8 2013-01-02  0.384913  0.080218  1.737480 -0.316140
9 2013-01-03  1.014396 -1.278679  0.873290 -0.339593
10 2013-01-04 -0.091760 -0.295883 -1.116633  0.434963
11 2013-01-05 -0.314056  1.311067  0.411571 -2.908431
12 2013-01-06  0.825971 -0.748699  0.828519  0.166611
13
14
15 # beliebige Datentypen sind moeglich
16 In [15]: df2 = pd.DataFrame({ 'A' : 1.,
17     ....: 'B' : pd.Timestamp('20130102'),
18     ....: 'C' : pd.Series(1, index=list(range(4)), dtype='float32'),
19     ....: 'D' : np.array([3] * 4, dtype='int32'),
20     ....: 'E' : pd.Categorical(["test","train","test","train"]),
21     ....: 'F' : 'foo' })
```

```
22 |
23 | In [16]: df2
24 | Out[16]:
25 |      A      B  C  D      E  F
26 | 0  1 2013-01-02  1  3   test  foo
27 | 1  1 2013-01-02  1  3  train  foo
28 | 2  1 2013-01-02  1  3   test  foo
29 | 3  1 2013-01-02  1  3  train  foo
30 |
31 |
32 | In [20]: df2.dtypes
33 | Out[20]:
34 | A      float64
35 | B  datetime64[ns]
36 | C      float32
37 | D      int32
38 | E      category
39 | F      object
40 | dtype: object
```

Datenbereiche auswählen:

```
1 | # Spalte
2 | In [59]: df['A']
3 | Out[59]:
4 | 2013-01-01    -0.281995
```

```
5 2013-01-02    0.384913
6 2013-01-03    1.014396
7 2013-01-04   -0.091760
8 2013-01-05   -0.314056
9 2013-01-06    0.825971
10 Freq: D, Name: A, dtype: float64
11
12 In [61]: df.A
13 Out[61]:
14 2013-01-01   -0.281995
15 2013-01-02    0.384913
16 2013-01-03    1.014396
17 2013-01-04   -0.091760
18 2013-01-05   -0.314056
19 2013-01-06    0.825971
20 Freq: D, Name: A, dtype: float64
21
22 In [63]: t=dates[3]
23
24 # Zeile ueber Namen
25 In [64]: df.loc[t]
26 Out[64]:
27 A   -0.091760
28 B   -0.295883
29 C   -1.116633
30 D    0.434963
```

```
31 Name: 2013-01-04 00:00:00, dtype: float64
32
33 # Zeile ueber Index
34 In [65]: df.iloc[3]
35 Out[65]:
36 A    -0.091760
37 B    -0.295883
38 C    -1.116633
39 D     0.434963
40 Name: 2013-01-04 00:00:00, dtype: float64
```

## 11.2 Funktionen zum Daten aufbereiten

```
1 In [48]: df3 = df.copy()
2 In [46]: t=df3.index[3]
3
4 # Zeile loeschen
5
6 In [49]: df3.drop(t)
7 Out[49]:
8           A          B          C          D
9 2013-01-01 -0.281995 -0.020310 -0.471895 -0.660280
10 2013-01-02  0.384913  0.080218  1.737480 -0.316140
11 2013-01-03  1.014396 -1.278679  0.873290 -0.339593
12 2013-01-05 -0.314056  1.311067  0.411571 -2.908431
13 2013-01-06  0.825971 -0.748699  0.828519  0.166611
14
15 # Spalte loeschen, Achse angeben und Dataframe direkt aendern
16 In [50]: df3.drop('A',axis=1,inplace=True)
17
18 In [51]: df3
19 Out[51]:
20           B          C          D
21 2013-01-01 -0.020310 -0.471895 -0.660280
22 2013-01-02  0.080218  1.737480 -0.316140
23 2013-01-03 -1.278679  0.873290 -0.339593
24 2013-01-04 -0.295883 -1.116633  0.434963
```



```

25 2013-01-05  1.311067  0.411571  -2.908431
26 2013-01-06 -0.748699  0.828519  0.166611
27
28
29 # Spalte hinzufuegen
30 In [70]: df4 = df.copy()
31
32 In [71]: df4['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
33
34 In [73]: df4
35 Out[73]:
36           A          B          C          D          E
37 2013-01-01 -0.281995 -0.020310 -0.471895 -0.660280  one
38 2013-01-02  0.384913  0.080218  1.737480 -0.316140  one
39 2013-01-03  1.014396 -1.278679  0.873290 -0.339593  two
40 2013-01-04 -0.091760 -0.295883 -1.116633  0.434963  three
41 2013-01-05 -0.314056  1.311067  0.411571 -2.908431  four
42 2013-01-06  0.825971 -0.748699  0.828519  0.166611  three
43
44 # Auswaehlen nach Inhalt
45 In [72]: df4[df4['E'].isin(['two', 'four'])]
46 Out[72]:
47           A          B          C          D          E
48 2013-01-03  1.014396 -1.278679  0.873290 -0.339593  two
49 2013-01-05 -0.314056  1.311067  0.411571 -2.908431  four

```

Und noch viel mehr...

```

1 In [78]: df
2 Out[78]:
3           A          B          C          D
4 2013-01-01 -0.281995 -0.020310 -0.471895 -0.660280
5 2013-01-02  0.384913  0.080218  1.737480 -0.316140
6 2013-01-03  1.014396 -1.278679  0.873290 -0.339593
7 2013-01-04 -0.091760 -0.295883 -1.116633  0.434963
8 2013-01-05 -0.314056  1.311067  0.411571 -2.908431
9 2013-01-06  0.825971 -0.748699  0.828519  0.166611
10
11 In [79]: df.apply(np.cumsum)
12 Out[79]:
13           A          B          C          D
14 2013-01-01 -0.281995 -0.020310 -0.471895 -0.660280
15 2013-01-02  0.102918  0.059907  1.265585 -0.976420
16 2013-01-03  1.117314 -1.218772  2.138875 -1.316014
17 2013-01-04  1.025554 -1.514655  1.022242 -0.881051
18 2013-01-05  0.711498 -0.203588  1.433813 -3.789482
19 2013-01-06  1.537469 -0.952287  2.262331 -3.622871
20
21 In [80]: df.apply(np.exp)
22 Out[80]:
23           A          B          C          D
24 2013-01-01  0.754278  0.979895  0.623819  0.516707
25 2013-01-02  1.469486  1.083523  5.683004  0.728957

```

