

Python

Grundlagen

Ulrich Cüber

Start 10:00

Am Besten kein VPN
nutzen!

Geschichte

1995 Guido van Rossum
(monty)python → Editor: idle

Verschieden: 2.7.X eigentlich seit
2010 abgelöst
„inecht“ ab 2019



3.X / 3.6 / 3.9
ab.

→ „pip“-env (Standard)*

„conda“-env (Anaconda)
Jupyter-Workbooks



Skript \Leftrightarrow Programm

```
print("Hello")
```

```
if Bed:
```

```
    print("1.")  
    print("2.")  
print("World")
```

Konvention: 4x |
2x | ok
PEP8
s.a. pylint

0

: \rightarrow Einrückung folgt
Blockbeginn

```
if ... :
```

```
for n in ...:
```

```
while ...:
```

```
def f():
```

```
class ...:
```

Programmstart

python3 skript.py

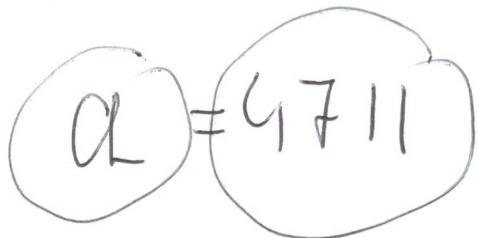
Linux / Windows

Linux: Hashbang in 1. Zeile
Datei chmod +x

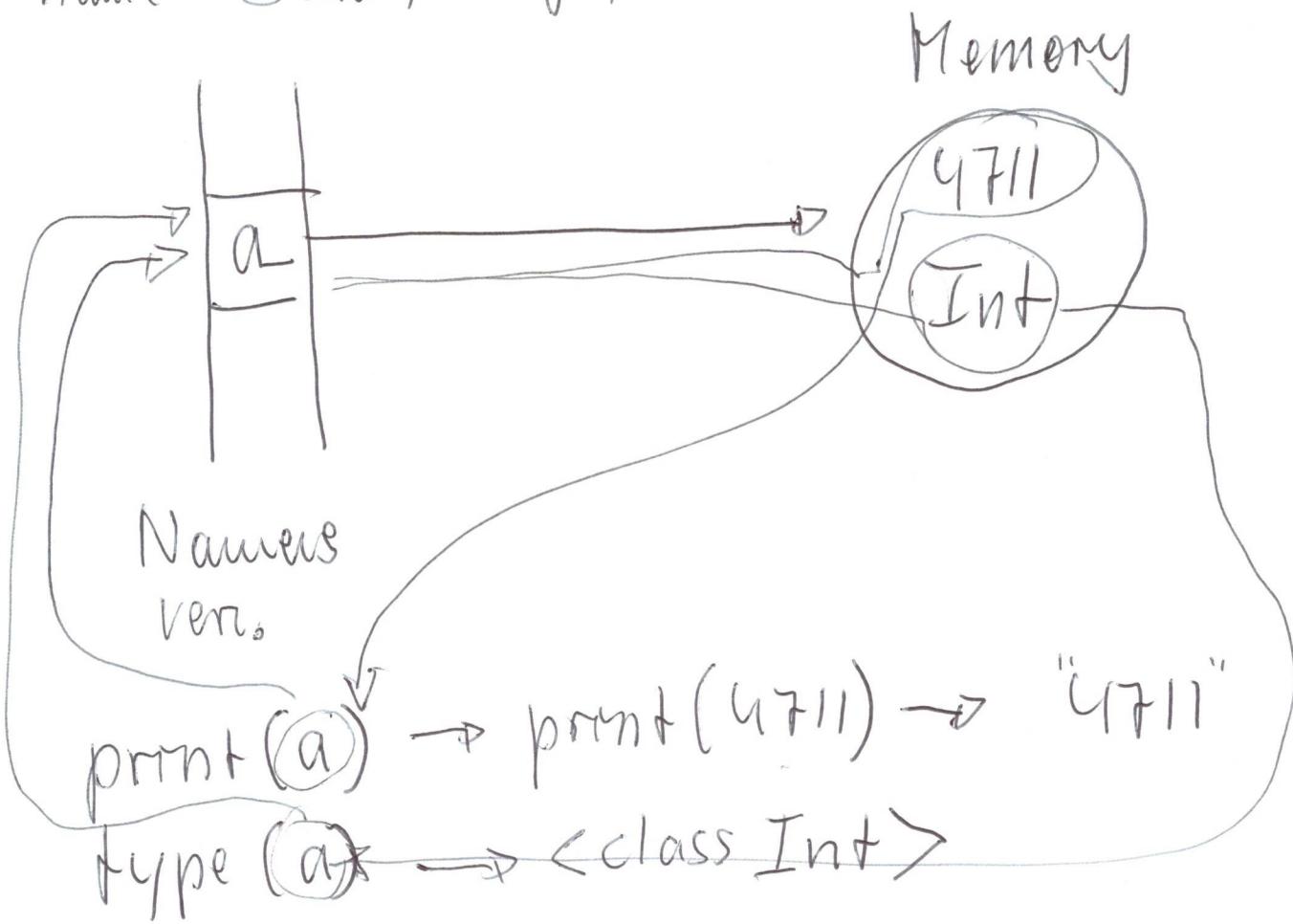
Aufruf: ./skript.py

Variablen & Datentypen

name der auf Daten zeigt



name Daten, Integer, 4711



Zahlen

Int - Ganzzahl - beliebig gross

Float - Fließkommazahl - grosser Wertebereich

Ungenau

→ Rundung

→ Max. Stellen 15



Decimal() wenn
genau sein soll

* / + - % **

== < <= >= >

Typstreuung! 'a'*5 → 'aaaaa' ✓

'a' + 5 → ↗ Exception

'a' + '5' → 'a5' ✓

Umwandlung:
int(variable) → zu int
float(variable) → zu float
str(variable) → zu str

Typeh

None \rightarrow "nix", nicht definiert

"String" } Text
"string" } Binärdaten \rightarrow jpg, mp4, mp3
b"xxxxxx..."

True / False

Liste []
Tupel () \rightarrow unveränderbare Liste
Set { . } $\rightarrow \{ 1, 2, 3 \}$ "Menge"
Dict. { } $\rightarrow \{ K:V, K:V, K:V \}$
Key } Paar
values }

Strings

Ausgabe \rightarrow String форматierung von printf()

printf("a", "b", "c") \rightarrow abc

printf("a", "b", "c", sep=":") \rightarrow a:b:c

printf("abc")

x = "a" y = 4711 z = 12.34

printf("x:", x, "y:", y, "z:", z) * benötigt

printf(f"{{x}} {{y}} {{z}}") * benötigt

printf("{{x}} {{y}} {{z}}".format(x, y, z))

ganz alt: 2.7 Stil

printf("%s %d %f", x, y, z)

Ein / Ausgabe

Ausgabe: print()
 str → > min

Eingabe: in = input("Prompt")
 str! → > Prompt

```
#!/usr/bin/env python3
eingabe = input("Ihr Name: ")
print(f"Hello {eingabe}")
```

Addierer

Ein Programm

Benutzer gibt 2 Zahlen ein (a, b)

