

Ulrich Cüber

kontakt@uc-it.de

Python GIL

Mo 9 -12 | 13 - ~17

Di 9 -12 | 13 - ~17

Mi 9 -12 | 13 - ~17

45 - 50 min

15 min Pause

Erklären

Zerlegen

Machen

Python

1994 Guido v. Rossum
2019

2.7.x

3.x

2008

3.9

3.6 ... 3.9

Python - pip-Installer

Anaconda - Conda-Installer

Linux:

Skript.py → chmod +X Skript.py

```
#!/usr/bin/python3
```

Interpreter

Answersheet

Syntaxcheck

Answering

Vorübersetzung

if Bedeutung:

Ausführung ↗

The diagram consists of two horizontal arrows originating from the left edge of the page and pointing towards a central rectangular box. The top arrow is above the bottom one. Both arrows point to the right, and they are positioned such that they appear to be merging or pointing at the same target.

Antworten

Antwortsung

Kommentar

Answers may

\$./Shmpt.py

C:\> python script.py

Eingabe / Ausgabe

- Quelle:
- * Interaktiv — input
 - * Kommandozeile — sys.argv
 - * Datei ~ open
 - Datenbank — Bib + Query
 - Nebquelle — Bib + Request

2.7

print « "Text"

raw_input → str

input → typ

Umwandlung

3.X

print("Text")

✓

input → str

Variablen (Anlage)

Namensraum

"main"

"a"

= 4711

int
4711

a = "Willi"

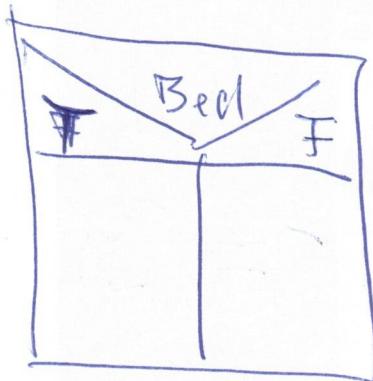
Adresse

Referenz

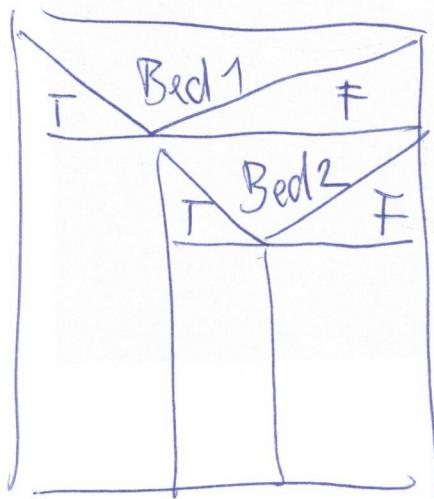
String
"Willi"

Daten

Verzweigung



if Bed:	\rightarrow	True / False
	$\neq 0$	/ 0
E		
else:	nicht None	/ None
E	Voll	/ Leer
	"m^n	/ "'''



Log / Vergleich

$a == b$

$a < b$

$a \leq b$

$a != b$

$a > b$

$a \geq b$



True / False (boolean)

not Bed. - Verneinung

1	0	1
0	0	0
1	0	1

$B1$ and $B2$ - und

$B1$ or $B2$ - ODER

1	0	1
0	0	1
1	1	1

$B1 \cdot and \ B2$

False \dashv

True $\longrightarrow ?$

$B1 \oplus B2$

False $\rightarrow ?$

True \dashv

short circuit eval.

Typumwandlung

Str → int → str

int(str) str(int)

Str → float →

float(str) str(float)

int → float →

int(float) float(int)

list()

dict()

BMI-Rechner

Eingabe Gewicht Grosser
V Berechnung
A Bewertung ob ob oder nicht

kg

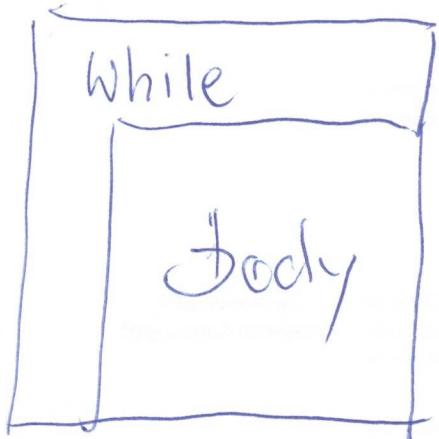
m²

Normal 18.5-24.9

Zavrel ≥ 25

Zuwenig ≤ 18

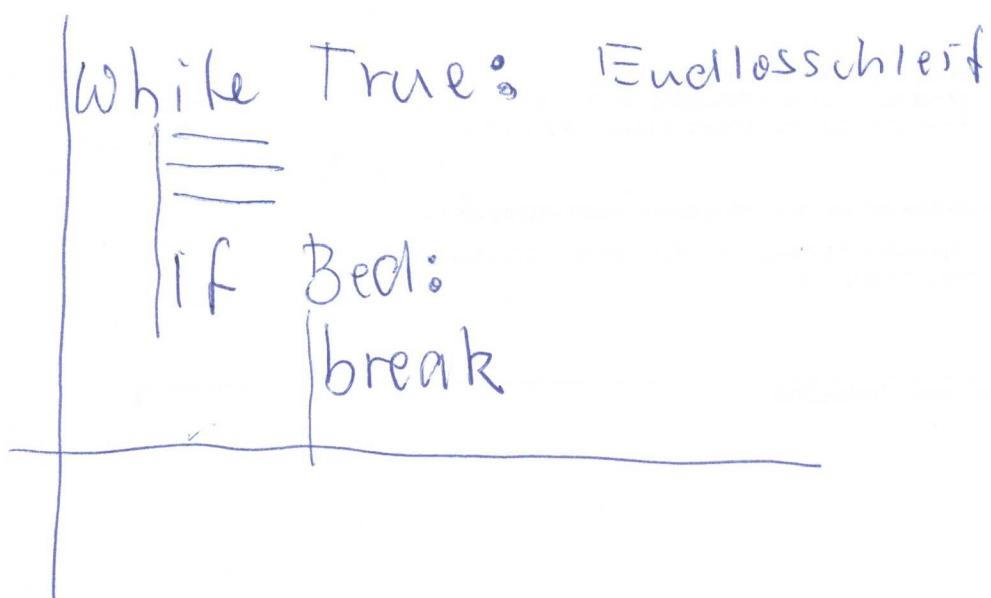
Schleife



while Bedingung:
 |
 | Dnw
 | Dnw
 | Dnw

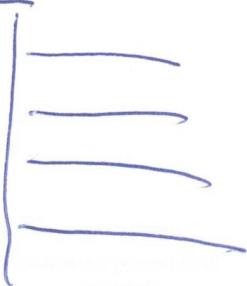
break — Springt raus
„Abbruch“

continue — Sofort weitermachen



Schleife

for Zielvariable(n) in Liste:



for zeile in text:
print(zeile)