```
26 2013-01-03  2.757696  0.278405  2.394776  0.712060
27 2013-01-04  0.912324  0.743875  0.327380  1.544906
28 2013-01-05  0.730478  3.710131  1.509187  0.054561
29 2013-01-06  2.284098  0.472981  2.289924  1.181294
30
31 In [81]: df.mean()
32 Out[81]:
33 A      0.256245
34 B     -0.158714
35 C      0.377055
36 D     -0.603812
37 dtype: float64
38
39 In [82]: df.sum()
40 Out[82]:
41 A      1.537469
42 B     -0.952287
43 C      2.262331
44 D     -3.622871
45 dtype: float64
```

## 11.3 Beispiel–1: WLCG Spreadsheet

```
1 # json einlesen
2 dwlcg = pd.read_json('https://wlcg-rebus.cern.ch/apps/pledges/resources/2016/all/json')
3 #
4 dwlcg.ATLAS.sum()
5 # Fehler wg Mischung strings und Zahlen
6 # —> leere Werte = leere Strings —> Null setzen
7 dwlcg[dwlcg==""]=0
8 # get sum per country and Pledgetype
9 r = dwlcg.groupby(['PledgeType', 'Country'])['ATLAS'].sum()
10 # Liefert Multi-index Series
11 r['CPU']
12 ....
13 r['CPU', 'UK']
14 108249
15 r[:, 'UK']
16 # back to DataFrame
17 dr=r.unstack()
18 dr['UK']['CPU']
```

## 11.4 Beispiel–2: Analyse Fussballdaten

Nach Artikel in Zeitschrift iX Nov/Dec 2015, Datenquelle <http://www.football-data.co.uk/mmz4281/1516/D1.csv>

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 # direct read from url
4 data = pd.read_csv('http://www-static.etp.physik.uni-muenchen.de/kurs/Computing/sw/source/D1s.
      csv')
5 # data = pd.read_csv('D1s.csv')
6 data
7
8      Date      Heim      Auswaerts  HTore  ATore
9 0  14/08/15  Bayern Munich      Hamburg      5      0
10 1  15/08/15      Augsburg      Hertha      0      1
11 2  15/08/15      Darmstadt      Hannover      2      2
12 3  15/08/15      Dortmund      M'gladbach      4      0
13 4  15/08/15      Leverkusen      Hoffenheim      2      1
14 5  15/08/15      Mainz      Ingolstadt      0      1
15 6  15/08/15  Werder Bremen      Schalke 04      0      3
16 7  16/08/15      Stuttgart      FC Koln      1      3
17 8  16/08/15      Wolfsburg      Ein Frankfurt      2      1
18 9  21/08/15      Hertha      Werder Bremen      1      1
19 10 22/08/15  Ein Frankfurt      Augsburg      1      1
20 11 22/08/15      FC Koln      Wolfsburg      1      1
21 12 22/08/15      Hamburg      Stuttgart      3      2

```

21	13	22/08/15	Hannover	Leverkusen	0	1
22	14	22/08/15	Hoffenheim	Bayern Munich	1	2
23	15	22/08/15	Schalke 04	Darmstadt	1	1
24	16	23/08/15	Ingolstadt	Dortmund	0	4
25	17	23/08/15	M'gladbach	Mainz	1	2
26	18	28/08/15	Wolfsburg	Schalke 04	3	0
27	19	29/08/15	Augsburg	Ingolstadt	0	1
28	20	29/08/15	Bayern Munich	Leverkusen	3	0
29	21	29/08/15	Darmstadt	Hoffenheim	0	0
30	22	29/08/15	FC Koln	Hamburg	2	1
31	23	29/08/15	Mainz	Hannover	3	0
32	24	29/08/15	Stuttgart	Ein Frankfurt	1	4
33	25	30/08/15	Dortmund	Hertha	3	1
34	26	30/08/15	Werder Bremen	M'gladbach	2	1
35	27	11/09/15	M'gladbach	Hamburg	0	3
36	28	12/09/15	Bayern Munich	Augsburg	2	1
37	29	12/09/15	Ein Frankfurt	FC Koln	6	2
38	30	12/09/15	Hannover	Dortmund	2	4
39	31	12/09/15	Hertha	Stuttgart	2	1
40	32	12/09/15	Ingolstadt	Wolfsburg	0	0
41	33	12/09/15	Leverkusen	Darmstadt	0	1
42	34	13/09/15	Hoffenheim	Werder Bremen	1	3
43	35	13/09/15	Schalke 04	Mainz	2	1
44	36	18/09/15	Mainz	Hoffenheim	3	1
45	37	19/09/15	Darmstadt	Bayern Munich	0	3
46	38	19/09/15	FC Koln	M'gladbach	1	0

47	39	19/09/15	Hamburg	Ein Frankfurt	0	0
48	40	19/09/15	Werder Bremen	Ingolstadt	0	1
49	41	19/09/15	Wolfsburg	Hertha	2	0
50	42	20/09/15	Augsburg	Hannover	2	0
51	43	20/09/15	Dortmund	Leverkusen	3	0
52	44	20/09/15	Stuttgart	Schalke 04	0	1
53	45	22/09/15	Bayern Munich	Wolfsburg	5	1
54	46	22/09/15	Darmstadt	Werder Bremen	2	1
55	47	22/09/15	Hertha	FC Koln	2	0
56	48	22/09/15	Ingolstadt	Hamburg	0	1
57	49	23/09/15	Hannover	Stuttgart	1	3
58	50	23/09/15	Hoffenheim	Dortmund	1	1
59	51	23/09/15	Leverkusen	Mainz	1	0
60	52	23/09/15	M'gladbach	Augsburg	4	2
61	53	23/09/15	Schalke 04	Ein Frankfurt	2	0
62	54	25/09/15	FC Koln	Ingolstadt	1	1
63	55	26/09/15	Augsburg	Hoffenheim	1	3
64	56	26/09/15	Hamburg	Schalke 04	0	1
65	57	26/09/15	Mainz	Bayern Munich	0	3
66	58	26/09/15	Stuttgart	M'gladbach	1	3
67	59	26/09/15	Werder Bremen	Leverkusen	0	3
68		...	...	...	...	...

