

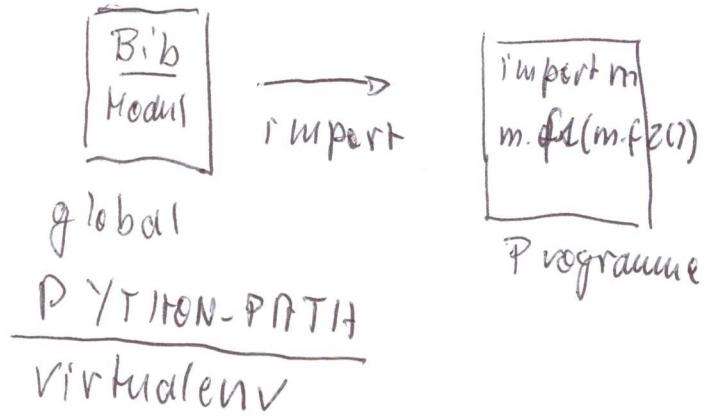
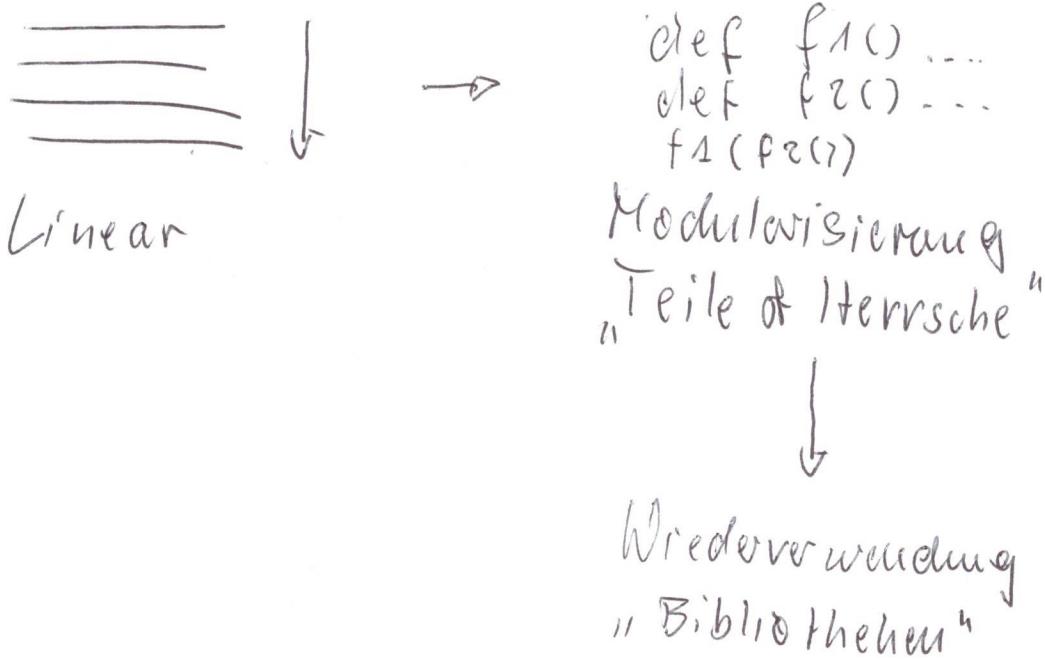
Python

Fortgeschritten

fnXX, training, uc-it.de
01-11

Kurs * 1234 * ucit

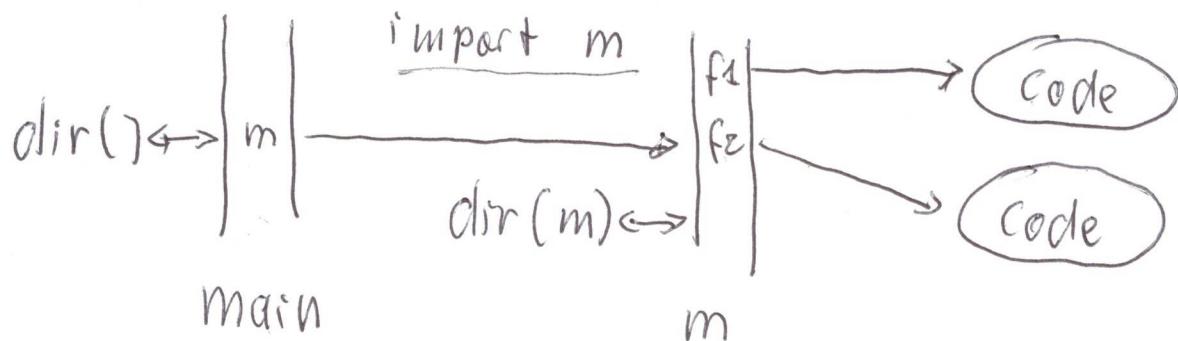
Programm = Struktur



Import

→ lesen & ausführen

z.B. Variablen gesetzt
Init-code ausgeführt
def - gelesen & gespeichert

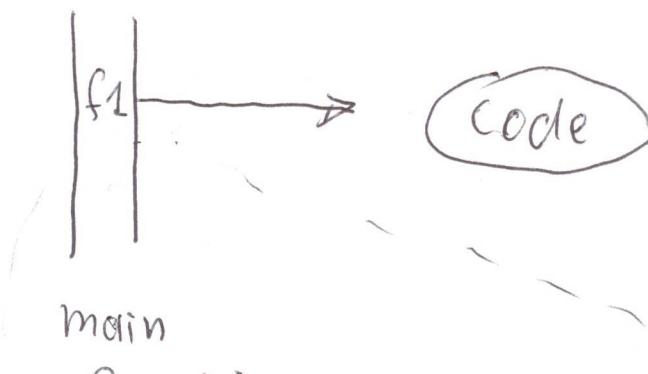


`m.f1()`

Suche in main nach m
in m nach f1
und führe aus

`from m import f1`

Bsp für Kollisionen



`def f1():`

`def f1():`
 ↳ x x x
 x x x

Neu!

→ ~~from m import f1~~
~~import m~~
~~f1() => m.f1()~~

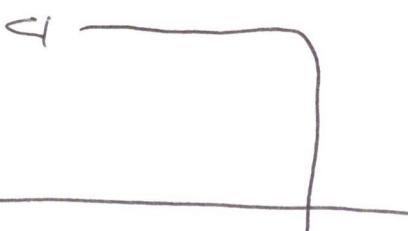
der letzte gewinnt

Programm of Lib in einer Datei

```
m.py |  
└ def f1()  
    ↓  
    def f2()  
    {  
        if __name__ == "__main__":  
            f1()  
            f2()  
    }
```

python3 m.py

```
main └ f1  
      └ f2  
      - - name - - = - - main - -
```



→ f1 ausget.
f2 -a-

import m

```
m └ f1  
      └ f2  
main └
```

→ f1 geladen
- f2 geladen

class CommandConfigData

≡ def m1()

if --name-- == "--menu--"

