

# Python

## Aufbau

---

Beginn: 9:30

Mo 9<sup>30</sup> - 12 | 13 - 17<sup>00</sup>

Di, Mi 9<sup>00</sup> - 12 | 13 - 16<sup>30</sup>

---

45-50 Min

Erklären - Zeigen - Üben

## Comprehension

$l = [v \dots, "Listengenerator"]$

$d = \{k:v \dots, "K-V-Generator"\}$

Nicht zu stark schachbeladen! Lesbarkeit

# Dictionary

Comprehension

$d = \{ \}$

for  $k$  in Werte:

    for  $v$  in Werte2

        if not  $k$  in  $d$ :  
             $d[k] = v$   
        else  
             $d[k] = v$

# Filter, Map

Filter: Funktion die T/F liefert ↗  
Fischt aus einer Liste die T-Werte raus  
`filter(function, liste)`  
→ Berechnung im Listenkontext z.B. for  
„Erinnerungsobjekt“, verbraucht semi Daten

---

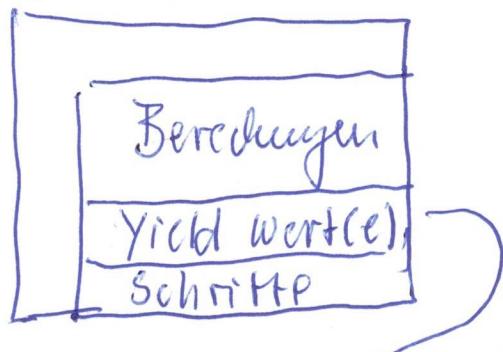
Map: Funktion die elementweise berechnet ↗  
Lauft über Liste und weidet Fkt an ↗  
`map(function, liste)`  
→ im Listenkontext  
Verbrauchend

# Generator

"Funktion" die in einem Listenkontext bei jedem Durchlauf eine Berechnung ausführt, den Wert zurückgibt, (noch) nicht endet und auf nächsten Aufruf wartet

Schlüsselwort: `yield`

`def gen_werte(Parameter):`



`for n in gen_werte(...):`  
≡ `print(n)`

## Zip / pack

$l1 = [k_1, k_2, k_3, \dots]$        $l2 = [v_1, v_2, v_3, \dots]$        $\rightarrow d = \{k_1:v_1, k_2:v_2, \dots\}$

→ Dict. comprehension?

→ (Geschachtelte) For-Schleifen

→ Zip → Liste von Tupel

zip ( $l1, l2$ )



$[ (k_1, v_1), (k_2, v_2), \dots ]$

z.B. um  $d$  zu erzeugen

$d = dict(zip(l1, l2))$

bei Über/Unterlängen der Partner

→ Liste mit Pärchen, Rest ignoriert

$l1 = [k_1, k_2, k_3]$

$l2 = [[\dots], [\dots], [\dots]]$



# Zip mit List Comprehension

$l1 = [a, b, c]$   
 $l2 = [1, 2, 3]$

$e = \left[ \text{list}(n) \text{ for } n \text{ in } \text{zip}(l1, l2) \right]$

$e = \left[ \begin{array}{l} \text{for } n \text{ in } [(a, 1), (b, 2), (c, 3)] \\ \quad \boxed{a} \quad \boxed{b} \quad \boxed{c} \\ \quad \quad \quad \boxed{1} \quad \boxed{2} \quad \boxed{3} \end{array} \right]$

# Aufgabe

file-compare.py

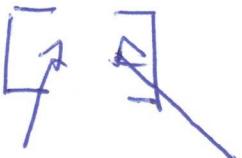
2 Pfade

Werden nicht rekursiv verglichen:

- Anzahl der Dateien
- Anzahl der Unterverzeichnisse } OS
- Dateien gleichen Inhalts  
durch md5 hash Abgleich

dict gefunden:

Key: md5hash

Value: 