69

70 [306 rows x 5 columns]

71

72 data.Heim[1:5]

```

73 # equiv: data['Heim'][1:5]
74 ...
75 data[data.ATore>3]
76      Date      Heim      Auswaerts  HTore  ATore
77 16   23/08/15   Ingolstadt      Dortmund    0    4
78 24   29/08/15   Stuttgart  Ein Frankfurt    1    4
79 30   12/09/15   Hannover      Dortmund    2    4
80 74   17/10/15  Ein Frankfurt  M' gladbach    1    5
81 93   31/10/15      Hertha      M' gladbach    1    4
82 113  21/11/15   Stuttgart      Augsburg    0    4
83 137  12/12/15   Darmstadt      Hertha    0    4
84 172  06/02/16  Ein Frankfurt      Stuttgart    2    4
85 207  01/03/16   Hannover      Wolfsburg    0    4
86 213  02/03/16   Leverkusen  Werder Bremen    1    4
87 301  14/05/16   Hoffenheim      Schalke 04    1    4
88 ...
89 #
90 # die meisten Auswaertstore
91 data.ATore.max()
92 ...
93 # Eintrag fuer meiste Auswaertstore
94 data[data.ATore==data.ATore.max()]
95 ...
96
97 # Heimsiege fuer Mainz
98 data[data.Heim=='Mainz'][data.HTore>data.ATore].count()

```



```
99 ...
100 # Mittelwert Heimtore
101 data.groupby('Heim')['HTore'].mean()
102 ...
103 # Heimtore Verlauf fuer Stuttgart
104 data.HTore[data.Heim=='Stuttgart'].cumsum().plot()
105 plt.show()
106 ...
107
108 # Tordifferenz zufuegen
109 data['TDiff']=np.abs(data.HTore-data.ATore)
110
111 # oder mit eigener function
112 def absdiff( d1, d2 ):
113     # function gets called for each value
114     diff=d1-d2
115     if diff >=0:
116         return diff
117     else:
118         return -diff
119 #
120 data['TDiff'] = map(absdiff, data.ATore, data.HTore)
```

## 11.5 Beispiel–3: Titanic

```
1 # Daten von der Titanic , Original auf https://raw.githubusercontent.com/mwaskom/seaborn-data/  
   master/titanic.csv  
2 # gleicher Datensatz wie f\"ur SQLite Aufgabe.  
3 >>> titanic=pd.read_excel('http://www-static.etp.physik.uni-muenchen.de/kurs/Computing/python2/  
   source/titanic.xlsx')  
4 >>> titanic.shape  
5 (891, 9)  
6 >>> titanic.columns  
7 Index([u'survived', u'sex', u'age', u'fare', u'embarked', u'class', u'who',  
8        u'deck', u'embark_town'],  
9        dtype='object')  
10 >>> titanic.dtypes  
11 survived          int64  
12 sex               object  
13 age              float64  
14 fare             float64  
15 embarked         object  
16 class            object  
17 who              object  
18 deck             object  
19 embark_town      object  
20 dtype: object  
21 >>> titanic.ix[0]  
22 survived          0
```

```

23 sex                male
24 age                22
25 fare              7.25
26 embarked          S
27 class              Third
28 who                man
29 deck               NaN
30 embark_town        Southampton
31 Name: 0, dtype: object
32 >>> titanic.describe()
33      survived      age      fare
34 count  891.000000  714.000000  891.000000
35 mean    0.383838   29.699118   32.204208
36 std     0.486592   14.526497   49.693429
37 min     0.000000    0.420000    0.000000
38 25%     0.000000   20.125000    7.910400
39 50%     0.000000   28.000000   14.454200
40 75%     1.000000   38.000000   31.000000
41 max     1.000000   80.000000  512.329200
42
43 >>> titanic.head()
44      survived      sex  age      fare embarked  class  who deck  embark_town
45 0           0   male   22    7.2500         S  Third  man  NaN  Southampton
46 1           1 female   38   71.2833         C  First woman  C    Cherbourg
47 2           1 female   26    7.9250         S  Third woman  NaN  Southampton
48 3           1 female   35   53.1000         S  First woman  C    Southampton

```

```

49 | 4          0    male   35   8.0500          S   Third    man   NaN   Southampton
50 |
51 | >>> titanic[['sex', 'age', 'embark_town']].head()
52 |      sex  age  embark_town
53 | 0    male   22   Southampton
54 | 1  female   38     Cherbourg
55 | 2  female   26   Southampton
56 | 3  female   35   Southampton
57 | 4    male   35   Southampton
58 |
59 | >>> titanic[titanic.age>70]
60 |      survived  sex  age   fare  embarked  class  who  deck  embark_town
61 | 96           0  male  71.0  34.6542         C   First  man   A     Cherbourg
62 | 116          0  male  70.5   7.7500         Q   Third  man   NaN   Queenstown
63 | 493          0  male  71.0  49.5042         C   First  man   NaN   Cherbourg
64 | 630          1  male  80.0  30.0000         S   First  man   A     Southampton
65 | 851          0  male  74.0   7.7750         S   Third  man   NaN   Southampton
66 |
67 | >>> sex_class = titanic.groupby(['sex', 'class'])
68 | >>> sex_class
69 | <pandas.core.groupby.DataFrameGroupBy object at 0x7f598ab0c4d0>
70 | >>> sex_class.count()
71 |      survived  age  fare  embarked  who  deck  embark_town
72 | sex      class
73 | female First      94   85    94        92   94    81          92
74 |      Second      76   74    76        76   76    10          76

```

```

75      Third      144  102  144      144  144      6      144
76 male   First    122  101  122      122  122     94     122
77      Second    108   99  108      108  108      6     108
78      Third     347  253  347      347  347      6     347
79  ...
80 >>> sex_class.mean()['survived']
81 sex      class
82 female First    0.968085
83      Second    0.921053
84      Third     0.500000
85 male   First    0.368852
86      Second    0.157407
87      Third     0.135447
88 Name: survived, dtype: float64
89 ...
90 # Weiteres in http://people.duke.edu/~ccc14/sta-663-2016/04A\_Data.html

```

## 11.6 Beispiel–4: Aktienkurse

Data-mining ist zentrales Thema im Finanz/Investment–Banking Bereich, Pandas unterstützt direkten IO von verschiedenen Web–Services mit Aktieninformationen und bietet viele Tools für Zeitreihen–Analysen.