A diagram illustrating nested loops. The outer loop is defined by 'for zeile in text:'. The inner loop is defined by 'print(zeile)'. Four arrows labeled 1 through 4 point from the start of the outer loop to the end of the inner loop: arrow 1 points to the 'for' keyword, arrow 2 points to the 'text' variable, arrow 3 points to the opening parenthesis of 'print', and arrow 4 points to the closing parenthesis of 'print'.

for i in range(1, 50):
 print(i)

for znr, zeile in enumerate(text)
 print(znr, zeile)

Adder ①

Solang der Benutzer nicht

{ "Ende" eingibt

Nimm die Eingabe als float
und
Addiere sie zu den vorherigen Eing

Gebe Resultsumme aus

String

Modifier

" "

String - unicode

u" "

unicode

r" "

raw - python interpretiert
Inhalt nicht

b" "

binary

f" "

format

f"m {x} m"

Inhalt

f"m {x:< } m"

{x:>}

{x:2}

Strings

$S = "Hello world"$

"Slicing"

$S[0:5] \rightarrow ["Hello"]$ „alles vor 5“
 $S[:5] \rightarrow ["Hello"]$ „alles vor 5“
 $S[7:] \rightarrow ["World"]$ „alles nach 7“

$S[5] \rightarrow \cancel{H} \rightarrow ,$

$S[6] \rightarrow \cancel{l} \rightarrow \square$

$S[::2] \rightarrow ["e l, w r d"]$ jeder 2. Buchstabe

String-Methoden

String. Methode()

"min". format()

s.startswith("Hello") \rightarrow True

Strings

$S = "uuuuuu"$

"Miss MM"

S[0]

S[-1]

77

```
S = "Hello, world"
```

Liste von Buchstaben statisch

$$\text{len}(S) \rightarrow 12 = n$$

Erstes Zeichen $s[0]$

Letztes Zeichen $s[\text{len}(s)-1]$
 $n-1$

$$S[-1]$$

Liste (Erste Schritte)

$l = [m_1, m_1, m_1, m_1]$

$l = \text{list}(x)$

$l = \text{funktion}()$

z.B. S. $\text{split}()$

for e in l:

 print(e)

 if e == "Hello":

 print("Begrüssung")

Aufgabe

Benutzereingabe mit input → String*

Erwartet werden 6 Zahlen

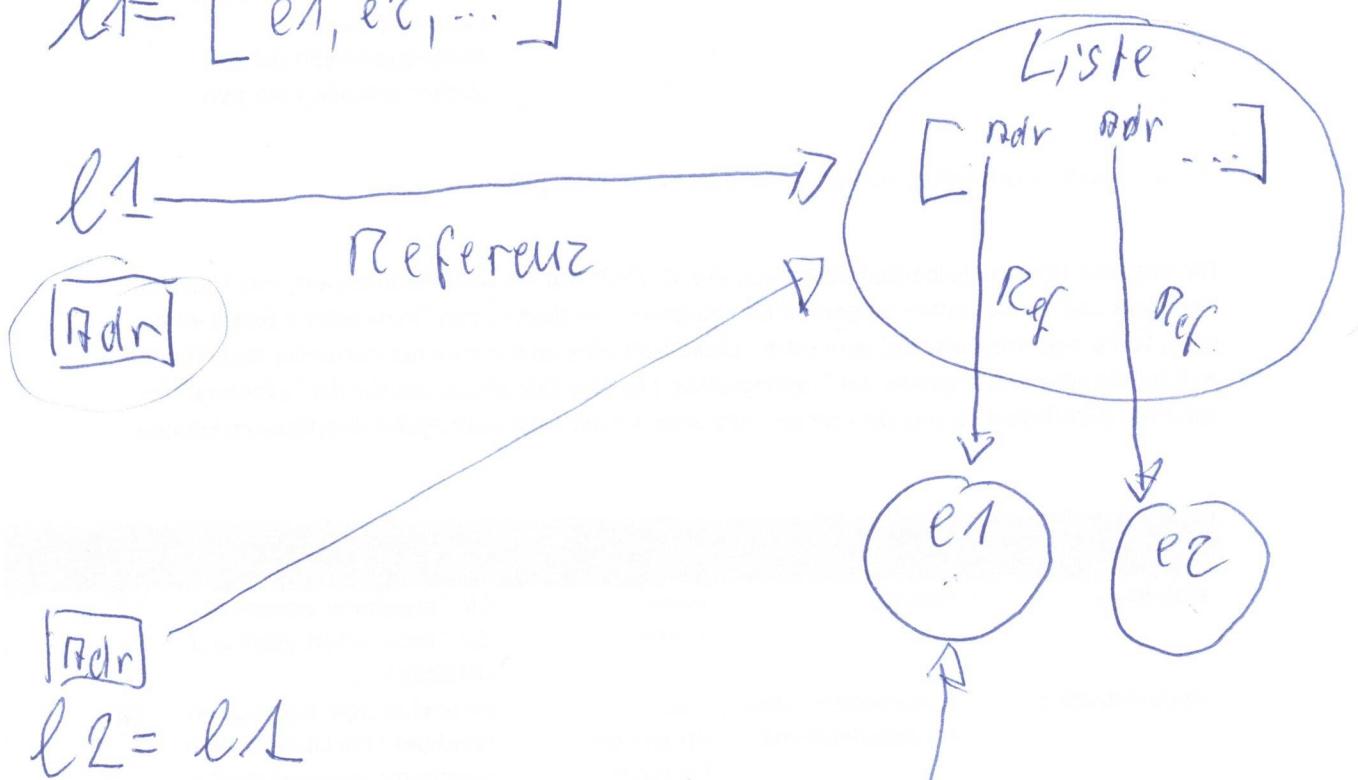
jede Zahl zwischen 1-49

* '123456'

```
for x in Eingabe.split():
    zähler hochzählen
    if "is Zahl"
        if "is 1-49"
            Meldung "OK"
        sonst
            Meldung "Fehler"
    sonst
        Meldung "Runder Fehler"
        if zähler l=6
            Meldung
```