Die Zahlen werden addiert $Z = a + b$

Das Ergebnis wird ausgegeben

Eingabe input \rightarrow int
 input \rightarrow int

Verarbeitung $Z = a + b$

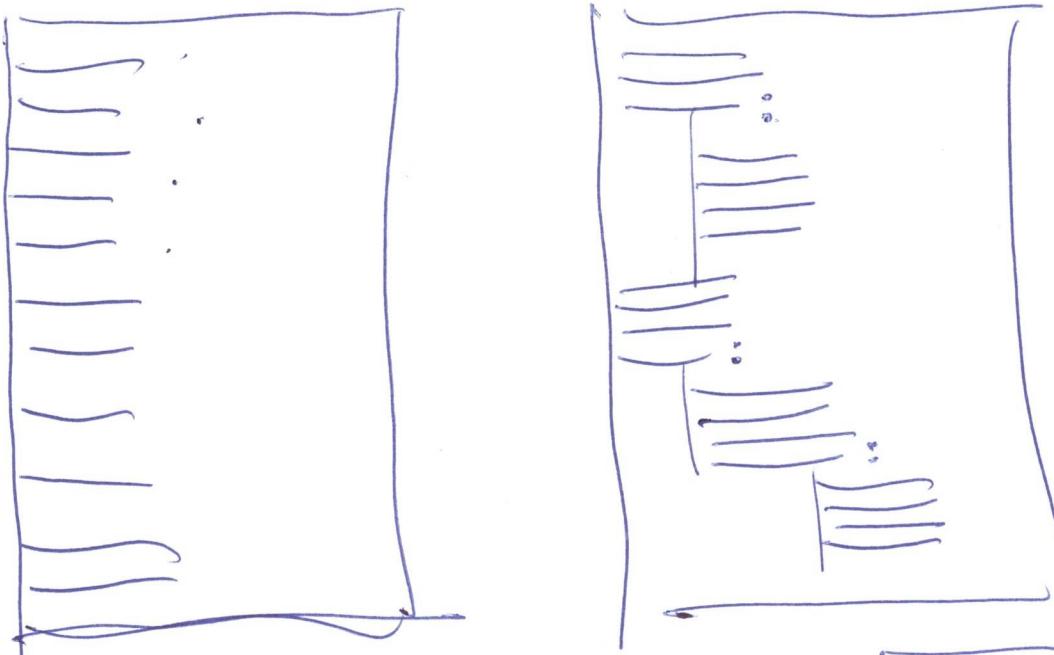
Ausgabe print

`int("2") \rightarrow 2`

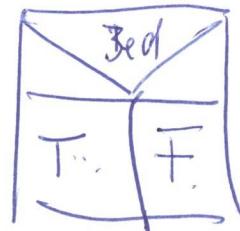
`float("2") \rightarrow 2.0`

`"1" + "2" \rightarrow "12"`

Programmfluss



if - Verzweigung



if Bed :

 | „True“-Zweig

{ else:

 | „False“-Zweig

- optional

if B1 :

 if B2 :

 =

 else:

 xxx

 yyy

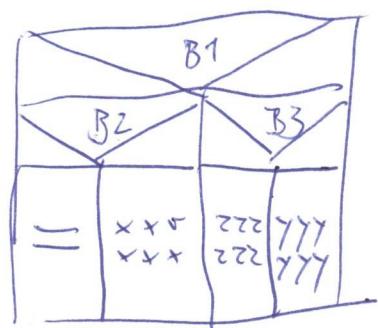
 else:

 if B3 :

 zzz

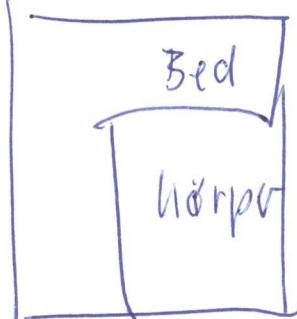
 else:

 yyy



→ „elif“ als Abhängigkeit

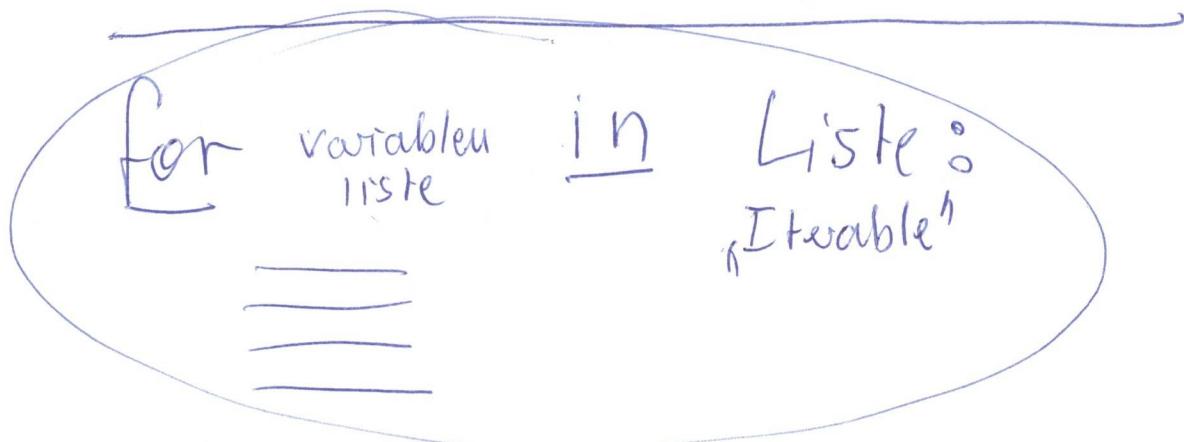
Programmfluss



while Bed : "offene"
 ≡ } Körper

Vorzeitig abbrechen "break"

Nächster Durchlauf
vorzeitig aufrufen "continue"



Schleife

for n in range(0, 10): → 0, 1, ..., 9
n-1
print(n)

for n in range(0, 10, -1): 9, 8, 7, ..., 0
print(n)

l = ["Willi", "Heinz", "Karl"]

for n in l:
print(n)

for i, n in enumerate(l)
print(i, n) 0, Willi
 1, Heinz
 2, Karl

f.b.c

for n in open("Dateiname"):
print(n)

Benutzer startet Prog.

→ Programm wartet auf Eingabe

Wenn Eingabe == "EXIT"

dann Programmende

Sonst Ausgabe "Eingabe war: ", ---

if ---:
 break

if ...:
 exit()

if eingabe.lower() == "exit"

a) Verlassen der
 Schleife: break
b) Verlassen Prog
 exit()

Fehler

Syntaxis auch zw Laufzeit!

Logische Laufzeit

→ Vermeiden → vor allen Schritten
testen ob alles ok ist

→ Mit umgehen

Versuch & Irrtum

= try - catch

try - except

"if Kashaden
mit Tests"

assert - Befehl

"pro-aktiv"!