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import pandas.io.data as web
4 import datetime
5 start = datetime.datetime(1980, 12, 12)
6
7 # get data from Yahoo for Apple from 1980–12–12 until today
8 stock_data = web.DataReader("AAPL", 'yahoo', start)
9 print stock_data.iloc[0]
10 print stock_data.iloc[-1]
11 #
12 # datum max('Adj Close')
13 tmax=stock_data['Adj Close'].idxmax()
14 print stock_data.ix[tmax]
15 #
16 stock_data['rel_intraday_loss'] = (stock_data['Close']–stock_data['Open']) / stock_data['Open']
17 date_min = stock_data['rel_intraday_loss'].idxmin()
18 date_max = stock_data['rel_intraday_loss'].idxmax()
19 # Kursverlauf–plot
20 p2=stock_data['Adj Close'].plot()
```

```
21 plt.show()
```

## 11.7 Beispiel–5: Wetterdaten

Der Deutsche Wetterdienst hat Archiv mit langjährigen Messdaten von vielen Wetterstationen: <http://www.dwd.de/DE/leistungen/klimadatendeutschland/klarchivtagmonat.html>, z.B. Daten von der Zugspitze seit 1. August 1900.

Analyse dieser Daten ist interessant, z.B. zur Suche nach Effekten vom Klimawandel

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import datetime
4 import numpy as np
5 # csv data file from zip file download from
6 # http://www.dwd.de/DE/leistungen/klimadatendeutschland/klarchivtagmonat.html
7 # Messdatum als index und Interpretation als Datum (nicht nur String)
8 df=pd.read_csv('http://www-static.etp.physik.uni-muenchen.de/kurs/Computing/sw/source/
    produkt-klima-Tageswerte-19000801-20151231-05792.txt',';\s*',index_col='MESS_DATUM',
    parse_dates='MESS_DATUM',skip_footer=1)
9 df.size
10 714867
11 df.columns
12 Index([u'STATIONS_ID', u'QUALITAETS_NIVEAU', u'LUFTTEMPERATUR', u'DAMPFDRUCK', u'BEDECKUNGSGRAD',
    u'LUFTDRUCK.STATIONS_HOEHE', u'REL_FEUCHTE', u'WINDGESCHWINDIGKEIT', u'
    LUFTTEMPERATUR.MAXIMUM', u'LUFTTEMPERATUR.MINIMUM', u'LUFTTEMP_AM_ERDB.MINIMUM', u'
    WINDSPITZE.MAXIMUM', u'NIEDERSCHLAGSHOEHE', u'NIEDERSCHLAGSHOEHE_IND', u'SONNENSCHINDAUER',
    u'SCHNEEHOEHE', u'eor'], dtype='object')
```



```
13
14 df[ 'LUFTTEMPERATUR' ]
15
16 df[ 'LUFTTEMPERATUR' ].plot()
17 plt.show()
18 # starke Schwankungen, kein Trend erkennbar
19
20 # nur fuer 1 Jahr
21 df[ 'LUFTTEMPERATUR' ][ '1901' ].plot()
22 plt.show()
23
24 # mitteln uebers Jahr
25 dfy=df[ 'LUFTTEMPERATUR' ].resample( 'A' )
26 dfy[ 'LUFTTEMPERATUR' ].plot()
27 plt.show()
28
29 # mitteln per Quartal
30 dfq=df[ 'LUFTTEMPERATUR' ].resample( 'Q-NOV' )
31
32 # Wintermonate
33 dfq[dfq.index.quarter==1].plot()
34 plt.show()
35 # Sommermonate
36 dfq[dfq.index.quarter==3].plot()
37 plt.show()
38 # seit 1970
```

```
39 dfq[(dfq.index.quarter==3) & (dfq.index.year>1970)].plot()  
40 plt.show()
```

## 11.8 Aufgaben

### 1. Beispiele

Versuchen Sie zunächst die gezeigten Beispiele nachzuvollziehen und zu verstehen ...

### 2. Fußball Daten

(a) Welche(s) Spiel(e) hatte(n) die meisten Heimtore bzw. die größte Tordifferenz?

(b) Machen Sie Plot zum Saisonverlauf für Heim und Auswärtstore für Ihre Lieblingsmannschaft.

(c) Ermitteln Sie die Verteilung der Zahl der Tore pro Spiel, d.h. wie häufig gibt es wieviele Tore, folgt das ungefähr einer Poisson-Statistik?

### 3. Titanic

Ermitteln Sie die Überlebensrate getrennt nach Klasse und für Kinder, Frauen und Männer.

### 4. Aktienkurs

(a) Wenn Sie am Anfang (1980) 10000 \$ in Apple Aktien investiert hätten, wieviel Wert wäre es heute? (Hinweis: Suchen Sie im Internet nach Erklärung für *Adjusted Close*)

(b) Untersuchen Sie Kurs der VW Aktie:

```
vw_data = web.DataReader("VLKAY", 'yahoo', start)
```

Gibt's auffällige Bewegungen seit Sommer 2015 ...

### 5. Wetterdaten

- (a) Untersuchen Sie den Verlauf anderer Größen, z.B. Varianz der Temperatur, Sonnenschein, Schneehöhe pro Jahr, Anzahl Tage mit Schnee, ...
- (b) Nehmen Sie Daten anderer Wetterstationen (DWD Archiv, z.B. Hohenpeißenberg, Helgoland, ...)

## 12 Netzwerk Programmierung

- Module der Python Standard Library
- Verteilte Programme mit XML-RPC

## 12.1 Überblick Netzwerkprogrammierung

Netzwerkprogrammierung ist die Kommunikation zwischen Programmen auf verschiedenen Rechnern. Beispiele: Client/Server-Modell oder Peer-to-Peer-Programme.

In Python gibt es die Auswahl zwischen mehreren APIs:

- Asynchron, Event-gesteuert: `Twisted`
- Synchron, Event-gesteuert: `SocketServer` und `asyncore/asyncat`
- Synchron, nicht Event-gesteuert: viele Netzwerk Module der Python Standard Library
- RPC-Modell (remote procedure call), Twisted: asynchron, XML-RPC: synchron (Andere Standards: SOAP, CORBA, etc.)
- Berkeley Socket API: Basis für meisten andere Pakete

Twisted ist ein sehr mächtiges Frameworks zum Erstellen von asynchronen Netzwerkprogrammen (client/server oder p2p). Das RPC-Modell versucht im Gegensatz zu Twisted die Kommunikation über das Netz vom Programmierer zu verbergen.

### Socket:

Ein Socket ist ein Software-Modul über das sich ein Computerprogramm mit einem Rechnernetz

verbinden und dort mit anderen Programmen Daten austauschen kann. Die Kommunikation über Sockets erfolgt in der Regel bidirektional, d.h. über das Socket können Daten sowohl empfangen als auch gesendet werden. Im Client-Server Modus ist ein Socket eine Verbindungsstelle zu einem bestimmten entfernten Programm, repräsentiert durch dessen Adress-Information (z. B. IP-Adresse und Portnummer). Ein Server muss auf Anfragen von unbekannten Rechnern warten können. Er kann sich auf eine bestimmte Adresse binden und auf Anfragen an diese Adresse warten. Es gibt zwei Socket-Arten: Stream Sockets und Datagram Sockets. Stream Sockets kommunizieren über einen Zeichen-Datenstrom (im Netzwerk: TCP, verlässlicher), Datagram Sockets über einzelne Nachrichten (im Netzwerk: UDP, effizienter und flexibler).