Tests von CommandConfigData

ccd.py

import ccd

class CCCompare

≡ def xyz()

if --name--

Tests

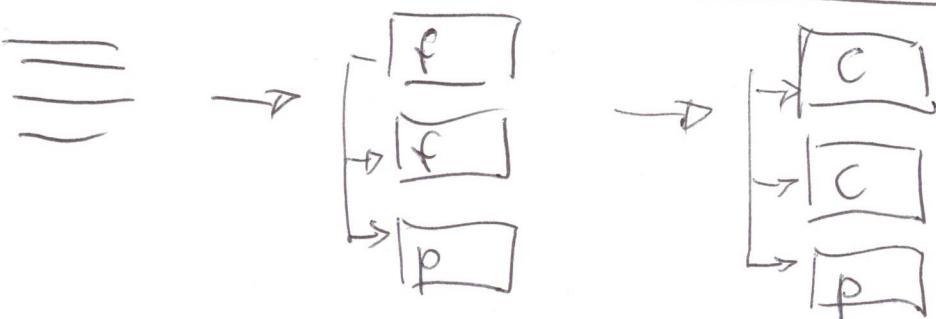
ccc.py

import ccd //

import ccc

Programm-arbeit()

p.py



o = CCCompare()

d1 = CommandConfigData()

d2 = CommandConfigData()

o = ccd.CCCompare()

o.mXYZ()

Daten

None

Skalare

(Einfache Variablen)

int

Bool

float

?

type(x)

String

Datentyp für alles

Text - Standarddaten

Listen

Dictionary

Class-Definitions

Daten \Leftrightarrow Klassen (builtins, user defined)



enthält



Methoden \Leftrightarrow Funktionen

Attribute \Leftrightarrow Variablen

Bauplan

Objekt

Entsprechend Speicher =

Speicher
Objekt

l = list()

Klasse

Referenz

l.count()

obj ~ methode

Fehlerbehandlung

- * Versuch macht klug!
- * Merchen und um Verzeihung bitten Wenns schiefgeht

try:

Versuch

→ alles ist gutgelaufen

except Fehlername A as e:

Fehlerbehandlung sorry!

except Fehlername B

=

Finally:

Wird immer ausgeführt!

Rufräumen sowohl bei Fehler
als auch OK

Funktronen

二

A hand-drawn diagram of a vertical pipe system. A horizontal pipe extends downwards from the left side of a vertical pipe. A valve is located at the bottom of the vertical pipe. An arrow points downwards from the valve, indicating the direction of flow.

def \sqsubset Name(\downarrow^*):

1

→ return
optional

* Pflicht

def f(a)

Optional mit Defaultwert def f(a=0)

Del. Länge Parameterliste def f(*a)

bel Lang key-value Parameter def f(**a)

Reihenfolge f(p, o=42, *args, **kwargs)

Pflicht-Opt-bez. Liste - bez K-V Paare

Bsp `f("Willi", 4711, *args "Hamburg", "Museums")`

P O

* args
"Hamburg", "Museumist",

**kwargs {Kunde = True,
Angestellt = False})

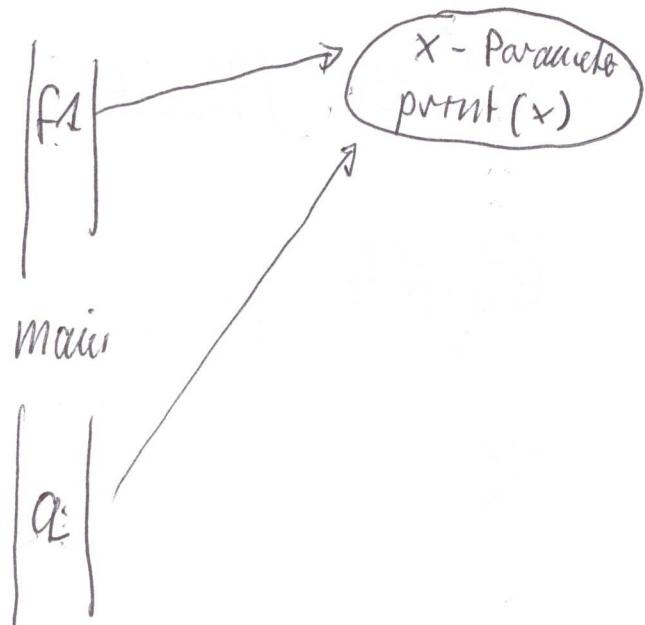
Funktoren als Objekte

```
def f1(x):  
    print(x)
```

```
f1("Ituhu")
```

```
a = f1
```

```
a("Mloha")
```



```
def caller(xf):
```

```
    if isinstance(xf, FunctionType):  
        xf("Hey")
```

```
caller(f1)  
caller(a)
```

z.B. bei Sortierprogramme
daraus entstehen Lambdas
und „Closures“

Übung

Programm zur Ausgabe einer

ASCII - Tabelle

Dez 32 - 127

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|-----|---|---|---|---|---|---|---|---|---|---|
| 30 | | | | | | | | | | |
| 40 | | | | | | | | | | |
| 50 | | | | | | | | | | |
| 60 | | | | | | | | | | |
| 70 | | | | | | | | | | |
| 80 | | | | | | | | | | |
| 90 | | | | | | | | | | |
| 100 | | | | | | | | | | |
| 110 | | | | | | | | | | |
| 120 | | | | | | | | | | |