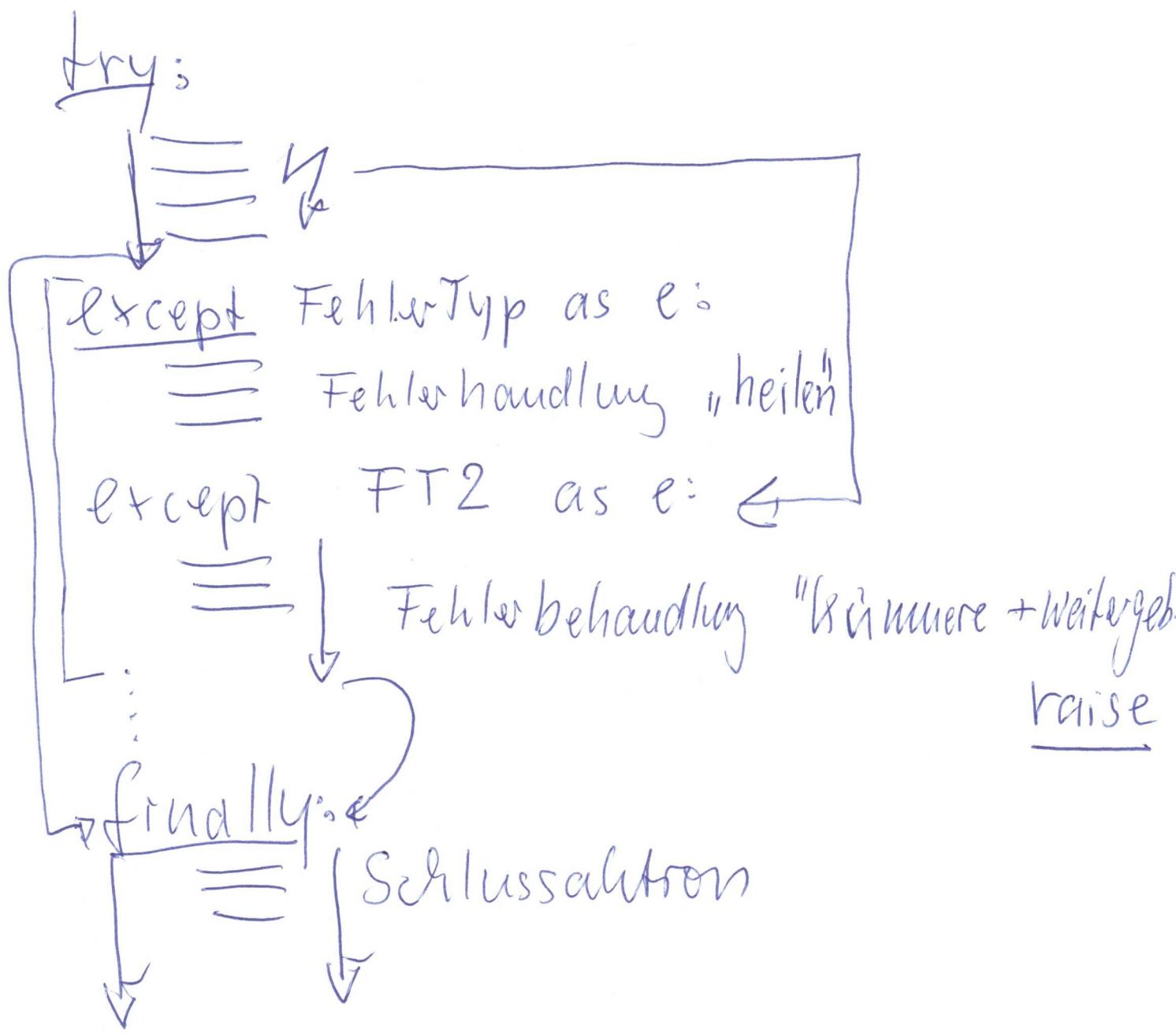
⇒ Aufwändig

⇒ Teuer

⇒ Fehleranfällig

⇒ nur begrenzt
Anwendbar

Try-Catch



try - except

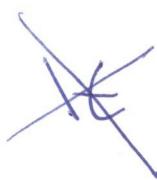
try - finally

try - except - finally

Zusammenfassung

→ indentation

PEP8 → 4x



if ... :

|

while ... :

for ... :

if ... :

|

else:

|

def f():

|

Text: " ... " 

AAA
|||
VVV

Str

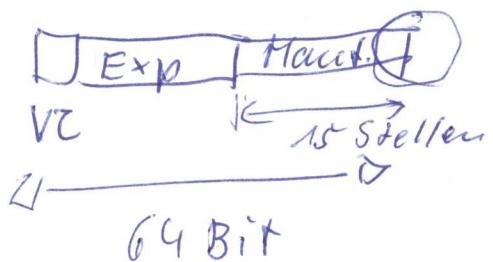
- Int Grundsätzlich

Float Fließkommazahl
 $a \cdot b / a \neq b$

Bool True/False

None

IEEE-Format



Liste / Tupel

Set

Dictionary

class X:

H

→

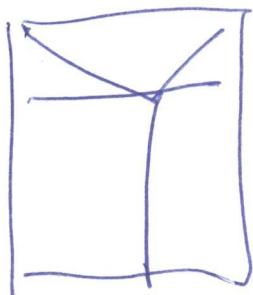
a = Wert

a →

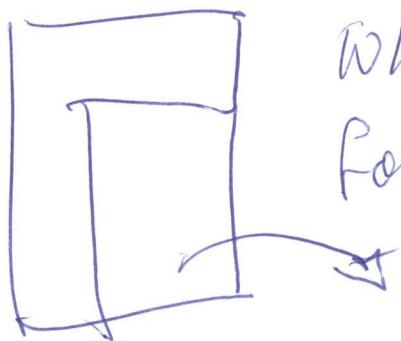
Wert
Typ

b = a
b ↗ Referent

del varname



if / else



while

for

break

Fehlerbehandlung

try

 Versuch

try

except

 Fehlerbehandlung

try

finally

finally

 I auwe!

z.B. Aufräumen

input / print

Strings

|| || ¶ ¶ Einzeilig

||||| ¶
 = =
 ||||| ¶
 = =
 ||||| ¶

Mehrzeilig

$s = "Hello, \n world"$

len → 12
 0 1 M n-1
 -2 -1

print(len(s)) → 12

print(s[0]) → "H"

print(s[-1]) → "d"

Sonderzeichen \zeichen → \t Tab
 \n Zeilenwechsel

Methoden / Funktoren: siehe Dok

in - Funktion

Slicing

$s = "Hello, world"$

$s[0]$

$s[-1]$

$s[0:5] \rightarrow "Hello"$

$s[:5]$

$s[3:9] \rightarrow "lo, wo"$

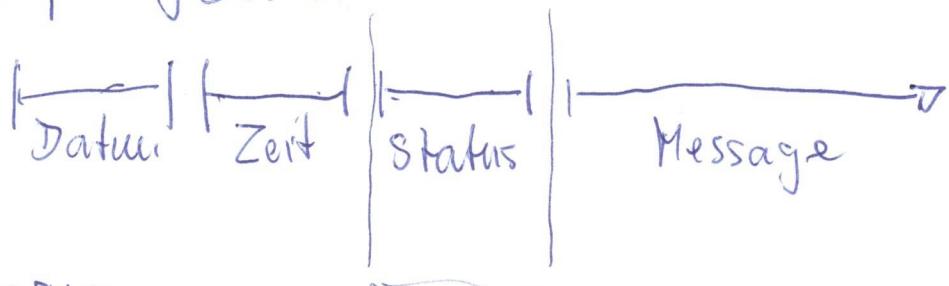
$s[7:-1] \rightarrow "worl"$

$s[7:] \rightarrow "world"$

$s[von:bis]$

erster
der
nicht
dabei
sein
soll

Bsp Log Datei



$s[:6:2]$

vom Anfang bis Ende(incl.!)
jeden 2. Buchstaben

$s[:5:2]$ $\rightarrow "Ho"$

vom Anfang bis Pos 5(exklusiv!)
jeden 2. Buchstaben

Listen

Zusammenfassung von Objekten

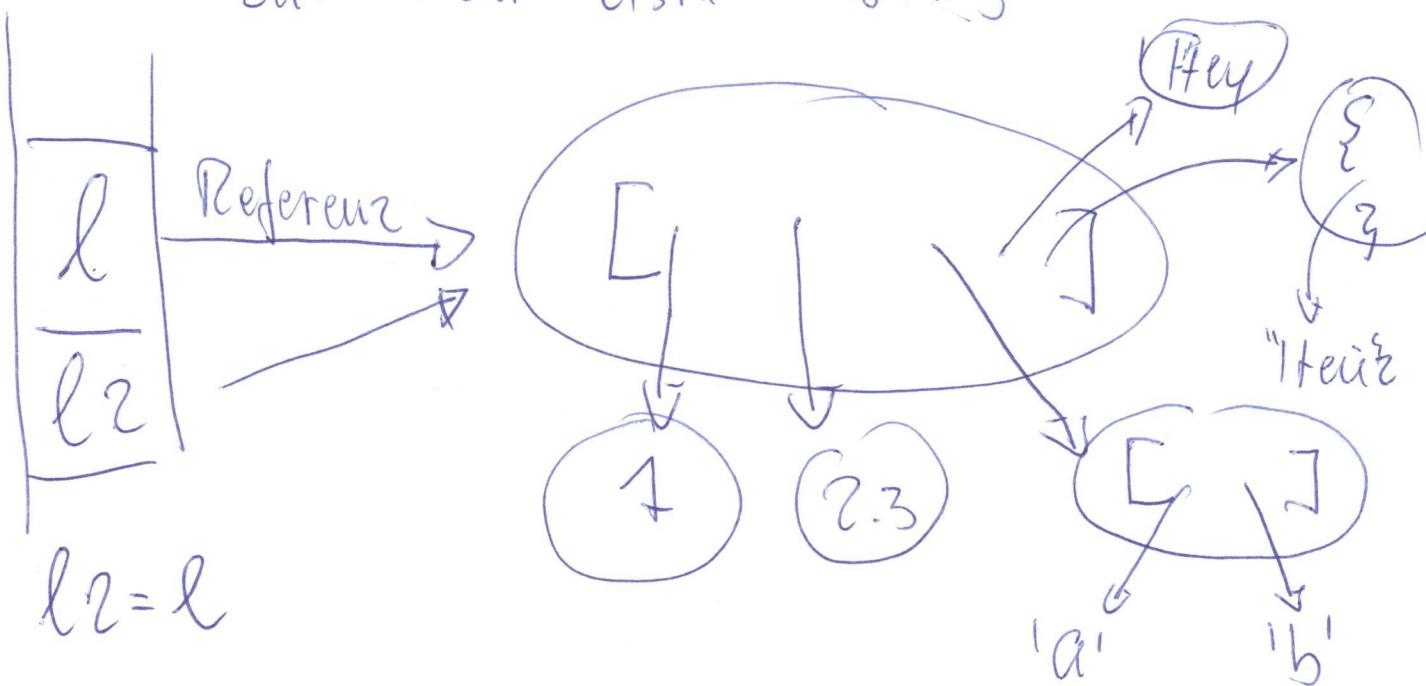
$l = []$ leere Liste (Boolean-Wrt: False)

