Python Grundlagen

Ulrich Cuber
kontakt@uc-it.de

Mi	9:00 - 12:00	13:00 - 17:00
Do	9:00 - 12:00	13:00 - ~~17:00~~ 16:45
Fr	9:00 - 12:00	13:00 - 16:00

45-50 min / 15 min

Python

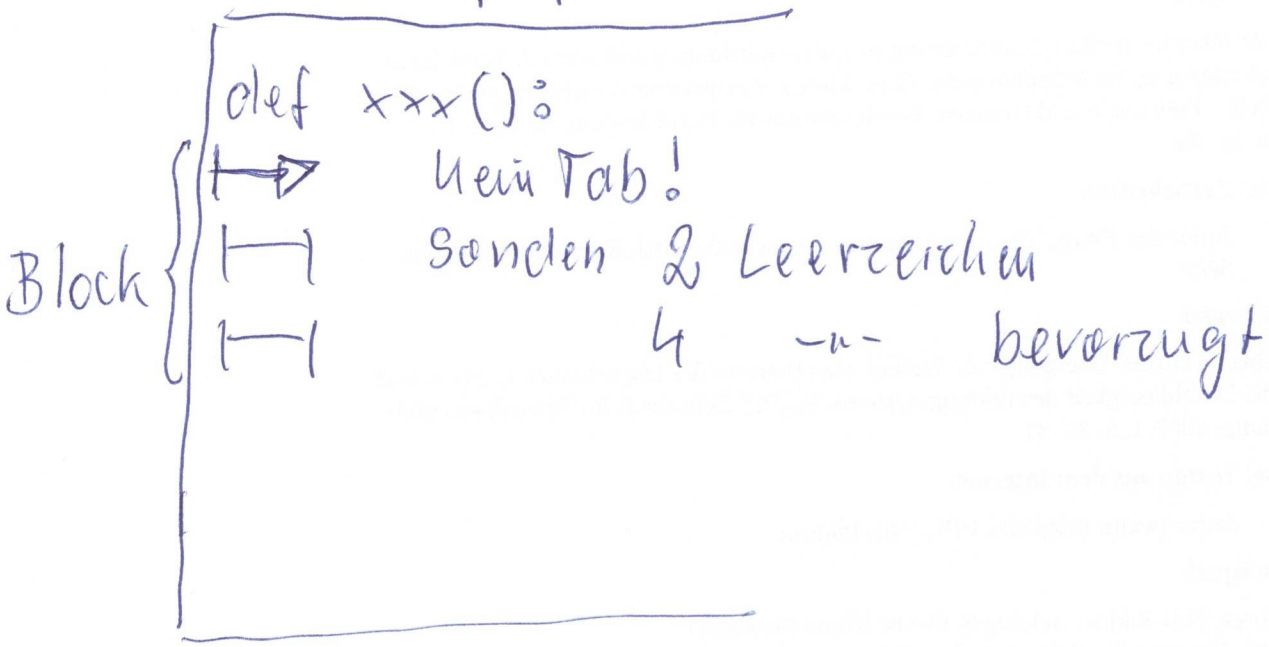
oder G. v. Rossum

objektorientiert

leicht lernbar

Guten Stil „erzwingt"

↳ pep8



obsolet: 2.7.X print "xxx"

aktuell: 3.X → 3.9 print("xxx")

3.6 → 3.9

Datentypen

V = ??? ← Datentyp ↔ typische Operation

Boolean True/False

Integer 1234

Float +12.34 −12.34

max 15 Stellen + Rundung

a*b/b ≠ a

Strings "____" '____'

(Complexe Zahlen)

List [ ]

Tupel ( )

Dict { }

„Nix" None

---

alle Klassen und ihre Objekte

Operatoren

\+ - * % // / **

% modulo

// integer Division

/ Division

=

Umwandlungsfunktion ("cast")

str → int int(x)

int → str str(x)

⋮

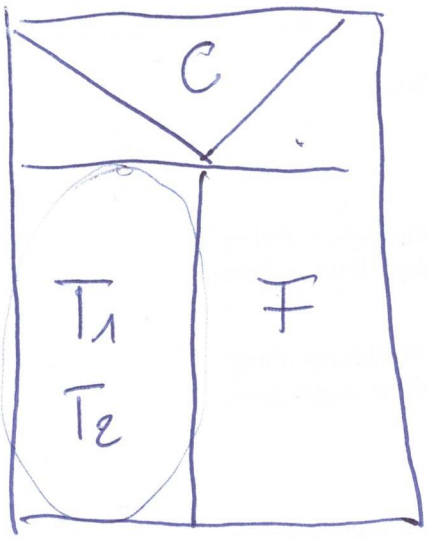
analog list, dict, bool, float, ...