## 12.2 Programme der Python Standard Library

Klassische Anwendungen wie FTP, eMails senden und empfangen oder Telnet können synchron abgearbeitet werden - d.h. eine Anforderung wird an einen Server gesendet, auf dessen Antwort gewartet und dann weitergearbeitet. Für zahlreiche Netzwerkdienste existieren Module in der Python Standard Library wie z.B. `ftplib`, `poplib`, `imaplib`, `smtplib`, `nntplib`, `gopherlib`, `telnetlib`.

### Server mit SocketServer

Wir gehen hier nicht auf die low-level Berkely Socket-API ein, sondern beschränken uns auf die Klassen des `SocketServer`-Moduls, die die Komplexität der low-level Socket-API verbergen. Bei einem `SocketServer` geht man wie folgt vor:

- Leite Handler-Klasse von `BaseRequestHandler` ab und überschreibe die `handle` Methode
- Benutze Serverklasse `TCPServer`, `UDPServer` und instanziiere diese mit Server-Adresse und Handler-Klasse
- Starte Event-Schleife des Servers

Ein Beispiel für einen einfachen Server auf TCP-Basis:

```
1 from SocketServer import TCPServer, BaseRequestHandler
```



```
2
3 class HelloHandler(BaseRequestHandler):
4     def handle(self):
5         print "Serving client:", self.client_address
6         self.request.sendall('Hello Client! I am a HelloHandler\r\n')
7
8 TCPServer.allow_reuse_address = True
9 srv = TCPServer(('', 7070), HelloHandler)
10 srv.serve_forever()
```

Man führt dieses Programm auf dem Server aus und kontaktiert das Server-Programm von einem Client auf der shell-Kommando-Zeile mit `nc` (`netcat`):

```
elmsheus@laptop:~$ nc localhost 7070
Hello Client! I am a HelloHandler
```

Auf dem Server erhält man die log-Meldung:

```
elmsheus@laptop:~$ python hellotcpserver.py
Serving client: ('127.0.0.1', 56983)
```

Ein Echo-Server sieht folgendermassen aus:

```
1 from SocketServer import TCPServer, StreamRequestHandler
2
```

```
3 class EchoHandler(StreamRequestHandler):
4     def handle(self):
5         print "Serving client:", self.client_address
6         for line in (self.rfile):
7             self.wfile.write("S:" + line)
8
9 TCPServer.allow_reuse_address = True
10 srv = TCPServer(('', 7070), EchoHandler)
11 srv.serve_forever()
```

Start man obigen Server und erhält man auf dem Client folgende Eingabe/Ausgabe:

```
elmsheus@laptop:~$ nc localhost 7070
Hallo
S:Hallo
Echo !
S:Echo !
```

Auf dem Server erhält man folgende log-Meldungen:

```
elmsheus@laptop:~$ python echotcpserver.py
Serving client: ('127.0.0.1', 55617)
Serving client: ('127.0.0.1', 55618)
```

Der Nachteil des gerade vorgestellten Echo-Servers liegt in der Tatsache, daß er nur einen Client zur

gleichen Zeit bedienen kann, da der Server single-threaded ist. Als Lösung bietet sich ein asynchroner Server (Twisted) an oder man führt die `handle`-Methode in einem eigenen Prozess oder Thread aus. Im folgenden Beispiel wird `ThreadingTCPServer` anstatt `TCPServer` verwendet:

```
1 from SocketServer import ThreadingTCPServer, StreamRequestHandler
2
3 class EchoHandler(StreamRequestHandler):
4     def handle(self):
5         print "Serving client:", self.client_address
6         for line in (self.rfile):
7             self.wfile.write("S:" + line)
8
9 ThreadingTCPServer.allow_reuse_address = True
10 srv = ThreadingTCPServer(('', 7070), EchoHandler)
11 srv.serve_forever()
```

### eMails senden mit `smtplib.SMTP`

Um eMails synchron zu senden, kann man die Klasse `SMTP` aus dem `smtplib`-Modul verwenden. In folgendem Programm wird demonstriert, wie man eine eMail mit dem `email.message`-Modul zunächst konstruiert und sich dann am LMU Physik Mailserver authentifiziert und die eMail an seine eigene Adresse versendet. Das Passwort wird über `getpass` in Laufzeit des Programms eingegeben:

`test_email.py`

## eMails lesen mit `imaplib.IMAP4`

Über die `IMAP4_SSL`-Funktion des `imaplib`-Moduls können eMails von einem Server gelesen werden:

```
imaplib_imap4client.py
```

## 12.3 Verteilte Programme mit XML-RPC

XML-RPC ist ein synchrones Protokoll zum Aufruf entfernter Methoden: es wird eine Methode in einem anderen Prozess oder Rechner aufgerufen und muss auf die Antwort warten. Argumente und Rückgabewerte mit Typen, werden als XML übertragen. Protokoll ist HTTP oder HTTPS.

XML-RPC ist etablierter sprachunabhängiger Standard. Client und Server können in unterschiedlichen Sprachen programmiert sein. In Python kann `xmlrpclib` und `SimpleXMLRPCServer` verwendet werden.

`xmlrpc_server.py` demonstriert, wie man einen XML-Server definiert und startet.

`xmlrpc_client.py` demonstriert, wie ein Client die Server-Methoden aufruft.