1. Dateiname

2. Dateiname für  
Datei gleichen  
md5-Wertes

E    V    R

/ \

Funktionen

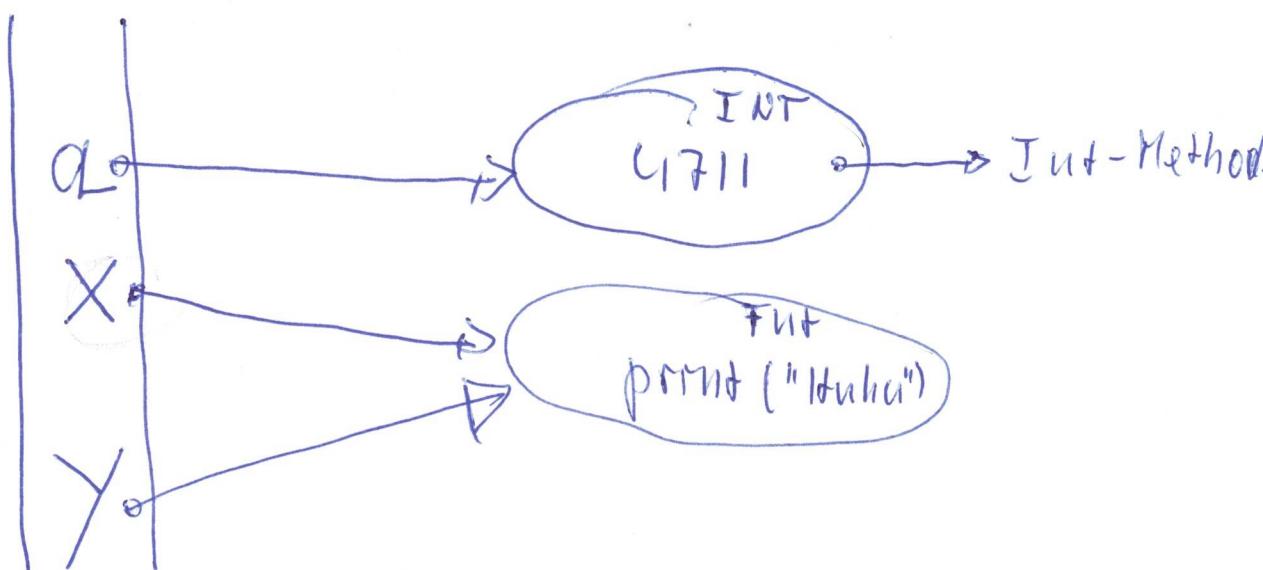
# Funktionen als Objekte

python  $\leftrightarrow$  OO-Sprache

- Daten basieren auf Objekten  $\leftarrow$  basieren auf Klassen
- Funktion  $\leftarrow$  Funktion Type

-i main()

Menu



c1 = 4711  $\rightarrow$  Int-Objekt

def x():  
 print("Huhu")  $\rightarrow$  Fkt-Objekt

y = x

# Generische Algorithmen

Eine Lösung für viele ähnliche Probleme/  
Daten

Anpassung an einen best. Daten via mitüber-  
gebener Funktion

- z.B. `sort (Daten, Cmp=None)`
- Listen von Dingen
  - Dinge sind vergleichbar
    - = = !=
    - < >
  - Vergleicher mit 2 Param a,b
  - Rückgabe a if  $a > b$  else b

# Springtabelle

Daten	A	B	C	X
	Text	Zahl	Liste	Text

```
jump-table = { "A": format-cell-text,  
               "B": format-cell-number,  
               "C": format-cell-list  
         } "X": format-cell-text
```

```
{ if key == "A":  
    format-cell-text()  
elif key == "B":  
    format-cell-number()  
...  
besser so:  
→ jump-table[key]()
```

# Wiederholung

List

Dict

- Comprehension

$$l = [e_1, e_2, \dots]$$

Bsp  $l = [x ** 2 \text{ for } x \text{ in } 1, 2, 3, 4]$

$$l = [1, 4, 9, 16]$$

Bsp  $d = \{ \text{calc-heis}(f\text{name}) : [f\text{name}] \text{ for } f\text{name} \text{ in } \text{glob}(\ast)$

$$d = \{ \begin{aligned} &x_4 \text{ 1234} : [ \text{claddel.} \cdot t \cdot x \cdot t ], \\ &\text{ab7} \cdot x \cdot 3 : [ \text{uraku.} \cdot t \cdot x \cdot t ], \\ &\dots \end{aligned} \}$$

$$l = [ \text{Ausdruck Schleifenkonstrukkt} ]$$

$$d = \{ k\text{-v-Ausdruck Schleifenkonstrukkt} \}$$

$$d = \{ k_1 : v_1, k_2 : v_2, \dots \}$$

# Map / Filter

$x = \text{map}(\text{func}, \cancel{\text{list}} \text{ iterable}) \rightarrow [ ]$   
 $y = \text{filter}(\text{func}, \text{ iterable}) \rightarrow [ ]$

```
def my_map(func, iterable):  
    result = []  
    for e in iterable:  
        result.append(func(e))  
    return result
```

```
for e in iterable:  
    if func(e) == True:  
        result.append(e)
```

# Generator

```
def my-gen( start, end, step):
    while True:
        yield start
        start += step
        if start > end:
            return start None
```

```
for x in my-gen(0, 10, 2):
    print(x)
```

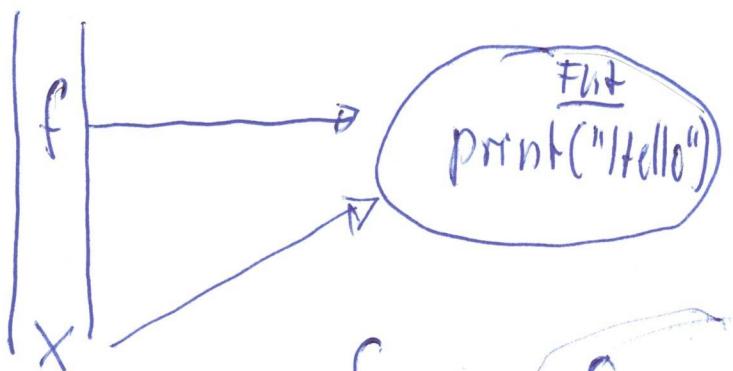
---

range (int, int, int)

range-chr (str, str)

chr-range (str, str)

# Funktionen



$f: \rightarrow$  Fkt  
 $f(): \rightarrow$  Ausführung

$x = f \Leftrightarrow$  Fkt. aufrufen via Parameter übergeben

z.B. Sprachtabelle

z.B. Sort für neue Datentypen  
"neuere Algorithmen"

→ def x(e):  
    return e \*\* 2

---

qz = map(x, [1, 2, 3, 4])

$x = \lambda e: e ** 2$

qz = map(x, [1, 2, 3, 4])

← qz = map(lambda e: e \*\* 2, [1, 2, 3, 4])