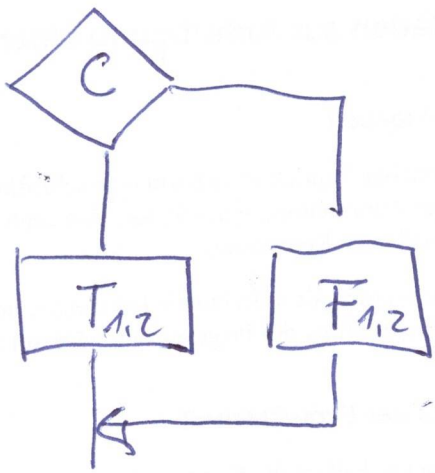
< > <= >= == != is

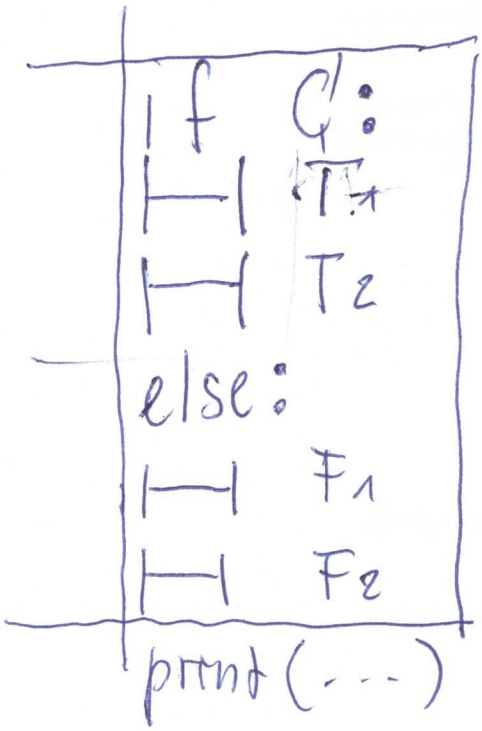
not and or

---

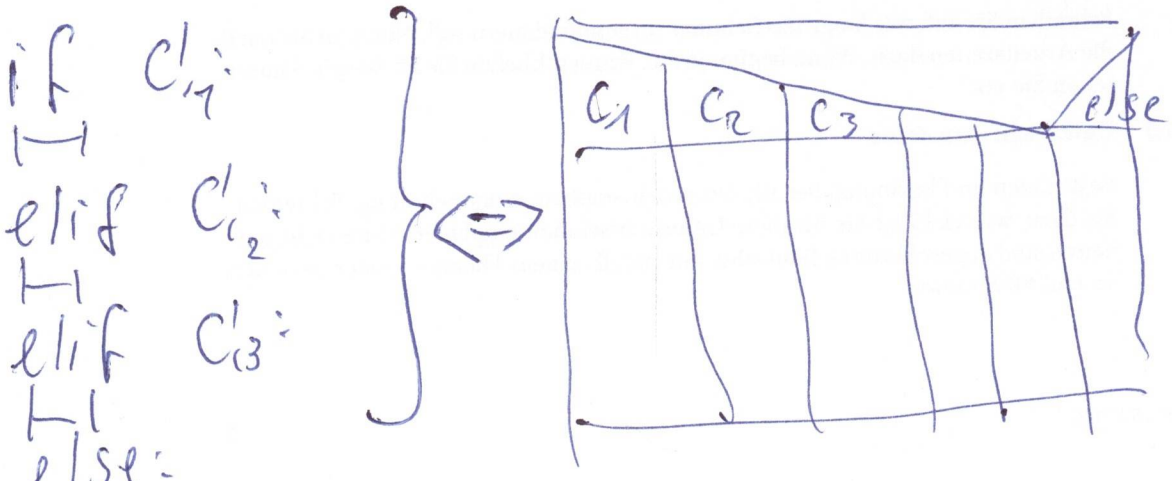
in

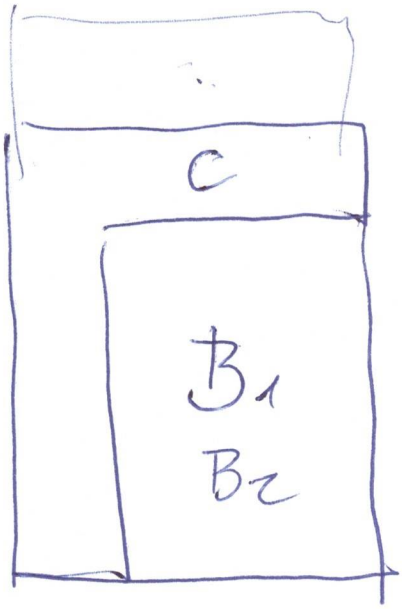






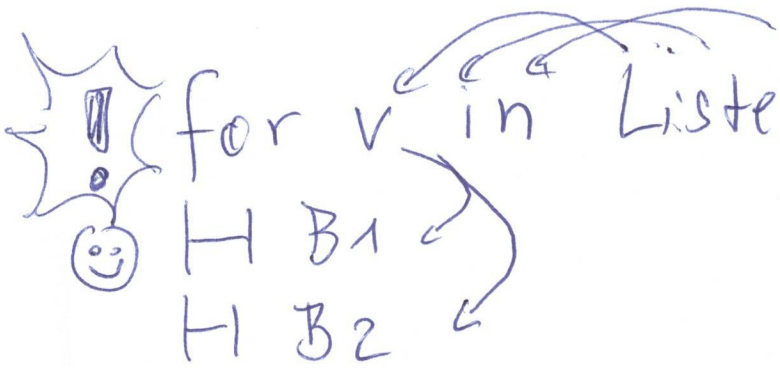

```
if val > 5:
    T1 print("ist grosser")
    T2 erg = val **3
    T3 print(f"{erg}")
else:
    F1 print("passt nicht")
```

