

# Python

# Fortgeschritten

Ulrich Cüber

# Programm

import - Section

Const - Section → Konvention GROSSGESCHRIEBEN  
→ global sichtbar

Variable - Section → global !?!

functions -

class

Code — entweder direkt

Oder

nach if \_\_name\_\_ == "\_\_main\_\_":  
=

Aufgabe → Zerlegung → modul(e) ↗  
+ app ↘  
import ↙

# Speicherlayout

modul.py

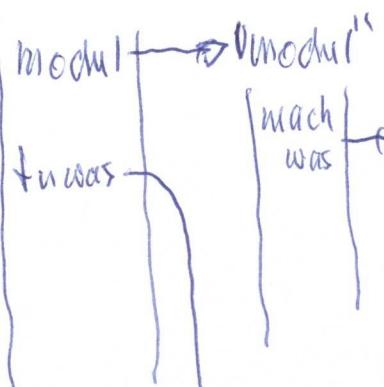
```
def machwas():  
    =
```

clapp.py

```
import modul  
  
def fuwas():  
    modul.machwas()
```

```
if __name__ == "__main__":  
    fuwas()  
    modul.machwas()
```

--main-- == --name--  
Global



# Datenstrukturen

liste	- list()	- [.....]	mutable, not unique
tupel	- tuple()	- (.....)	immutable, not ->
menge	- set()	- {.....}	mutable, unique

-----

dictionary - dict() - { k:v, k:v, k:v, ... }

mutable, key unique

$l = (1, 2, 2, 3)$

$l \rightarrow [1, 2, 2, 3]$

$t = \text{tuple}(l)$

$t \rightarrow (1, 2, 2, 3)$

$s = \text{set}(l)$

$s \rightarrow \{1, 2, 3\}$

-----

$kvtl = [(\underline{\underline{k}} \underline{\underline{v}} \underline{\underline{l}}), (\underline{\underline{k}} \underline{\underline{v}} \underline{\underline{l}}), (\underline{\underline{k}} \underline{\underline{v}} \underline{\underline{l}})]$

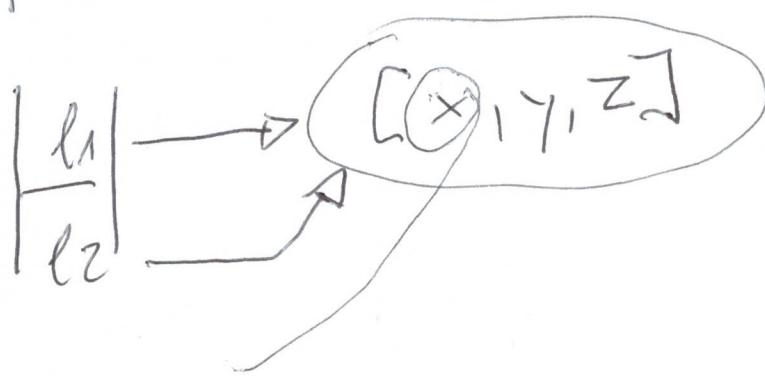
$d = \text{dict}(kvtl)$

Copy?!



$$a = 4711$$

$$b = a$$



$$l_1 = [x, y, z]$$

$$l_2 = l_1$$

Nur ein Kopie der Liste!  
2 Referenzen auf 1 Objekt

$$l_2 \text{ ist } 4711$$

$$\text{print}(l_1[0]) \rightarrow 4711$$

→ Copy - Verfahren

$$l_2 = l_1[:]$$

$$l_2 = l_1.\text{copy}()$$

Aber: es sind flache Kopien!

$$l_1 \rightarrow [ \quad ] \xrightarrow{\text{Elemente}} \{ \quad \} \xrightarrow{\text{Kopie}} [ \quad ]$$

$$l_2 = l_1.\text{copy}$$

$$l_2 \rightarrow [ \quad ]$$

deep copy  $\rightarrow$  siehe Bibliothek  
und selbst gebaut z.B. bei klassen

# Comprehension,

Filter, Mapper, Generatoren,  
Iteratoren

Comprehension: Listen, Dict-Erzeugung „on the fly“

$l = []$

for  $x$  in range(1, 100):  
 $l.append(x)$

$l = [x \text{ for } x \text{ in range}(1, 100)]$

$l = []$

for  $x$  in range(1, 100):  
if  $x \% 2 == 0$ :  
 $l.append(x)$

$l = [x \text{ for } x \text{ in range}(1, 100) \text{ if } x \% 2 == 0]$

# Filter

Funktion  $\rightarrow$  True/False  
w  $\leftarrow$  Wert } Filter-Funktion

Daten  $\rightarrow$  Iterable z.B.  
Liste

$l = []$

for w in Daten:

if Funktion(w) == True:  
 $l.append(w)$

---

$\Leftrightarrow l = list(filter(Funktion, Daten))$

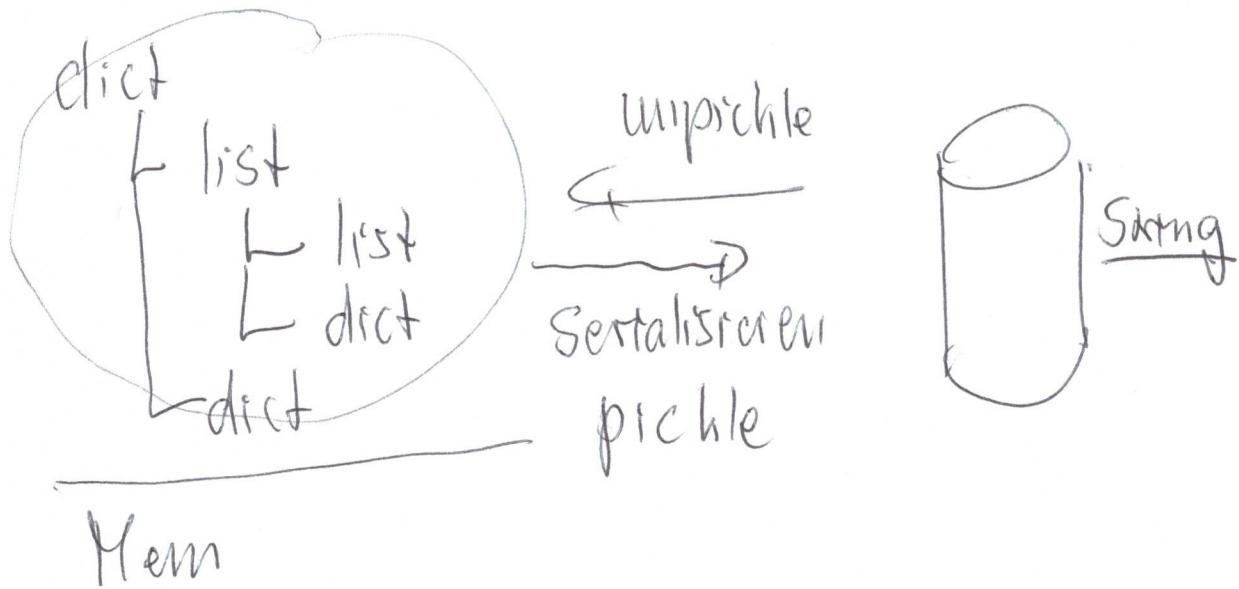
$\Leftrightarrow l = [w \text{ for } w \text{ in Daten} \text{ if } Funktion(w) == True]$

# Map

Funktion       $\rightarrow$  Ergebnis  
                   $\leftarrow$  Wert  
Daten

$l = \text{list}(\text{map}(\text{Funktion}, \text{Daten}))$

# Pickle



OOP

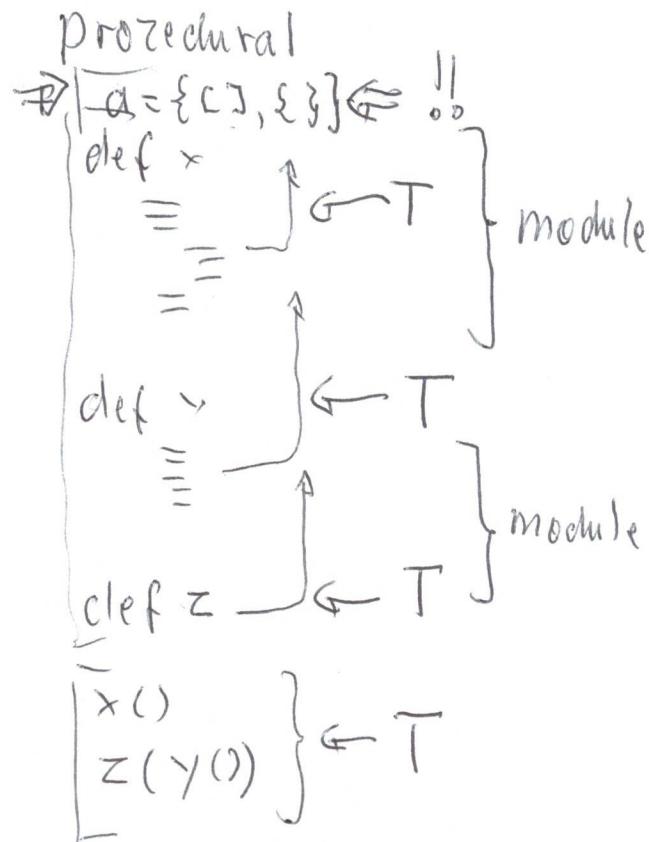
Giegeben



50A

nicht testbar  
→ -/ wortbar  
kann

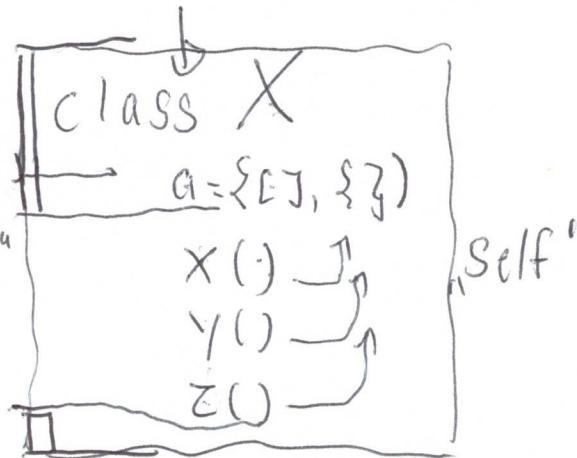
Ctrl C - V



testbar  
wartbar

Wiederver-  
wertbar

Klasse →  
„hidden“ / „private“ →  
API / Contract / „public“



## initialism

Ergebnis auf X-Objekt - r1 = X(config-1)  
r2 = X(config-2)

if r1.dns == r2.dns:  
    ==

# Klassen elemente

class Name:

→ \_\_init\_\_(self)

{ - hidden(self) protected

  - totally-hidden(self) private

  public-method(self)

\_\_name\_\_

„dunder-methoden“ - für Python

Automatismen

# "Lotto Bude"

"6 aus 49"

"Eurolotto"

→ Klasse

a) beim Start: 6 aus 49 / Eurolotto

b) Eingabe der Zahlen: String, : getrennt

z.B. 6/49      1:2:3:4:5:6

Euro      1:2:3:4:5:6:1:2

→ passend zerlegen

prüfen: ② stimmt der Range 1-49  
1-50 u. 1-

① Strumen die Zahlen

③ keine Doppelten

Except

$l1 = \text{Lotto}("6 aus 49")$

1-49

6 aus 49

$l2 = \text{Lotto}("Euro")$

1-50 | 1-12

5 aus 50 +  
2 aus 12

$l1.\text{eingabe}("1:2:3:4:5:6")$  ✓

$l1.\text{eingabe}("a:2:\dots")$  ↗

$l1.\text{eingabe}("50:2:\dots")$  ↗

$l1.\text{eingabe}("1:1:\dots")$  ↗

$l1.\text{eingabe}("4\leftarrow 4\rightarrow 7\leftarrow\right.\left.\rightarrow")$   
bzw  $\leftarrow 7\rightarrow$  ↗