# Errors / Handling

Exception-Klassen - Voredefiniert / user defined

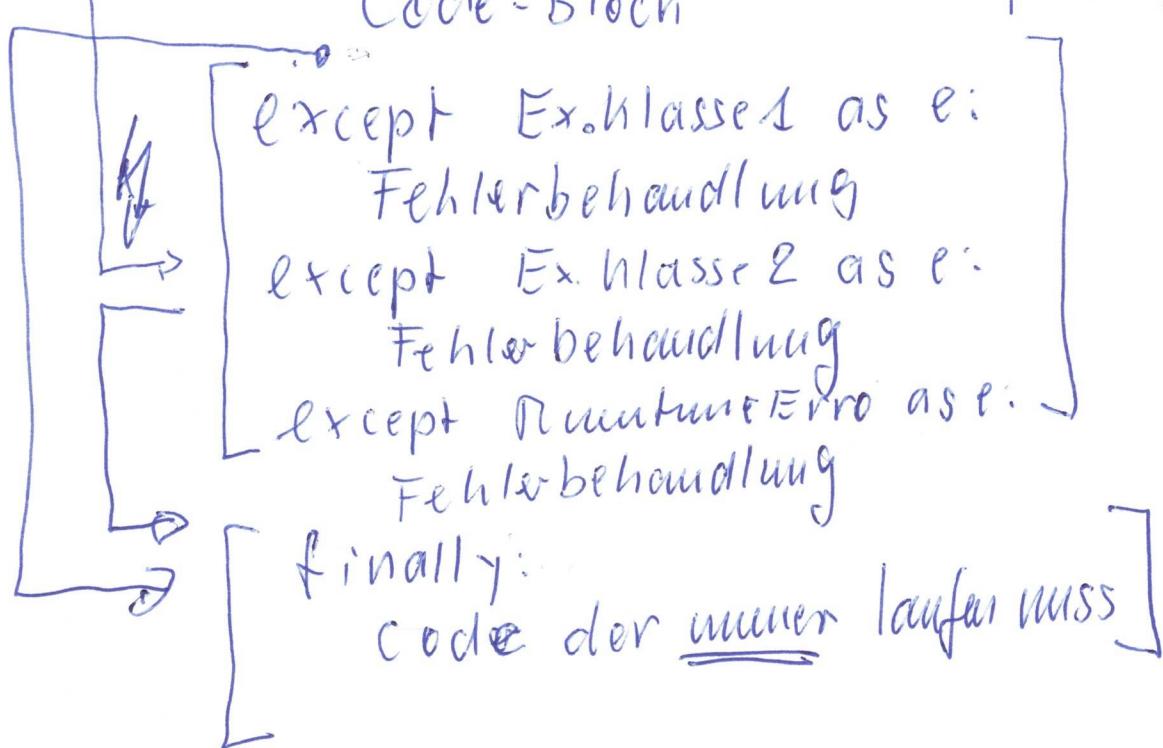
Risierung: durch einen Fehler → python vom Programmierer → raise Ex.Klasse

Bsp if zahl < 1 or zahl > 49:  
raise RuntimeError("ungültige Zahl")

## Error handling

try: # Versuch  
Code-Block

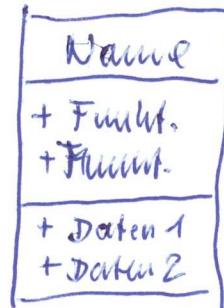
try - finally  
try - except  
try - except - finally



# Klassen / OOP

Container / Bauplanc

Name / Funktionalen  
→ Daten



Notation: UML → Klassendiagramme  
Sequenzdiagramme  
Statusdiagramme

Prinzipien (in py basierend auf Konvention)

Kapselung

Information Hiding

Festes/Stabiles RPI

Tests, Tests, Tests

Vererbung }  
Abstraktion }

## Klassen

class EineKlasse(Elternklassen):  
 -> pass  
  
 => "Leere Klasse"

```
class Calculator:    # python nutzt das
    def __init__(self, p1, p2=None):    # methoden
        "dunder"                      # bei Objekt-
        self.p1 = p1                   # anlage
        self.p2 = p2
    def mach_was(self):
        print(self.p1)
        print(self.p2)
```

im Programm:

$\theta = \text{Calculator}(4711, 1234)$

print(o.p1, o.p2)

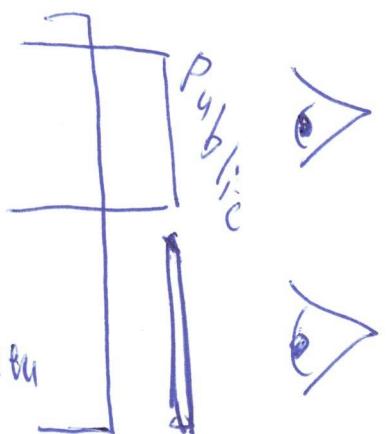
O. mach-Was()

$\textcircled{O} \equiv \text{self}$   
mission intern

# Klassen

Offizielle  
Schnittstelle  
+

Interne Funktionen



hidden  
private / protected

— name / - name  
hart private

hidden / protected  
Konvention!

class X:

— status = True ← hidden

— \_\_init\_\_(self): ← indirekt Aufruf

— self.\_status = not self.\_status

def \_\_ = hilfsrechnung(self): ← echt private, nur  
in X

— nebenrechnung(self): ← hidden, nur in  
X und Kindklassen

hauptrechnung(self): ← public, für alle  
self.\_\_hilfsrechnung()  
self.\_nebenrechnung()

# Übung

Lottoerie:

Ein Spieler macht eine Eingabe\* von  
6 Zahlen\* (1-49, keine Doppelten).