$l = \text{list}(\dots)$ legt eine Liste an / Umwandlungsfunktion

$l = \text{list}(a, b, c) \rightarrow$ erzeugt Liste mit a, b, c

$l = [1, 2.3, ['a', 'b'], 'Hey', \{\text{"Name": "Heinz"}\}]$

↑ ↑ ↑ ↑ ↑
Int Float Liste String Dict

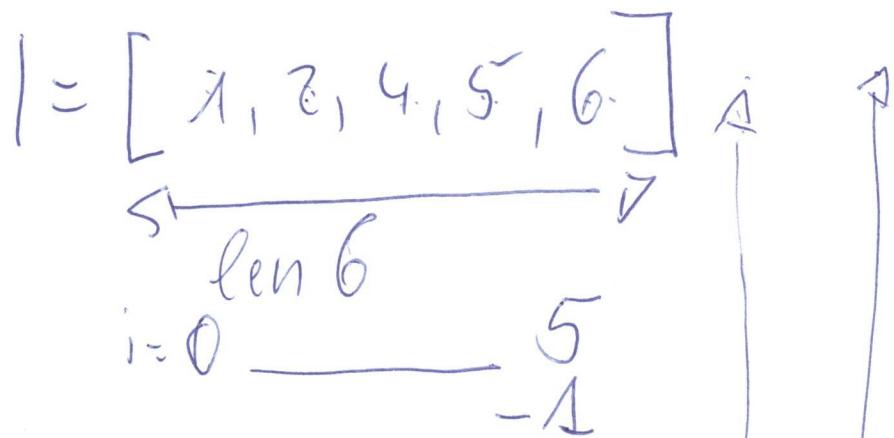


Liste

Länge $\text{len}(L)$
 $L.\text{count}()$

Index $L[i]$
 $\text{print}(L[i])$
 $L[i] = 4711$

Voraussetzung: es gibt den Platz



$l[6] = 7$
 $\text{print}(l[5])$

Liste

append	}	+	Verändert Liste
extend		+	
pop		-	
insert		+	
remove		-	

$l = []$ List-Init

for w in Daten:

 → „Verarbeitung“

$l.append(w)$



$$l = [] + e_1$$

$$l = [e_1] + e_2$$

$$l = [e_1, e_2] + e_3$$

⋮

$\text{len}(l) \neq \emptyset$

Liste

Slicing

$l[0], l[-1]$

$l[3:6]$

$l[:2]$

...

$l = \{ "a", "b", "c", "d", \} [47/1]$

$\underbrace{\qquad\qquad\qquad}_{l[: -1]}$ $\underbrace{\qquad\qquad\qquad}_{l[-1]}$

$a, b = l[: -1], l[-1]$

$a, *b = l$

$\uparrow \quad \nwarrow$
 $l[0] \quad l[1:]$

Liste vs Tupel

$l = [1, 2, 3]$ veränderbar

$t = (\underline{1}, 2, 3)$ unveränderbar

d.h. $t[0] \neq l[0]$ ✓
 $t[0] = 4711 \cancel{\checkmark}$
 $l[0] = 4711 \checkmark$

$l = \text{list}(t)$ Liste angelegt
Inhalt von t kopiert

$t = \text{tuple}(l)$ Tupel angelegt
Inhalt von l kopiert

Tupel gerne als Rückgabe von Funktionen
als „Konstanten“
wenn Daten nicht versehentlich
geändert werden soll

Aufgabe

Benutzereingabe:

input

6 Zahlen

(int)

Wertebereich

1 - 49

Keine Doppelten

~~KAT~~

raise Exception

eingabe = input ("Bitte 6 aus 49: ")

bsp 1 2 3 3 5 4 7 13

:
:
?

sortieren bitte

try-except
Im Fehlerfall

Ausgabe wenn ordentliche Zahlen

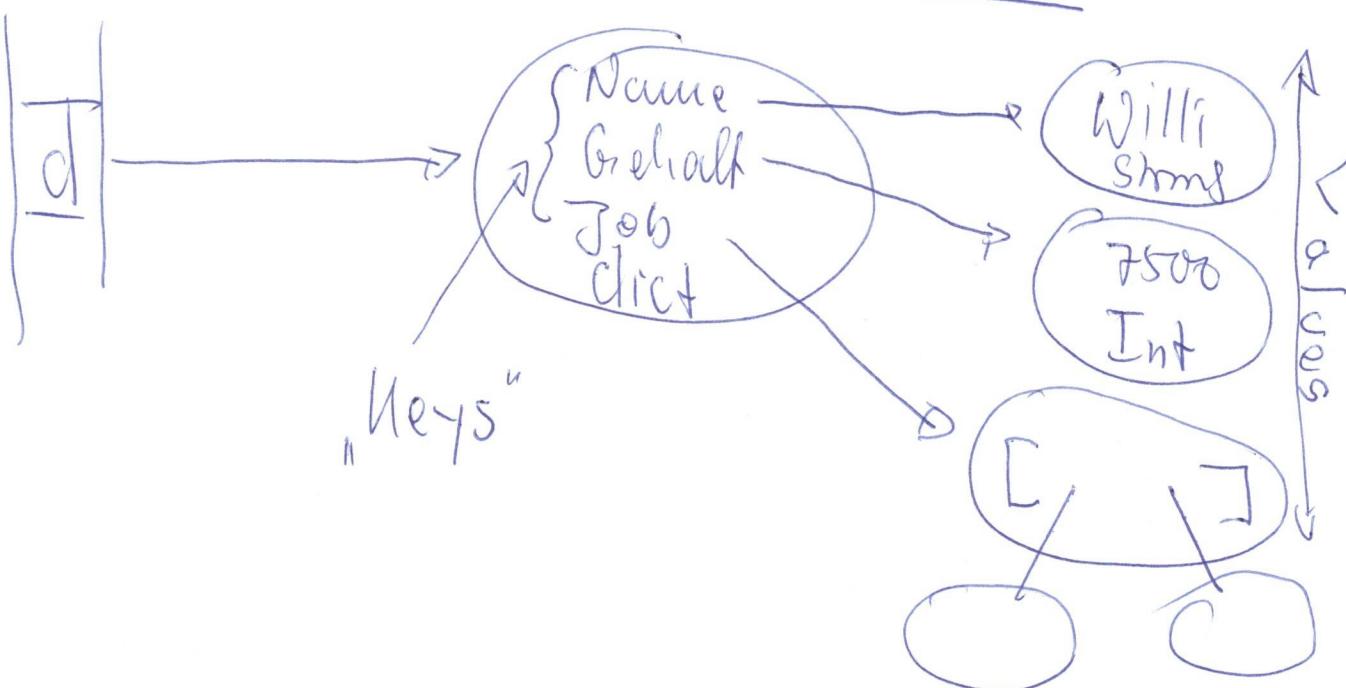
Dictionary

"key-Value-Store"

$$d = \{ \}$$

$$d = \text{dict}(\star) \leftarrow v = [(k, v), (k, v), (k, v)]$$

$$\overbrace{d = \{ \begin{array}{l} \overset{k}{\text{"Name": "Willi",}} \\ \overset{v}{\text{"Gehalt": 7500,}} \\ \text{"Job": ["CTO", "Programmierer"]} \end{array} \}}$$