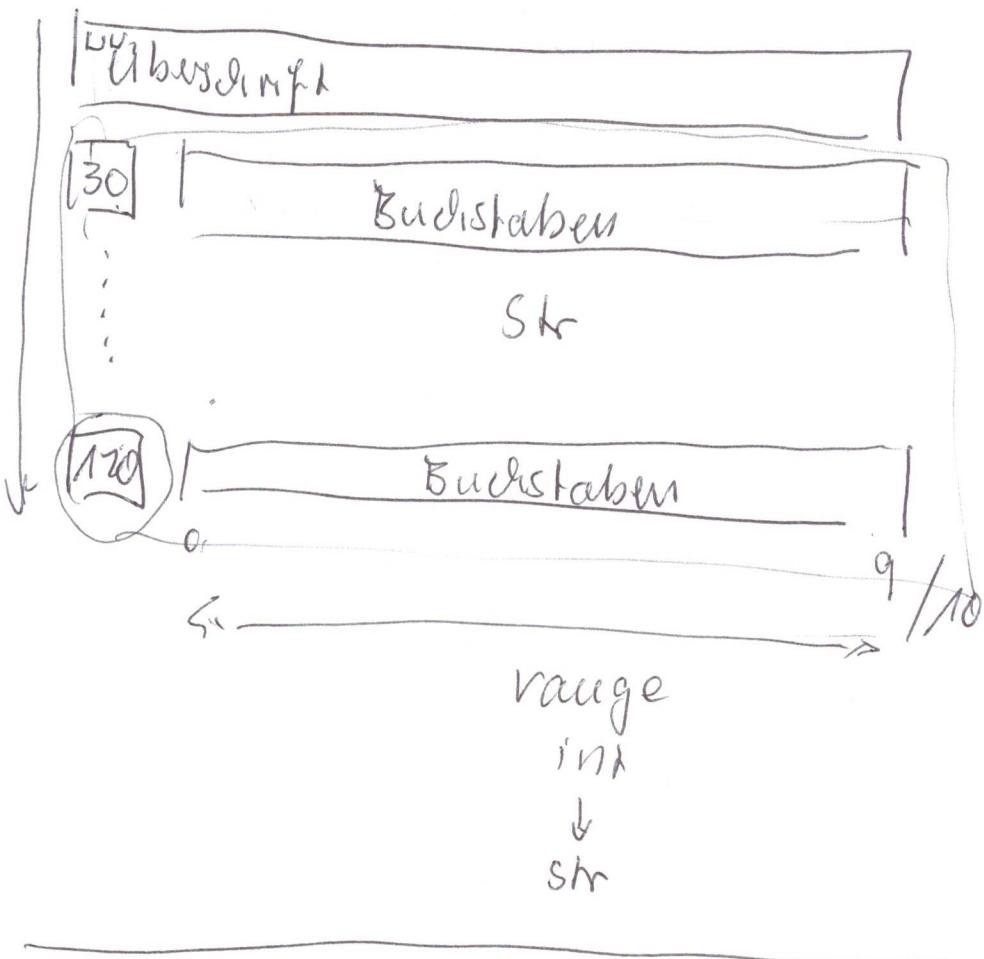
X V → R: std::out

32_{dez} ? → Welcher Buchstabe ist das?

30

range
int
↓
str

130



bau überschrift

```
header=[['\u20ac',] range(0,10)]
for n in range(0,10):
    header.append(str(n))
zeilen=[]
for i in range(30,130,10):
    Zeile=[str(i)]
    for j in range(0,10):
        Zeile.append(chr(i+j))
    zeilen+=Zeile
return header, zeilen
```

Tricks mit Listen / Sets / Dicks

String: "L" * 80 → "← 80 →"
 "—" * 80 → "————— 80 —————"

List : [0] *80 → [0, 0, ..., 0]
0 → 79

[None] * 1024 → [None, None, ..., None]
0 9 — 0 1023

| | | | |
|---------|-----------------------------|-----------------|---------------------------------|
| String: | "Hello, world" | [3:] | 3 bis Ende "lo, word" |
| | ↑ A ↑ O ↑ 11 | | |
| | len → 12 | [:3] | 0 bis 3 "l+el" |
| | | [:2] | Jeder 2 Buchstabe "el, wrld" |
| | | [-1] | "d" |
| | | [-2:] | "ld" |

lst : ["A", "b", "e", "r"]
 len → 4
 0 1 2 3

~~[::2]~~ → ["b", "r"]

~~[::1]~~ → "r"

[2:] → ["e", "r"]

[:-1] → ["A", "b", "e"]

List Comprehension

Eingabedaten → Text, mit IP-Adressen

REX.compile(r"([0-9]\{1,3\}.)\{3\}[0-9]\{1,3\}")

IP-liste = []

for zeile in eingabedaten.splitlines():

if REX.search(zeile):

IP-liste.append(zeile)

print(IP-liste)

IP-liste = [zeile for zeile in eingabedaten.splitlines() if REX.search(zeile)]

print(IP-liste)

List Comprehension

- Kompakter Code
- Verkürzung langer Konstrukte mit for
- potenziell schneller

$[1] \rightsquigarrow [1, 2] \rightsquigarrow [1, 2, 3] \rightsquigarrow [1, 2, 3, 4]$

① $l_1 = [x \text{ for } x \text{ in datenliste}]$

② $l_2 = [x \cdot \text{ for } x \text{ in datenliste if } x\text{-Bedingung}]$

③ $l_3 = []$
for x in datenliste:
 $l_3.append(x)$

④ $l_4 = []$
for x in datenliste:
if x -Bedingung:
 $l_4.append(x)$

Dictionary Compr.

list $\rightarrow [e, e, e, e, e, e, e]$

↑

index or int

index > n-1



dict $\rightarrow \{k_1:v, k_2:v, k_3:v\}$

↑

key \leftrightarrow str

Bsp d = { "Name": "Willi", "Alter": 40, "PLZ": "22765" }

d["Name"] \leftrightarrow "Willi"

key not in d



Wertepaare \rightarrow dict - Überführung

["abc123", "xyz456", "lm000"]

{ "abc": 123 }

"xyz": 456

"lm": 000 }

d = {}

for v in Wertepaare:

k = v[:3]

w = v[3:]

d[k] = w

d = { v[:3]: v[3:] for v in Wertepaare }

print(d)

Aufgabe

a) Wie könnte man eine ASCII-Tabelle mit einer Dict erzeugen?

$$\text{ASCII} = \{ . \text{ ? } \text{ ? } \}$$

b) Wie könnte man das mit dict-Compr. machen?

2 Varianten Zahl \rightarrow Buchstabe
Buchst \rightarrow Zahl

ASCII 32 - 127

Iteratoren, Generatoren, Mapper, Filter

Iterator - ein Ding das immer wieder aufgerufen wird und das nächste Element liefert
Quelle: statisch, dynamisch berechnet

Generator - erzeugt Daten mit jedem Aufruf

map - Wendet eine Funktion auf ein Iterable an

filter - Wendet eine Funktion auf ein Iterable an, die Funktion wählt etwas aus

Bsp - map

```
d = ["apple", "cherry", "banana"]
```

```
def calc_len(x)  
    return len(x)
```

```
for n in map(calc_len, d):  
    print(n)
```

5, 6, 8

```
l = list(map(calc_len, d)) → [5, 6, 8]
```

$l = list(map(len, d))$

als list comprehension machbar

```
l = [len(x) for x in d]
```

Bsp filter

~~def max~~

def cmp(x):
 if x > 100:
 return True
 else:
 return False

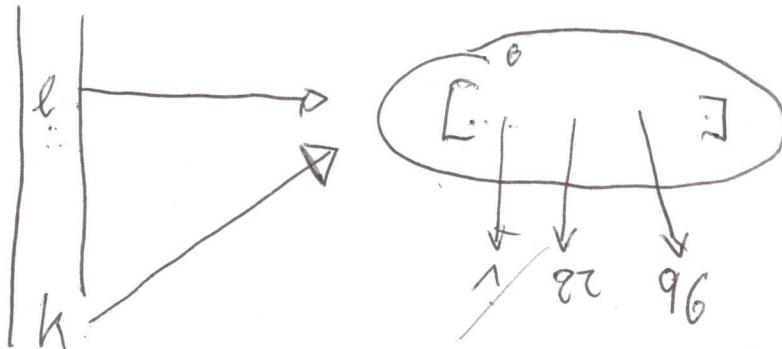
def cmp(x)
 return True if x > 100 else False

daten z.B aus Datei, als Liste

+ liste = list(filter(cmp, daten)) $\rightarrow [T, T, T]$

+ f-liste = [cmp(x) for x in daten] $\stackrel{z.B.}{=} [T, F, T, T, F]$

Shallow vs deep Copy



$$l = [1, 22, 96]$$

print(l)

$k = l$

Neue Kopie der Liste ??
Zusätzlicher Verweis !