Drei Eingabe kann seine Web, Kommadozeile, Datei  
mechanisch

Es werden 6 Zufallszahlen gezogen und  
mit den Eingaben ("Tipp\*\*" genannt) verglichen

Das Ergebnis wird ausgegeben. Je nach  
Wahl des Spielers als Web, Kommadozeile, Datei

---

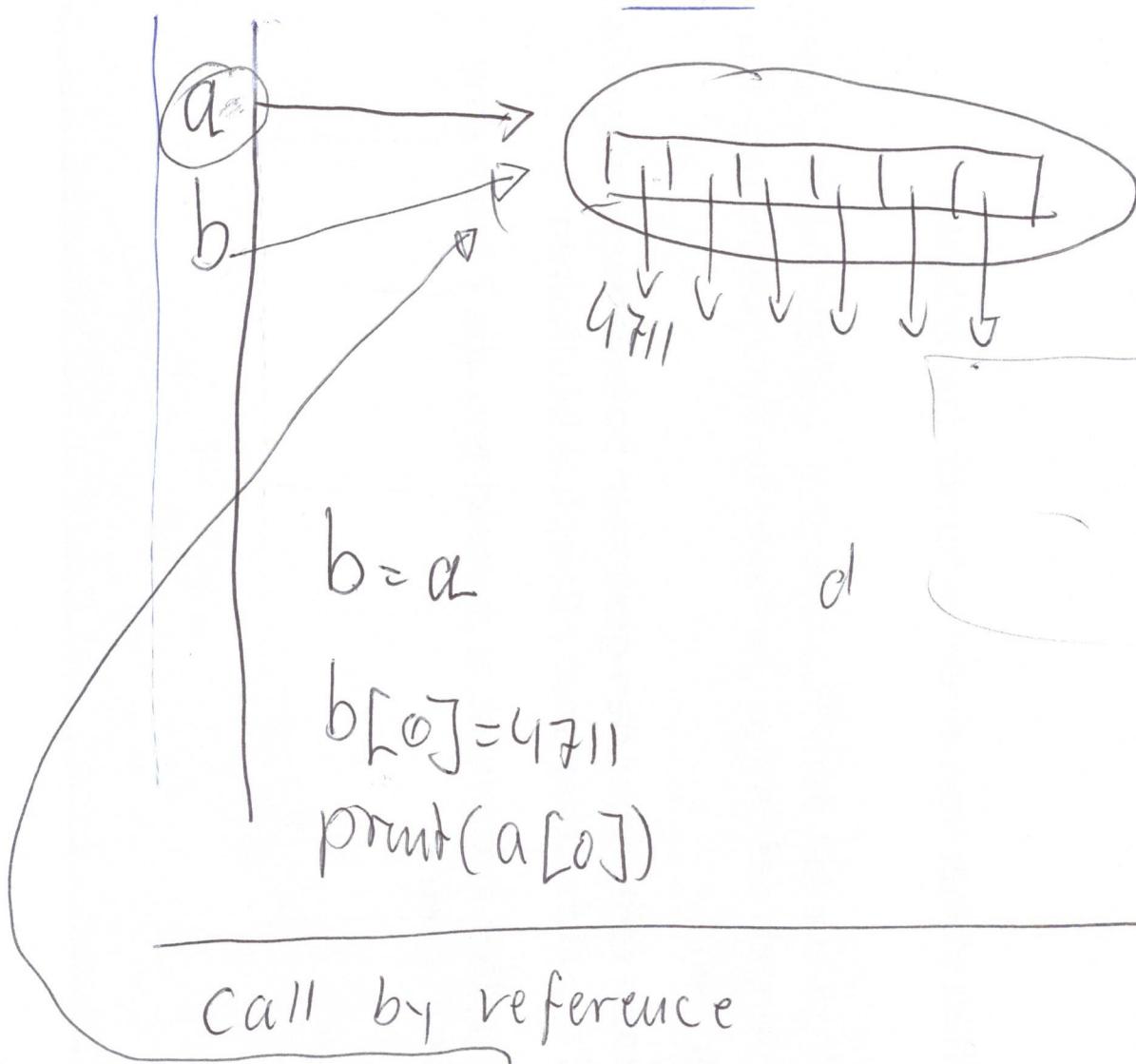
Welche Klassen Konnekt es geben?

Welche Methoden -n- sie haben?

Welche Daten -n- sie haben?

# Copy Problem

---



call by reference

`def x(y):`

$y = []$

---

`x(a)`

`print(a)`

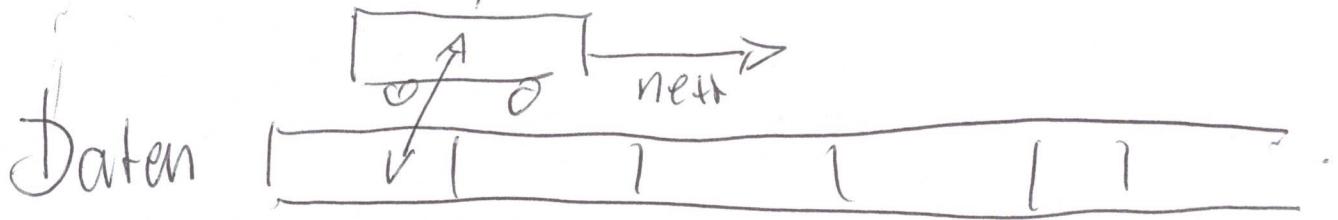
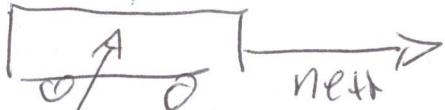
`a.copy()  $\leftrightarrow$  a[:]`

shallow copy!

