

# Python

~ 1995er

Guido v. Rossum

2.7.x alt!!!

3.x neu!!.

ab > 3.6

aktuell 3.9 - 3.10



Python 3.8

pip ← library → conda  
fetcher

Linux: python → 2.7 „module“

pip → 2.7

---

python3 | → 3x  
pip3

# Struktur

Linux

```
#!/usr/bin/env python3  
print("Hello, world")
```

```
def mach_was():  
    2 oder  
    4 Leerzeichen  
    nie TAB
```

} block-

indentation

a)

python3 hello.py

Win/Linux

PATH

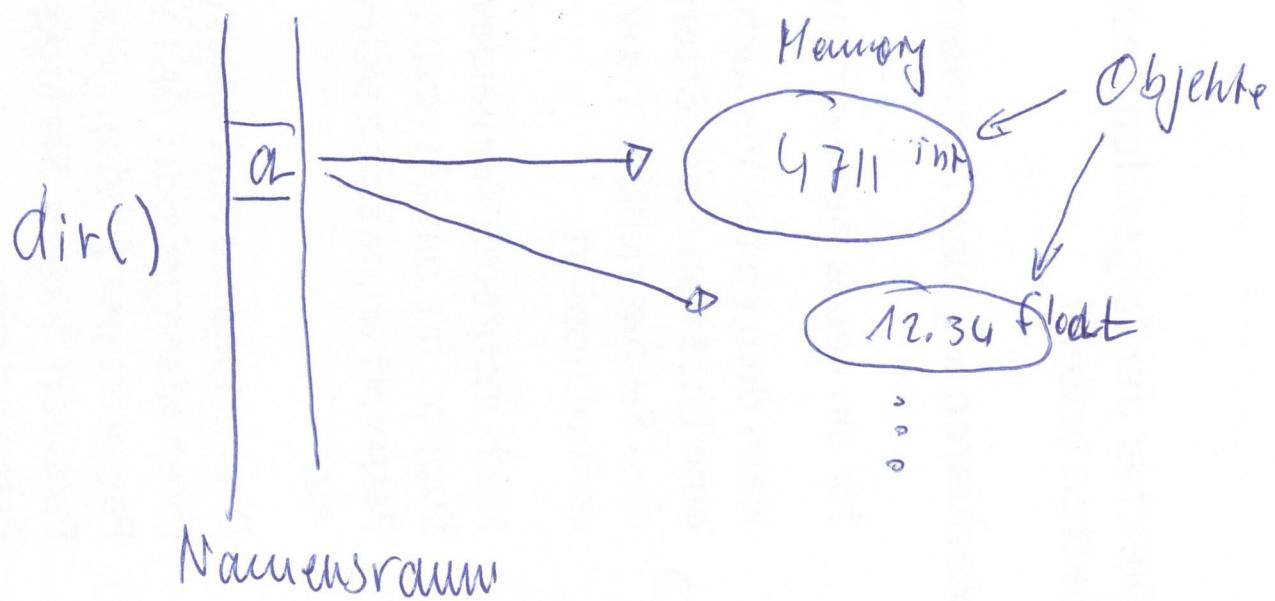
b)

chmod u+x hello.py  
./hello.py

Linux

# Variablen & Datentypen

Variablen Name	Zuweisung ,	Wert ≡ Typ	
a int	=	4711	bei. groß
a float	=	12.34	ungenau!
alltwech typ	→ a str	= "Text" oder 'Text'	bei. 11
	a list	= [ ]	Liste
	a dict	= { }	Dict
	a Klasse	= Name()	Objekt
	a NoneType	= <u>None</u>	„Nix“



# Kontrollstrukturen

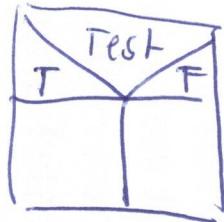
if Test:

H    T

else:

F    F

optional

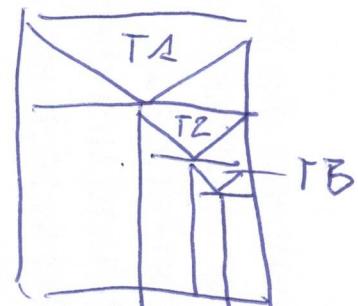


if T1: True

elif T2:

elif T3:

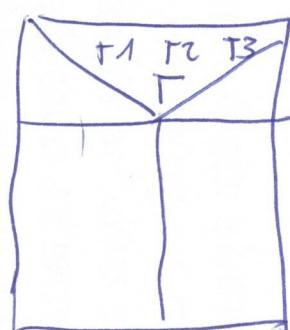
else:



if T1 and T2 and T3:

H  
else:  
H

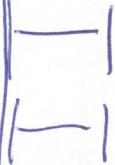
Logik: not  
and  
or



Vereinig  
UND  
ODER

# Kontrollstruktur

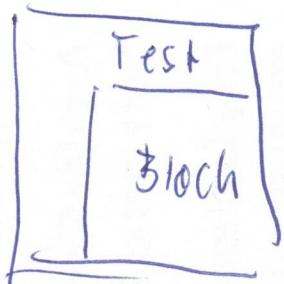
While - Test:



i =  $\emptyset$

while i < 10:

```
    print(i)
    i = i + 1
```



break - Sprung rans

continue - Nächste Aus-  
führung

for  
var in "Liste von Objekten":

```
    H
    H
```

for i in range(0, 10):

```
    print(i)
```

# Übung Calculator for integers „cfi.py“

Eingabe 1 :	Zahl	input/if
Eingabe 2 :	Zahl	input/if
Eingabe 3 :	„OP-Code“ add mul div sub	input/if print

→ Ausgabe

5	5	5	5
6	6	6	6
add	sub	mul	div
$\frac{11}{+}$	$\frac{-1}{-}$	$\frac{30}{*}$	$\frac{0}{}$
			(Rest 5)

Granzahl div //  
+ Modulo %

Wenn keine Zahlen → Meldung dan das  
Ausma ist  
Wenn kein Opcode → Meldung ---

# Strings



Zeichensatz: utf-8

Text      "Hello, world"

s = "Hello, world"

len(s) → 12

s[0] → H

s[1] → o

s[12] → Exception

$$s[0] - s[\text{len}(s)-1]$$

↑ s[-1]

# Strings

in - Operator

if "Hello" in s: ✓

if "w" in s: ✓

if "x" in s: X

---

Methoden:

s.methodenname()

X = s.upper()

→ HELLO, WORLD

X = s.substr(0, 5)

X = s.replace("U", "#")

→ Hello, # world

X = s.split(",")

→ ["Hello", "world"]

# Strings

`"-" * 80` → `"-----" +`  
                        4              80              →

`"1" * 80` → `"11111.....1111"`

`"[" * 80` → `"[" + "`

`"Hallo" * 10` → `"HalloHallo.....Hallo"`

---

Templates z.B. für print

`t = "...{}...{}..."` Template

`f.format(var1, var2)` → `"...***...***..."`

`var1 = 4711`

`var2 = "Willi"`

`x1 = "Hallo {} , du hast {}".format(var2, var1)`

`x2 = f"Hello {}, du hast {}"`

---

# Strings

## "Slicing"

$s = "Hello, world"$

← 0      12      →  
-1 = 11

$s[4:6]$   
← "o,"

$s[-3:]$   
← "rld"

$s[7:]$   
← world

$s[:5]$   
← Hello

$s[:]$   
↑ Kopie von s

$s[::2]$  ↑ rufaus    ↑ Ende    ↑ Step  
jede 2. Buchstabe

← "el, wrd"

$$a=4$$

$$e=7$$

-  $s[a:e] \doteq s[4:7] \rightarrow "o,"$

$s[:-3] \rightarrow "Hello, wo"$

# Übung

Benutzerzugabe solange bis "exit"

Benutzer gibt IP-Adresse an

input

Wenn Port mit angegeben, diesen auslesen

Dann testen ob Loopback 127.0.0.1

Adresse in Triplets zerlegen

Jedes Triplet testen ob gültig 0-255

Bereit ausgeben

while

eingabe != exit

\* Port auslesen - Ausgabe

Loopback ? - Ausgabe

Triplet 1 gültig ?  
--- 2 ---  
--- 3 ---  
--- 4 --- } - Ausgabe

\*

|xxx,xxx,xxx,xxx| YYY | ... | 192.148.1.12 |

# Wiederholung

→ Indentation → Wenn ein Block anliegt  
     $\Leftrightarrow$  immer nach : in der  
    Vorhergehenden Zeile

for n in 1, 2, 3, 4:  
 H  
 H      Breite: 2 / 4  
if x == 99:  
 H      Nicht \Tab, es sei denn  
 H      die IDE übersetzt in Leerzeichen

while x < 100:  
 H

python 3.x  
 $x > 6$

python 2.7.x  
nur noch Pflege

Pip / python.org

vs.