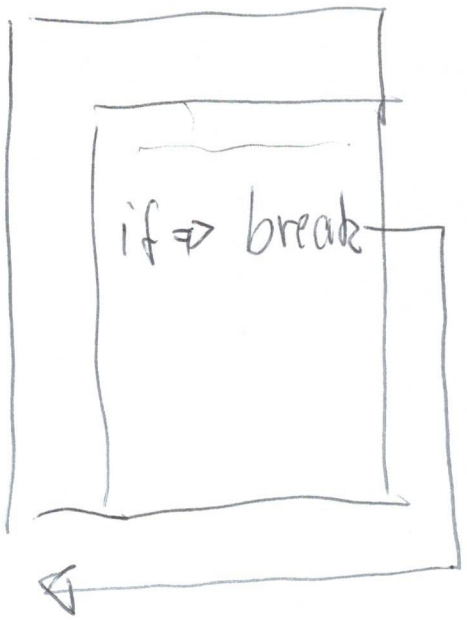


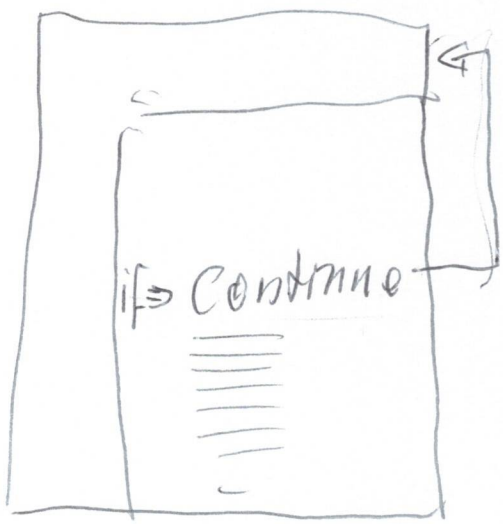


while C:
⊢ B1
⊢ B2

„offene Schleife"







if eingabe == "EXIT":
break

not
if eingabe.nothmeth
continue

String

"Text" 'Text'

✓ "Willi's" 'Willi\'s' "Er sagte:\"ich bins\""

Escape

'Er sagte:"ich bins"'

"Text" Unicode, von py ausgewertet

u"Text" Unicode

r"Text" Roh, nicht interpretieren durch p

b"xxxx" Binär zu interpretieren

f"Text{}" Formatiert

pyformat.info

String

konstante

s = "Hello, world" len(s) → 12

list(s) → ['H', 'e', 'l' ..... 'd']

index | 0 1 2 ..... 11 |

0 - 11

-1 - -12

s[0] → 'H'

s[11] → 'd'

s[len(s)-1] → 'd'

s[-1] → 'd'

s[-2] → 'l'

Slice

s[:5] → Hello

s[7:] → world

s[2:5] → llo

s[::2] → Hlo ol

String

Methoden

String.methode(Parameter)

Aufgabe

Benutzereingabe bis "EXIT" eingegeben wurde, → Ende

Ausnahmen: wenn die Eingabe ein Wert ist (int oder float) soll das aufsummiert werden
wenn nicht: kurze Meldung und weiter geht die Schleife

Ausgabe der Gesamtsumme

Exception

Versuch – Fehlerreaktion

```
try:
    Versuche
( except:          "catch all"
    Fehlerreaktion )

except ValueError as e:
        Fehlerbeh.

except IndexError as e:
    =

except FileNotFoundError as e:
    print(e)
```

Exceptions

```
try:
    ...
except:
    ...
finally:
    ...
else:
    ...
```

try: / except

try / finally

try / else

finally: auf jeden Fall ausgeführt

else:

wirft Exception: raise

Liste

len(l) ⇒ N = Anzahl Elemente

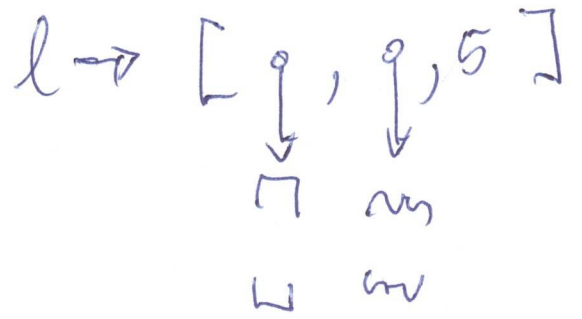
l.index() 0 - (N-1)

l[0], l[-1]

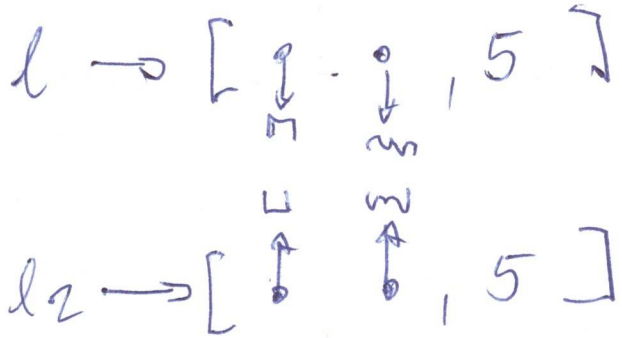
l[:3], l[5:], l[::2] Slicing

---

l[:] ⇔ l.copy() flache Shallow Copy



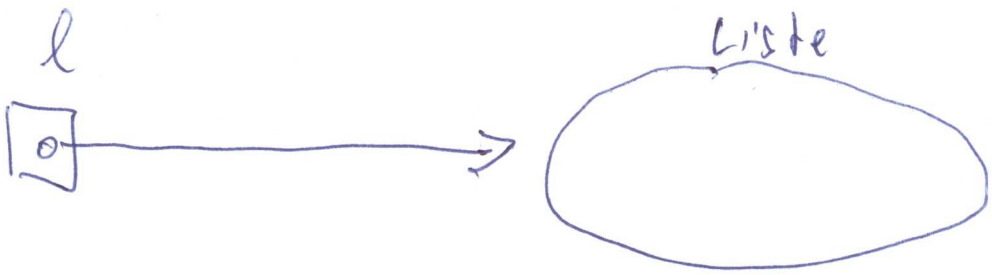
l2 = l[:] oder l2 = l.copy()