S.o.l. import copy

# Iterable

Iterator von Daten



Class Daten

u.a. --iter-- definieren, er gibt  
Iterator-von-Daten zurück

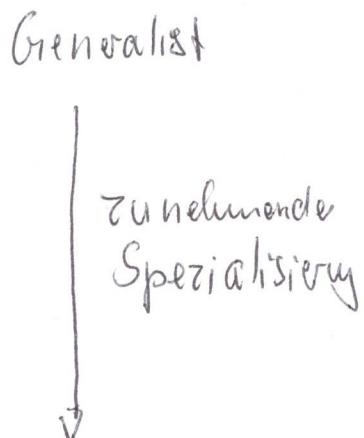
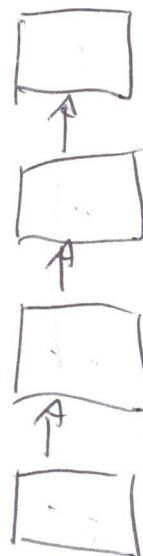
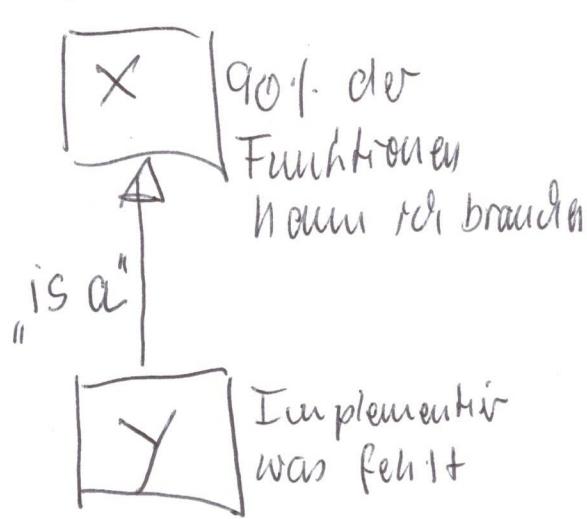
class Iterator-von-Daten

u.a. --next-- definier

---

# Vererbung

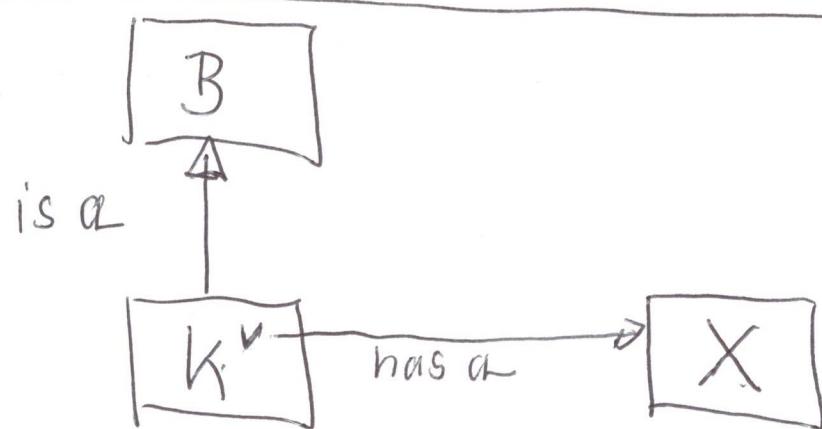
## Software Reuse



⇒ tiefe Klassenhierarchien



⇒ breite Klassenhierarchien



auch interessant: Code Injection

# Vererbung

class B:

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

=

def mach\_was(self)

=

def \_\_helfer(self)

=

def \_\_geheim(self)

=

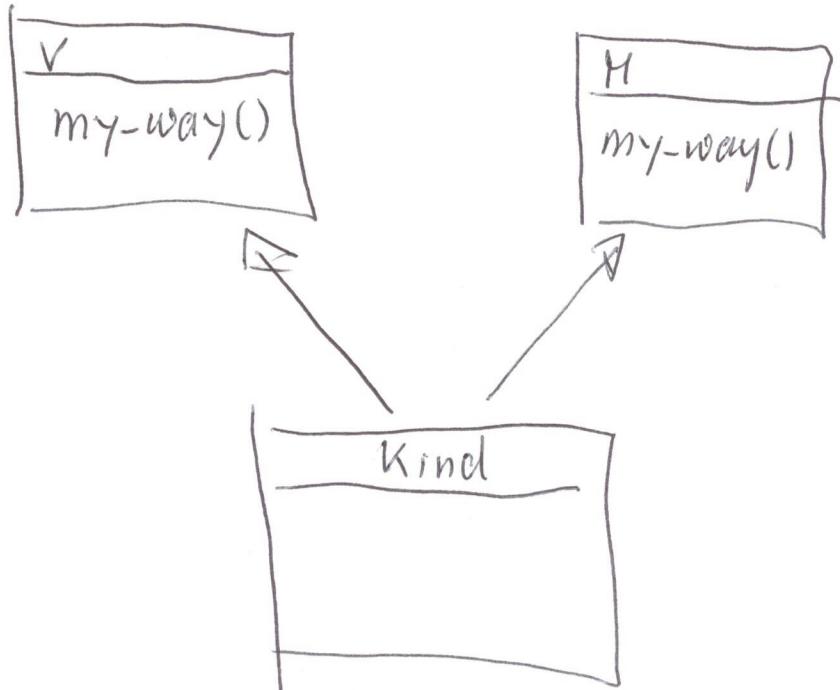
class K(B)

pass

\_\_init\_\_ → Super

# Vererbung

python erlaubt: Mehrfach-Vererbung



class Kind(V, M):