Dict

$d = \{ \}$

Wert hinzufügen / überschreiben

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

$\text{type(key)} = \text{str}!$

Wert abfragen

Value $\text{print}(d["Name"]) \rightarrow$ Value

existiert ✓

nicht vorhanden ✗

Projekt: if "Name" in d:

$\text{print}(d["Name"])$

Oder : $\text{print}(d.\text{get("Name", "nicht vorhanden"))}$

Dict

```
d = { "Name": "Willi",  
      "Gehalt": 7500  
 }
```

d.keys() → ["Name", "Gehalt"]

d.values() → ["Willi", 7500]

d.items() → [(Name, "Willi"),
 ("Gehalt", 7500)]

Z.B.: for k, v in d.items():
 print(f"Key {k} → Value {v}")

Dict

Bsp db-abfrage

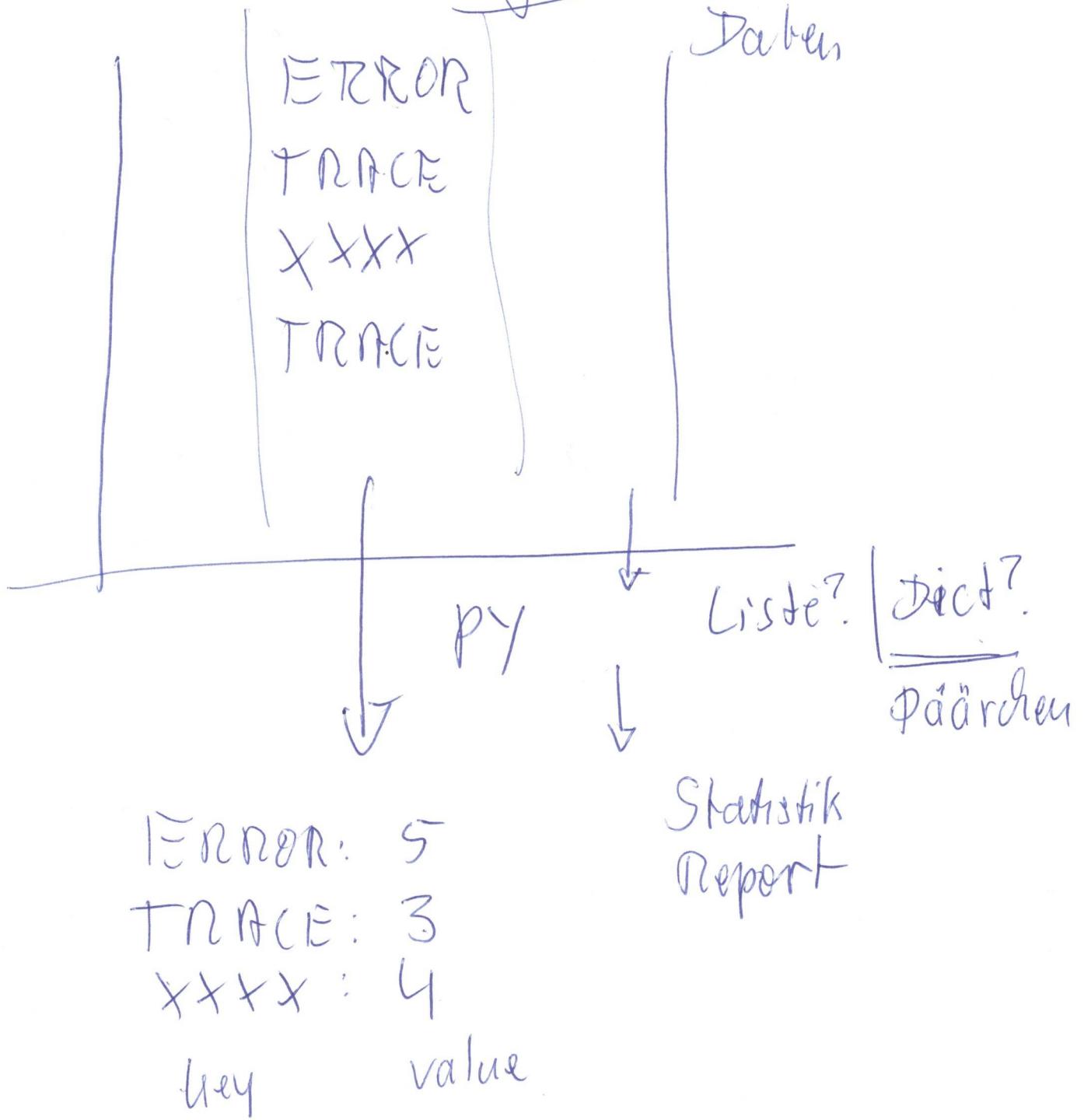
$e = [$ { "Id": "4711", "Name": "Willi"},
→ { "Id": "0815", "Name": "Heinz"},
{ "Id": "1234", "Name": "Karl"}
 $]$

liste von dicts, in den dicts: "row"

$e[1]["Name"] \rightarrow "Heinz"$

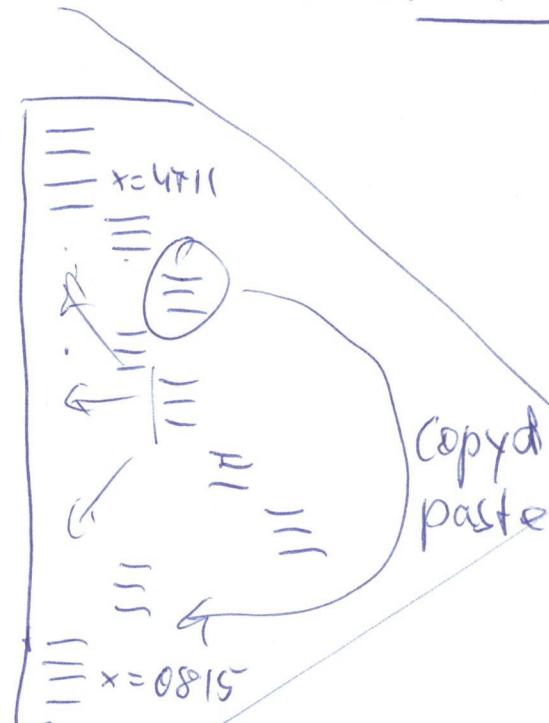
distanz	dict.
index	key
inner Int	string

Aufgabe



Faulttouren

1
15700



unlesbar
untestbar
unwarrbar
Einfel Lösung

Teile & Hersteller
Kopiere nicht
Fasse dir kurz
Richte auf Wiederverwendbarkeit
Mache es testbar



↓
Nachdenken!
Datenstruktur finden
Faulttouren einsetzen

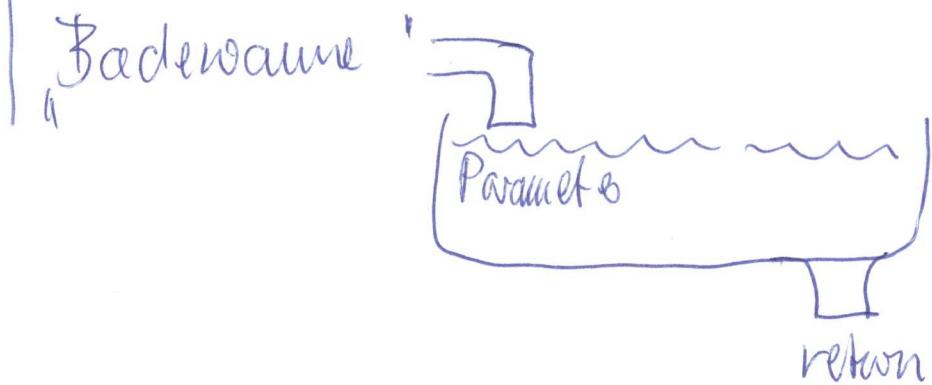
ergane?

