

# Python Fortgeschritten

---

Ulrich Cüber

---

Start 10:00

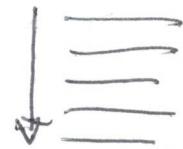
Ulrich Güber

Kontakt@uc-it.de

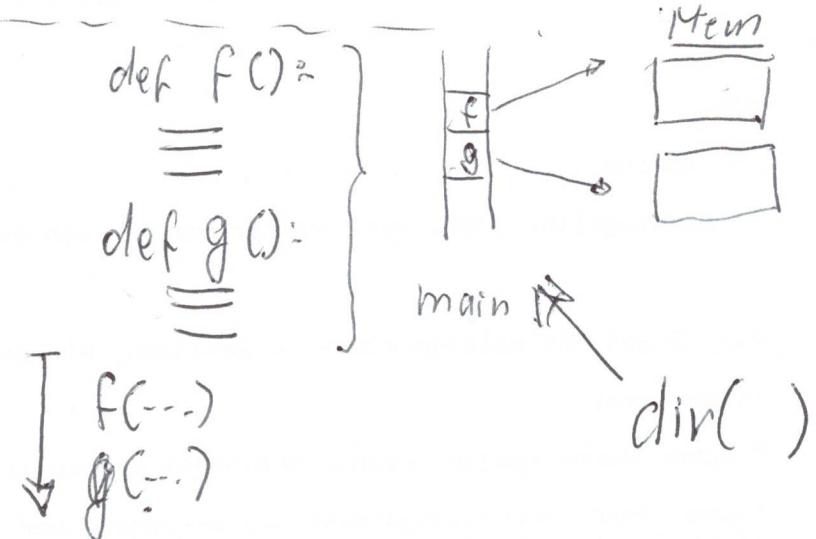
---

# python - Skript $\Leftrightarrow$ Programm

Enfach Linear



Funktionen Einsatz



Einsatz als Standalone  
oder Modul

machWas.py

def f():  
---

def g():  
---

if \_\_name\_\_ == "\_\_main\_\_":  
 f(...)  
 g(...)

„dunder“

Ruf ruf: import machWas  $\rightarrow$  if nicht true  
nur Laden der Def

Nutruf: machWas.py  $\rightarrow$  Name Raum ist mein  
f(), g() aufrufen

# Datentypen + Variablen

a = 4711      type(a) → Int

a = "....."      Str  
      ↑  
      f

      r      Raw  
      b      Binary ← Bilder, Musik, ...  
( u      Unicode )

a = 47.11      IEEE-Format !!

$10^{\pm 300} \equiv 300$  Stellen

Rückrundigkeit      15 Stellen

→ "Big float"

"Festkommazahl"

"Currency"

a = None      "nix"

a = True/False      Bool

a = "

## Operatoren

+ - \* / \ % \*\* { } ==

$a = a + 1 \Leftrightarrow a += 1$

$a = a \text{ op } x \Leftrightarrow a \text{ op=} x$

< <= == >= //

not and or

---

"a" \* 20 ✓ aaaaa...aaa

"a" + 20 ↴ + nicht def für  
str | Int str u. Int

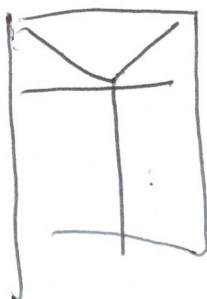
"a" + str(20)

pythonachtet auf

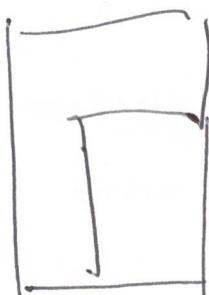
Operatoren, Kontext u. Datentyp!

# Kontrollstrukturen

Einrichtung! Einrichtung! Einrichtung!



if Bed:  
→  
2 für Leerzeichen!  
else:  
→



while Bed:  
→

← break  
continue  
(else ...)