---

class Lotto:

def \_\_init\_\_(self, stil):

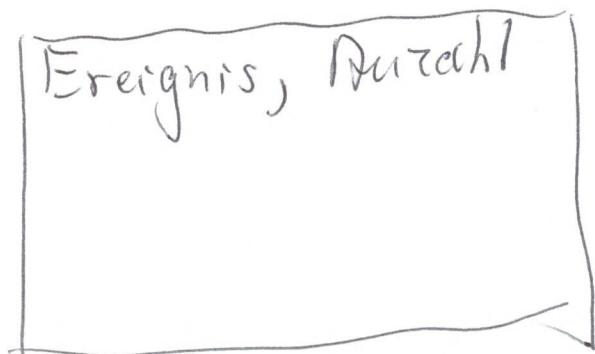
— ==

# Log Analyse

①

Welche Ereignisse liegen wir oft vor  
Annahme: Dateistruktur immer gleich  
Wir kennen noch nicht alle Ereignisse

Ergebnis: Ausgabe → Screen  
→ Datei  
mit CSV-Stil



Idee: nutzen Sie Funktionen  
lassen für später den Datename  
offen  
⇒ Parameterliste

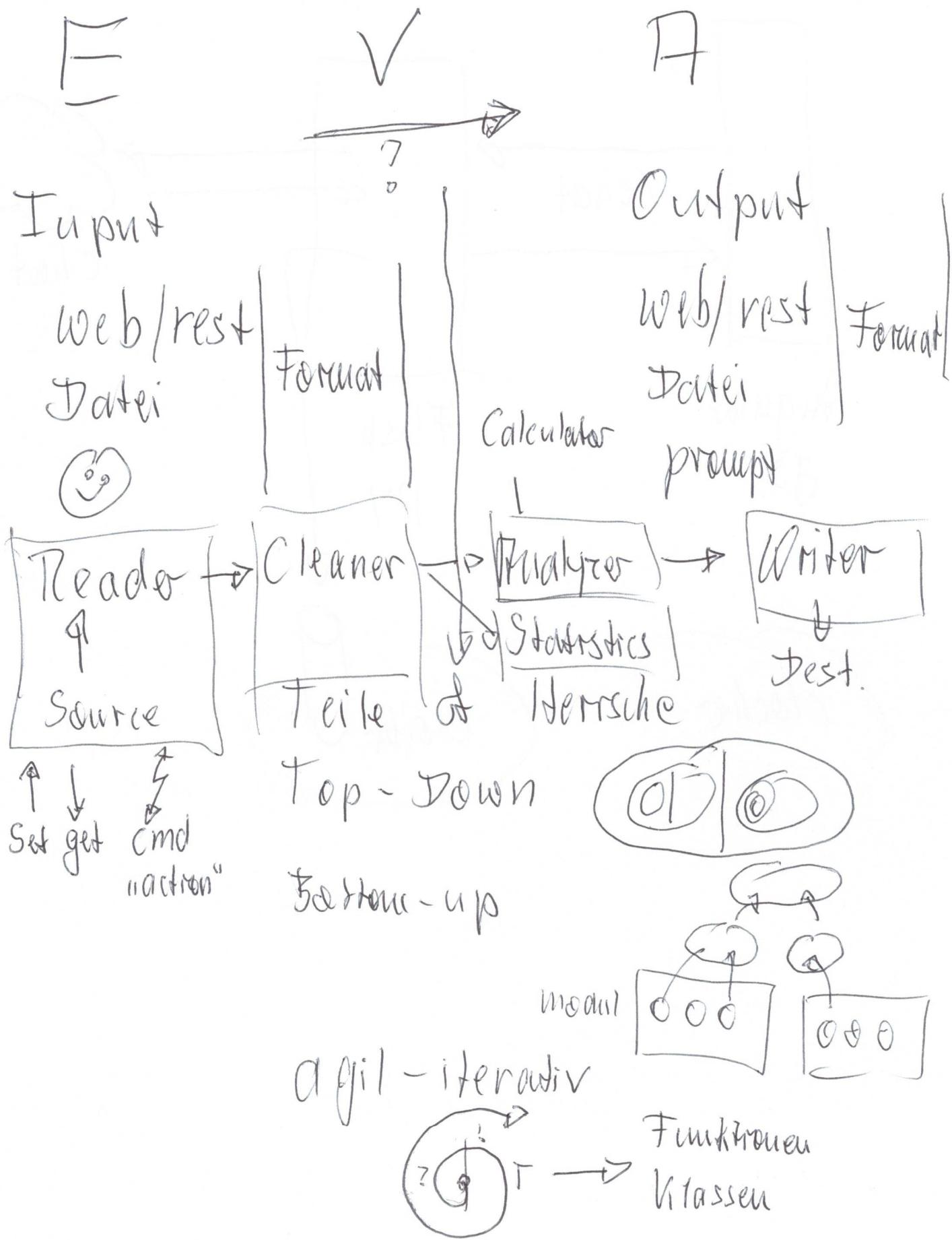
log-analyzer -i ... -o ...