→ fertige? \leftrightarrow gibt es
Module/Bibs?

Funktion

Erst definieren - dann nutzen

def verb_subjekt(Parameter):
 | Daten
 | Code . ↓
 | return xxx } ← optional
 | Daten



def slurp(fname):
 with open(fname) as fd:
 → daten = fd.read()
 return daten

lokale
Variable

daten
Werte = slurp("thesaurus.txt")
str

Parameter

Keine	$f()$	
Pflicht	$f(a, b, c)$	3x Pflicht! Typ ist beliebig!
Optional	$f(a=None, b=0, c="")$	

Variabel/Liste $f(*args) \rightarrow$ interne Liste

Variabel/Dict $f(**kwargs) \rightarrow$ interne Dict

def $f(a, b=0, c="")$:

print (a)

print (b)

print (c)

$f()$ ↴

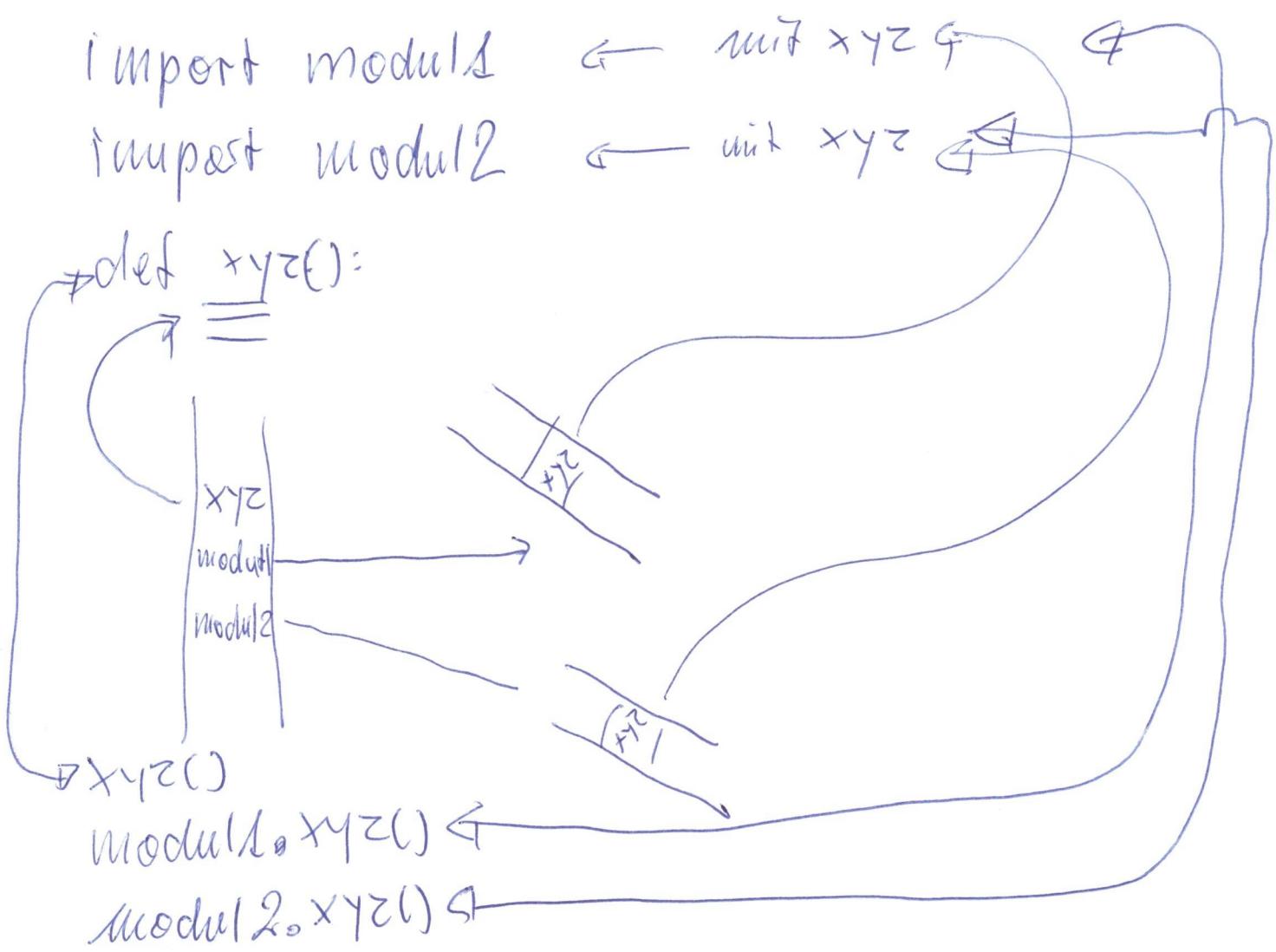
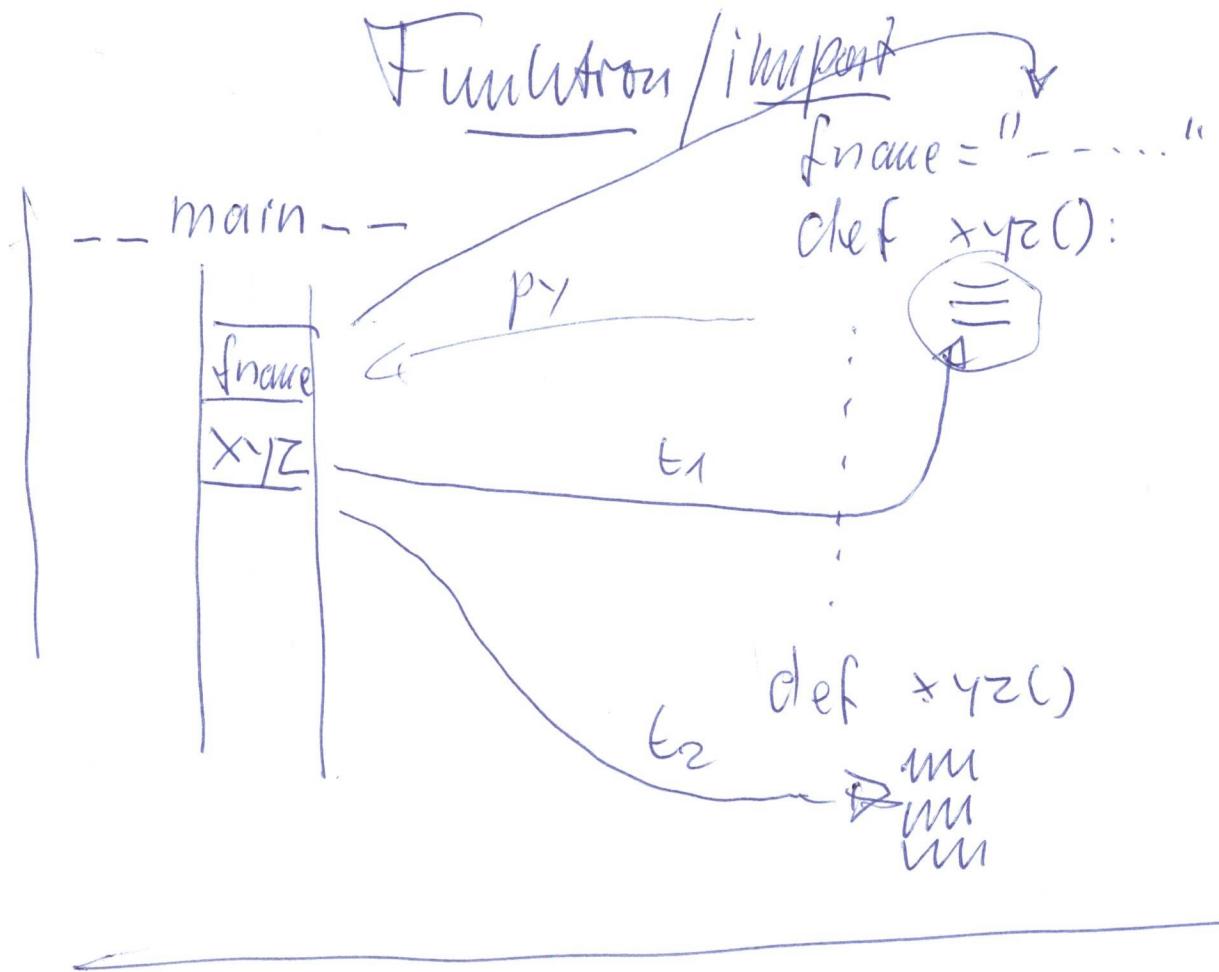
$f(4711) \rightarrow 4711, 0, ""$