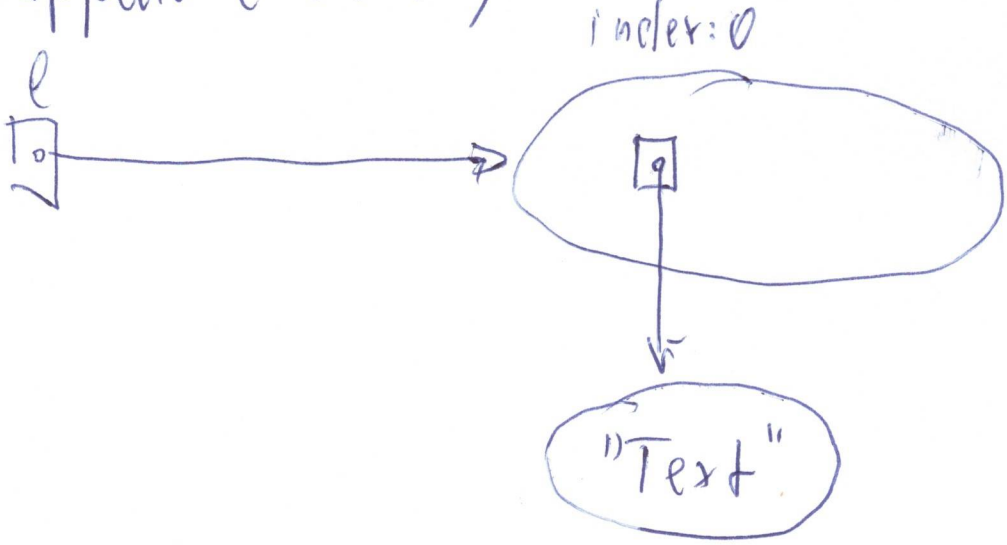
Liste

l = Objekt

l = [ ]



l.append("Text")



Listen

l = [ ] → liste ✓

l = list( ) Inhal → None = ∅

l = [1, 2, 3, 4] l vom Typ Liste mit Int Elementen

l = [1, 2.3, "Hey", [ ], { }]

---

l = [ ] ← "Text"

↑ neuer Speicher

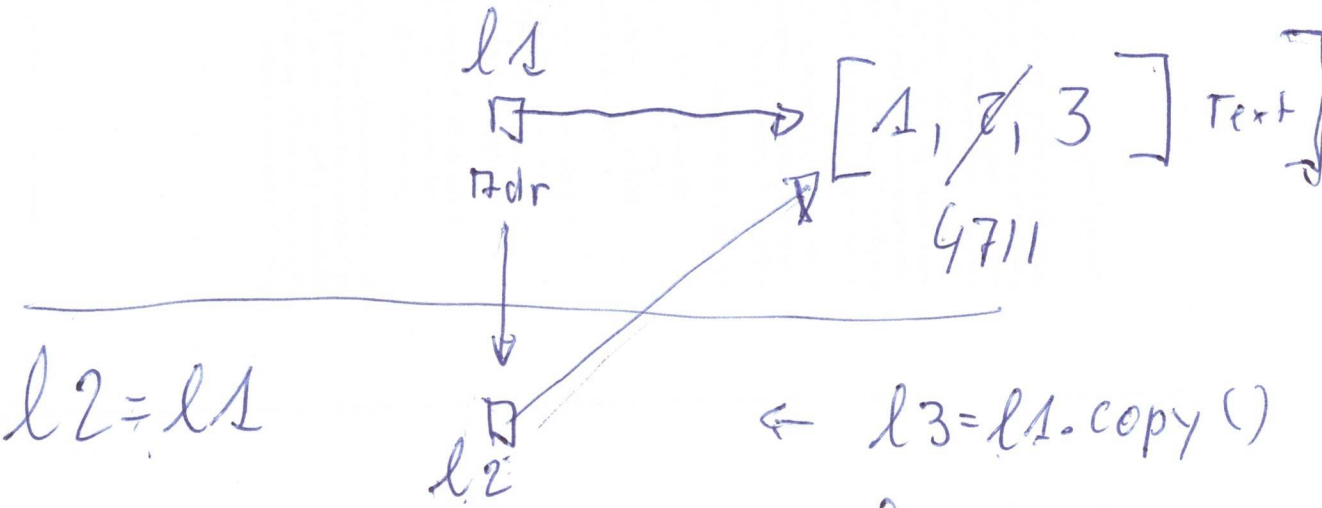
l.append("Text") l.insert("Text")

↓ ↙

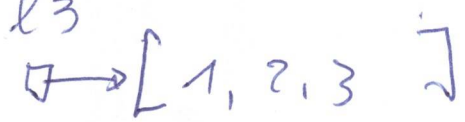
Speicher für 'Text' angelegt

Liste

$l1 = [1, 2, 3]$



$l2 = l1$ ← $l3 = l1.copy()$

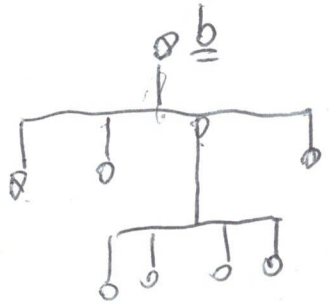



```
l2.append("Text")
l2[1] = 4711
```