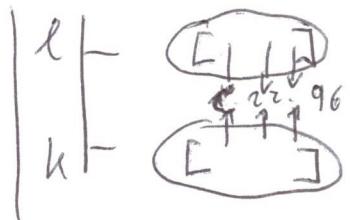
$$k[0] = 4711$$

$\text{print}(l) \rightarrow [4711, 22, 96]$

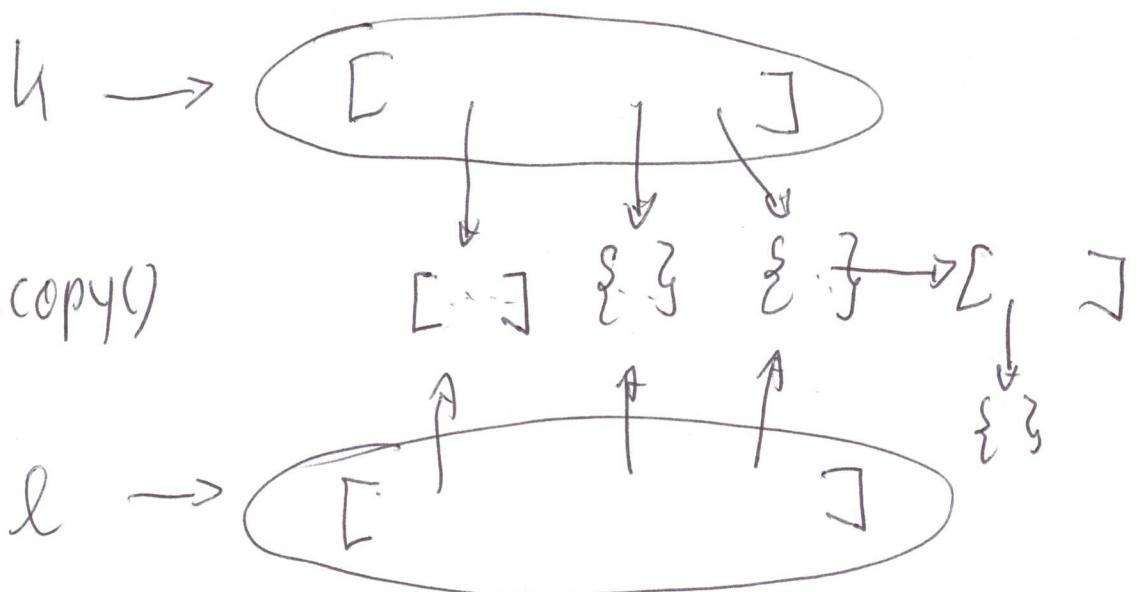
Wie wird k eine Kopie von l ??

a) $k = l[:]$ slice ist nur Kopie

b) $k = l.copy()$



flache/shallow Copy



$h[0] \neq l[0]$

$h[0][1] = l[0][1]$ ↗

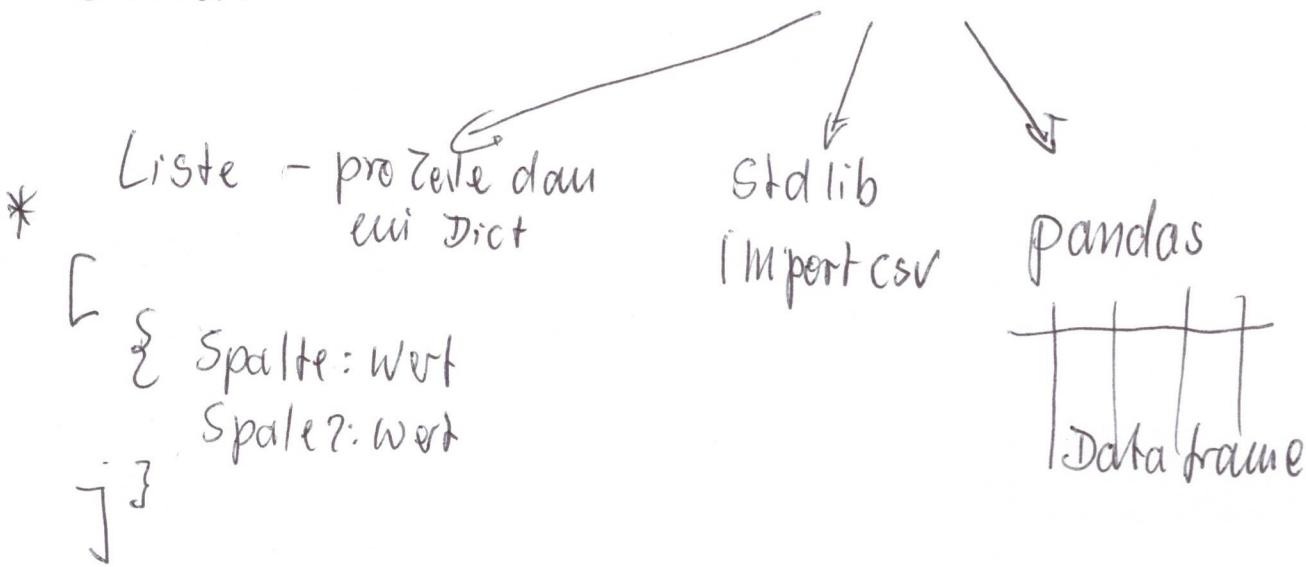
Aufgabe

Einlesen von Börsendaten

- Datenstruktur ?
- Deutsches Datumsformat
- \$-Werte in float

Materialien / Historical Quotes.csv

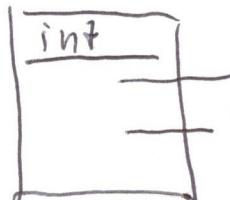
Daten → Textdatei → CSV-Werte



Lese Daten → Zeilenliste ["str"] ohne "\n"
splitlines()

Klassen / OOP

Python ist eine OO-Sprache
→ alles ist Objekt von einer Klasse



Spercherung
Funktionalität

a = 4711

a →



| | |
|---------|--------------------|
| public | Kapselung |
| private | Information Hiding |
| | Entkoppelung |
| | Vererbung |

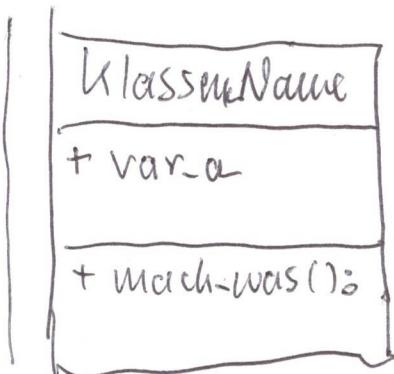
class KlassenName:

 |__def __init__(self, ...):

 |__ self.var-a = 4711 #Attribut

 def mach_was(self): #Methode
 print(self.var-a)

Schnittstelle



Beispiel

class Verarbeiter:

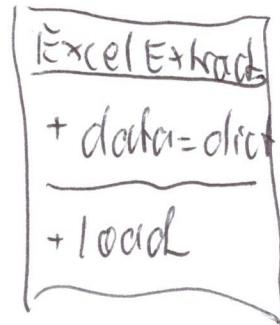
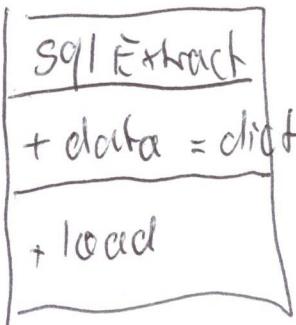
def __init__(self, source-name, sql-text):
 self.source = source-name
 self.sql = sql-text

def load_data(self):