for n in Liste:  
→

## Exception

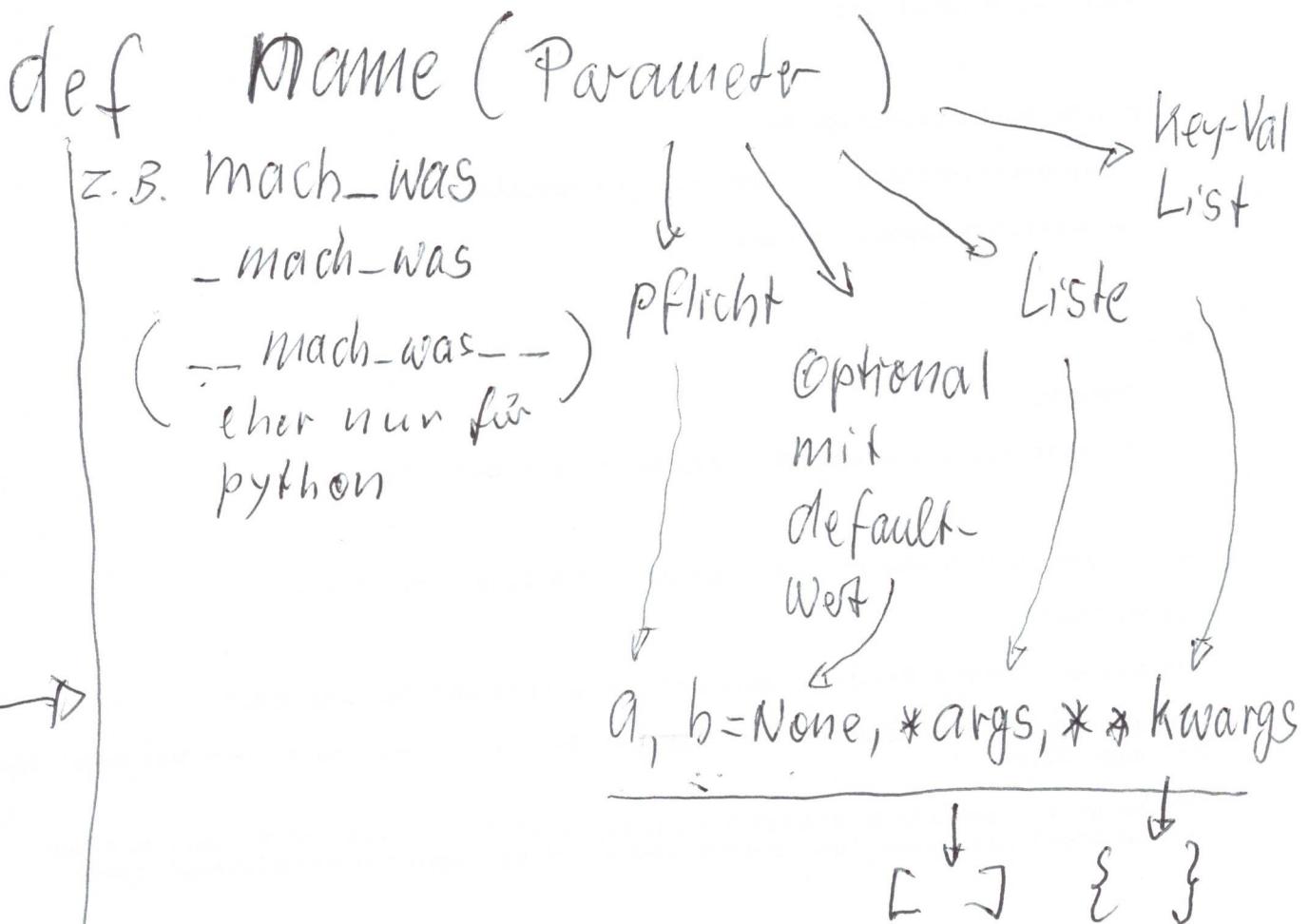


try:  
→ Versuch

{ except:  
→ Fehlerbehandlung }

finally:  
→ immer aufrufen

# Funktionen



return Ergebnis

Badezimmersprinzip

Parameter



Vermieden:

glob. Var

viele return

## Module

Fatteries included! = Standard bib

pypi ← pip

( · anaconda ← conda )

Eigene : Daten mit Fkt/Klassen

import Modulname

`dir()` → main=[..., Modulename, ...]