Listen

$l_1 = [e_1, e_2, \dots]$



$l_2 = l_1$

$l_2[0] = 4711$

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

Referenz - flache Kopie
shallow copy

Liste

$l = []$ leere Liste

$l = list()$

$l = \text{listen_erzeuger}(\text{daten})$
z.B. " ".split()

neu Anhängen: $l.append(x)$

neu Einfügen: $l.insert(x, wo)$

Entfernen: $l.pop$

Löschen del $l[index]$

Liste

```
l = [ "Willi", 1234, [ "97765", "H14" ] ]
```

0 1 2

↓ ↓ ↓

str int str

↓ ↓ ↓

Hauz H14 str

print(l) liste

$$\text{len}(\ell) \rightarrow 3$$

$$\text{len}(l[2]) \rightarrow 2$$

$\ell[0] \rightarrow \text{Willi}$

$\ell[-1] \rightarrow ["22765", "411"]$

$\ell[-1][0] \rightarrow "22765"$

$\ell[-1][-1] \rightarrow "HH"$

$\ell[-1] \mathbb{J}[-1]$ = "Hamburg"

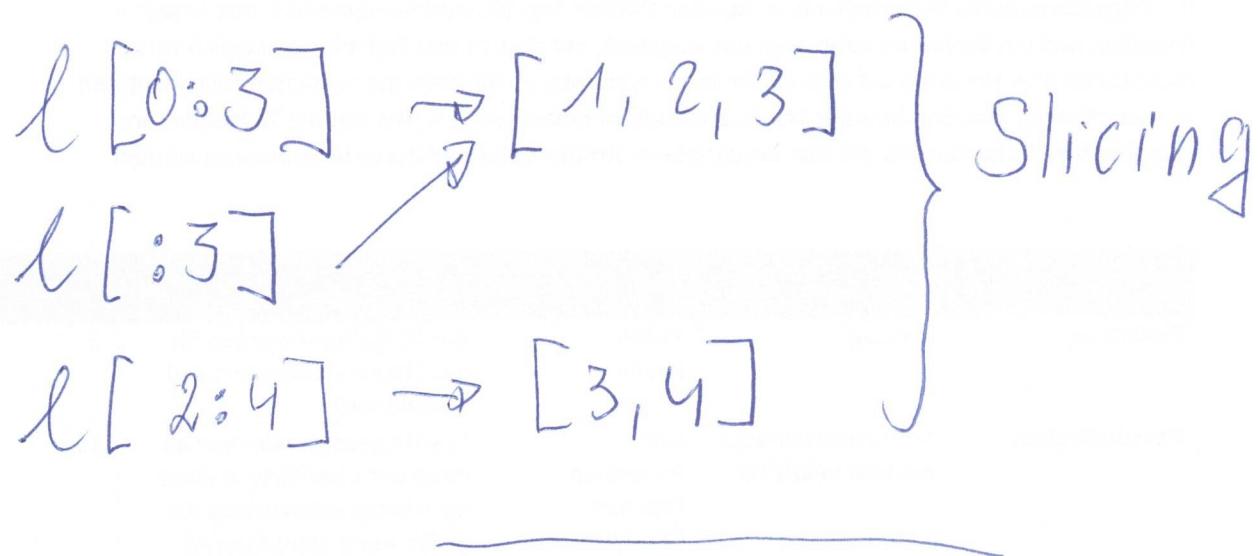
$\ell[0]$ = "Heinz"

Liste

$$l = [1, 2, 3, 4, 5]$$

$$\text{len}(l) \rightarrow 5 = n$$

$$i \text{ lauft } 0 - 4 = (n-1)$$



$$l_1 = [1, 2]$$

$$l_2 = [5, 6]$$

$$l_3 = l_1 + l_2 \rightarrow [1, 2, 5, 6]$$

Liste

$$l = [1, 2, 3, 4, 5]$$

$$a, b = l \rightarrow a \rightarrow 1 \\ b \rightarrow 2$$

$$a, b, *c = l \rightarrow a \rightarrow 1 \\ b \rightarrow 2$$

$$c \rightarrow [3, 4, 5]$$

$$x = 100 \rightarrow \\ y = 0 \rightarrow$$

$$y, x = x, y$$

$$x = 0$$

$$y = 100$$

Interessant: list comprehension
Generatoren
Iteratoren

Name Naps

$l_8 \rightarrow [[], [], [],] \rightarrow$

$l_9 = l_8$

l_9

$\boxed{[], [], [], X}$

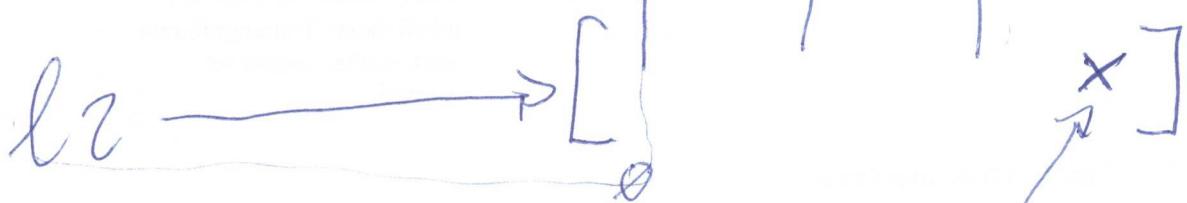
$l_9, \text{append}(X)$

print(l_8)

Flache Kopie



$l2 = l1.copy()$



$l2.append(x)$

$l2[0] = "ALOHA"$

Siehe: Lib \rightarrow shallow / deep copy !

Listemops

$$l1 = [1, 2]$$

$$l2 = [5, 6]$$

$$l3 = l1 + l2 \rightarrow \begin{matrix} \text{nein:} \\ \text{flach} \end{matrix} \xrightarrow{\text{copy}} [1, 2, 5, 6]$$

$$l1.append(l2) \rightarrow \text{in } l1: [1, 2, [5, 6]]$$

Tupel

$l = [1, 2, 3]$ ← veränderbar

$t = (1, 2, 3)$ ← nicht veränderbar

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

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

Lotto bude - I

Spieler gibt 6 Zahlen ein

Prüfen: Sind es 6

Sind es Zahlen

Sind sie zwischen 1 - 49

Nennt Doppelten?

Alles ok → merken

Wann nicht → Meldung o.
Programmende

Techniken: input try-except
 for
 if
 liste
 Umwandlung

Lotto bude - II

Ziehung (Zufallszahlen!)