`print(l1)` ⇒ 1, 4711, 3, Text

Datenstrukturen

Idee: Baum



---

b = {

"child": [{}, {}, {}, {}]

}

[{}, {}, {}, {}]

⇒ class Baum

__init__(self, **kwargs):

Liste: veränderbar [ ]

l[i] = neuerWert

l.append()

Tupel: unveränderbar ( )

„immutable"

Lotterie

(1) Erzeuge eine Liste:

- 6 Zahlen
- 1-49
- keine doppelten
- Zufallszahlen
  ⇒ random

Tip: erst suchen
und dann wenig code

(2) Benutzer soll 6 Zahlen eingeben

- 6 Zahlen ⇒ int
- 1-49
- keine doppelten

Eingabeschleife bis 6 komplett
- input
- int() → try-catch
- if 1 - 49
- ~~for~~
- if neue zahl in Liste
- neue zahl anhängen

Dictionary

Hash, assoz. Array, Key-Value-map

```
d = {}
d = dict()
```

↳ [(k,v), (k,v), (k,v)]

z.B. Umbrinatron
pack, zip

```
d = {"key": value, "key2": value}
d = dict(key=value, key2=value)
```

Dictionary

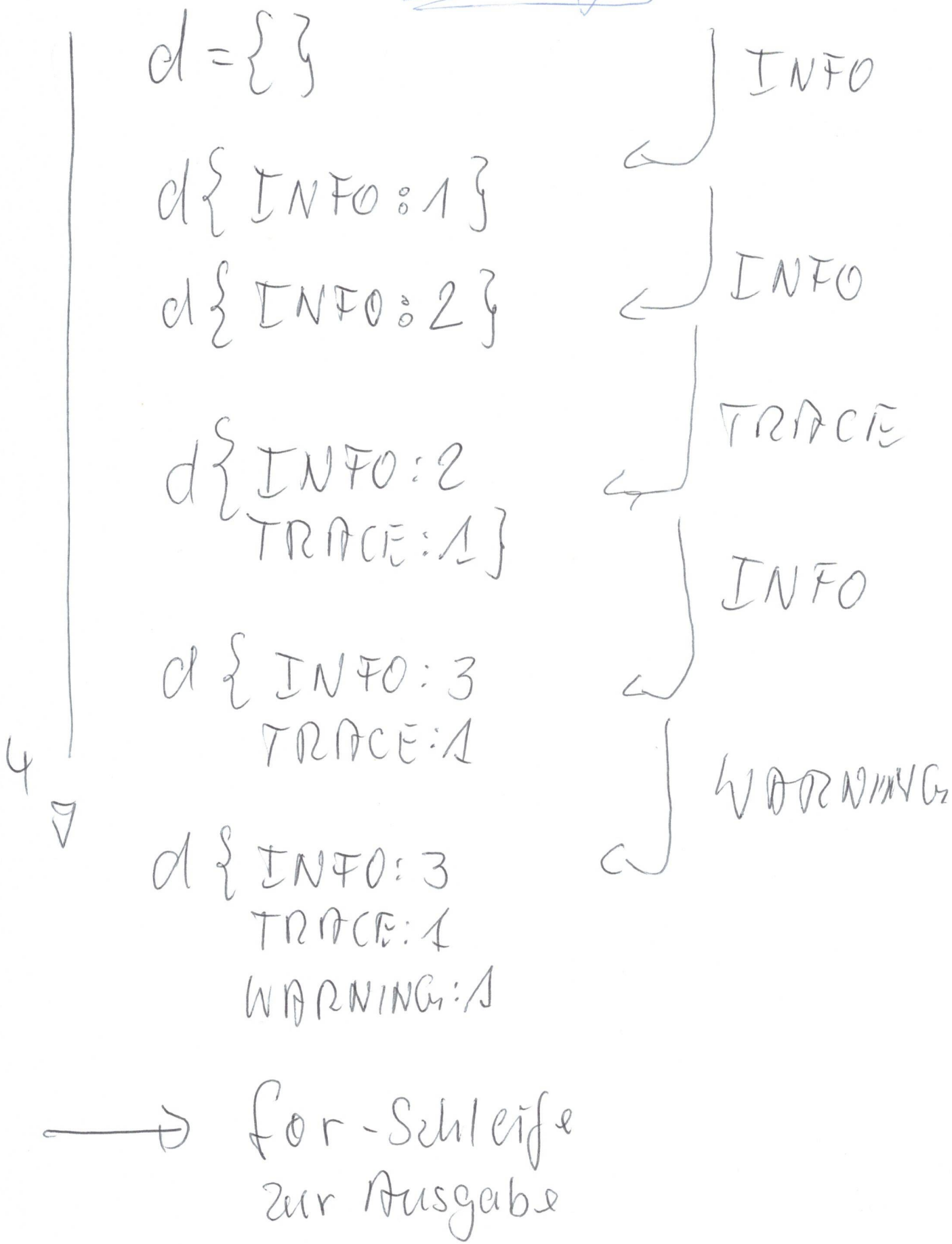
```
d = {}
d["Name"] = "Willi"
d["PLZ"] = "01234"
```


```
d["Name"] = "Heinz"
```

```
print(d["Name2"])
```
Name2 ex. nicht ↯