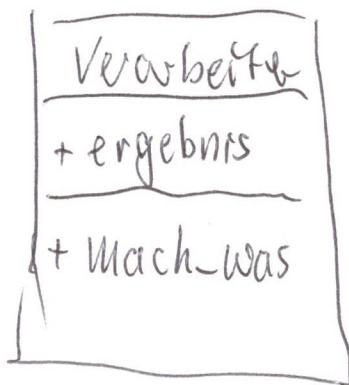
csr=sqlbib.connect(self.source).execute(self.sql)
self.data = csr.readall()

def berechnung(self):

Zusammenspiel



...



```
data = SQLExtract(...).load()
```

```
---  
result = Verarbeiter().MachWas(data)
```

Bestandteile

Class Integer:

Initialisierung

```

def __init__(self, value)
    self.value = value
    self._check(self.value)

    hidde, aber aufrufbar von extn
    oder
    self._check(self.value)

    „fully hidden“, nicht aufrufbar von extn
  
```

Python
„dunder“

```

def __bool__(self):
    if self.value != 0:
        return True
  
```

Offizielle
Schwartzfille

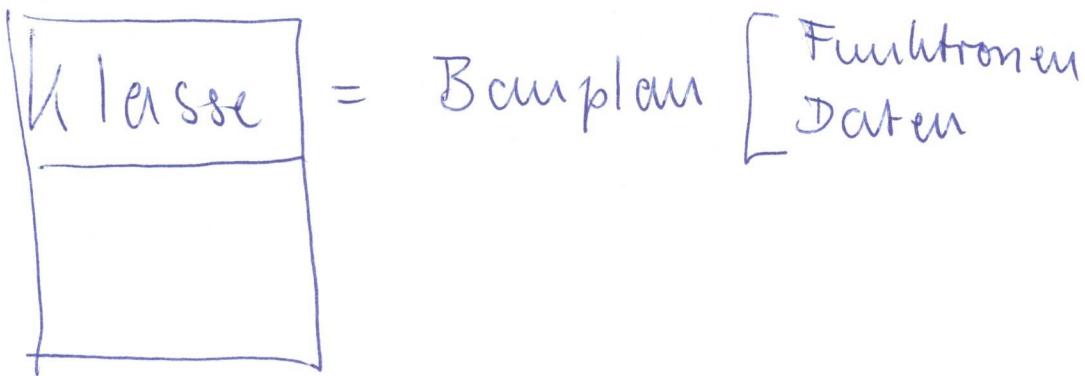
```

def bit_pattern(self):
    # Macht aus self.value
    # einen Bitumterschlag
    return pattern_string
  
```

private/
hidden

```

def _check(self):
    ...
    oder
    def __check(self):
        ...
  
```



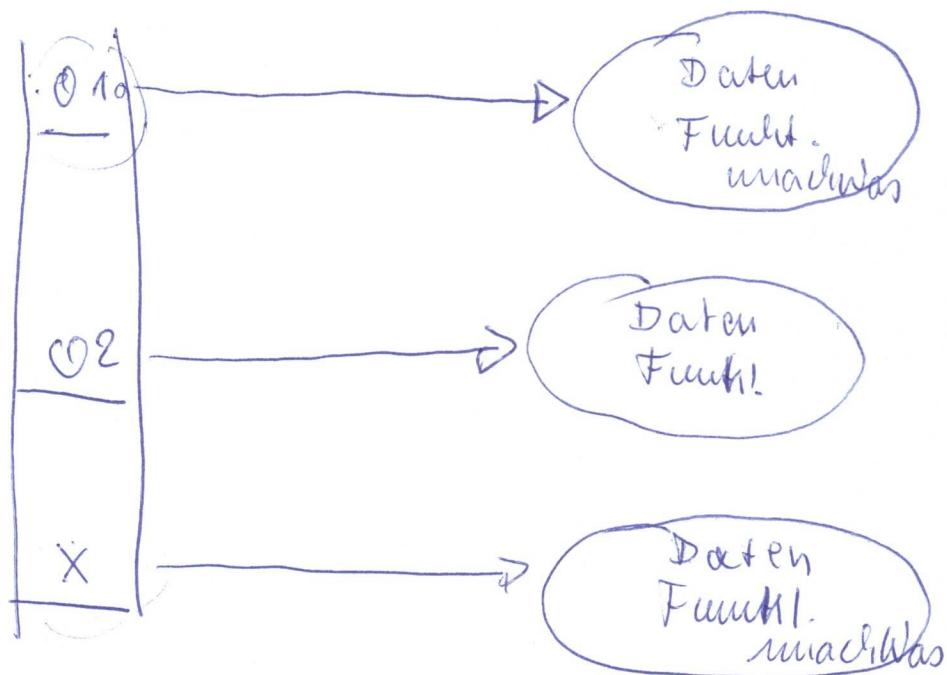
im Programm: 1 - n Objekte vom Typ "Klasse"