6 aus 49

"Dice, Lottery"

Vergleich Spieler eugabe und
Ziehung
Ausgabe der Treffer

Dictionary

map key → value
 Strong! beliebig

$d = \{ \}$ leer

$d = \text{dict}()$

$d["Name"] = "Willi"$

→ $\{ "Name": "Willi" \}$

$d["Alter"] = 42$

→ $\{ "Name": "Willi",$
 $"Alter": 42 \}$

Dict

`print(d["Name"])` → "Willi"

`print(d["Ort"])` →  liegt
nicht
vor
eleganterer Zugriff

`print(d.get("Ort", None))` → None

`d.keys()` → ["Name", "Alter"]

`d.values()` → ["Willi", 42]

`d.items()` → ~~["Willi",~~

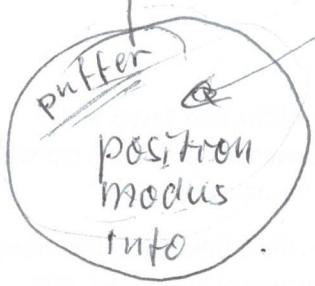
[("Name", "Willi"), ("Alter", 42)]

Datei lesen

=

fd = open(fileName)

①



Filedescriptor

& BS →



fileName

bereit
Zugriff vor

↑
koppe

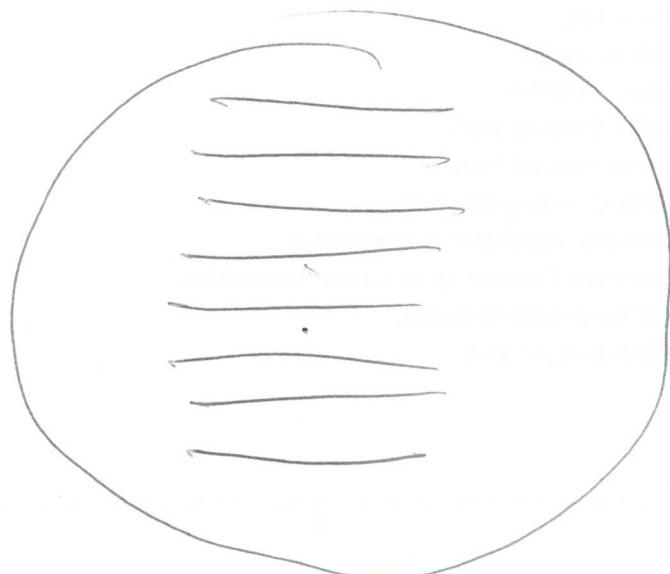
fd.read() das liest erst

text = fd.read()

②

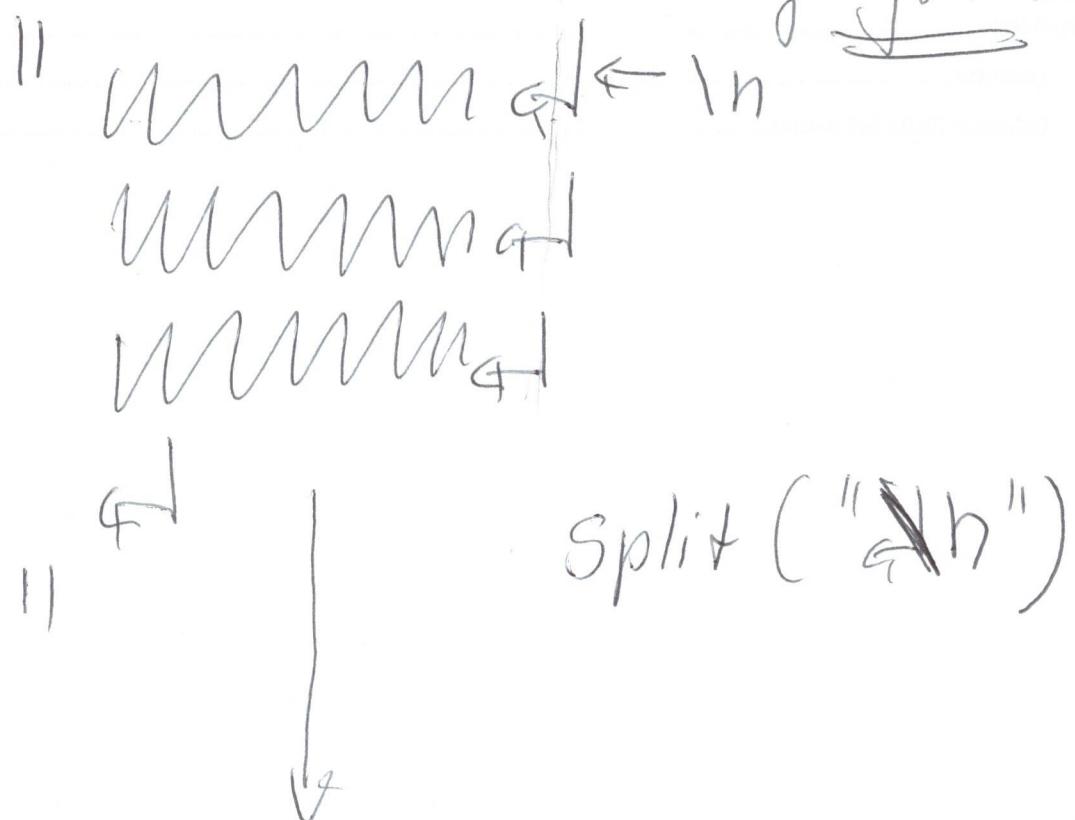
—n— und

Speichert Dateiinhalt
intern ab



fd.close()

Überlegungen üb-2



$\ell = [$

"WWWW",

"WW",)

"WW"

]

readlines ist oh → aber $\backslash n$ ist noch drin

besser: read → split("\n")
read → splitlines()

Überlagerung

"Konspalting"; X; Y; Z"

↓ split();

1 2 ... →

z["Konspalting", "X", "Y", "Z"]

z[0]