Bei Ausführung beider Programme, gibt der Server ähnlich einen Webserver folgende Zeilen aus:

```
elmsheus@laptop:~/python/kurs09$ python code/xmlrpc_server.py
localhost.localdomain - - [08/Oct/2009 15:53:07] "POST /RPC2 HTTP/1.0" 200 -
localhost.localdomain - - [08/Oct/2009 15:53:07] "POST /RPC2 HTTP/1.0" 200 -
localhost.localdomain - - [08/Oct/2009 15:53:07] "POST /RPC2 HTTP/1.0" 200 -
localhost.localdomain - - [08/Oct/2009 15:53:07] "POST /RPC2 HTTP/1.0" 200 -
localhost.localdomain - - [08/Oct/2009 15:53:07] "POST /RPC2 HTTP/1.0" 200 -
```

Um die XML Nachrichten des Clients zu sehen, schaltet man `verbose=True`

```
prx = xmlrpclib.ServerProxy('http://localhost:7070',  
                             allow_none=True, verbose=True)
```

Ein Nachteil des bis jetzt konfigurierten XML-RPC-Servers ist, dass er single-threaded ist. Nachteile daraus sind:

- Funktion benötigt viel Zeit oder blockiert
- Viele Daten werden zwischen Client und Server gesendet
- Verbindung zwischen Client und Server ist besonders langsam
- Client nimmt Daten nur sehr langsam entgegen

Der Server sollte also Multi-threaded gestaltet werden. `xmlrpc_threadedserver.py` demonstriert, wie solch ein Multi-threaded XML-Server definiert und gestartet wird.

Zum Testen wird in einem Fenster eine langsame Methode aufgerufen:

```
In [1]: import xmlrpclib
In [4]: prx = xmlrpclib.ServerProxy('http://localhost:7070')
In [5]: prx.SlowFunc()
Out[5]: 42
```

Gleichzeitig wird in einem anderen Fenster eine schnelle Methode aufgerufen:

```
In [1]: import xmlrpclib
In [2]: prx = xmlrpclib.ServerProxy('http://localhost:7070')
```

```
In [3]: prx.FastFunc()
```

```
Out[3]: 4711
```



## 12.4 Aufgaben

- Erweitern Sie den Echo-Server aus dem SockerServer-Abschnitt, so daß dieser als Taschenrechner benutzt werden kann., d.h. Eingaben von z.B. '1+1' mit dem richtigen Ergebnis antwortet. Prinzipiell kann die gewünschte Funktionalität mit nur einem zusätzlichen Befehl zum existierenden Code erreicht werden - aber dies kann zu einer erheblichen Sicherheitslücke führen. Warum ? Wie sollte der Taschenrechner anstatt realisiert werden ?

Lösung: [test\\_echo.py](#)

- Schreiben Sie ein verteiltes Programm, dass ihnen die Zahl  $\pi$  nachdem in einer der vorigen Kapitel vorgestellten Aufgaben auf mehreren Rechnern berechnet. Ist die gewählte Methode sinnvoll, um die Genauigkeit der Zahl  $\pi$  zu erhöhen ?

Lösung: [xmlrpc\\_pi\\_server.py](#), [xmlrpc\\_pi\\_client.py](#)

## 13 Webprogrammierung und Django

- Web Programmierung
- Django Web-Framework

## 13.1 Webprogrammierung

Aus Performancegründen werden Webserver normalerweise in C/C++ geschrieben, wie z.B. Apache oder Lighttpd. Nicht immer ist Performance nötig wie z.B. in eingebetteten Systemen.

Die Python Standard Library bietet zwei von `Socket.Server.TCPServer` abgeleitete Klassen für einen Webserver: `SimpleHTTPServer` und `CGIHTTPServer`. Beide Klassen basieren auf dem `BaseHTTPServer`.

Der einfachste Webserver:

```
1 import BaseHTTPServer
2
3 server_address = ( '', 9090)
4 handler_class = BaseHTTPServer.BaseHTTPRequestHandler
5 server_class = BaseHTTPServer.HTTPServer
6
7 server = server_class(server_address, handler_class)
8 server.serve_forever()
```

Führen Sie dieses Programm aus und rufen Sie in einem Webbrowser folgende Adresse auf:

<http://localhost:9090/>.

Das funktioniert noch nicht richtig, der Server kann die `GET` Methode nicht interpretieren, da die `handle`-Methode des `BaseHTTPRequestHandler` keine `do_GET`-Methode finden konnte.

Eine einfache Erweiterung:

```
1 import BaseHTTPServer
2
3
4 class MyHandler(BaseHTTPServer.BaseHTTPRequestHandler):
5     def do_GET(self):
6         print "Got GET Request from", self.client_address
7         self.wfile.write('Sorry, I do not speak HTTP. Go away.\r\n')
8
9 server_address = ('', 9090)
10 handler_class = MyHandler
11 server_class = BaseHTTPServer.HTTPServer
12
13 server = server_class(server_address, handler_class)
14 server.serve_forever()
```

Dieses Beispiel kann man nun z.B. zu einem einfachen Taschenrechner erweitern:

[basehttpcalc.py](#)

Etwas mehr Webserver-Funktionalität bietet der **SimpleHTTPServer**, damit kann man Verzeichnisse anzeigen und Dateien übertragen:

```
1 import BaseHTTPServer
2 import SimpleHTTPServer
3
```

```
4 def run_server(port=9090):  
5     server_class = BaseHTTPServer.HTTPServer  
6     handler_class = SimpleHTTPServer.SimpleHTTPRequestHandler  
7     server_address = ('', port)  
8  
9     server = server_class(server_address, handler_class)  
10    server.serve_forever()  
11  
12 if __name__ == '__main__':  
13     run_server()
```

Nächster Schritt ist nicht nur Inhalte anzeigen sondern auch Skripte auszuführen. Am einfachsten geht das mit den sog. **CGI-Skripts** unter Verwendung von **CGIHTTPServer**. Mit CGI-Skripten, die im `cgi-bin` Unterverzeichnis des Webserver platziert werden, können Programme auf dem Webserver ausgeführt und die Ergebnisse an den Client übertragen werden. Allerdings ist das unter Sicherheits-Aspekten heikel. Die Skripte können in beliebigen Programmiersprachen sein, nicht nur Python (solange es server-seitig unterstützt ist).

Ein **CGIHTTPServer**:

```
1 import BaseHTTPServer
2 import CGIHTTPServer
3
4 def run_server(port=9090):
5     server_class = BaseHTTPServer.HTTPServer
6     handler_class = CGIHTTPServer.CGIHTTPRequestHandler
7     handler_class.cgi_directories = [ '/cgi-bin' ]
8     server_address = ( '', port )
9
10    server = server_class(server_address, handler_class)
11    server.serve_forever()
12
13 if __name__ == '__main__':
14     run_server()
```

Ein kleines CGI-Skript platziert man in das `cgi-bin` Unterverzeichnis, macht es ausführbar und ruft

es in einem Webbrowser mit folgender Adresse auf:

<http://localhost:9090/cgi-bin/cgiprintenv.py>:

```
1 #!/usr/bin/env python
2 import os
3 from sys import stdout
4
5 stdout.write("Content-type: text/plain\r\n\r\n")
6
7 for key in sorted(os.environ.keys()):
8     print "%s=%s" % (key, os.environ[key])
```

Ein weiterführende elegante CGI-Lösung ist der **WSGI-Standard**, auf den hier aber nicht weiter eingegangen wird.