# Log-Analyzer

Klasse  $\leftarrow$  Name/Pfad einer LogDatei

- $\hookrightarrow$  sofort lesen  $\leftrightarrow \text{--init--}$
- $\hookrightarrow$  get-event(Event-Typ)  $\Rightarrow$  Liste aller Ereignisse mit Message
- $\hookrightarrow$  get-event-count(Event-Typ)  $\leftarrow$  Zahl
- $\hookrightarrow$  --str-- & Ein Teilausschnitt

```
la = LogAnalyzer("Dateiname.log")
print(la.get-event-count("INFO"))
for line in la.get-event("INFO");
    print(line)
print(la)
```



# Reader

Quelle  $\leftrightarrow$  Datei  $\rightarrow$  Tabelle  
CSV.

class Reader

public

RPT

def __init__(self, filename, typehint="csv")
def read(self) -> <del>None</del> Return: self
def get(self) -> pandas dataframe
def __str__(self) -> str
def __repr__(self) -> str

if \_\_name\_\_ == "\_\_main\_\_":

E      r = Reader("xyz.json", "json")

C      c = Cleaner()

A      a = Analyzer()

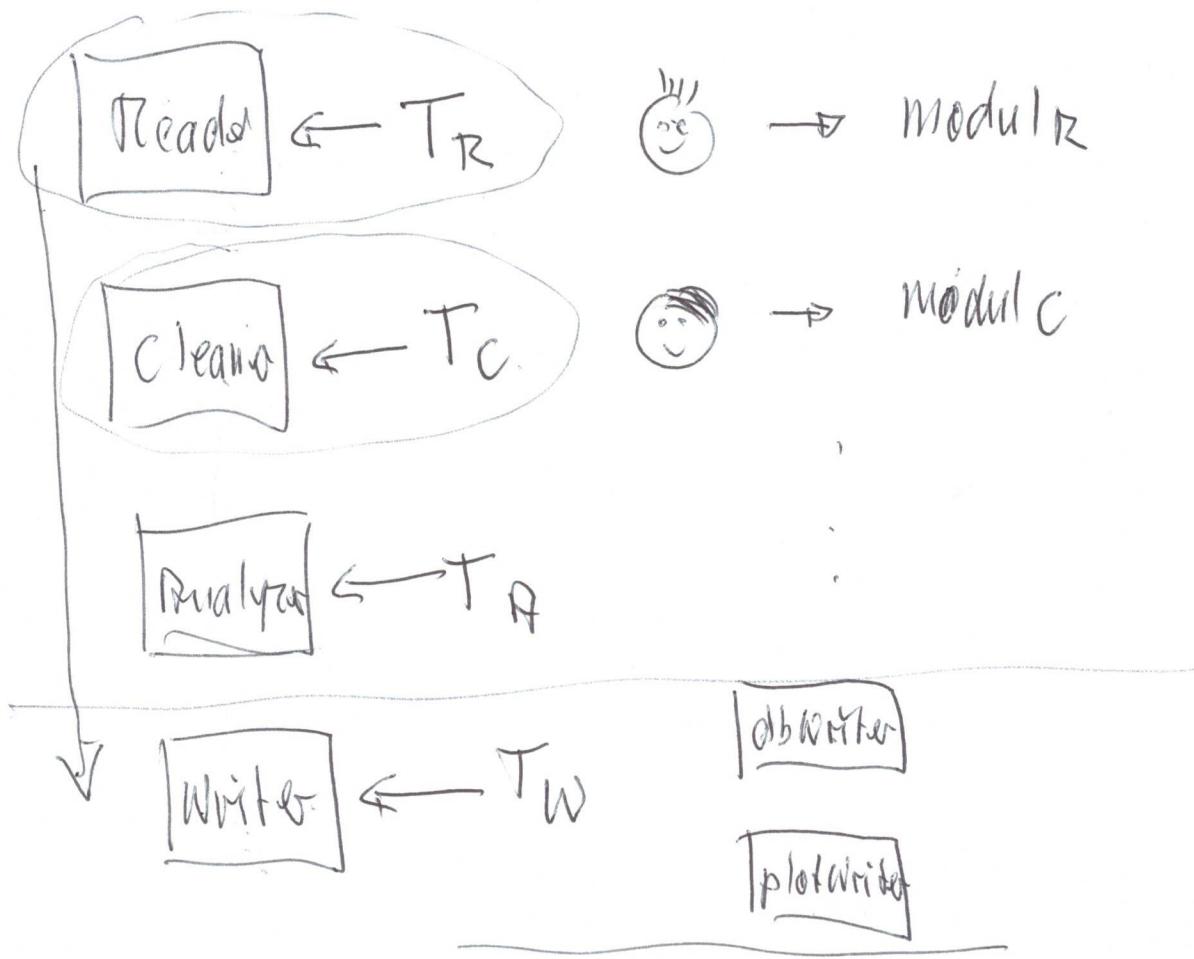
D      w = Writer("xyz.csv", "csv")

d = r.read().get()

d = c.clean(d)

e = a.analyze(d)