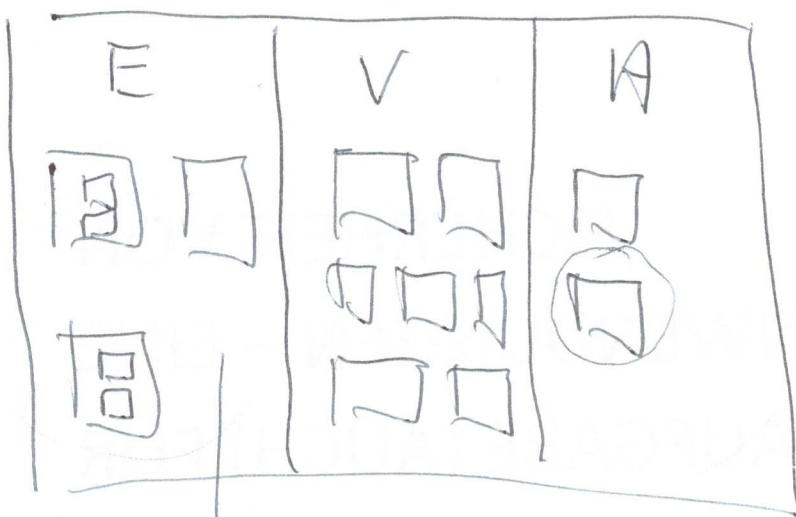
↓ thesaurus[z[0]] = z[1:]

{ ("Konspalting"); ["X", "Y", "Z"] }

↓
thesaurus["Konspalting"] → Ausgabe

Funktion

- Organisation
- Kein copy&paste!
- Teile & Herrschen



≡ Programm

Lib



Funktionen
Klassen

Funktionen

Parameterliste

def mach_was(\downarrow):

====

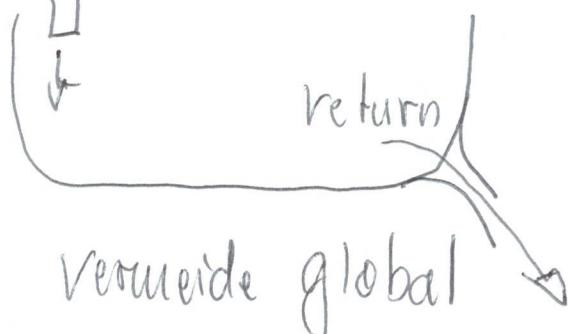
return Wert

def fu_was()

====

"Badewannen-Prinzip"

Parameter



Funktion: klein, dünn, effektiv

Schwerpunkt → Fehler raise Exception

↓
try
~~except~~

Funktion

def f() keine P. erlaubt

def f(a) ein Pflichtparam.

def f(a, b) zwei — —

def f(a, b=None)
 |
 |
 Pflicht Optional

a - Pflicht

b - optional, vorbelegt

f(4711) a - 4711
 b - None

f(4711, 123) a - 4711
 b - 123

f(a=4711, b=123)

f(b=123, a=4711)
named parameters

Funktron

```
def ff( a, *args )
```

Pflicht legal: 0, -2 - 20 - 200 ...
da eine Liste

f(4711)

F(4711, 1)

`f(4711) { 1, 2, 3, 4, 5, 6, 7 }
 ^
 |
 a args`

```
def f(a, **kwargs)
```

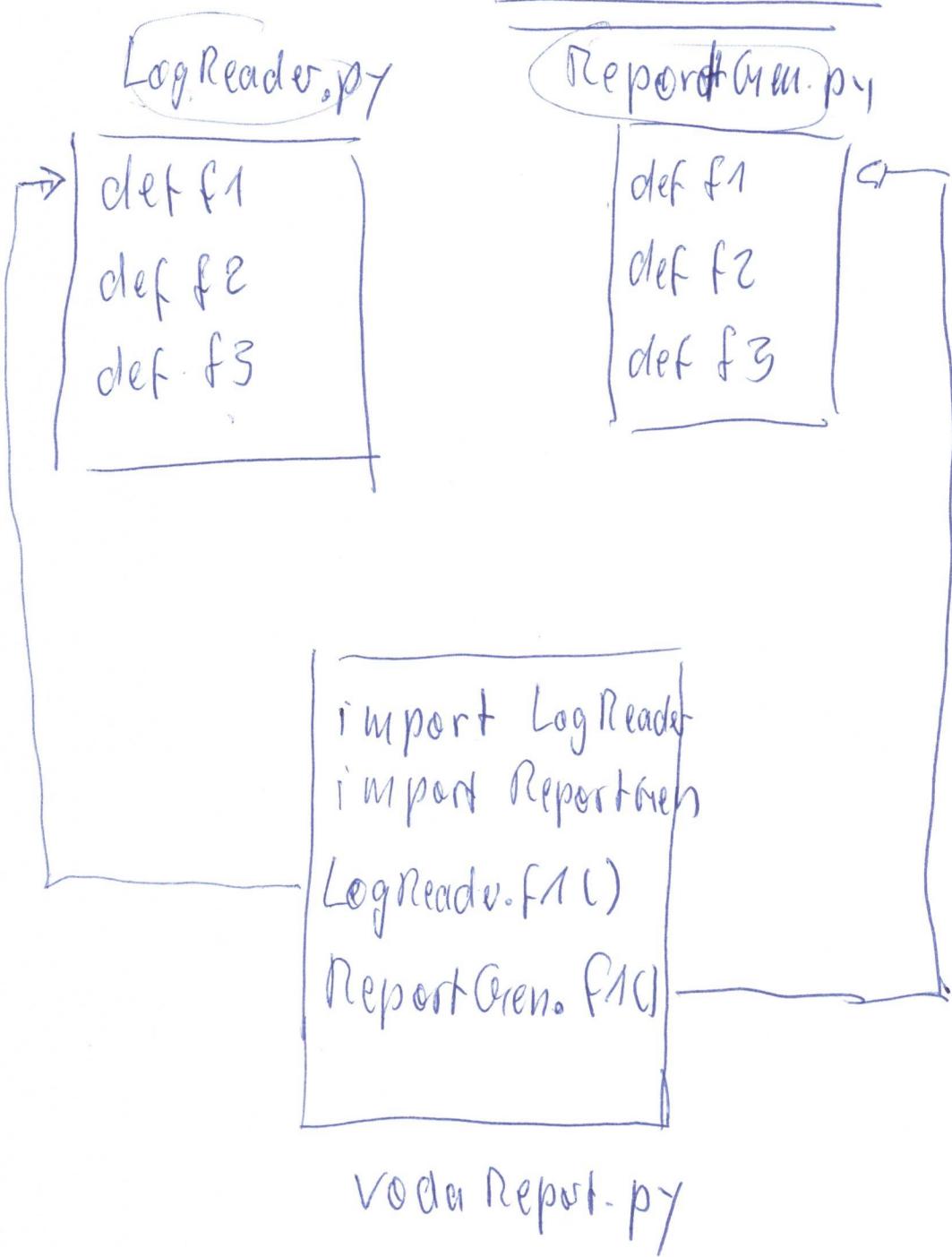
f(4711)

f(4711, name)

f(1711, name="Willi",
alter=50,
ort="Naturgarten")

```
def f(a, b=None, *args, **kwargs)
```