Conda / anaconda

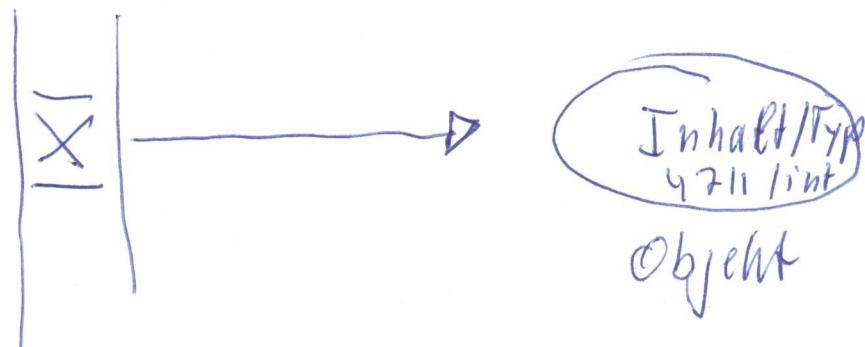
# Kommentar

#! Hashbang  $\hookrightarrow$  Shell  $\rightarrow$  startet Programm  
aufrufe

# Wiederholung

Variable : Name der auf Spercher zeigt  
der ein Typ hat

$$x = 4711$$



`type(x)` → Typangabe

`isinstance(x, int)` → Testmöglichkeit  
True/False

`if isinstance(x, int):`

Berechnung()

`else:`

`raise RuntimeError("kein Int")`

oder

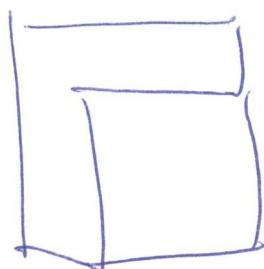
Fehlerbehandlung()

# Wiederholung

Skalar Ein	int ()	
	str ()	" ... " / ' ... '
	float ()	
Datum	None → NoneType()	≡ False
	bool ()	True / False
	complex ()	



if Test:  
|  
[ else: ] [ elif Test: ]



while Test:  
|  
for n in Liste:  
|

break / continue  
Schleife endet      nächster Durchlauf

# Listen

Liste, Tupel, Menge

↔ Ansammlung von Objekten

Liste: dynamisch, veränderbar, doppelte

Tupel: ✓, ✓, ✓ "statische Liste"

Menge: ✓, ✓, ✓

Liste      list()      [ ... ]

Tupel      tuple()      ( )

Menge      set()      { }

$l = [1, "a", ["x", "y"], \text{True}]$

Index 0 1 2 3  
 $\stackrel{\leftarrow}{\text{len}} = \overbrace{4}^{\stackrel{\rightarrow}{= \text{len}(l)-1}}$

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

Slice  $l[:2], l[::2], l[2:4]$

Methoden z.B.  $l.append(\text{wert})$

Operatoren + \*

# Listen

direkte Aulage  $\rightarrow l = [1, 2, 3, 4, 5]$

aus Funktion  $l = \text{list-generator}(10)$

aus Range  $\rightarrow l = \text{list(range}(1, 6))$   
 $\text{list}(1, 2, 3, 4, 5)$

aus Zerlegung,  $l = "xx;yy;zz".\text{split}(";) \leftarrow ["xx", "yy", "zz"]$

Mutatoroperation: join  
hier: ":".join(l)

aus z.B Dateien,  
Datenbankabfragen,  $f = \text{open}(\text{Dateiname}, "r")$   
...

$l = f.\text{readlines}()$   
oder

$l = f.\text{read}().\text{split}("\n")$   
oder

$l = f.\text{read}(), \text{splitlines}()$

# Listentrick

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

$*l \rightarrow$  Liste wird in Einzelwerte  
aufgelöst

z.B.  $\text{print}(l) \rightarrow [1, 2, 3, 4]$   
 $\text{print}(*l) \rightarrow 1 2 3 4$

$a, *X = l \rightarrow a : 1$   
 $X : [2, 3, 4]$

$a, b, c, d = l \rightarrow a : 1$   
 $b : 2$   
 $c : 3$   
 $d : 4$   
~~e~~

$$\begin{array}{l} a = 1 \\ b = 2 \end{array}$$

$$a, b = b, a$$

$$\begin{array}{l} a = 2 \\ b = 1 \end{array}$$

# Aufruf von Funktionen / Methoden

// einfache Funktion"

z.B. len(Objekt)

name( Parameter )



len(l)