$\text{dir}(\text{ModulName}) \rightarrow [\dots, \dots, \dots]$

PYTHONPATH wo module liegen

```
import sys
```

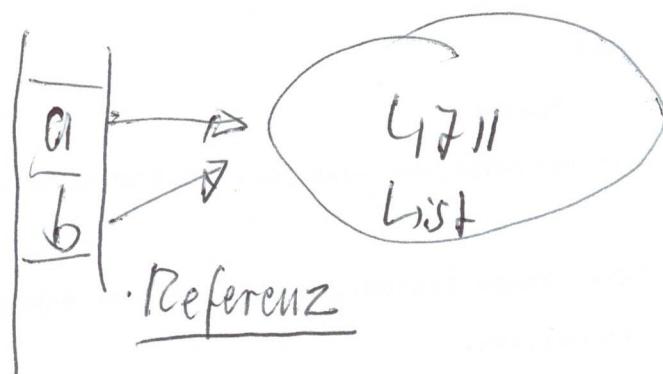
```
print(sys.path)
```

# Liste

`list( )`      } Liste  
`[ ]`

`a = [ 4711 ]`

`b = a`

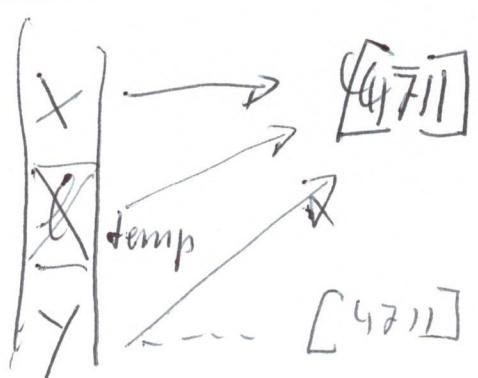



---

`def ahtron(l):`  
 `l.append(4711)`  
 `return l[:]`

`x = []`

`y = ahtron(x)`




---

Kopie: Copy

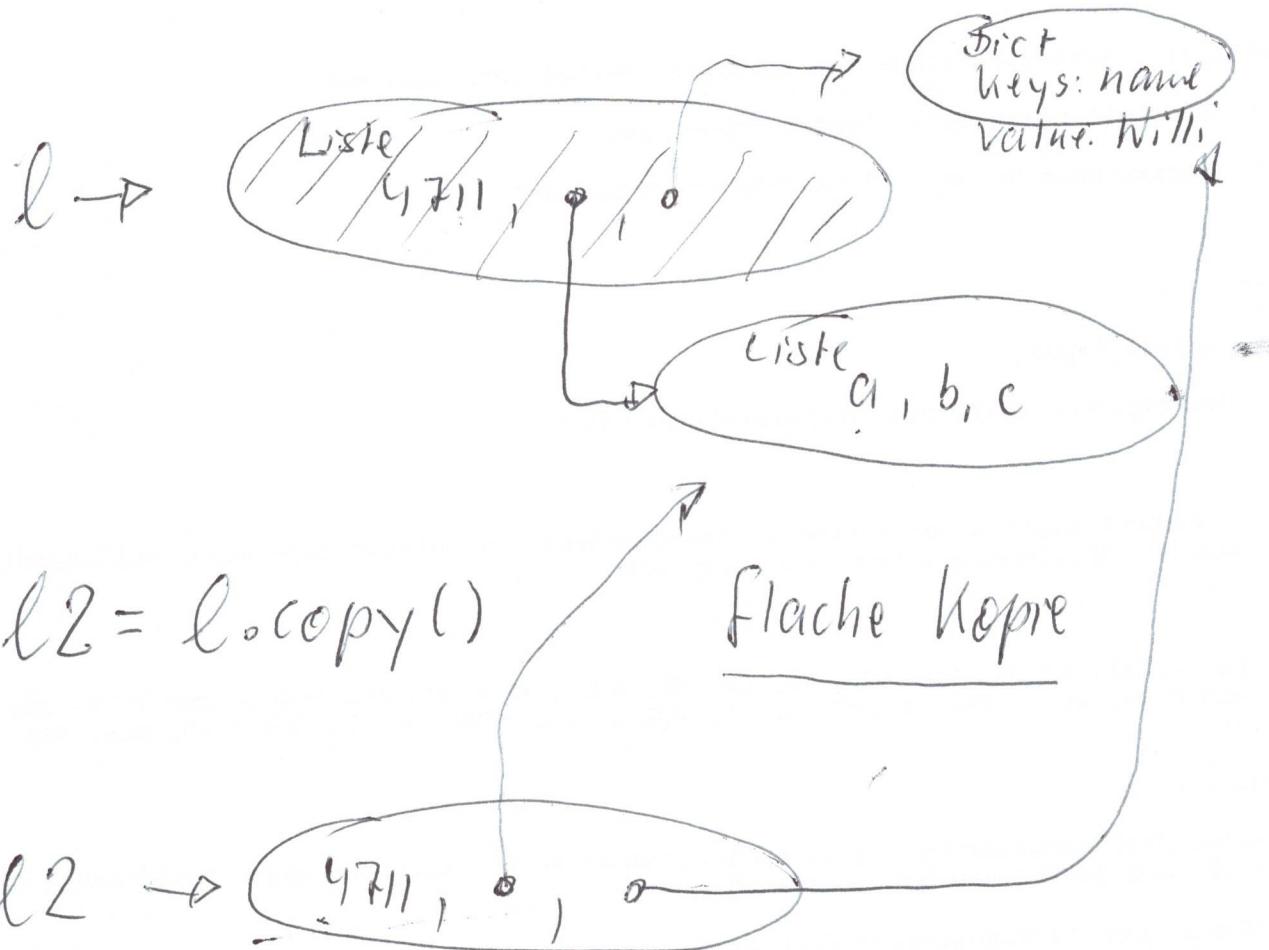
`b=a.copy()`

$b = \bar{a}[:]$

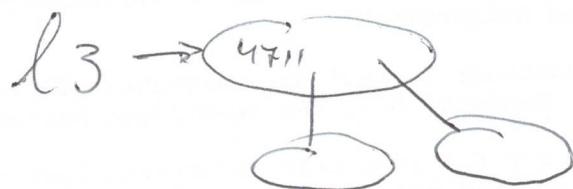
Slice

# Tiefe der Kopie

$l = [4711, ["a", "b", "c"], \{"name": "Willi"\}]$



Tiefe Kopie —  $l_3 = TK(l_2)$



# Slicing Liste/String

$l = [^{\circ} "a", ^{1} "b", ^{2} "c", ^{3} "d", ^{4} "e"]$

$l[von:bis:Schritt] \rightarrow$  neue Liste

Bsp  $l[3:] \rightarrow$  von 3 bis Ende

$l[:3] \rightarrow$  von 0 bis 2

$l[1:3] \rightarrow$  von 1 bis 2

$l[::2] \rightarrow$  von 0 bis Ende  
jeden 2.