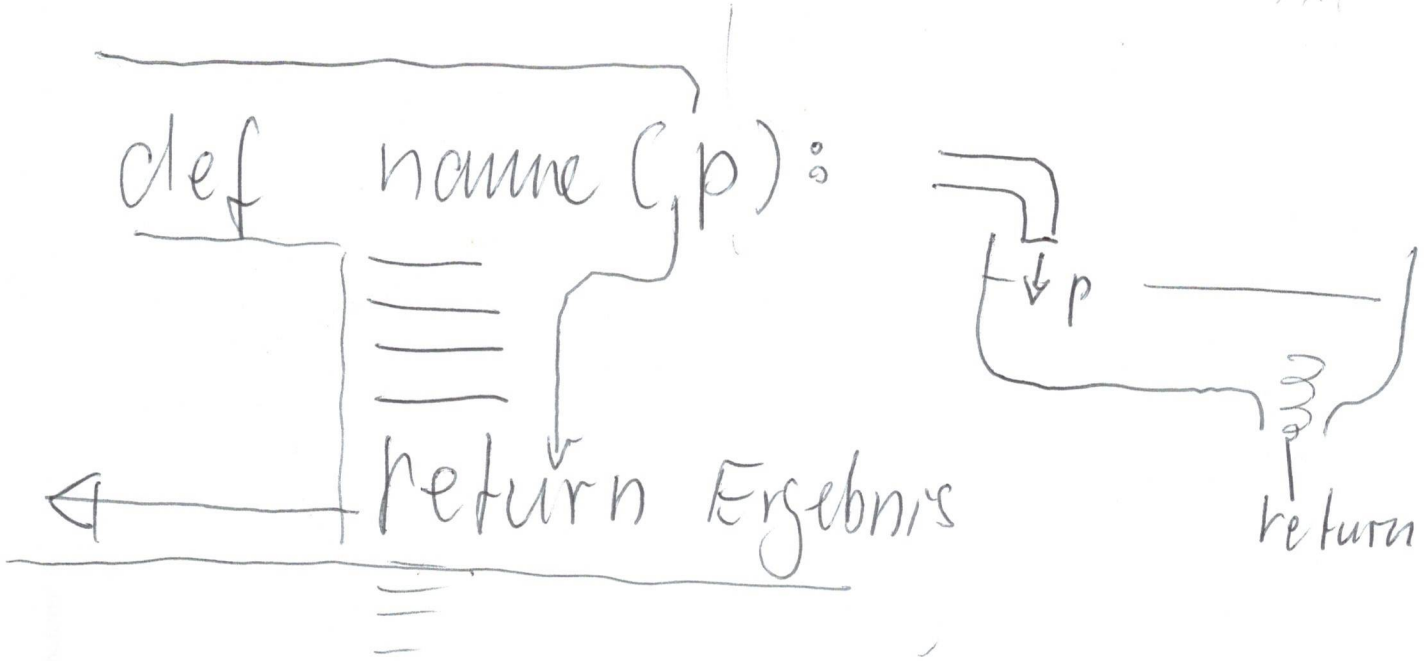
```
d.get("Name2", None)
```
Name2 ex. nicht → None

Zur Aufgabe



→ for-Schleife zur Ausgabe

Funktionen



def ....

yield ⟷ generater F.
F. def. in Funktion

} python F.
geschrie

Funktionen

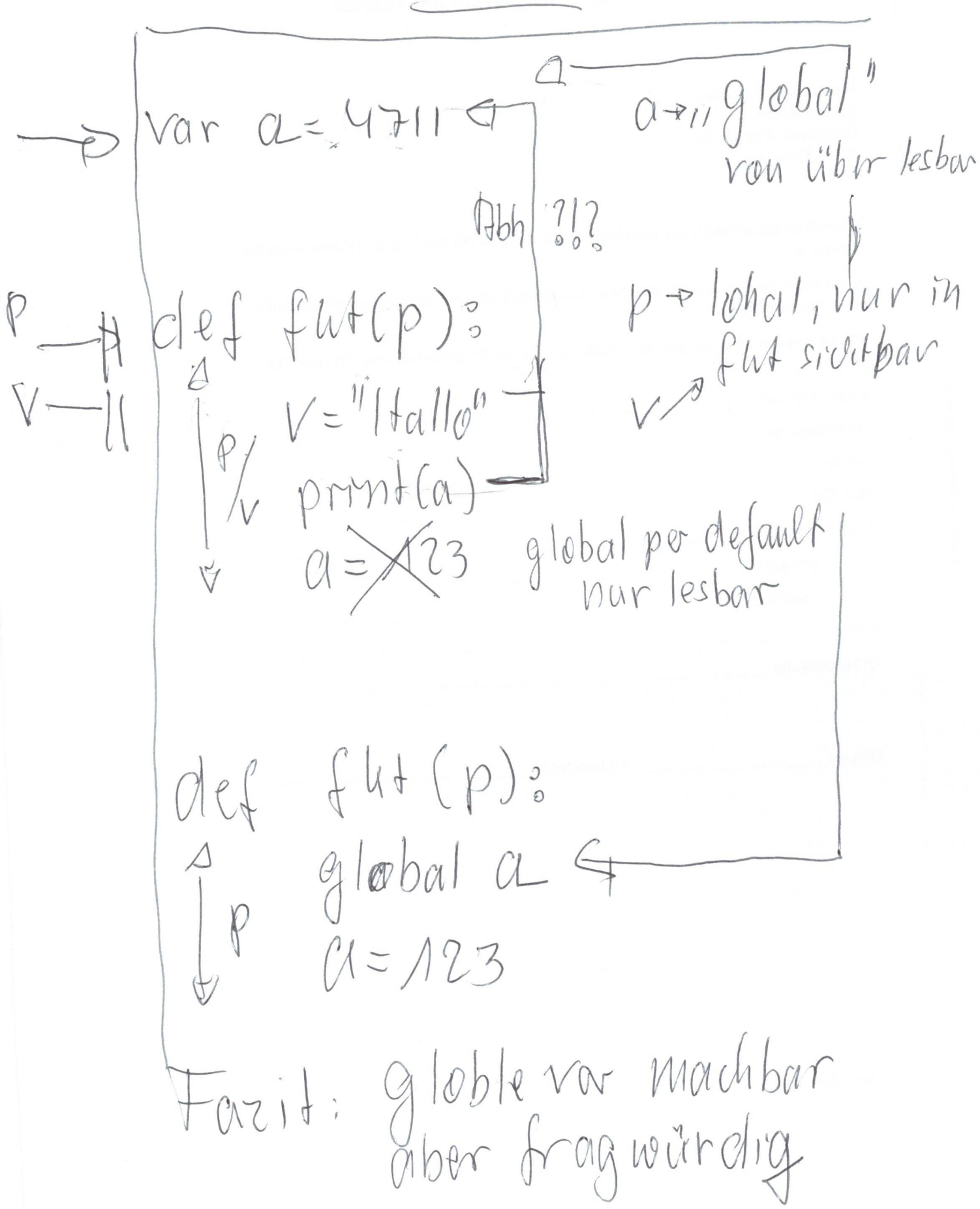
```
def name(.):
    Funktionskörper
```

```
def name():
    """
    Dok. String   <- pydoc -> HTML
    """
    Funktionskörper
```