$o_1 = \text{Klasse}()$

$o_2 = \text{Klasse}()$

$x = \text{Klasse}()$

3 Variablen die auf Objekte vom Typ zeigen

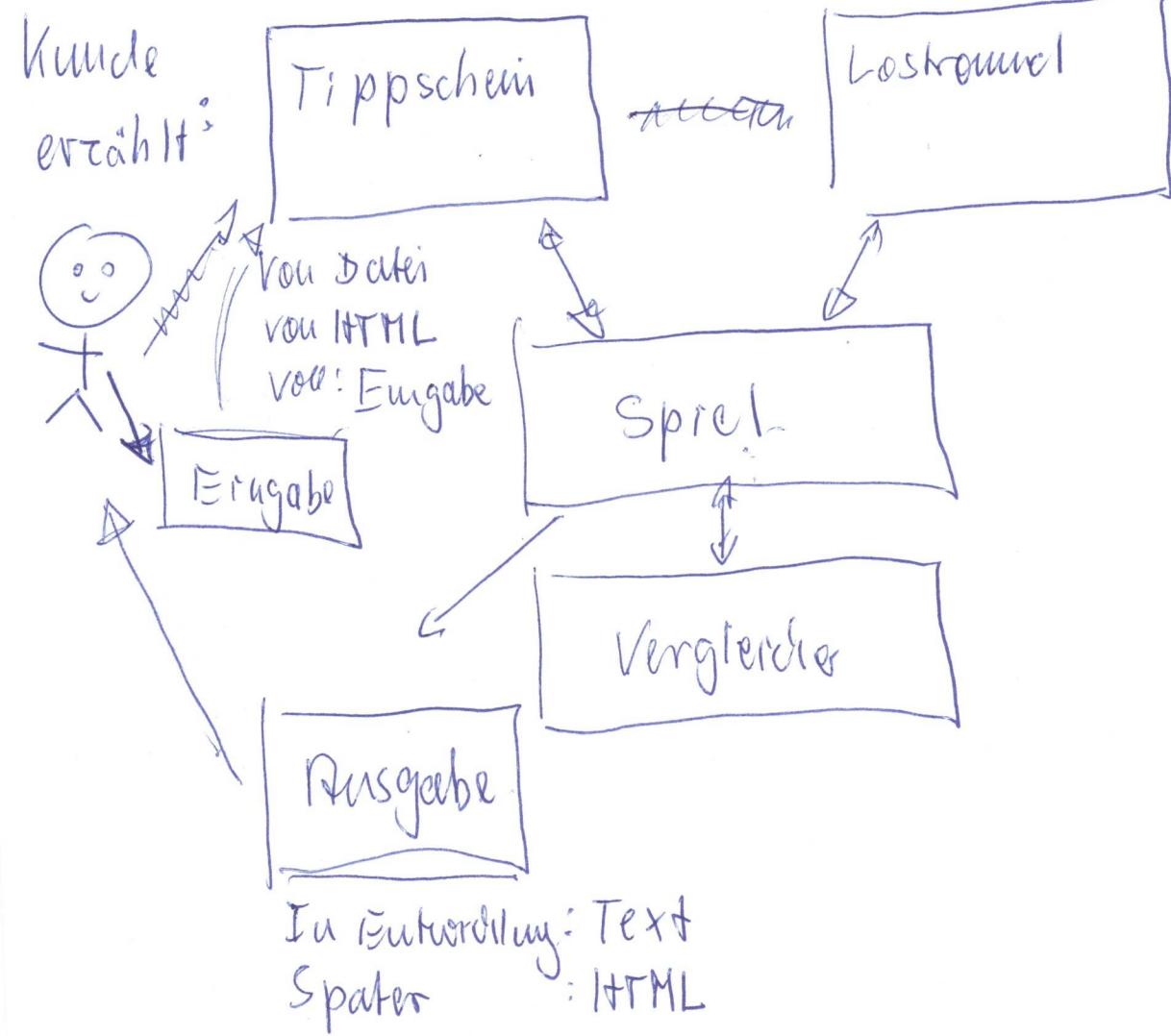


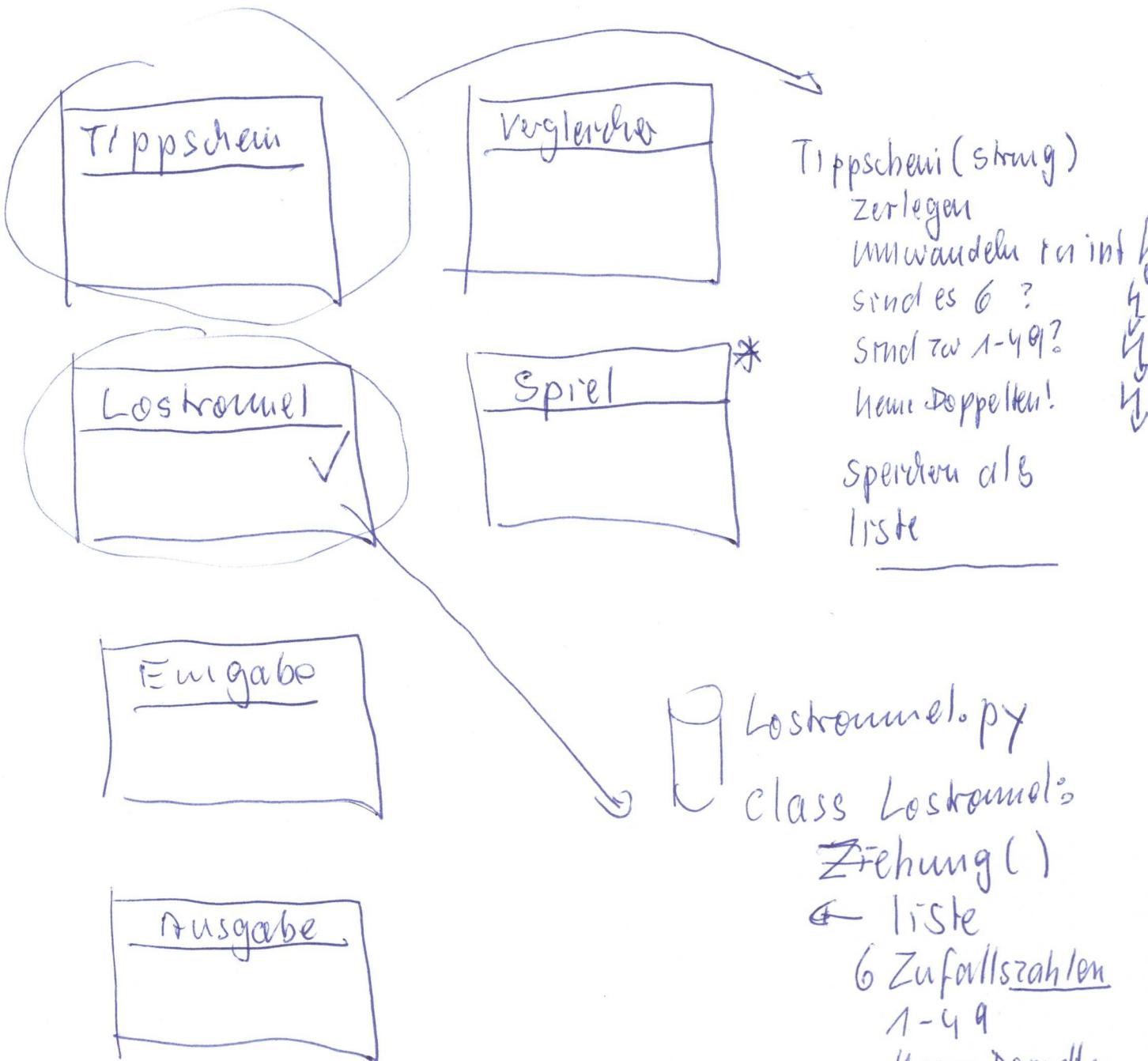
$o_1, \text{ machWas}()$

$x, \text{ machWas}()$

Objekt oder Zustand vom Typ "Klasse"

Übung: Online-Lottobude





Tippschein (strung)
 zerlegen
 umwandeln + in int
 sind es 6?
 sind zw 1-49?
 keine Doppelten!
 speichern als
 liste

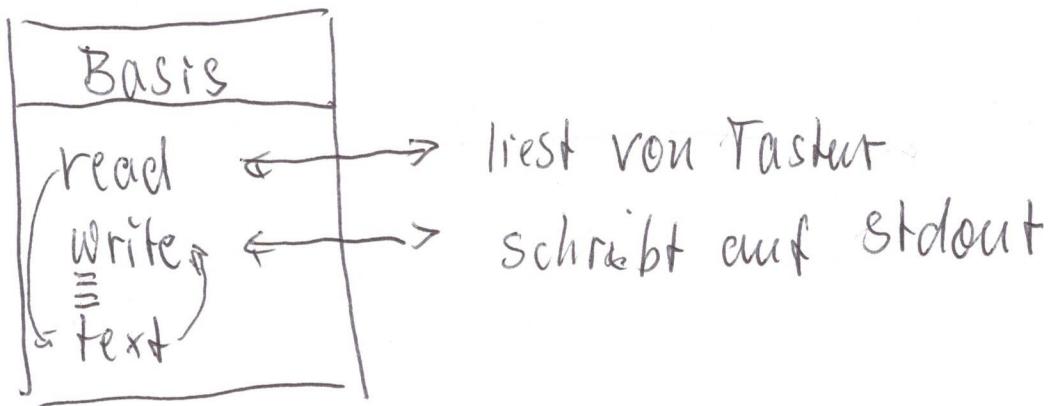
Lostrommel.py
 class Lostrommel:
 ziehung()
 & liste
 6 Zufallszahlen
 1-49
 keine Doppelten

```

if __name__ == "__main__":
    l = Lostrommel()
    print(l.ziehung())
  