$f(4711, 123) \rightarrow 4711, 123, ""$

$f(4711, 123, "Willi") \rightarrow 4711, 123, "Willi"$

$f("Itemz", "Karl", 99) \rightarrow "Itemz", "Karl", 99$

$f(4711, c="Hit") \rightarrow 4711, 0, "Hit"$



Aufgabe

"Lottochine"

11:30

→ Eingabe: Benutzer gibt ein
6 aus 49
Name Doublette
6 Zahlen
als Liste verwaltet
"Tipp" ← von Int.

→ Ziehung: 6 Zufallszahlen
Wählen Sie
6 aus 49
aus dem Netz, als Liste

→ Auswertung: vergleicht Tipp mit Ziehung
gibt Anzahl Treffer

→ Bericht: Ausgabe Tipp, Ziehung, Treffer

Modul

- Datei → Klasse & Funktionen def
 - Verzeichnis → Dateien mit → —
-

import Modulname

Bsp

sys
os } „batteries included“
random
re
...

openpyxl
cx-oracle
pandas } mit pip / conda
my-lib.py } eigene

Virtualenv

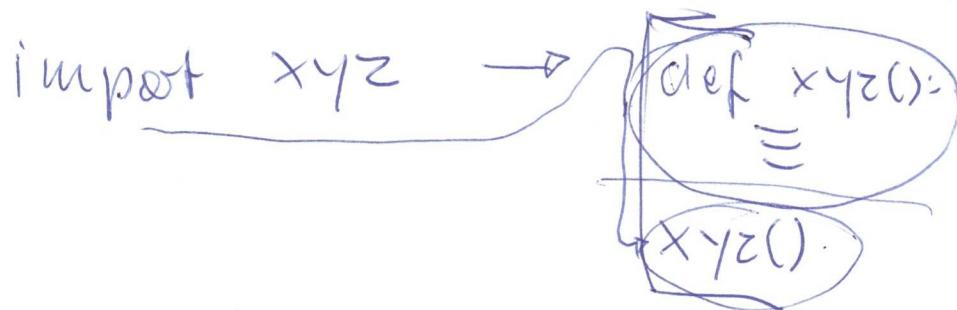
Modulname.Name()

Modulname.Function()

...

Import

import lädt Datei und führt aus!



import xyz → Aufruf xyz.function()

from xyz import function → function()
da function in
--- main ---
liegt.

import xyz.abc.module.alles as xyz-alles

statt xyz.abc...alles.function()

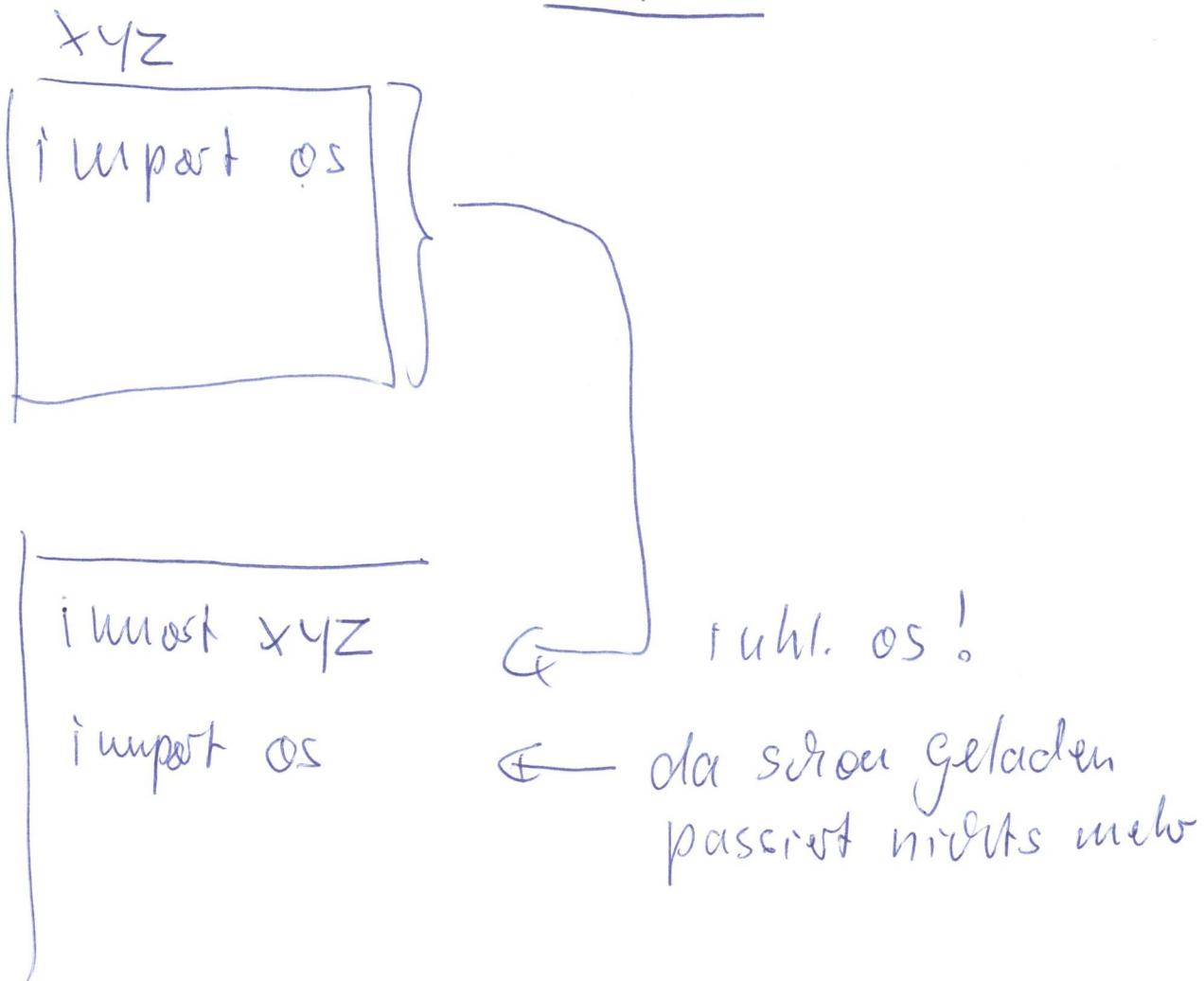
xyz-alles.function()

Alias

from xyz import function as f-xyz

Aufruf: f-xyz()