$l[3::2] \rightarrow$  von 3, jeden 2. bis Ende

## List Comprehension

Liste direkt mit "gefilterten" Werten erzeugen

daten = [n, n, n]  
daten = [{"Name": "Willi"}, {"Name": "Karl"}, {"Name": "Heinz"}, {"Name": "Hans"}]

"Filtere alle Personen aus HH heraus"

~~for k in~~

namen = [n["Name"] for n in daten if n["Ort"] != "HH"]  
Liste =

[Willi, Karl]

$\Leftrightarrow$

$n = []$

for n in daten

if n["Ort"] != "HH"

n.append(n["Name"])

# Iterator, Iterable, Generator

Iterable → durch das Objekt kann Schrittweise durchgegangen werden  
next() nächste Element  
iter(.) erzeugt ein Iterable

string = "Hello, world"

i\_string = iter(string)

① next(i\_string) → H e l l o , w r l d

Iterator → Objekt das als Iterable genutzt werden kann

class X\_Iterator: str\_iterator  
def \_\_iter\_\_(self):  
 =  
 def \_\_next\_\_(self):  
 =  
 return x

# Generator

Funktion, die bei jedem Aufruf ein Folgewert zurückgibt

kein return    sondern yield

=  
Generator erzeugen

def char\_gen(v, b):

====

while c <= b:

    yield(c)

    c += 1

---

# Dict

{key : value} / dict(...)

↑      ↑  
string    bel  
/ hash-val

d = { k1:v, k2:v, }

d[k1] → v      }      in try/except ~ block  
d[k3] → ↴      }      order  
d.get(k, default)      v ↴      k3:4711

## Dict.-Comprehension

$d = \{ \text{k: v for } x \text{ in L} \}$

$= \{ \text{num: num * num} \text{ for num in range(1, 11)} \}$

The diagram illustrates the mapping from the key 'k' to the value 'num' in the dictionary comprehension. A horizontal line connects the two parts of the equation. From the left side of this line, a vertical arrow points down to the 'k' in 'num'. From the right side, another vertical arrow points down to the 'num' in 'range(1, 11)'.

---

aus List-Comprehension

$d = \text{dict}([n, n * n \text{ for } n \text{ in range(1, 11)}])$   
[ $(1, 1), (2, 4), (3, 9), \dots \]$

---

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

$v = [1, 4, 9, 16, 25]$

$d = \text{dict}(\text{zip}(k, v))$

# Aufgabe

# Dikt als ASCII-Tabelle

$a_{15011-\text{tab}} = \{$

- 'a': 97,
- 'D': xx,
- '#': xx,
- .
- .
- .

}

Von  $\text{ASC}(32)$  -  $\text{ASC}(127)$

# Reguläre Ausdrücke

Text  $\xleftarrow{\text{gesucht}}$  Muster  $\rightarrow$  Matches

Reg. Ausdruck

Sonderbedeutung eines Zeichens

das Zeichen selbst

Bsp • „alles, beliebig, jedes“

b „kleines B“

\. „der Punkt“

\b „ein Wortanfang“

# Reguläre Ausdrücke

## Quantifier

$\forall \exists^*$  o,n alles beliebig oft was links  
beschrieben wird

• \* "egal was, egal wieviel"



1|2|3|4  $\leftarrow$  •\*

1|2|3|4  $\leftarrow$  a\*



Neutra

= a Ø mal

+ 1,n

a+ ist nie in 1234

? 1 oder  $\emptyset$

{min, max}

# OOP

python ist oop

4 → Objekt / Inst

None → Objekt / NouerType

def f() → Objekt / FunktionsTyp

Klasse: Kombi von Daten | Funktion

|  
Bauplan      Kapselung  
                Information Hiding } Konvention

Schnittstelle / API

Objekt  
| Attribut  
|  
Attribut

↳ Methoden (meist)  
↳ Attribute (selten)

API-Aufruf

Objektname.methode()