```

Unit-Tests

Vererbung



① = Basis()

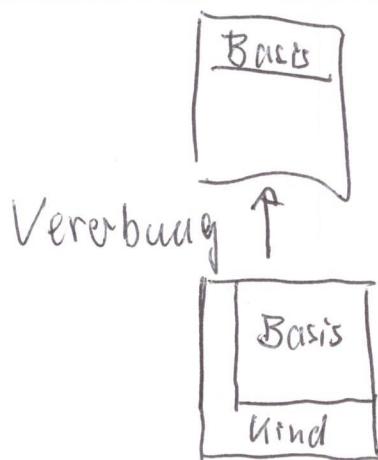
② .read()

③ .write()

Erweiterung: Ursprünglich: def write(self):

Variante: def write(self, datei=None):
 ≡ if datei:
 print(self.text, file=datei)
 else: print(self.text)

Oder:

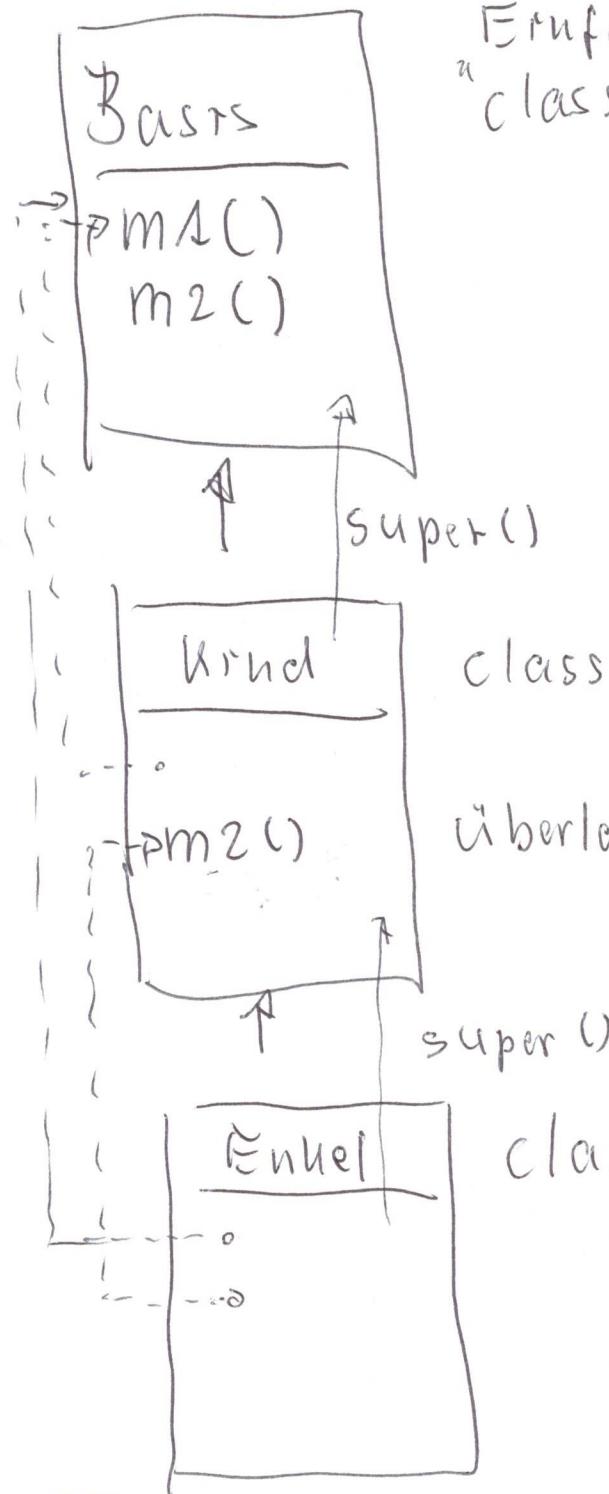


Basis Klasse
Elteruskasse

Kinder Klasse
Abgeleitete Klasse



Vererbung



"Einfache Vererbung"
"class Basis"

`--init--`

wird NICHT vererbt!

class Kind(Basis)

`--init--`

`super.--init--`

überladen! „override“

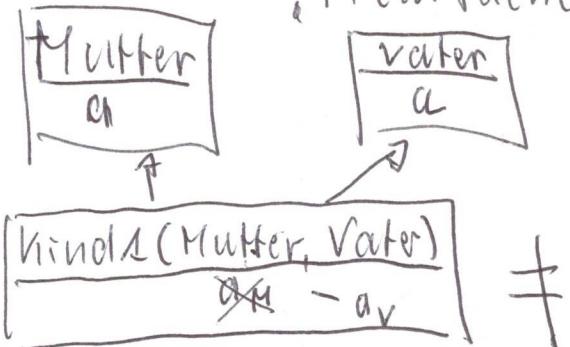
`super()`

class Enkel(Kind)

`--init--`

`super.--init--`

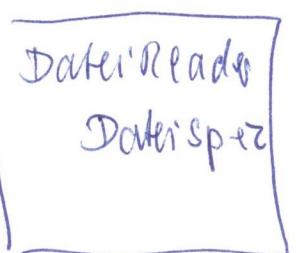
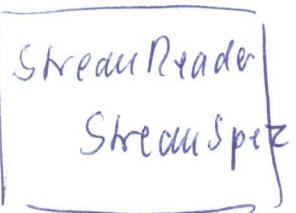
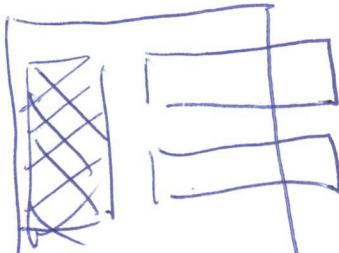
"Mehrfache Vererbung"



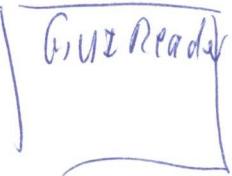
Kind1(Mutter, Vater)
~~aM - av~~

"Interfaces"

Datenverarbeitung



def machwas(reader):
 \daten = reader.read()



Später:

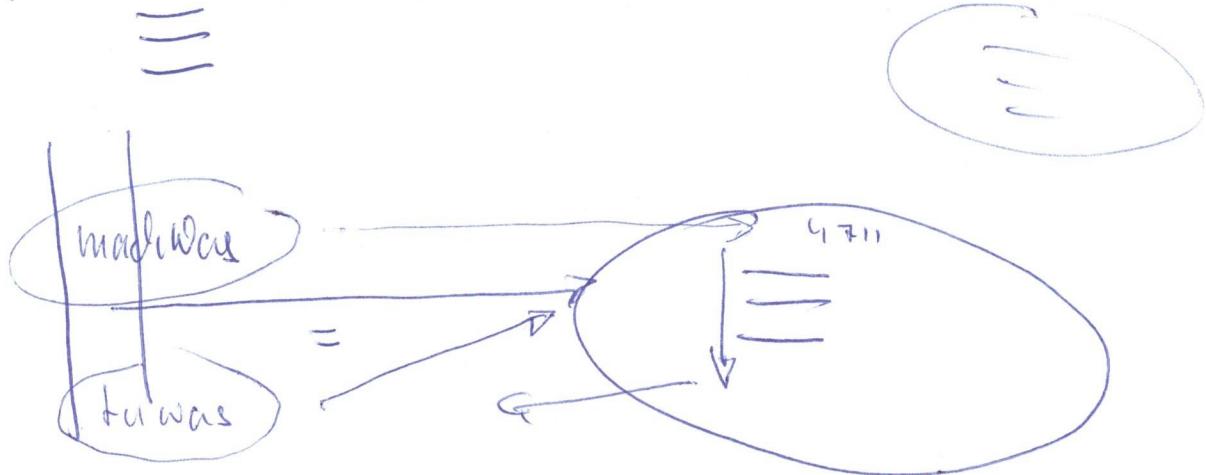
"Entkoppelung"

machwas(StreamReader())

machwas(ITmlReader())

Fun hofraum

def machwas(p):



machwas(v1711)

Machwas ← Class FunctionType

fuwas = machwas

fuwas()

def executor(f):
 f()
 =

Lambda-Funktionen

anonyme einzeilige Funktion mit automat. return

$f = \lambda x: X * X$

\Leftrightarrow def $fuwas(x):$
return $x * x$

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

$e = \text{map}(fuwas, l) \rightarrow [1, 4, 9, 16]$

$e = \text{map}(\lambda x: X * X, l)$

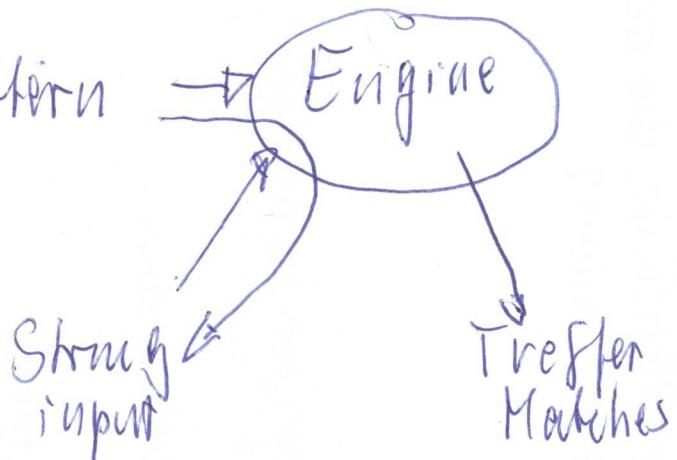
map, filter, sort, search, ...

„generischer Algorithmus“

RegEx

Parser : aus Pattern \rightarrow Engine

import re



Zeichen mit / ohne Sonderbedeutung

Sonderbedeutung: • „bel. Zeichen“

• Punkt

Escape

$^0 \rightarrow \infty$

*

$= 0$ oder 1

?

$\leq - \infty$

+

von-bis {v, b}

Quantifizier

„wiederholt“

R = r \cup r \cup r

() Gruppe

[] Zeichenklammern

\uparrow Aufruf
\$ Ende Anker

Doku

Kommendare #

- # kurz
- # notwendig
- # nicht trivial

elektro: der Kommentar ist der nicht
geschrieben, weil der Code so gut ist

↔ müssen gepflegt

Z. B. an einem RegEx

hier: IP4 kein IP6
Quelle: regex101.com
ret = r"\b\d\{\dots\}\b"

Doku

pydoc (einfach) → sphinx (luxuriös ☺)

"" " " " "
''' ...''' } an bestimmten Stellen
''' ...''' Noamen Platzhalter haben

| | | |
|-----|-------------------|--|
| """ | Modulebene: | Sinn & Aufgabe Historie bes. Hinweise |
| """ | Class X: | Sinn & Aufgabe Hinweise Schnittstelle |
| """ | def m(self, ...): | Sinn & Aufgabe Parameter Returns Raises |

help(X) interaktiv
help(X.m)

Testen

Testen

- Systematisch
- basiert auf Anforderungen
- Interaktiv oder
Automatisch

↳ Unit-Test

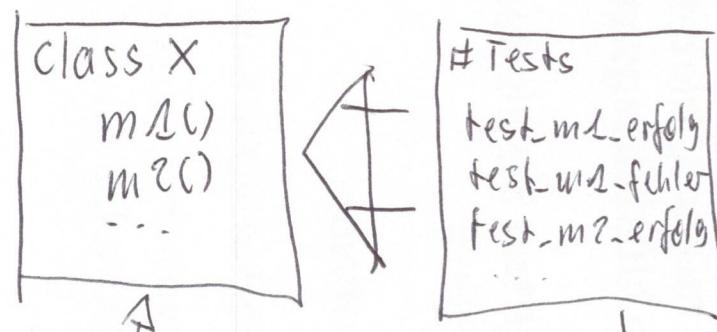
↳ Regressionstest

nach Änderung

⇒ pytest

Debuggen

- Auslöser: Fehler
 - Codeausführung
und Beobachtung
- IDE
→ Interaktiv



✓ test_m1_ergebnis ??
✗ test_m1_fehler ??
✓ ...
✓ ...

Testen

Was testen:

good case

bad case

edge case

z. B. bei Zahlen

[0

..... "aⁿ" .. - . . . 10]

Lotto bude

festen

class Eingabe:

def __init__(self, str_eingabe):

- * Zerlege
- * 6 Stück?
- * int ?
- * Doppelte?

↓
Self. Fipp

} → raise Runtime Error

| | | |
|-------|----------------|------|
| Test: | 1: 1,2,3,4,5,6 | good |
| 2: | 1,2,2,4,5,6 | bad |
| 3: | 1,2,3 | bad |
| 4: | ä,1,2,3,4,5,6 | bad |
| 5: | 1,2,3,4,5,99 | bad |
| 6: | "Blablabla" | bad |

Rufgabe

Klasse

file_walker

- init - und String = Startverzeichnis

String = Dateinamenmuster

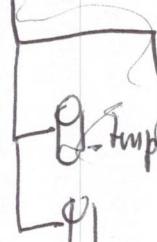
Walk() → Wandert rekursiv ab
← [] Startverz. Mwh gefundene Datei

return: Liste der Namen

~~Beispiel~~ Testen → umfassend oder umflest

C:/

tmp



fw = File_Walker("C:\\tmp", "*tmp")

e = fw.walk()

print(e) → [{"fn": "01\\01.tmp", "typ": "sub1"}]