Funktionen / modules

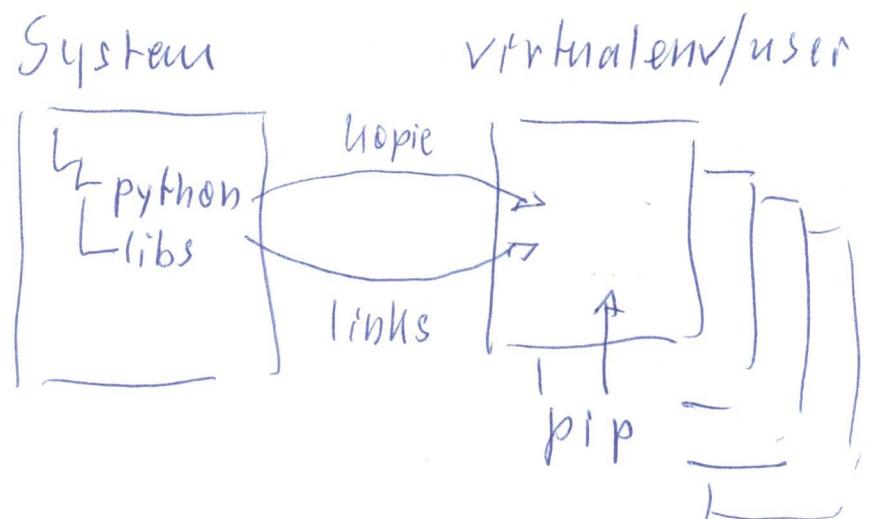


Module

Spercherort → Distr.-Abhängig *

→ User-Abhängig *

→ virtuellenv-Abhängig



`PYTHON_PATH` ← Ergänzung import

Sys. path
↓
Import - Pfact

Liste: append
insert

Modul

• PY

egg

~~Paket mit modulen + Manifest~~

Wheel

Import-Varianten

Laden alles

=

Import x → x in "__main__"
x.ahtron()

Import x as y → (x→y) →
y.ahtron()
x.ahtron()

Gesiebtes Laden

from x import ahtron → ahtron in "__main__"
ahtron()

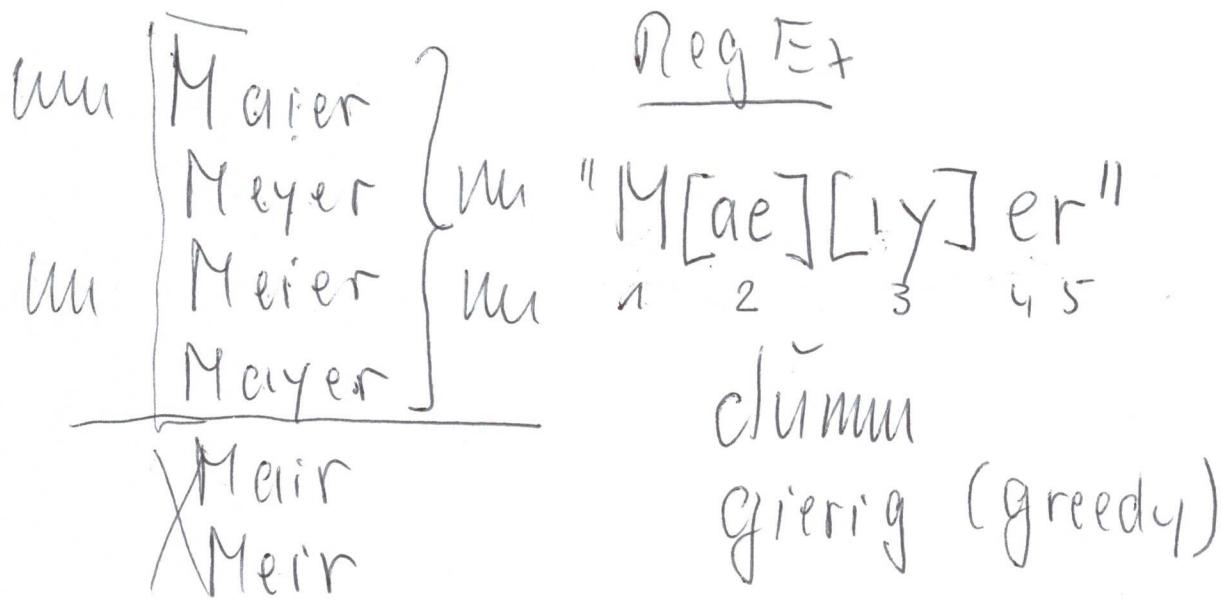
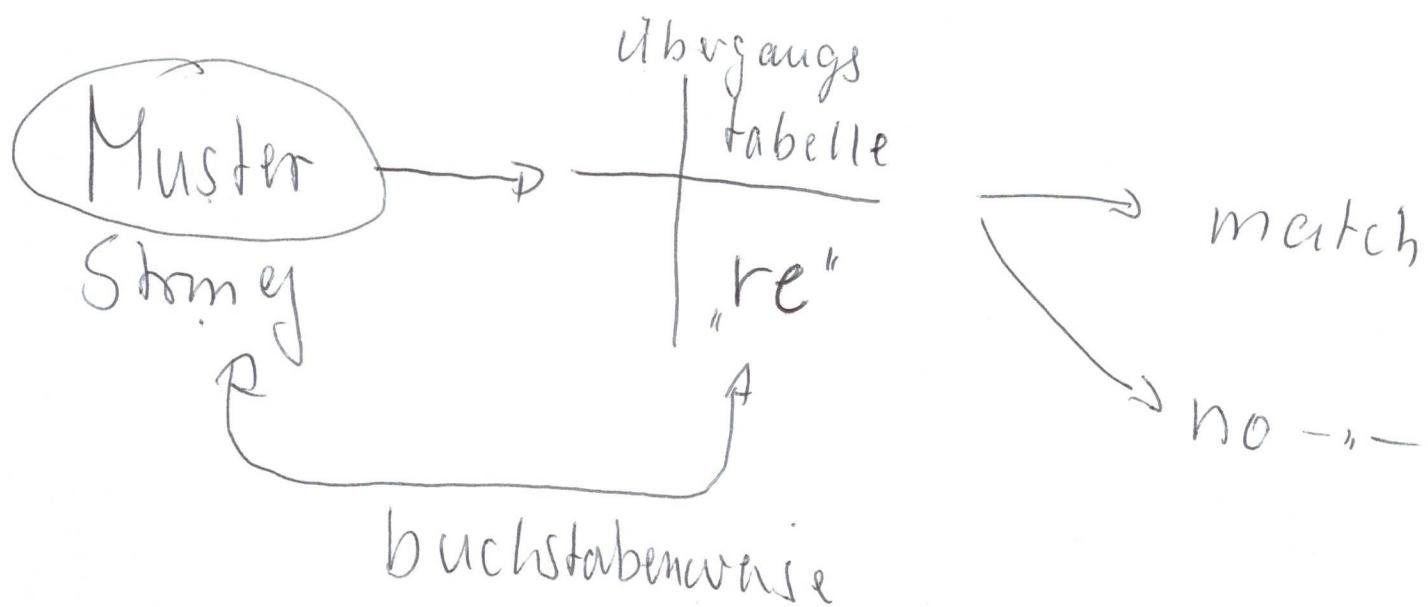
from x import ahtron as x-ahtron → x-ahtron in __main__
x-ahtron()
ahtron()