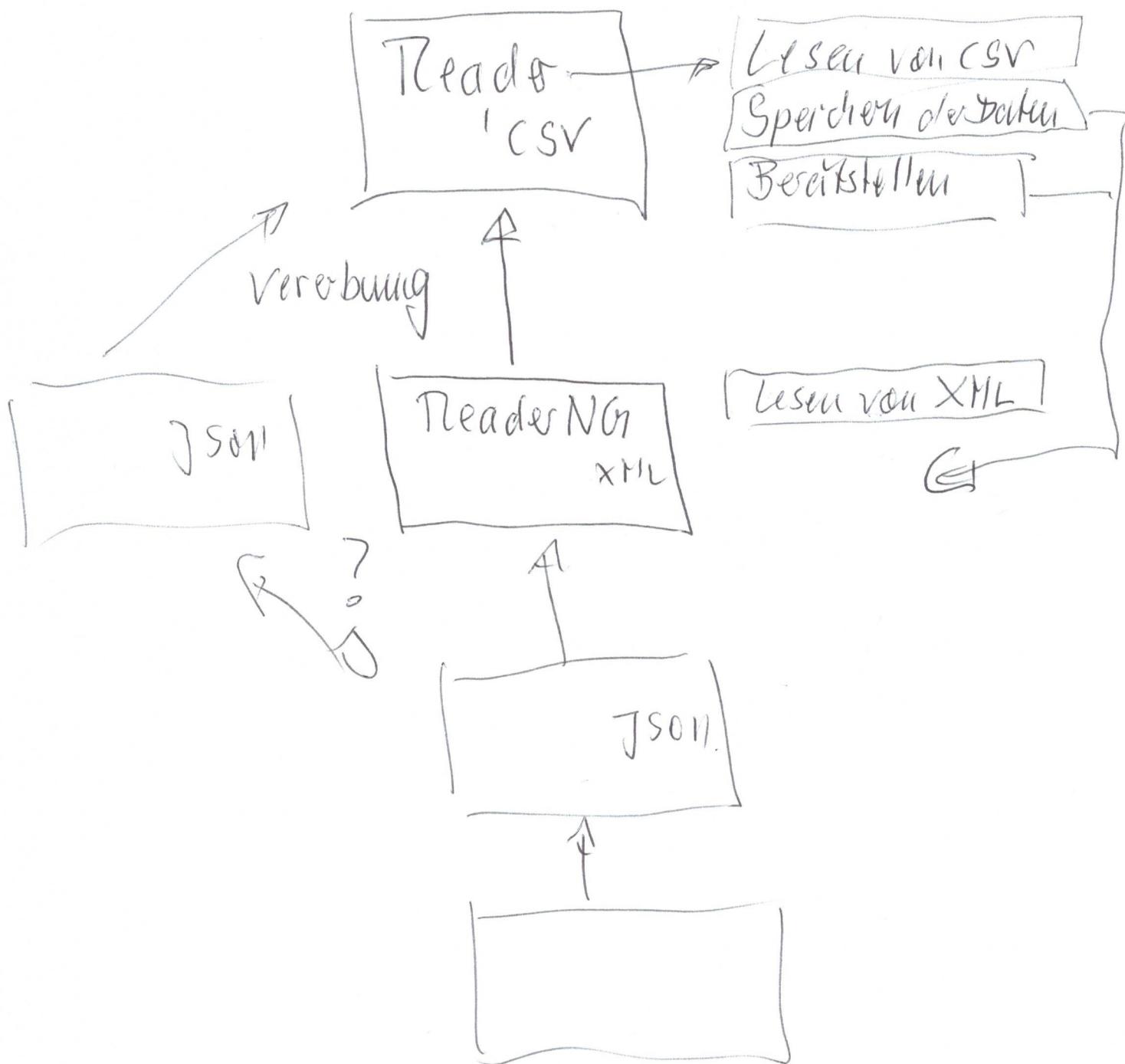
w.write(e)



import Reader

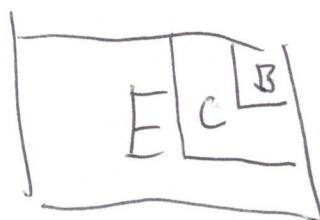
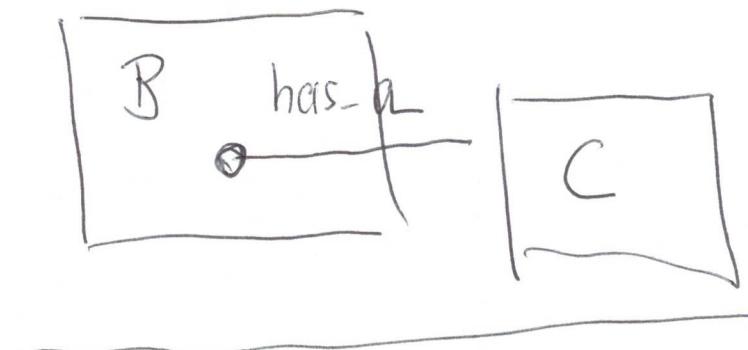
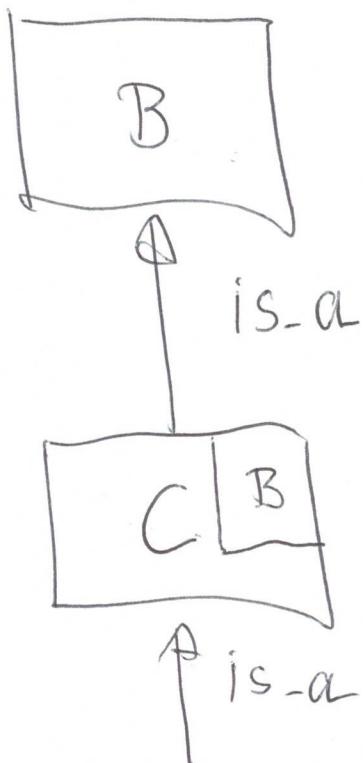
import Cleaner

---

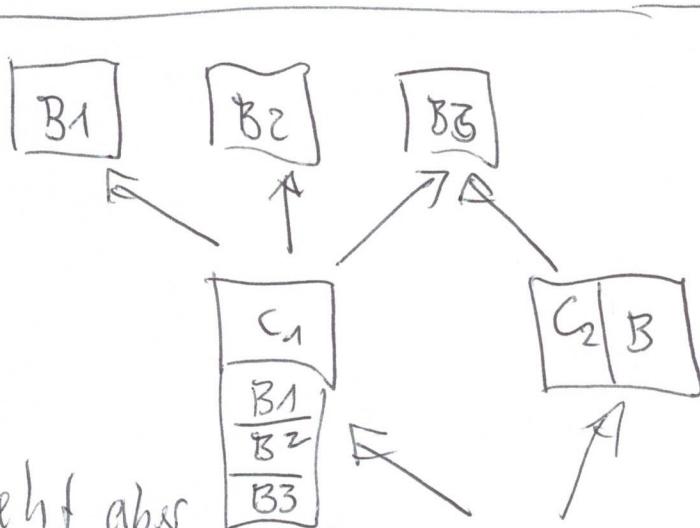


is-a vs has-a

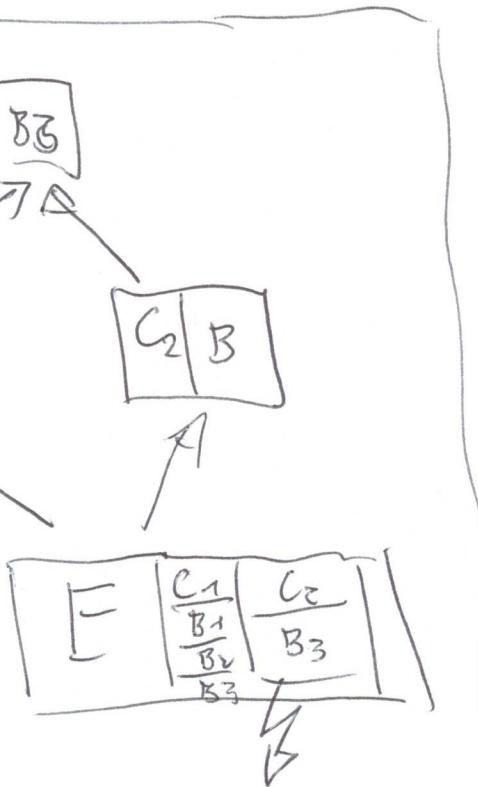
### Berechnung



Dependency Injection



geht aber  
ist zweifelhaft  
in python!