# Vererbung

file-tools



has a  
Backup }      hat einen String  
              }      hat einen Finder

~~Vererbung~~



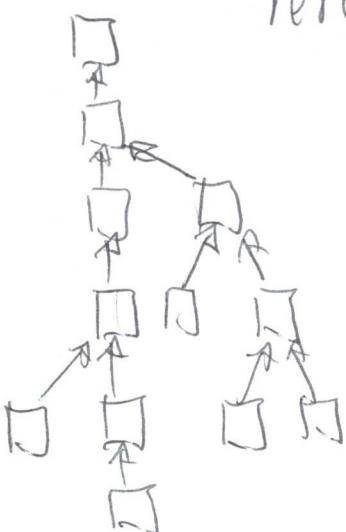
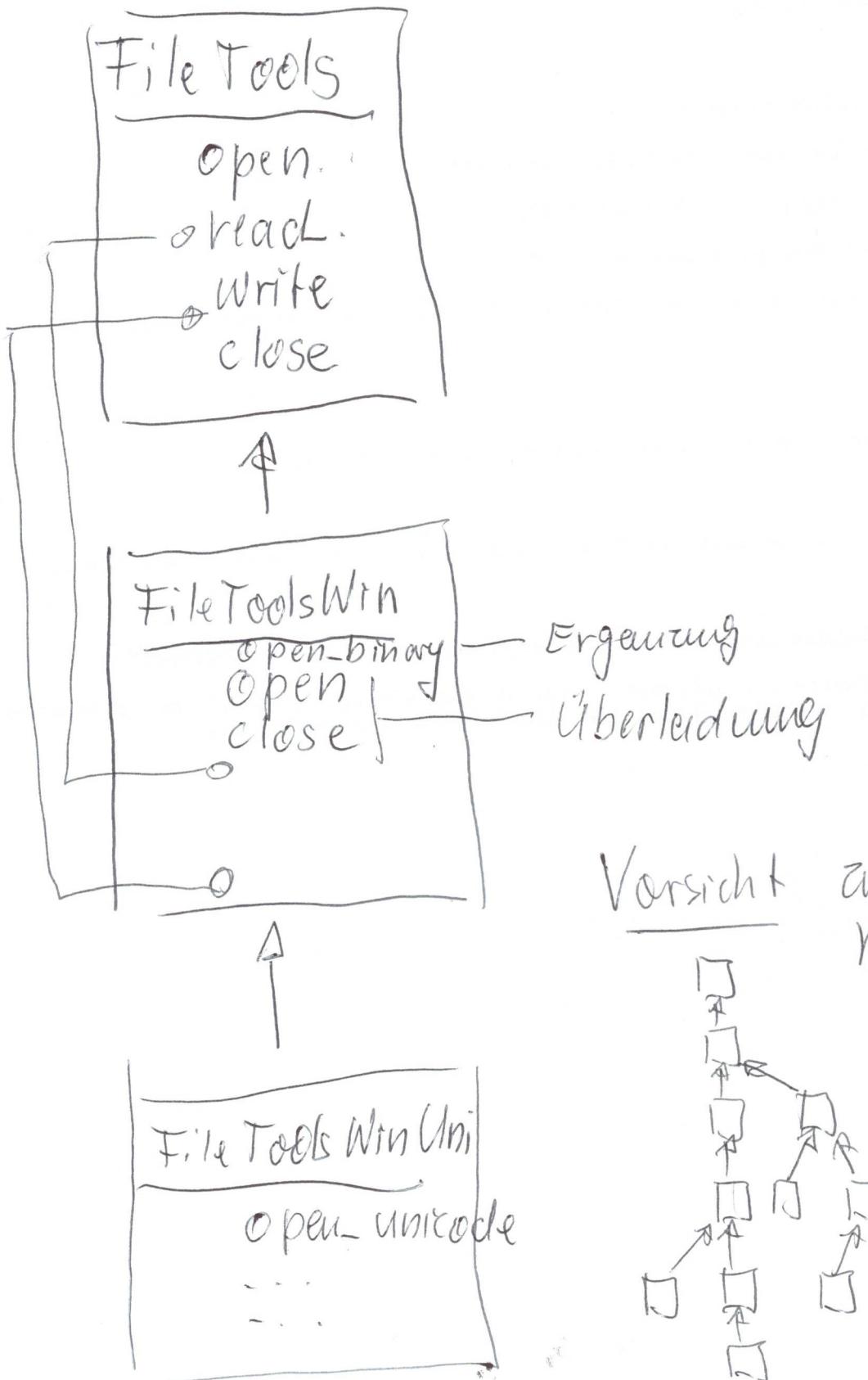
Idee: Don't repeat yourself  
Was schon da ist wird  
genutzt, ggf ergänzt

file-finder-ng is a ist eine Abhängigkeit  
von file-finder

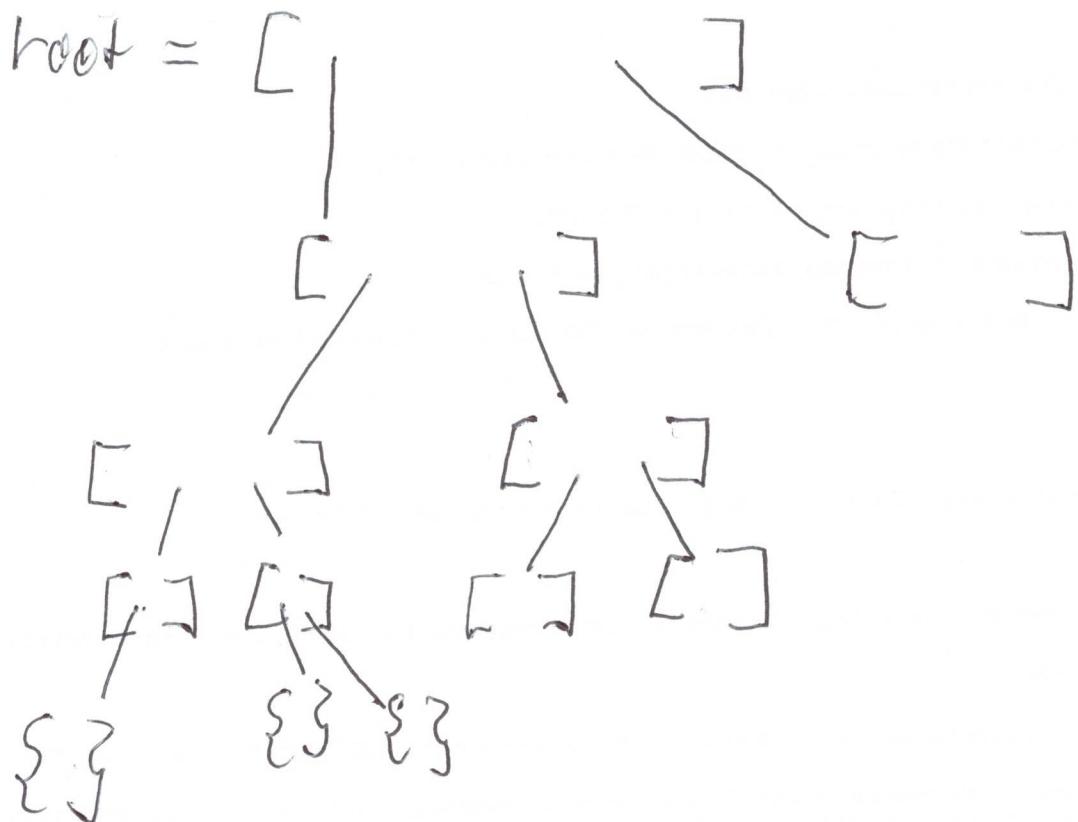
## Vererbung

ff.ng kann alles was ff kann (Erbe)  
Und noch mehr (Was hier programmiert  
wurde)