Vervierufen: from x import *

in dem Fall alles aus x → __main__
besser import x

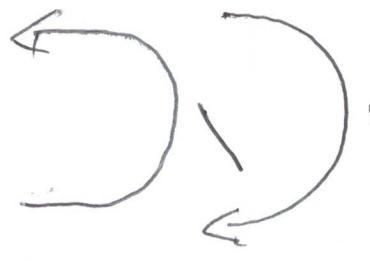
Reguläre Ausdrücke

z.B. grep, perl, python, ...



Regulär Ausdrücke

Zerchen ohne Sonderbedeutung



Zerchen mit Sonderbedeutung

Bsp "b" Buchstabe

"_a" bel. Zerchen

"\b" Wortgrenze

"\." Punkt

"\n"

Reguläre Ausdrücke

bel. Zerchen

Bsp

1

0-∞ oft

Quantifier

$$5 * \begin{array}{c} \diagup \\ 5 \\ \diagdown \end{array} \begin{array}{c} \diagup \\ 5 \\ \diagdown \end{array} 55$$

۱۷

Order 1

5? < $\frac{11}{5}$... 5

r +

1 -∞ oft

$$\begin{array}{r} 5 \\ + 5 \\ \hline 55 \end{array}$$

4
r }

r^a { von, bis } von - bis mal .

$$5 \overline{) \{ 1, 3 \} - 55} \begin{matrix} 5 \\ \diagdown \\ 555 \end{matrix}$$

Λ
δ

Bufang

Eucle

Leere Zeile: \$

[Auswahl]

[a-z]

$\begin{bmatrix} 0 & -9 \end{bmatrix}$

[a-zA-Z]

()

Coupprering

卷之二

Quantifier Fails

$\forall d \ast$

4711
471
4711

$\exists d$

W | | | | |

matches!!