Methoden mit Rückgabewert

Objekt.split(":")

Objekt.method()

bei einigen Methoden wird das Objekt oder  
eine Kopie zurück d.h. man  
kann Methoden verketten

lang: s = "Blau:zuBlau"

(str) s1 = s.replace("u", "")

(list) sl = s1.split(":")

kurz      sl = s.replace("u", "").split(":")  
verketten      ↓————→

# Lottoerie Teil 1

Benutzer gibt Zeile ein input

→ Zeile soll haben:

6 Zahlen zwischen 1-49 ohne Doppelte

→ Wenn das nicht der Fall ist

⇒ Meldung und Ende mit exit

---

→ Liste Tipp = [int, int, int, int, int, int]

---

Ziehung → 1-49 random  
6 Stich  
Kein Doppelte

Treffer

treffer = list(set(tipp)) & intersection(set(ziehung))

print(treffer)

print(f"Treffer: {treffer}")

# Dictionary

## key - value - Store

```
dict() d = {"k1:v", "k2:v", "k3:v"}  
d
```

Zugriff:  $d["k_1"]$   V  
     ↗ wenn  $k_1$   
    ↗ nicht erreicht

Sicherer Zugriff  $d.\text{get}("k_1", \text{None})$

Wertzuweisung/  
K-V-Anlage

# Dictionary

```
d = { "Header": ["Datum", "Last", "High", "Volume"],  
      "Values": [  
          [ ]  
          [ ]  
          [ ]  
      ]}  
    }
```

---

```
apple_stock = [  
    { Datum: 21.1.91,  
      Last : 395,0  
      High: 400,0  
      Volume: 400000  
    },  
    { Datum:  
      Last  
      High  
      "Volume": 5000  
    }  
    ...  
]
```

```
apple_stock[-1]["Volume"] → 5000
```

# Aufgabe

data = { }



data = { "07/07/2021":

{  
  low  
  high  
  close  
  volume  
},

"08/07/2021":

{

---  
---  
---

}

}

:

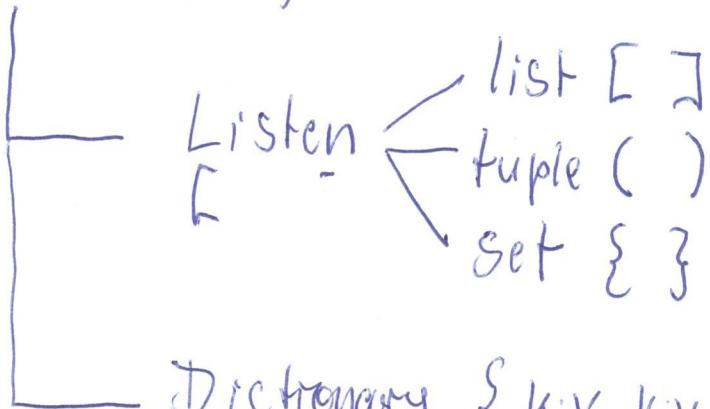


X|s → CSV | vergleichen

# Niederholung

Daten → Einfache, "Skalare"

→ Strukturen, "Vektor"



beliebig kombinierbar

Quelle

DB  
Datei  
Format  
Messgerät

{ json  
CSV  
XML  
Text }

Prog

read → v →  
↳ interne  
Format  
Listen + Dict  
+ Skalare

E

v

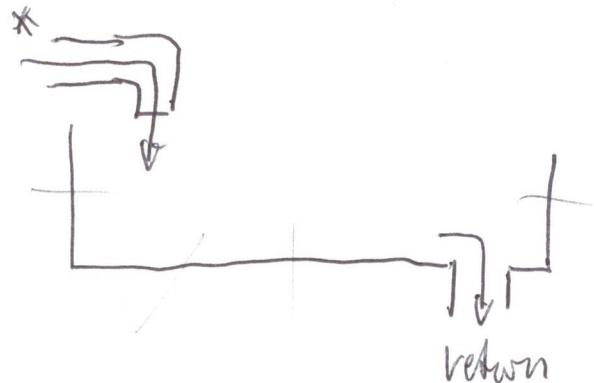
A

→ hilfreich: Pandas  
openpyxl

# Funktionen

Def� mach-was (\*): → ohne Parameter  
[ ] Körper mit Parameter  
[ return ] → mit Rückgabe  
def .. ohne -

Datenübergabe: "Badewannenprinzip"



Kein Aufruf vor der Definition!