# Vererbung



# Datenstruktur



1. DS initialisieren / aufbauen

2. Suchen

3. Löschen

4. Einfügen

5. Tausch

Tree-API

Tree

self.root = []

def search(self, kriterium)

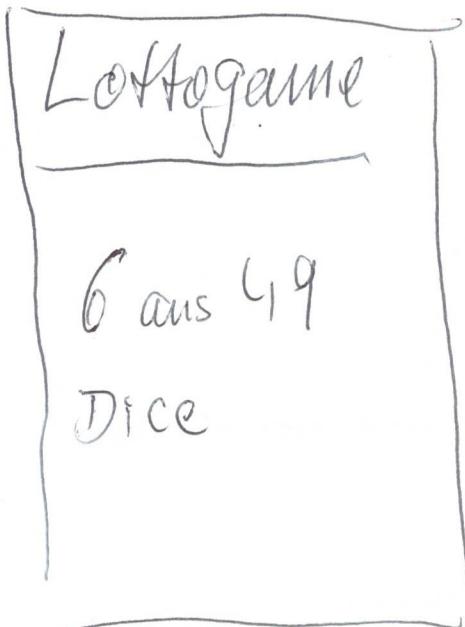
def delete(self, kriterium)

def insert(self, daten, krit)

def swap(self, ...)

f = Tree(), insert([Werte])

# Aufgabe



Ziehung-durchföhren() \*<sup>1</sup>  
Tipp-auswerten(Tipp) \*<sup>2</sup>  
Statistik-lesen() \*<sup>2</sup>

\*<sup>1</sup> 6 Ziehungszeichen erzeugen  
merken

\*<sup>2</sup> Vergleich 6 Zahlen mit der  
letzten Ziehung  
im Statistik Teil Treffer merken

\*<sup>3</sup> Liefert Liste mit den Treffern  
der Ziehungen

Lottogame()

Q 10.000 x einen Tipp mit einer neuen Ziehung  
Auswerten 3,4,5,6 er

# Funktionen

```
def name(parameter):  
    _____ Local Variables  
    _____ Aweisungen  
    [return Ergebnis] | optional
```

Stichwort Badewanne      ↗ Parameter  
                                    ↘ return

globvar = 4711

```
def machWas(x):  
    erg = globvar * x  
    return erg
```

print(machWas(5)) → 5 \* globvar  
                                    5 \* 4711

# Funktion

class X:

→ def machwas(self, ...):

≡

return optional

def fuwas(self, ...):

self.machwas(...)

X = X()

Objektanlage

X.machWas()

X.fuwas()

# Parameter

def name( Parameter ):

    pflichtparameter z.B. Wert      bel. Viele

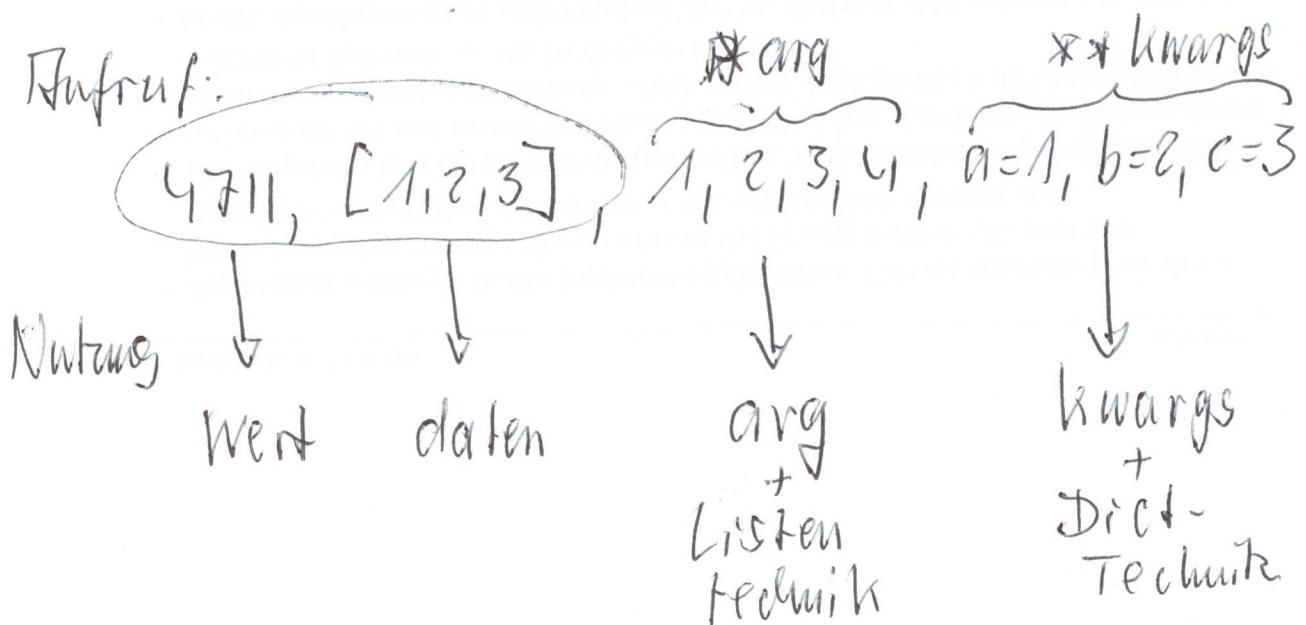
    optionale Parameter z.B. daten=[ ]      bel. Viel  
  ↑  
  Default  
  value