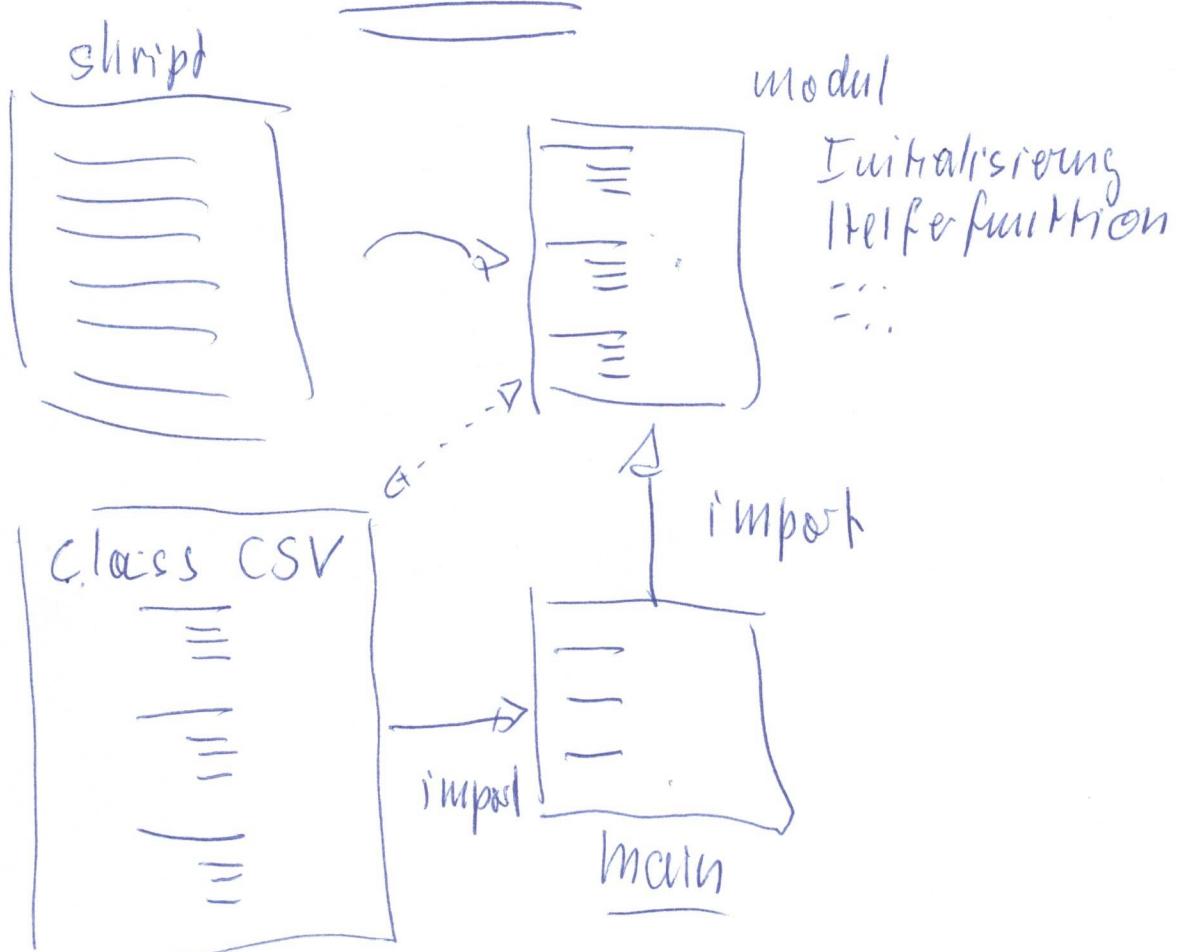
$\forall d L^+$

4711

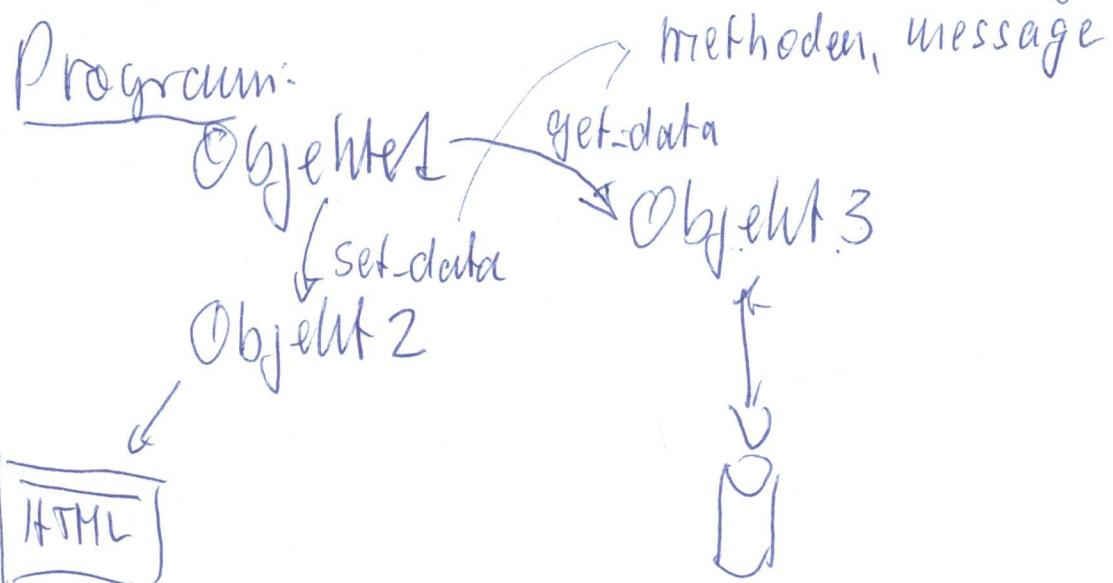
471
4711

No match

Klassen



- Objektbildung
- Daten + Code zusammen, Kapselung
- Implementation interessiert nicht, Hiding



my-class.py

class Reader: "Ich"-Bewusstsein
↓ des Objekts

def __init__(self):

≡

def set-fname(self, name):

≡

def read(self):

≡

def get-data(self):

≡

return

def -machwas(self):

≡

—||
Konvention

— ≡ private

main.py

import my-class

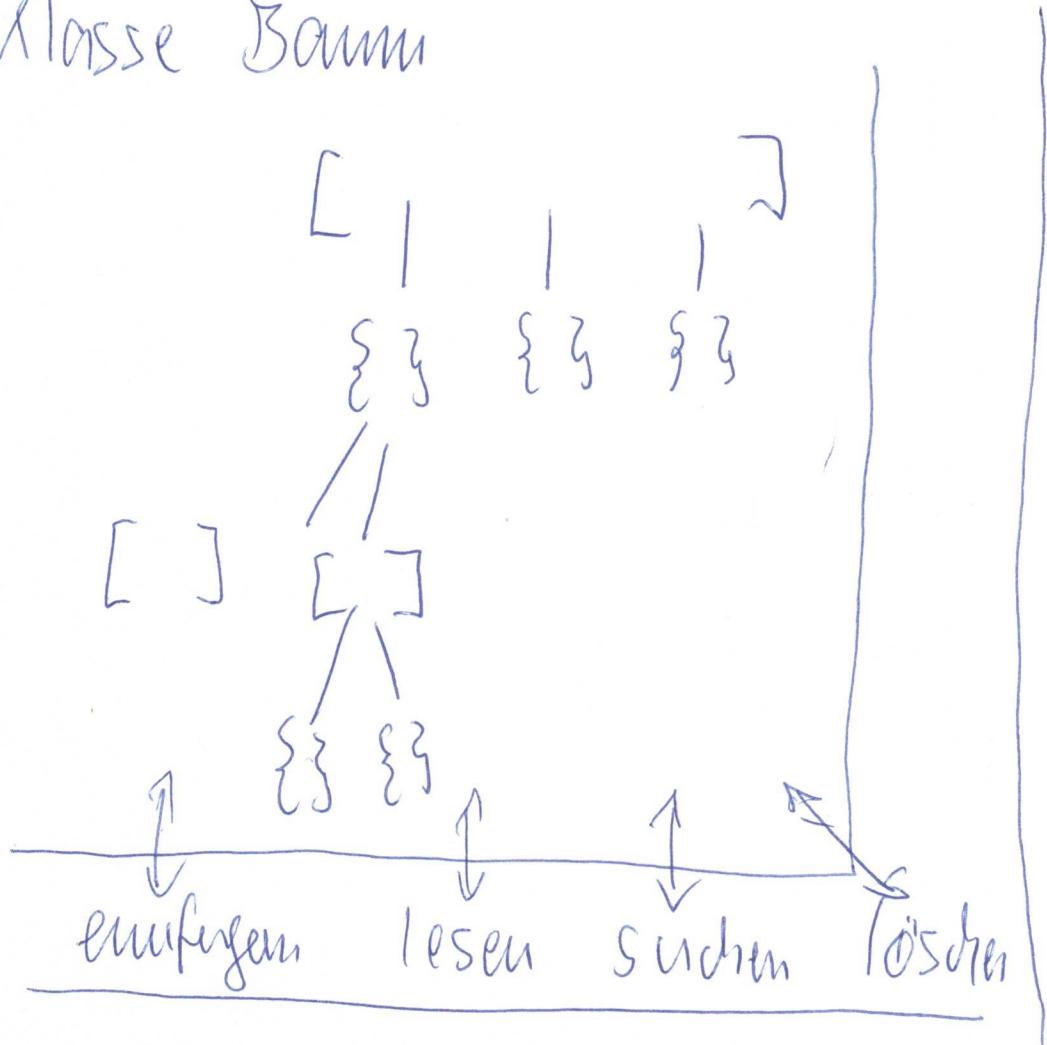
o = my-class.Reader()

o.set-fname("daten.txt")

o.read()

print(o.get-data())

Klasse Baum



$b = \text{Baum}()$

$b, \text{lesen}(\text{Quelldatei})$