## 13.2 Web-Framework Django

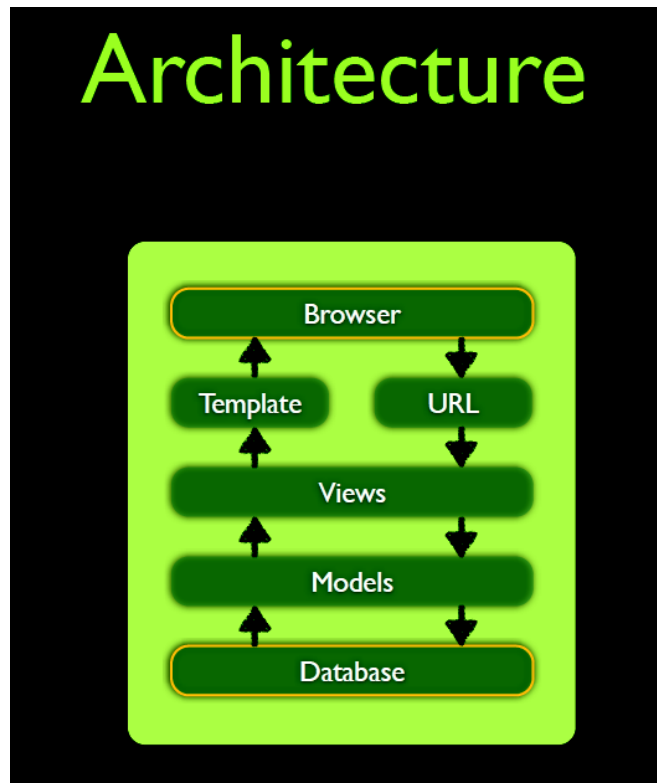
Web-Frameworks fassen mehrere Komponenten zusammen:

- integrierter WSGI-kompatibler Webserver
- Templating-System (Text- oder XML-basiert)
- Anbindung an eine Datenbank

Django ist ein sog. 'leichtes' Web-Framework. Eine Django-Webseite besteht aus einem sog. Django-Projekt, das in der Regel aus mehreren Applikationen besteht. Eine Applikation besteht dabei aus 3 Komponenten:

- Modelle: objektorientierte Repräsentation der persistenten Daten der Anwendung
- Views: Controller, die Modelle abfragen und Daten mit Hilfe von Templates generieren. Templates bestimmen das Aussehen der Webseite.
- URL-Mappings: Abbildungen von Teil-URLs auf einzelne Views





## Model-Template-View

- **Models** : What things are
- **Views** : How things are processed
- **Templates** : How things are presented

Als Beispiel für eine Django Applikation erstellen wir einen einfachen **Wiki Service**.

Dazu müssen wir folgende Schritte ausführen

**1. Ein Django Projekt starten:**

```
cd $HOME
mkdir.djangoprojs
cd.djangoprojs
django-admin startproject wikicamp
```

Es werden mehrere Dateien erzeugt:

```
elmsheus@laptop:~/djangoprojs$ cd wikicamp/
elmsheus@laptop:~/djangoprojs/wikicamp$ ls
manage.py wikicamp
```

Starten Sie den Entwicklungsserver auf Port 9090 mit:

```
python manage.py runserver 9090
```

und öffnen in einem Webbrowser folgende URL:

```
http://localhost:9090/
```

**2.** Da passiert noch nicht viel, man muss erstmal noch die Applikations-Templates anlegen mit:

```
elmsheus@laptop:~/djangoprojs/wikicamp$ python manage.py startapp wiki
elmsheus@laptop:~/djangoprojs/wikicamp$ ls
manage.py  wikicamp  wiki
```

Editieren Sie [wikicamp/settings.py](#) und ändern Sie folgende Zeilen – damit erzeugt man eine SQLite3 Datenbank und definiert die verwendeten Applikationen bzw. Django-Module

```
....
TIME_ZONE = 'Europe/Berlin'
....
INSTALLED_APPS = (
    ...
    'wiki',
)
```

Editieren Sie die Datei [wiki/models.py](#):

```
from django.db import models

# Create your models here.
```

```
class Page(models.Model):  
    name = models.CharField(max_length="20", primary_key=True)  
    content = models.TextField(blank=True)
```

Führen Sie folgenden Befehle aus:

```
python manage.py syncdb  
python manage.py makemigrations  
python manage.py migrate
```

Beantworten Sie die Fragen nach dem superusers/admin entsprechend.

Folgender Befehl startet den Django Server:

```
elmsheus@laptop:~/djangoprojs/wikicamp$ python manage.py runserver 9090
```

Öffnen Sie die Datei [wikicamp/urls.py](#) und definieren Sie die verschiedene URL-Mappings:

```
...  
url(r'^wikicamp/(?P<page_name>[^/]+)/edit/$', 'wiki.views.edit_page'),  
url(r'^wikicamp/(?P<page_name>[^/]+)/save/$', 'wiki.views.save_page'),  
url(r'^wikicamp/(?P<page_name>[^/]+)/$', 'wiki.views.view_page'),
```

Wenn Sie nun in einem Webbrowser die URL <http://localhost:9090/wikicamp/> wieder laden werden Sie eine **Page not found** Meldung erhalten. Und die folgende URL

`http://localhost:9090/wikicamp/Start/` liefert einen Fehler: **ViewDoesNotExist**.