main

- import DataReads

DataReads

import pandas

pandas

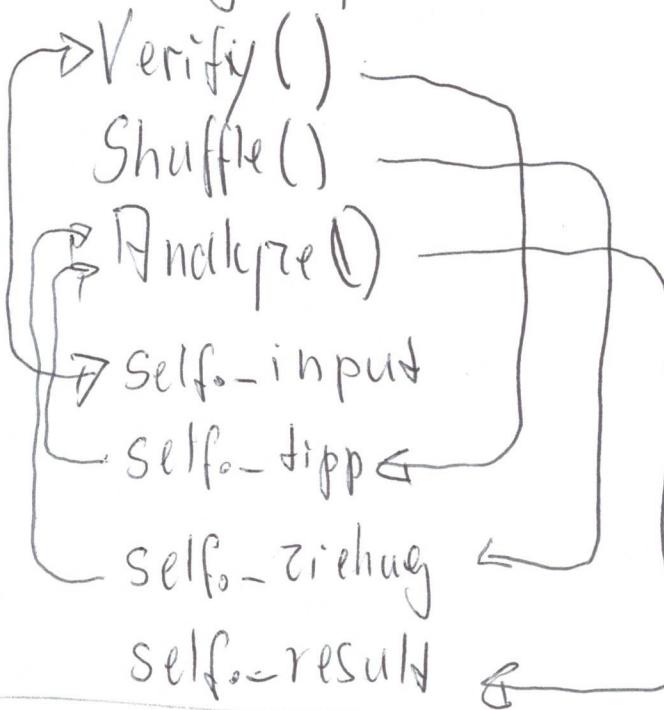
import pandas

pandas



# Lotto NG

Lotto Engine | 6 aus 49



Reader

read

return [1, 2, 3, 4, 5, 6]

Writer

write

print  
print  
print

\*1 Read ( Reader-Objekt )

\*2 Write ( Writer- Objekt )

\*1 def read(self, reader):

    self.-input = reader.read()

\*2 def write(self, writer):

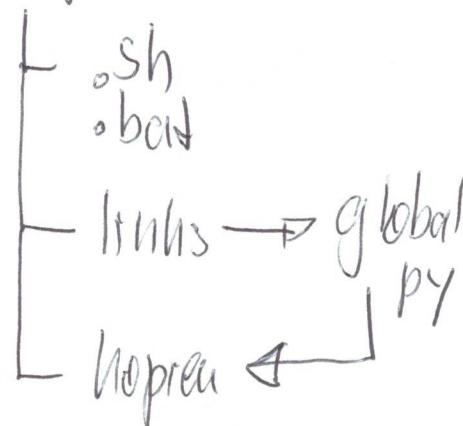
    writer.write(self.-tipp,  
                  self.-ziehung,  
                  self.-result)

# Projektarbeit

- eigene Klassen
- Tools
- Module
  - eigene
  - pypi
  - ...

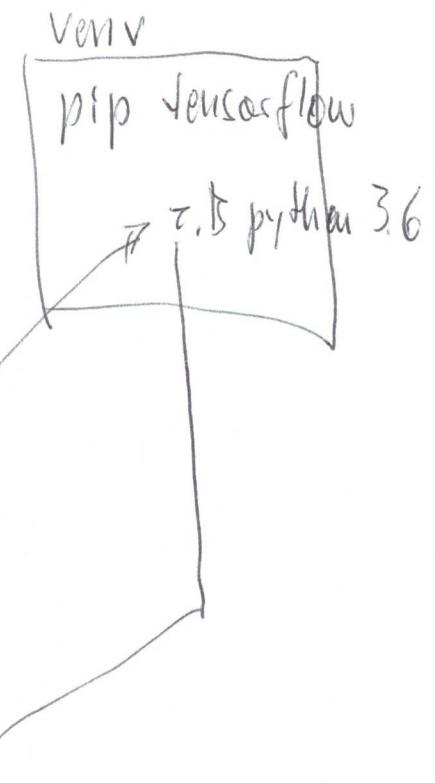
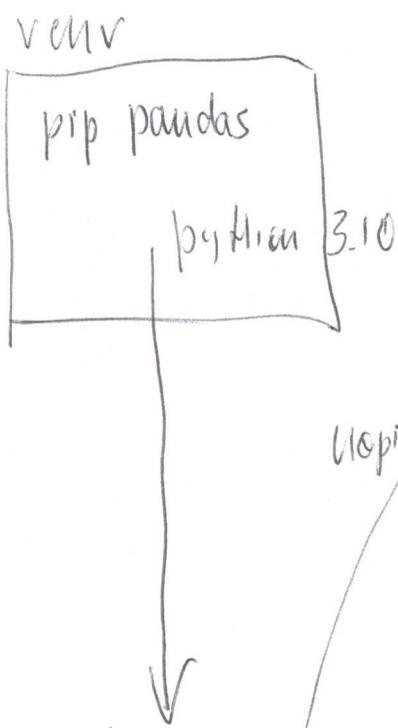
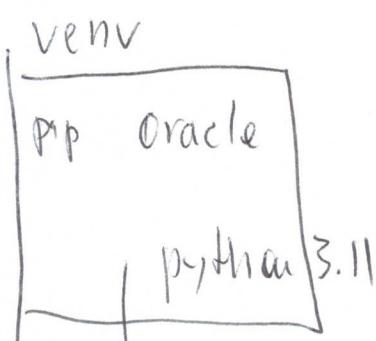
→ Virtualenv - Arbeitsbereich  
für Projekte