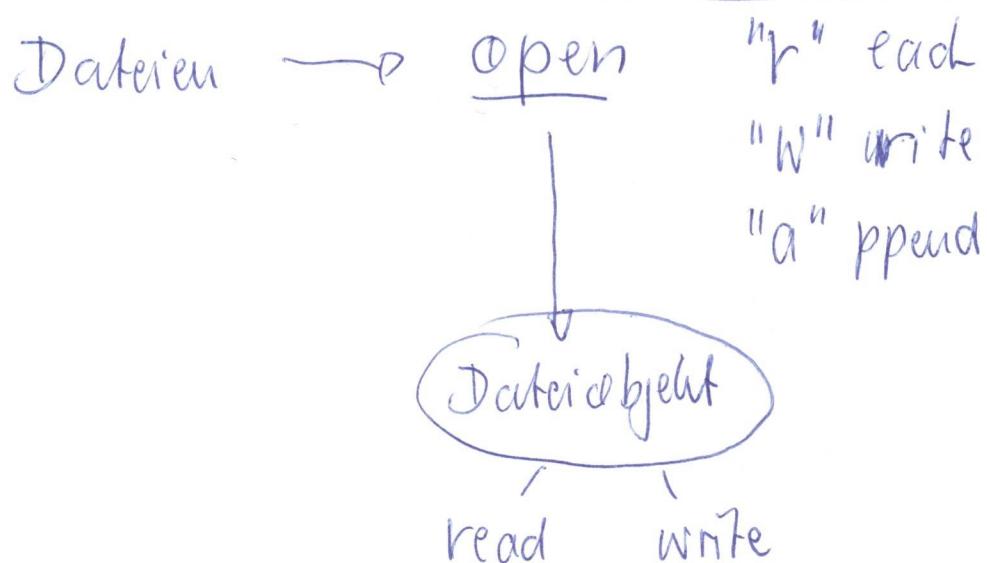
Import



Datenquellen

Interaktion: input / print

Kommandozeilе import sys
sys.argv



Formate: csv.

(xls xlsx)

json

yaml

xml

html

...

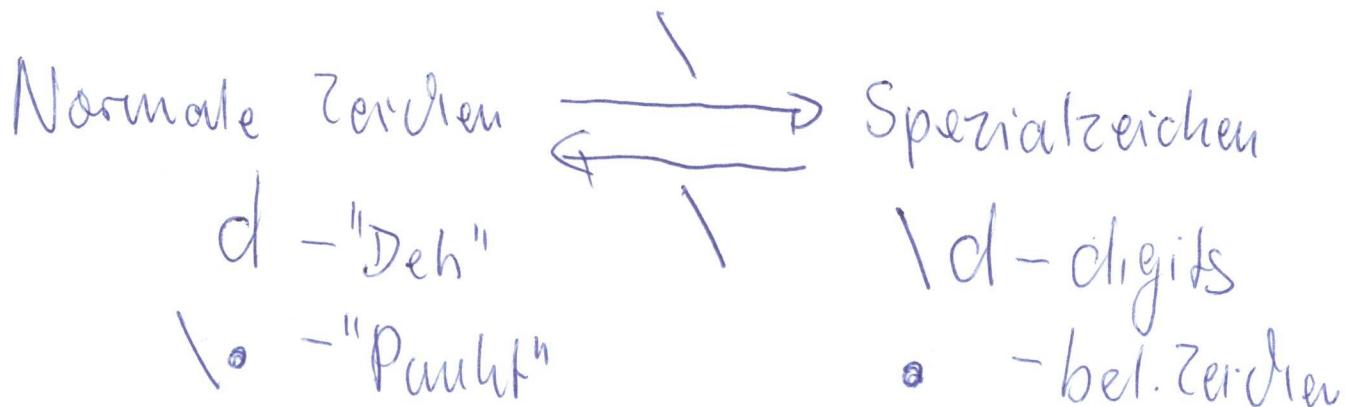
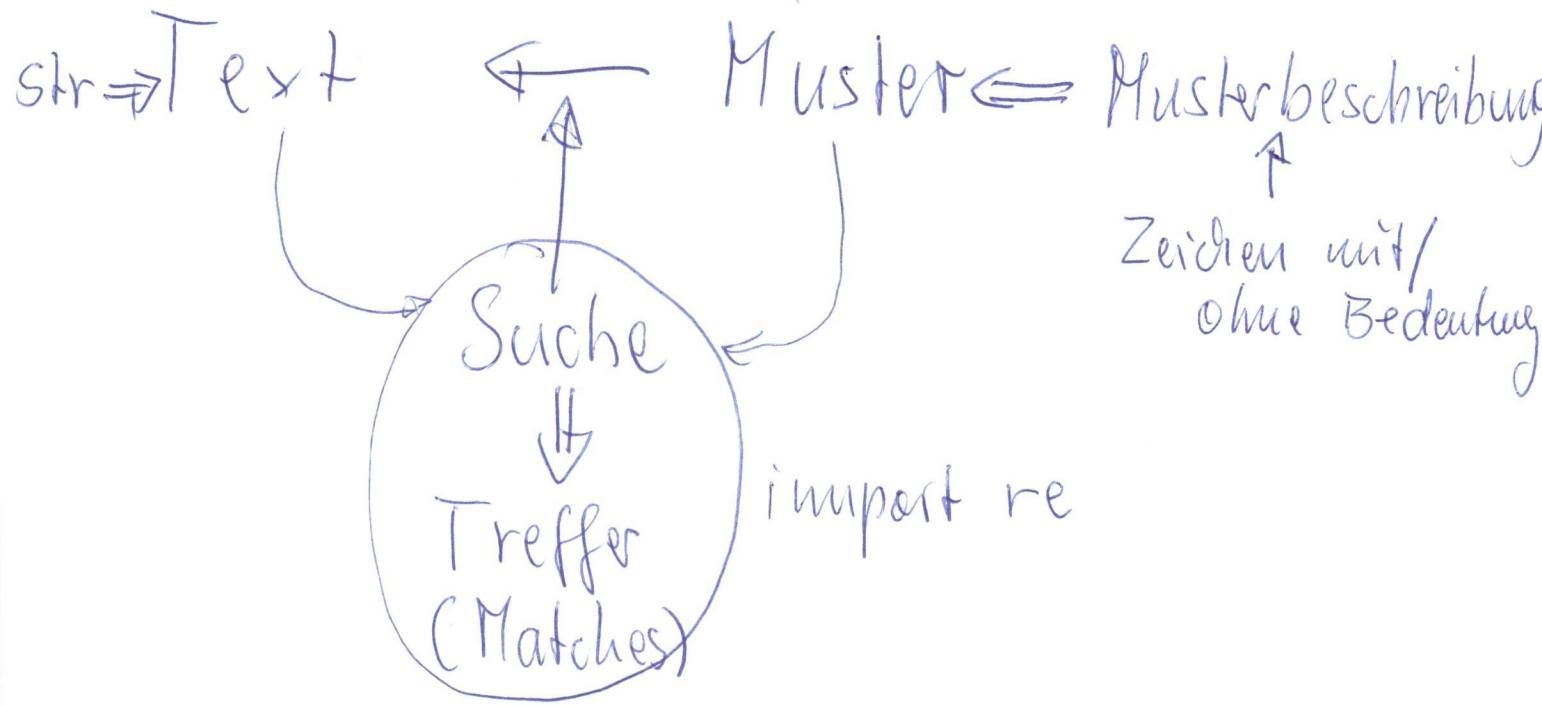
| File - Klassen

import json

daten = json.loads(open("Datei.json"))

↑
Dict

RegEx



RegEx

• bel. Zeichen

[] Zeichen aus Gruppe [a-z]

^ Beginn $\lambda \$$ = leere Zeile
\$ Ende = " "

| oder a | b

() Treffergruppe

"Quantifier"

R * R Ø mal 1234 & ✓

Oder bel. oft

R + 1 - ∞ 1234 & neu

R ? 0, 1

R { von, bis } von-bis mal
das R