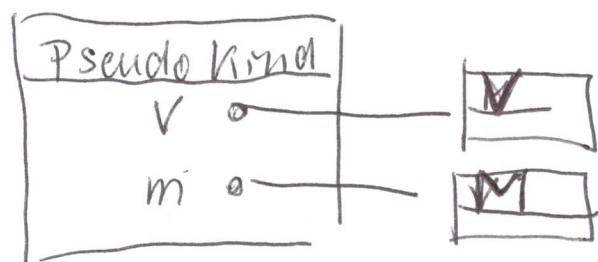
1. Vater importieren,
  2. Mutter --
- kennt my-way von M

class Kind(M, V):

1. Mutter
  2. Vater
- my-way von Vater

Empfehlung: Einfach Vererbung ist oft besser

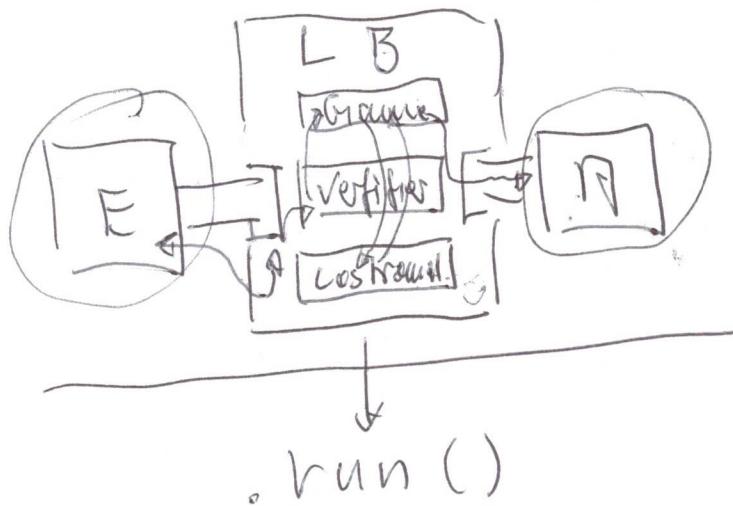
Muss V und M genutzt werden → has-a Vorgehen



# Programm

↓  
LottoBude / Eingabe / Ausgabe  
Grundklasse mit  
└ benötigte Klasse init  
└ Lostrommel  
└ Verifrier  
└ Grauue  
"Infrastruktur" für den Lauf