Mehrfachdefinition ⇒ der letzte gewinnt!

# Function

return

```
def mach_was():
    result = random.sample(range(1,50), 6)
    return result
```

mach_was() ✓ ? print(result) ↴	<del>X</del> = mach_was() ✓ print(X) ↴
-----------------------------------	---

---

# Funktion

## Parameter

- def f()  
→ Ø-Parameter
- def f(a)  
→ 1 Pflicht
- def f(a, b=Ø)  
→ 1 Pflicht  
1 optional mit Default
- def f(b=Ø)  
→ Ø Parameter  
1 optionaler
- def f(a, b=Ø, c=None) → 1 Pflicht  
2 Optionale  
p → 0

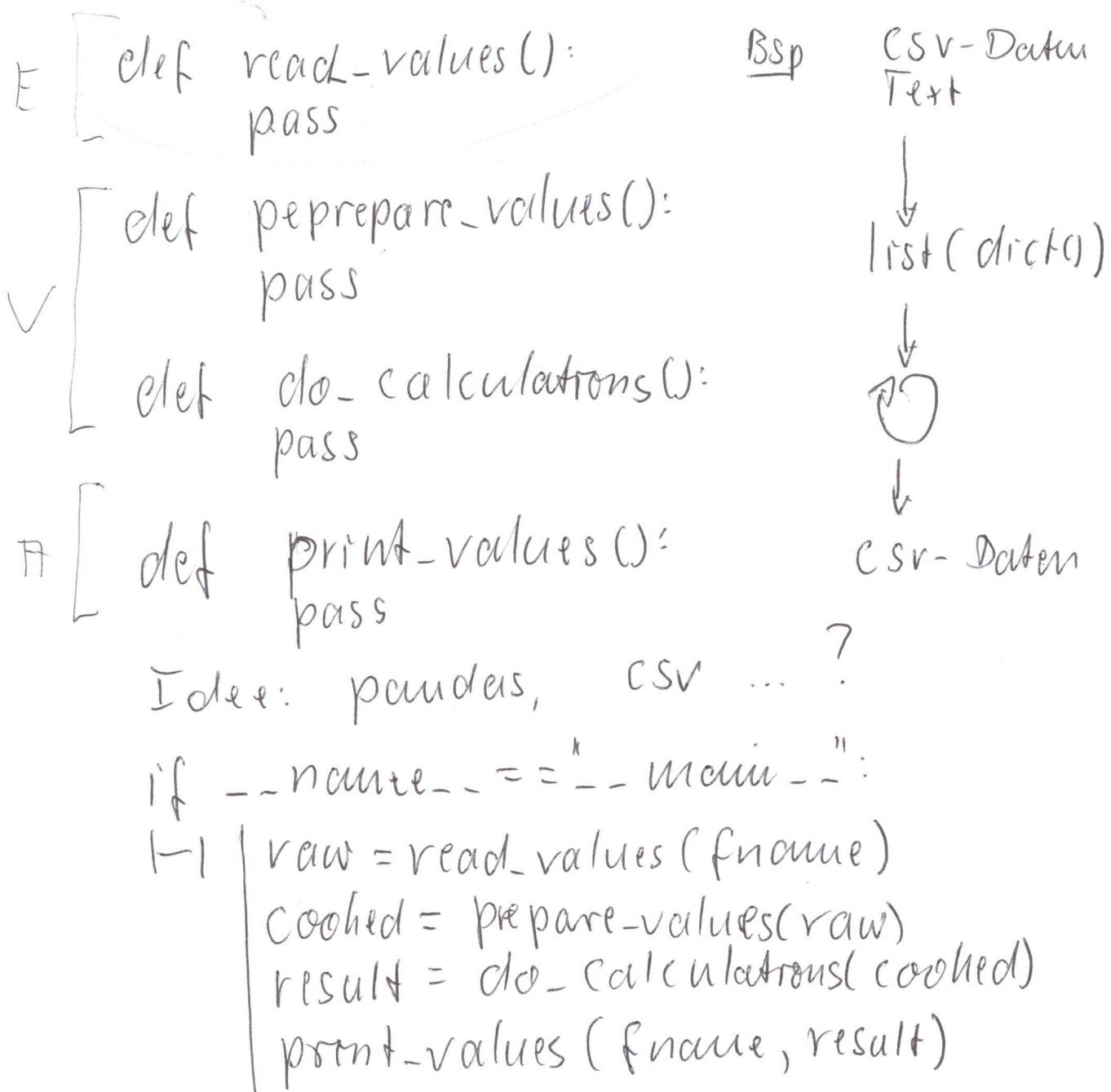
- 
- def f(\*args) → bel. viele Werte,  $\cong$  liste
- def f(\*\*kwargs) → bel. viele k-v Paare  $\cong$  dict
-

# Programm entwurf

Grob: E - V - A

Quelldaten - internes - Zieldaten  
Format

Ferner: Funktionen?!? Welche, was, womit  
→ gibt es das als Lib?