Parameterlist                                        \*args      1\*

Parameterdictionary                                \*\*kwargs    1\*

Reihenfolge: Pflicht, Optional, PListe, PDict.

Rufruf:



# Return

def x():

return Einzelwert

Liste

Dict

Int

Float

String

Funktion  $\rightarrow$  lambda-Fkt

Referenz!

e = x()

def x():

$[-1]$   
↓

return 1, 2, 3  $\Leftrightarrow$  Tupel / Liste

e = x()  $\rightarrow$  e enthält (1, 2, 3)

a, b, c = x()

a  $\rightarrow$  1, b  $\rightarrow$  2, c  $\rightarrow$  3

a, \*c = x()

a  $\rightarrow$  1, c  $\leftarrow$  (2, 3)

a = x()[-1]

a  $\rightarrow$  3

a = x()[1:]

a  $\rightarrow$  [2, 3]

## Liste vs Tupel

Liste = [ ] } 90%, gleiche Funktionalität

(tuple()) ↗ list() Veränderbar nach Anlage

Tupel = ( ) Unveränderbar

# Parameter + Funktionen

```
def auswerte(daten, fkt=None):  
    erg = None  
    if fkt:  
        erg = fkt(daten)  
    else:  
        erg = len(daten)  
    return erg
```

$l = \text{auswerte}([1, 2, 3, 4])$   
 $\underline{l \rightarrow 4}$

$\underline{\text{def } x(l):}$   
 $\underline{\text{return str}(l)}$

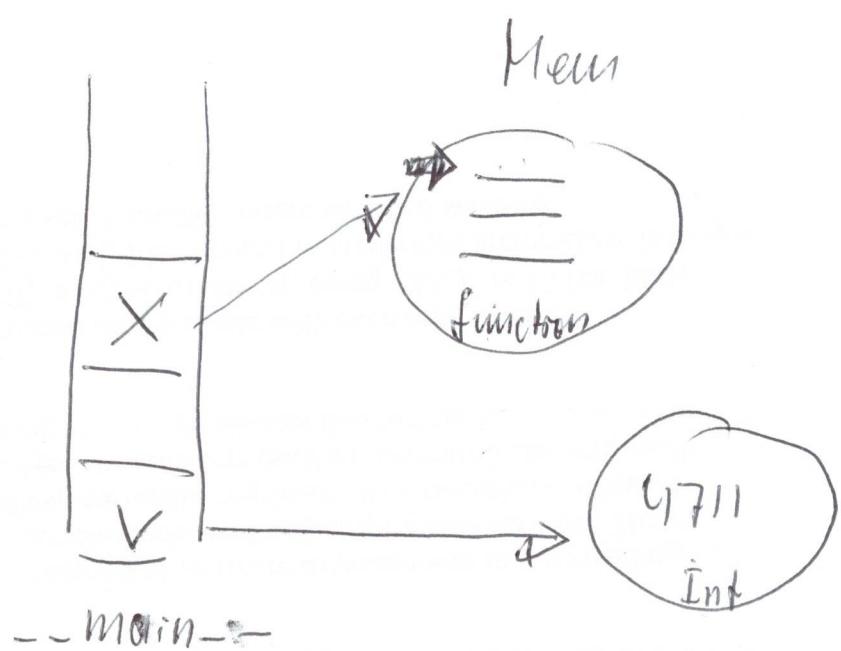
$l = \text{auswerte}([1, 2, 3, 4], x)$   
 $\underline{l \rightarrow "[1, 2, 3, 4]"}$

# Parameter & Funktionen

def x():

=

V = 4771



print(V) → 4771

print(X) → function x at 0x47110815

print(X()) → was and now x als  
return hat



```
def X(w):  
    print("Hallo", w)
```



```
def machwas(wert, f=None):  
    if f:           4711  
        f(wert)    0x47110815 ausfuehren mit '4711'  
    else  
        print(wert)
```

f ist alias für X

Machwas(4711) → Machwas(4711, None) ✓

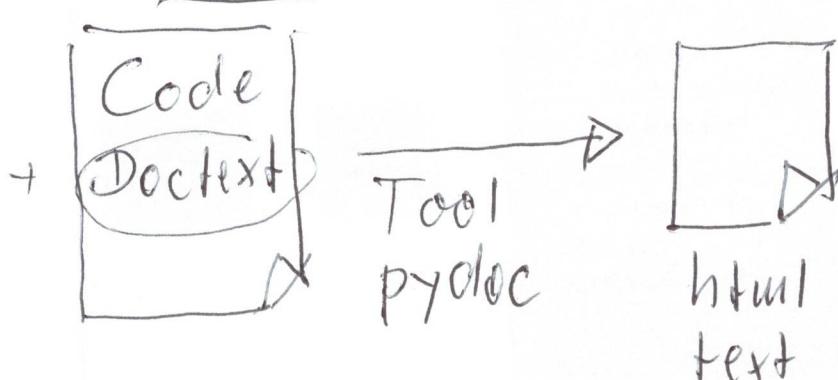
Machwas(4712, X) → machwas(4711, 0x47110815)

# Dokumentation

# Einzeiliger Kommentar in Code

---

Russendok.