Rulage → Ordner - scripts



Häkchen

ab dann passiert alles  
mit python & pip nur hier,  
lokal



python  
global

z.B. std-module

# Dokumentation

Im Code:

- „sprechender Code“
- Variablen, Funktionen, Klassennamen
- Konstante ABC
- Namen verb\_objekt  
daten\_verwendung
- Klassen Eine Einfache Klasse
- pythonic Style  
nicht zu tricky!
- Kommentare ! #

Clean Coding

User-Docs:

Doc-Strings

"""  
""" } normale „lange“ Strings  
an besonderen Orten

# DocString

```
""" Modul beschreibung
```

```
class X:
```

```
    """ Klassenbeschreibung
```

```
    """
```

```
    def machwas
```

```
        """ Methode beschreibung
```

```
        """
```

```
def fuwas
```

```
    """ Funktionsbeschreibung
```

```
    """
```



Von der IDE  
pydoc → HTML | PDF

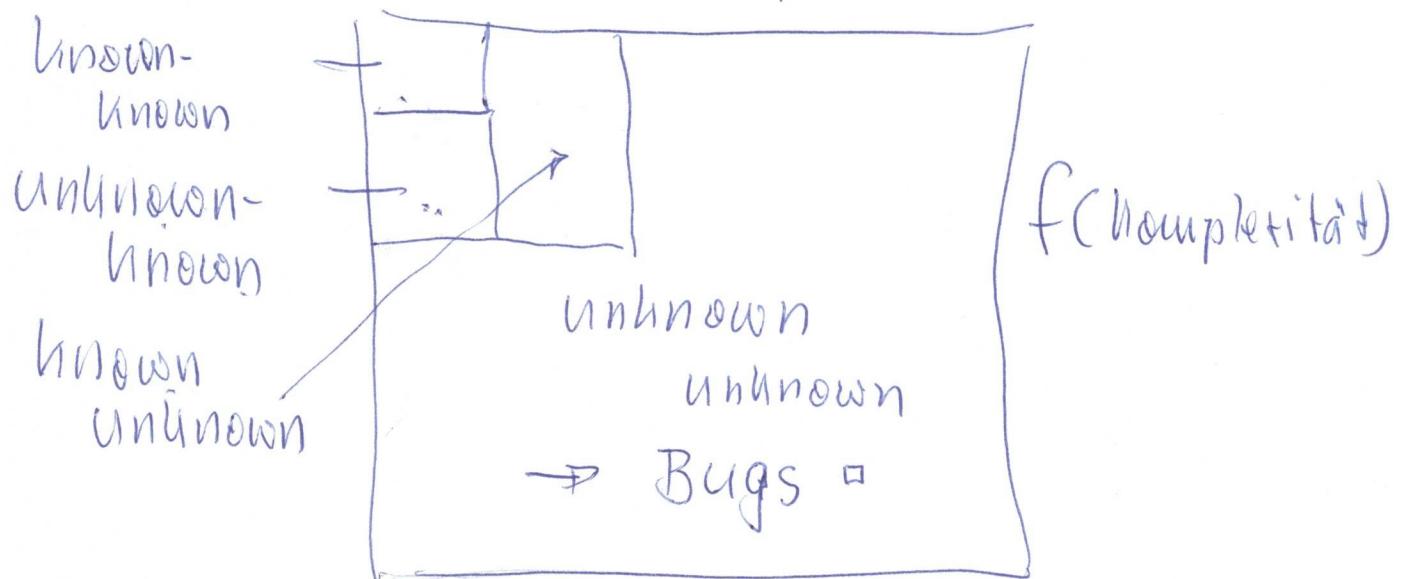
Tools: Sphinx eine Art Mockup für  
return, parameter ...