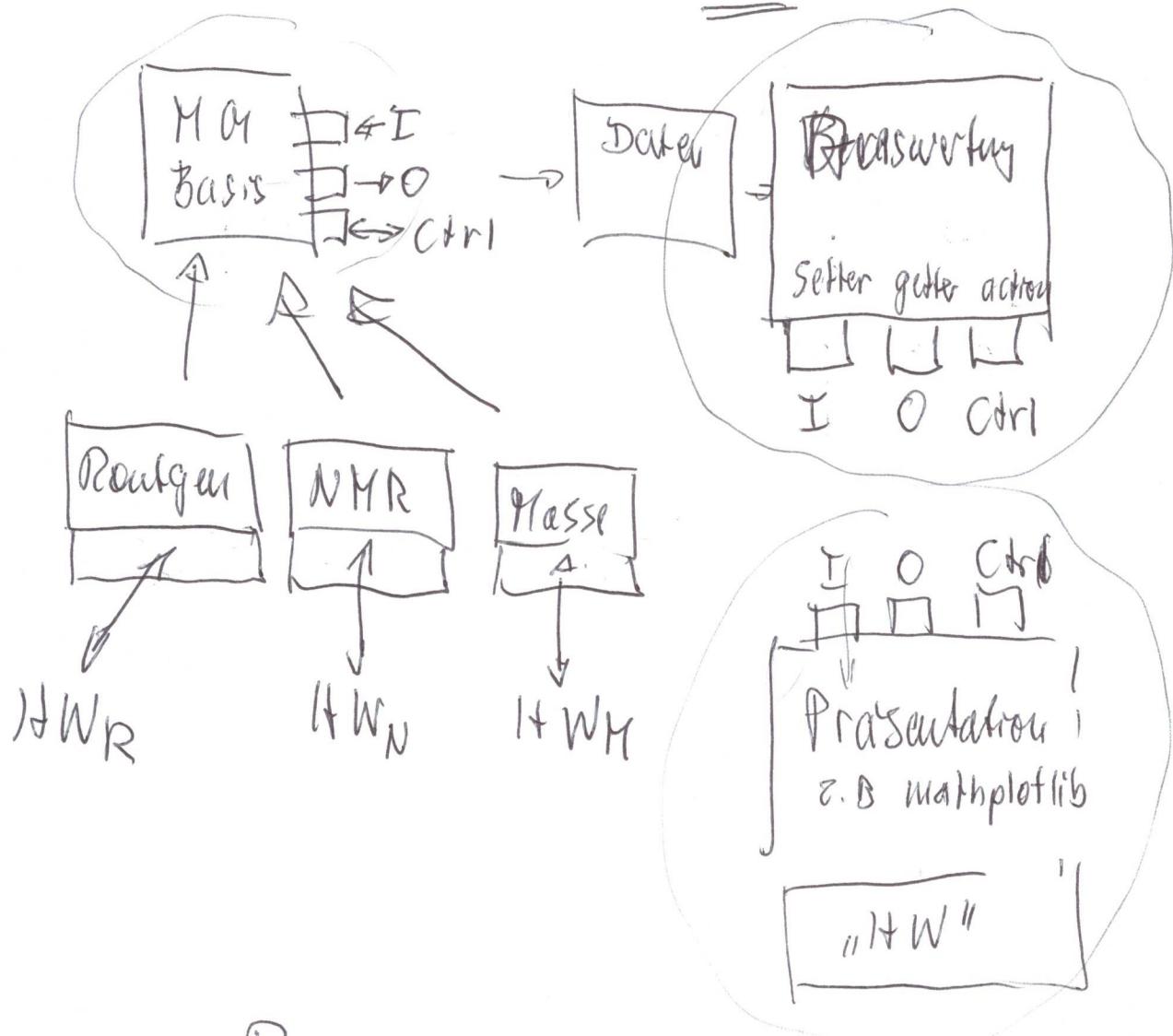
↓  
t



.run()

s.a. UML-Tools zur  
Unterstützung

# Versuchsaufbau



## Programm

a = Auswertung()

p = Presentation()

mg = [Röntgen(), NMR(), Masse()]

daten = Daten()

for h in mg:

daten.append(h.messung().get\_daten())

a.set\_daten(daten)

a.calculate()

~~a.send\_data(p)~~

p.set\_daten(a.get\_daten())  
p.print()

E

V

A

# Wiederholung

## Klasse

┌── methoden  
 ┌── properties = öffentliche Daten  
 │ lesen/schreiben  
 └── „magic“ — — dunder —  
 ┌──

} öffentlich  
 } hielten - name  
 } private -- name

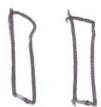
— — init —  
 — — str —  
 — — repr —

— — eq —  
 — — ne —  
 — — lt —  
 — — gt —  
 ...  
 — — old —

Objekt / Instanz  
 wird im Prog.  
 genutzt

→ self-Referenz

zw. Objekt variable  
 + Code der Klasse  
 + Daten des Objekt



Variablen auf Objekte: immer Referenzen

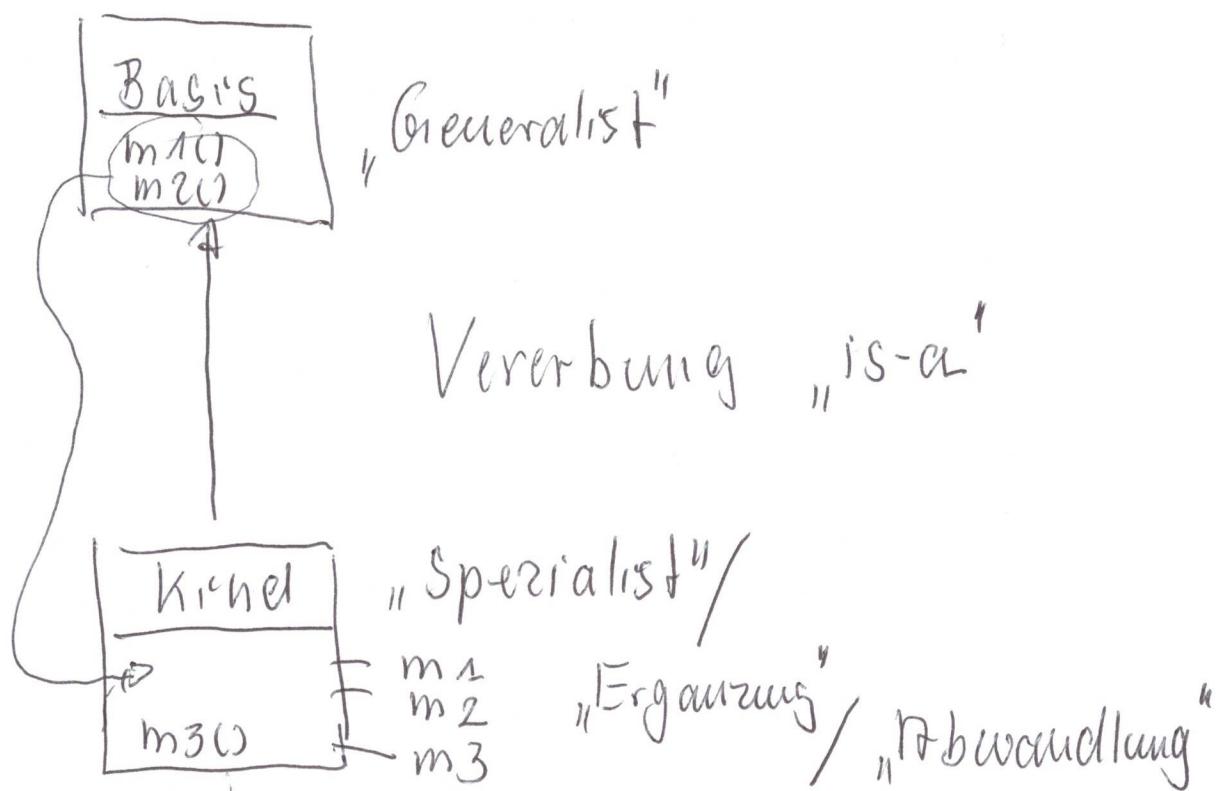


$\Leftrightarrow$  evtl. evne Copy-Problematik

evtl. copy-Methode realisieren!