name ← „Adresse" wo die F. liegt

name() ← Aufruf der F.

Scope von Var.



Farit: globle var machbar aber fragwürdig

Parameter

f() null Parameter!

f(a) 1 Pflicht

f(a,b) 2 Pflicht

---

f(a, b=None) 1 Pflicht f(4711)
optional 1 Opt f(4711, 123)
f(4711, b=123)
↑
named param.

f(a, b=None, c=4711)
opt

f(a, b, c=None, d=4711)

Pflicht opt.

Parameter

Parameter Liste

f(a, *args) ⇒ f(1) f(1,2,3)
f(1,2) f(1,2,3,4...)

a: 1
Rest: [2] [2,3] [2,3,4,...]

Parameter Dict

f(a, **kwargs) f(1, x=5)
f(1, x=5, y=9)

Rest:
{"x"=5} {"x"=5, "y"=9}

---

f(a, b, o1=None, o2=None, *args, **kwargs)

p p o o Liste Dict

f(1, 2, 3, 4, 5, 6, 7, a=3, b=5)

Module

import Modulname

„Namensraum"

Hauptprogram: „main"

[..., ..., ..., a, Modulname]

Modulname = Namensraum [..., ...) ***, ***]

„Modul liegt auf der Platte"

Pfad + (Name ⇐ Namensraum)

default pfade → Speicherort der py-Version

user pfade

distrib-package - Standardbib

site-package - Nachinstallierte Pakete

pip conda

s.a. „virtualenv"

Module

import xxx

lädt die ganze Bib

first come first serve in sys.path

main
└[..... , xxx]

Aufruf xxx.f()

---

import xx.yy.zz

Aufruf xx.yy.zz.f()

lädt nur f

a) from xx.yy.zz import f

f()

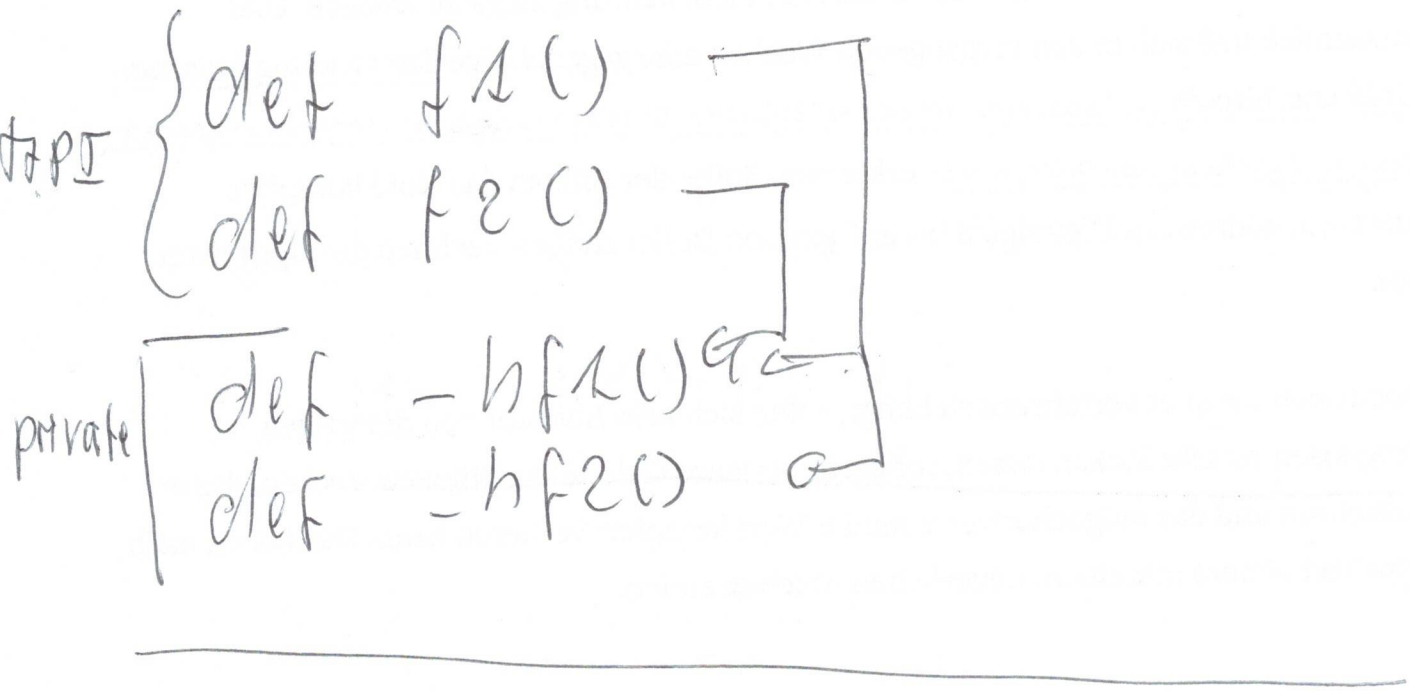
main
└[..., ..., f]