# Testen & Realität

100% Fehlerfrei ist nicht möglich  
bzw. nicht zu beweisen

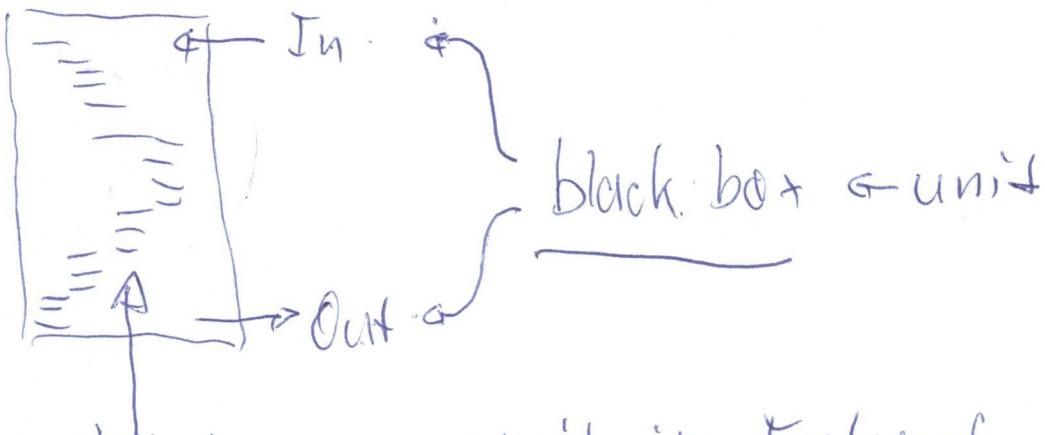
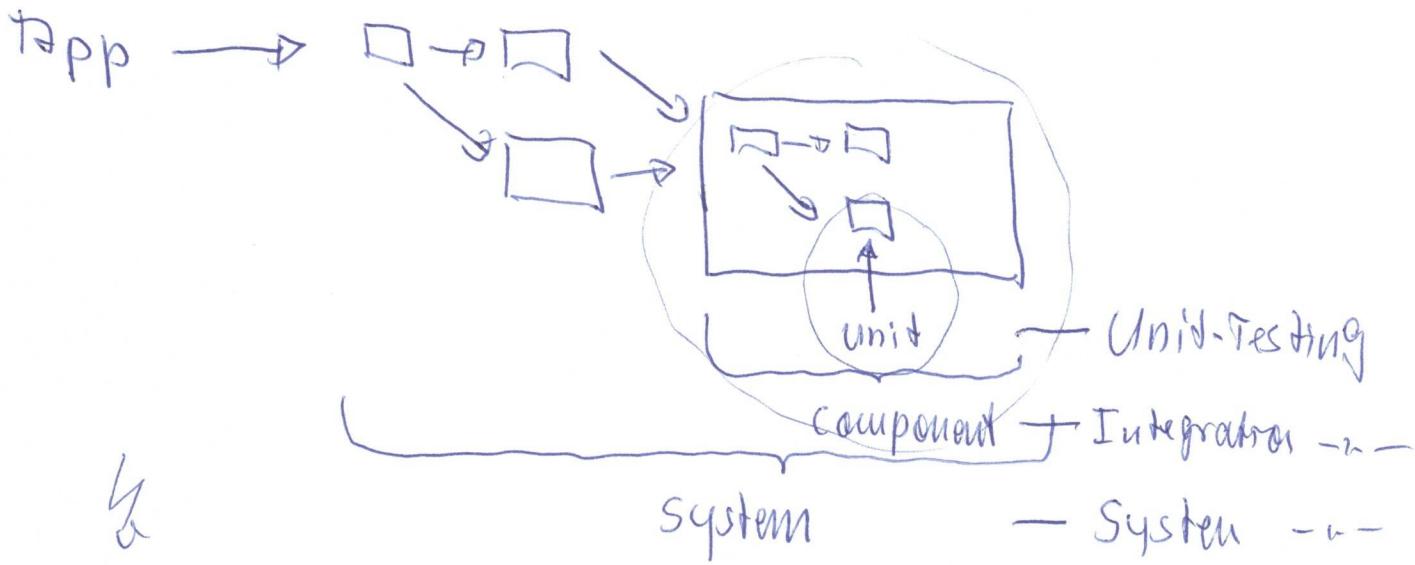
Testen → prüft themenbezogen ob sich  
die Anwendung wie spez. verhält  
bzw. für bekannte Fehlsituationen wie  
sich die Anw. verhält

## Die Welt nach D. Rumsfeld



# Unit-Tests

=



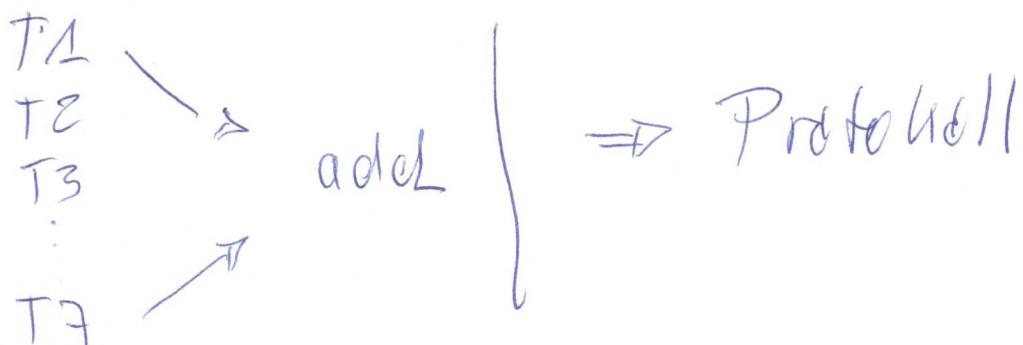
Unit im Entwurf:

Unit-Test → 

- klein
- einfache → „~~Experten~~ schlicht“ Spezialist"
- gut festbar

Erwartung g.m.  
Spez

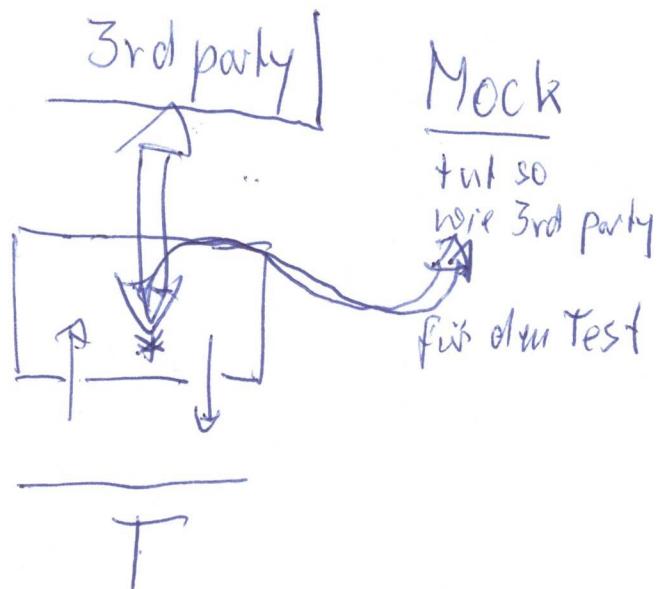
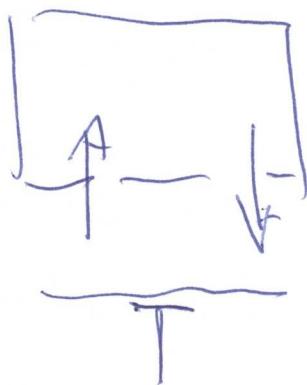
	1,2	None, 1   $\rightarrow$ 6
	int int	$\downarrow$
3?	<u>add(a,b)</u>	"a", 1   $\rightarrow$ 6
	return a+b	"a", "b"   $\rightarrow$ 6
		-1, 0
		-1, -2
		oo, 1



Unit test - standard bib

pytest - extra modul

isolierte  
Unit



T-Suite