"doc string"

→	▼▼▼		Mehrzeiliger String
→	:	:	Standalone
	:	:	Spez. Formativwörter
	A A A	""""	Darstellungs Konvention
stellt hinter	class X --		
hinter	def fmaue ...		

# Qualität

Programm ist testbar

Monolith 1 - 1000 / nicht testbar



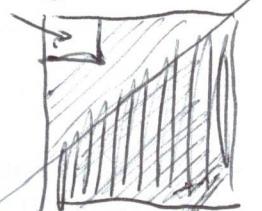
Funktionalen  
Methoden  
→ klein



Tests

Test bed<sup>1</sup>

Tests



- monothematisch
- "Badewanne"!



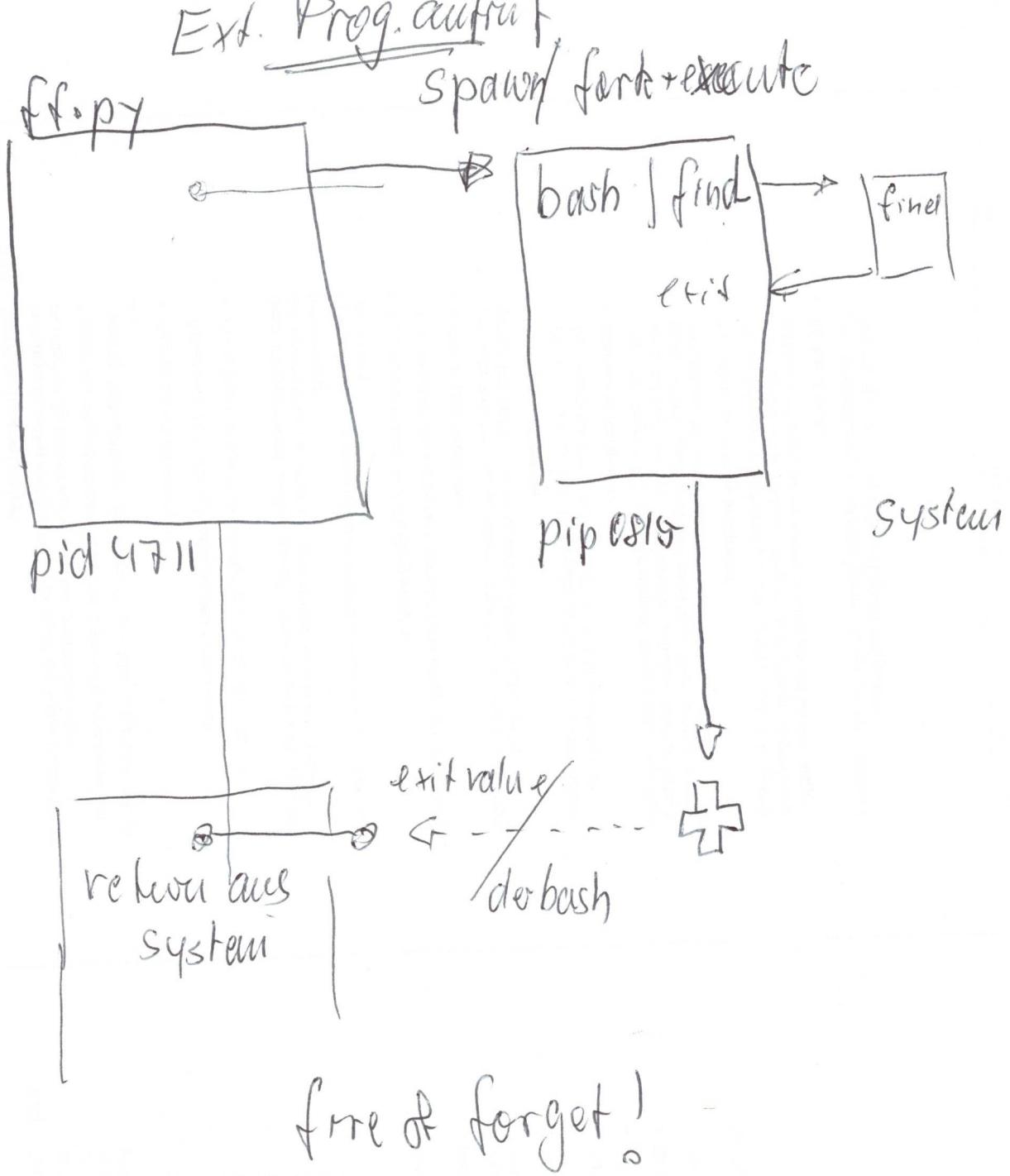
Unit test



Framework

unit test

py test



# Ex. Prog. Nutruit