3. Den 'view' erzeugen wir nun, in dem wir folgende Datei editieren: [wiki/views.py](#):

```
1 # Create your views here.
2 from wiki.models import Page
3 from django.shortcuts import render_to_response
4 from django.core.context_processors import csrf
5
6 def view_page(request, page_name):
7     try:
8         page = Page.objects.get(pk=page_name)
9     except Page.DoesNotExist:
10         c = {"page_name": page_name}
11         c.update(csrf(request))
12         return render_to_response("create.html", c)
```

Ein erneuter Aufruf der URL <http://localhost:9090/wikicamp/Start/> liefert einen neuen Fehler: **'TemplateDoesNotExist at /wikicamp/Start/ create.html'**

Dieses template erzeugen wir nun. Erzeugen Sie das Unterverzeichnis [djangoprojs/wikicamp/templates](#) und tragen Sie in [wikicamp/settings.py](#) folgende Zeile ein:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
```

```
        'DIRS': ['templates'],  
    )
```

Erzeugen Sie die Datei `templates/create.html` mit folgendem Inhalt:

```
<html>  
  <head>  
    <title>{{page_name}} - Create</title>  
  </head>  
  <body>  
    <h1>{{page_name}}</h1>  
    This page does not exist. <a href="/wikicamp/{{page_name}}/edit/">Create?</a>  
  </body>  
</html>
```

4. Ein erneuter Aufruf der URL <http://localhost:9090/wikicamp/Start/> liefert nun eine korrekte HTML Seite. Wenn der *Create* link geklickt wird, erhält man eine Fehlermeldung über den fehlenden view `'edit_page'`. Diesen fügen wir nun in `wiki/views.py` hinzu:

```
def edit_page(request, page_name):
    try:
        page = Page.objects.get(pk=page_name)
        content = page.content
    except Page.DoesNotExist:
        content = ""

    c = {"page_name": page_name, "content": content}
    c.update(csrf(request))
    return render_to_response("edit.html", c)
```

Ein erneuter Aufruf von <http://localhost:9090/wikicamp/Start/edit> liefert eine Fehlermeldung, dass das Template noch nicht existiert: **TemplateDoesNotExist at /wikicamp/Start/edit/**

Wir legen nun das template `templates/edit.html` an:

```
<html>
  <head>
    <title>{{page_name}} - Editing</title>
```



```
</head>
<body>
  <h1>Editing {{page_name}}</h1>
  <form method="post" action="/wikicamp/{{page_name}}/save/">
    {% csrf_token %}
    <textarea name="content" rows="20" cols="60">{{content}}</textarea><br/>
    <input type="submit" value="Save Page"/>
  </form>
</body>
</html>
```

Zum Speichern des Textfeld-Inhalts definieren wir nun die Methode `save_page()` in `wiki/views.py`

```
....
from django.http import HttpResponseRedirect
def save_page(request, page_name):
    content = request.POST["content"]
    try:
        page = Page.objects.get(pk=page_name)
        page.content = content
    except Page.DoesNotExist:
```

```
        page = Page(name=page_name, content=content)
    page.save()
    return HttpResponseRedirect("/wikicamp/" + page_name + "/")
```

und komplettieren die *view\_page()* Methode:

```
...
def view_page(request, page_name):
    try:
        page = Page.objects.get(pk=page_name)
        content = page.content
        c = {"page_name": page_name, "content": content}
        c.update(csrf(request))
        return render_to_response("view.html", c)

    except Page.DoesNotExist:
        c = {"page_name": page_name}
        c.update(csrf(request))
        return render_to_response("create.html", c)
...
```

Erzeugen Sie die Datei `templates/view.html` mit folgendem Inhalt:

```
<html>
  <head>
    <title>{{page_name}}</title>
  </head>
  <body>
    <h1>{{page_name}}</h1>
    {{content}}
    <hr/>
    <a href="/wikicamp/{{page_name}}/edit">Edit this page?</a>
  </body>
</html>
```

Füllen Sie die Seite *Start* mit dem Text: *Dies ist die Startseite.* und klicken Sie den *Submit* Button. Die Seite wird in der Datenbank gespeichert und mit `templates/view.html` angezeigt. Klicken Sie den link *Edit this page?* – es wird erneut der Editier-Dialog angezeigt.

Wir haben hiermit einen Prototyp eines Wiki-Systems erzeugt mit Änderungen bzw. Erzeugung der folgenden Dateien:

- `wiki/models.py`
- `wiki/views.py`

- `wiki.db`
- `templates/create.html`
- `templates/view.html`
- `templates/edit.html`
- `settings.py`
- `urls.py`

**Bonus (I): Die Django Admin Seite:**

Fügen Sie in `wiki/models.py` folgende Zeilen in der `class Page()` ein, um die Klassen-Objekte in einer lesefreundlicheren Art anzuzeigen:

```
def __unicode__(self):  
    return self.name
```

Erzeugen Sie die Datei `wiki/admin.py` mit folgendem Inhalt, um in der späteren admin-Webpage das `Page`-Objekt sichtbar zu machen:

```
from wiki.models import Page  
from django.contrib import admin  
  
admin.site.register(Page)
```

Rufen Sie die admin Seite in einem Webbrowser unter folgender Adresse auf `http://localhost:9090/admin` und loggen Sie sich mit dem eben gesetzten username und Passwort ein. Sie haben auf der admin-Seite volle Kontrolle über ihr Projekt und die Daten-Einträge.

**Bonus (II): Markdown:**

Um die Wikiseite nun wie gewohnt mit einer vereinfachten Auszeichnungssprache zu editieren, muss jetzt noch das **markdown** Modul eingefügt werden. In `wiki/views.py` fügen Sie folgende Zeilen ein:

```
1 .....
2 from django.template import RequestContext
3 import markdown
4 .....
5
6 def view_page(request, page_name):
7     .....
8     content = page.content
9     c = {"page_name": page_name, "content": markdown.markdown(content)}
10    c.update(csrf(request))
11
12    context_instance=RequestContext(request)
13    context_instance.autoescape=False
14
15    return render_to_response("view.html", c, context_instance)
16 .....
```

Dies Zeilen bewirken, daß der Text mit **Markdown**-Befehlen editiert und dargestellt werden kann.

## 13.3 Aufgaben

- Machen Sie sich mit dem Programm `basehttpcalc.py` vertraut und erweitern Sie dieses mit weiteren Rechenbefehlen.
- Erweitern Sie das `CGIHTTPServer` Beispiel, indem Sie Eingabe-Parameter an das CGI-Skript über die HTML-Adresse übergeben und auf der Webseite wieder ausgeben.

Lösung: Erzeugen Sie ein Verzeichnis `cgitest`, platzieren folgende Datei darin: `test.cgiserver.py`, dann erzeugen Sie ein Unterverzeichnis `cgi-bin` im Verzeichnis `cgitest` und platzieren folgende Datei darin: `cgiprintenv.py`. Diese Datei muss ausführbar sein. Starten Sie den `CGIServer` im Verzeichnis `cgitest` und rufen Sie in einem Webbrowser folgende Adresse auf: `http://localhost:9090/cgi-bin/cgiprintenv.py?42`

- Erzeugen Sie eine Index-Seite für die URL `http://localhost:9090/wikicamp/`, die einen Index mit allen verfügbaren Wiki-Seiten der Datenbank anzeigt.

Lösung: `urls.py`, `views.py`, `index.html1` (in `index.html` umbenennen !)