# Aufgabe

0 1 2 3 ...  
archiv [out] datei1 datei2 ...

für alle Dateien:

- Lese Datei als Text
- Sammle in Variablen z.B Liste von dict

sys.argv[0]  
sys.argv[1]  
sys.argv[2:]

Ausgabe nach out

[ {Name: }, {Name:  
Inhalt}, {Name:  
Inhalt}, ... ]

Dateiname - Anzahl Zeilen | Anzahl Buchstaben

Inhalt =  
=====

Dateiname -

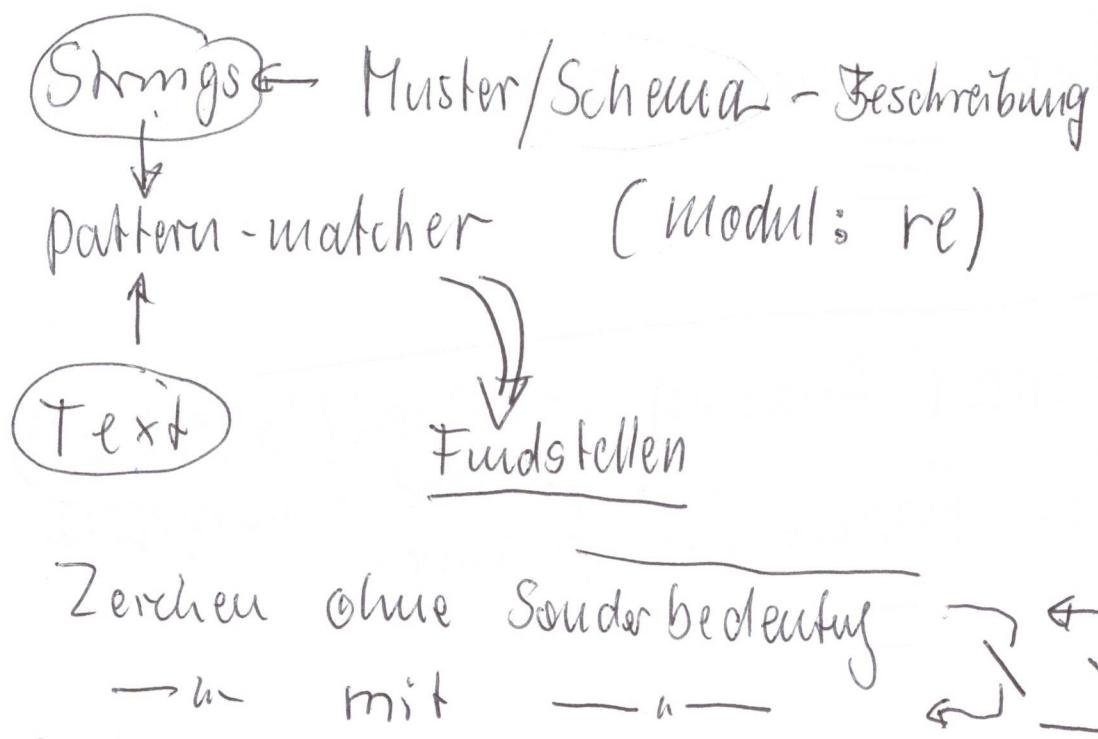
Inhalt =  
=====

daten = read(sys.argv[2:])

write(sys.argv[1], daten)

python archiv.py 2021-11-archiv \*.py

# Reguläre Ausdrücke



Quantifier: „wie oft“

$$\begin{matrix} * & 0 - \infty & \longrightarrow & a * \\ + & 1 - \infty & & \swarrow \\ & & & a, aa, a \dots a \end{matrix}$$

9 0, 1

$\{v, b\} \quad v = b - Ma$

Zeichengruppen: ( )

Zerden kllassen [...]

Risher

## \b Wortgruppe

↑ \$

Wortgrau  
Aufgang  
Ende