— copy — / — deepcopy —

# Wiederholung



Ergänzung: zusätzliche Methoden

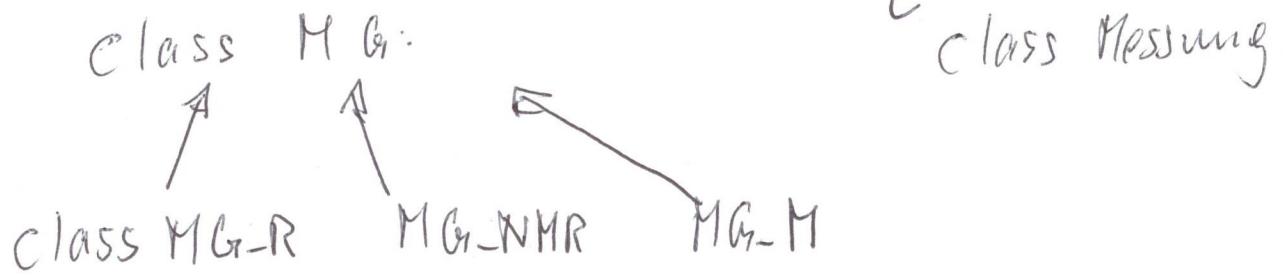
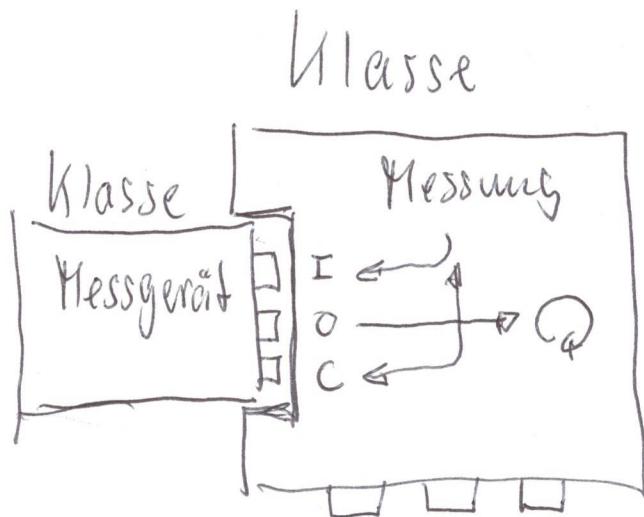
Transformation: geerbte Methoden werden

überdeckt.

ggf mit super()  
auf geerbte Methode  
intern zugreifen

- ▷ ein überdeckter `--init--` wird
- nicht automatisch in der Basis aufgerufen  $\Leftrightarrow$  Initialisierung in der Basis fehlt  
Methoden mit `--name` werden hier offiziell vererbt.

# Übung



$m = \text{Messung}(\text{MG}())$

$m.\text{run}()$

for  $mg$  in  $\text{MG-R}(), \text{MG-NMR}(), \text{MG-M}()$ :

$m = \text{Messung}(mg)$

$m.\text{run}()$

$m.\text{print}()$

# Dokumentation

doc - String - Infos für Benutzer

#Kommentar - Erklärung am Code  
für Programmierer

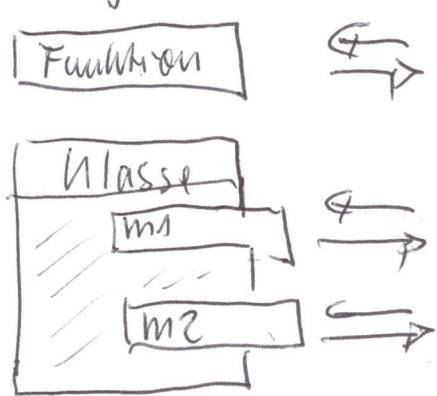
{ help (Objekt) liefert doc-Text interaktiv  
pydoc auf Konsole  
oder HTML

doc - muss existieren

- Modulbeginn
- Klasse beginn
- Funktion/Methode beginn

Teslen

## Programm

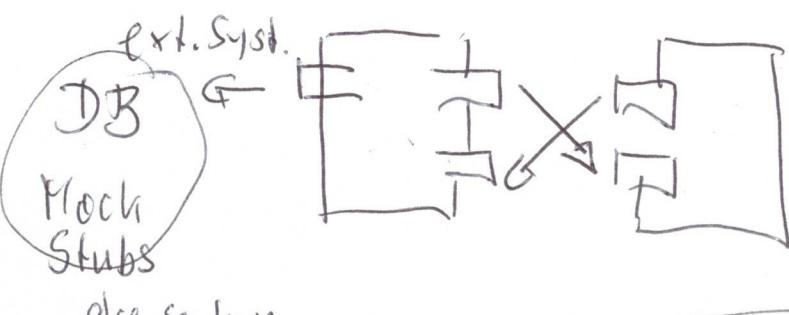


## Unit-Tests

{ test1  
test2  
test3

{	test1	good/happy path
	:	bad cases
	:	edge cases
	testNN	

Testanrede: Eingabe  $\Rightarrow$  Ausgabe



## Komponent-Tests Integrations-Test

drei so fun  
als ob DB Waren  
und auf vordefinirte  
Eingaben / Abfragen  
kommen gleiche Ergebnisse  
Liefern  $\Leftrightarrow$  Reproduzierbarkeit!!!

unittest  
pytest  
moch-Bibs  
z.B. mochito