import xx.yy.zz as lib_1

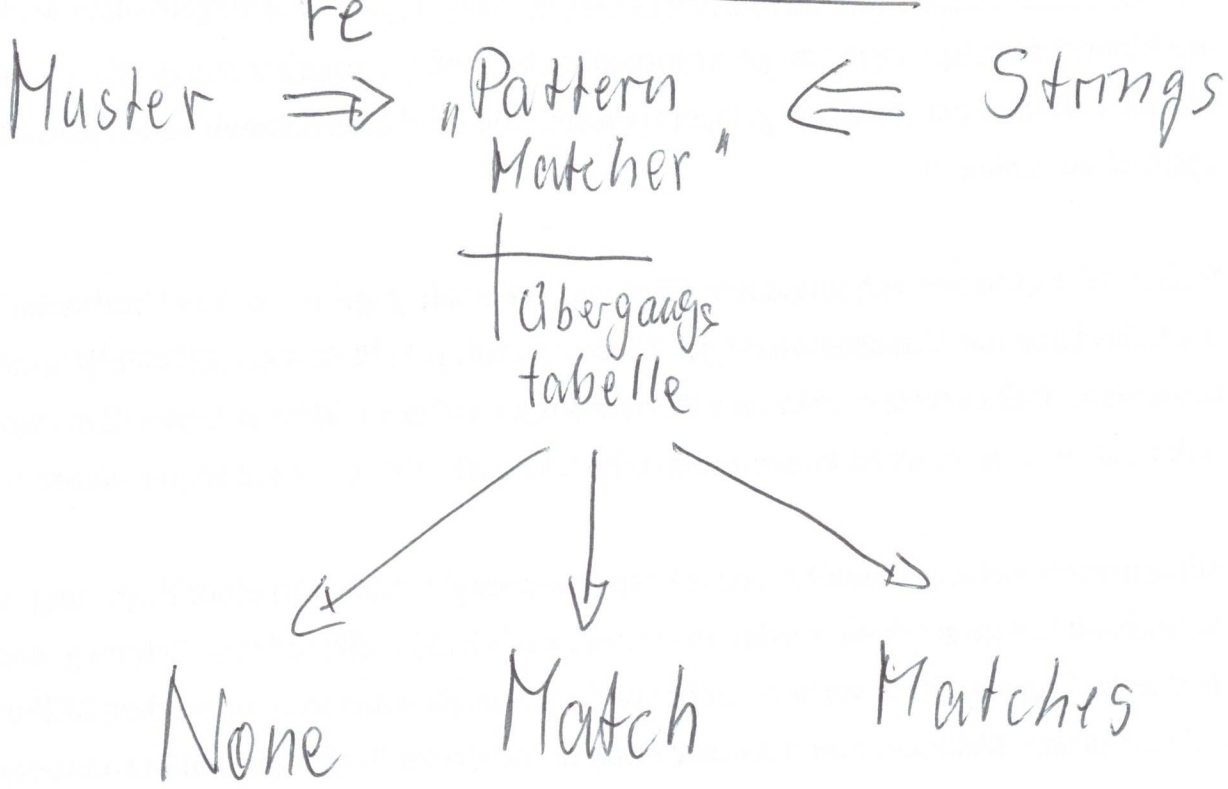
main
└[..., ..., lib_1]

lib_1.f()

Module layout



Reguläre Ausdrücke




```
import re

re.search ( Muster, String)
re.find (                  )
re.findall (               )
re. ...
```

```
daten = read_very-big-file()
for zeile in daten:
    if re.search(M, zeile):
        mach_was(zeile)
```

2 ms 1. M parsen
1 ms 2. M → Tabelle umwandeln
3 ms 3. Zeile mit M untersuchen

6 ms × 1 Mio

---

für grosse Dateien besser:

3 ms `rex = re.compile(M)`

```
for .....
    if rex.seach(_...)
```

3 ms × 1 Mio

---

search → 1. Treffer
findall → alle Treffer

Rex

.	jedes Zeichen	
\	hebt Bedeutung	z. B \. Punkt
	gibt Bedeutung	\d Digit
[von-bis]	Zeichenklasse	||d
	eine Position/Zeichen	
[^]	Gegenteil	
|	oder r1|r2	entweder r1 oder r2
( )	Gruppenbildung	

---

^	Anfang einer Zeile	^Bla
$	Ende — " —	\.mp4$

Rex

r *	0-∞
r +	1-∞
r ?	0,1
r{v,b}	v - b mal
r{n}	n mal

\d* c -?-> Hello
⇓
m!

IPv4 Adr: |xxx.xxx.xxx.xxx|

email xxx@xxx.xxx