Statement ("Anweisung"): One logical executable line of code, e.g. `1+1`

Expression ("Ausdruck"): Code snipped that yields a certain value when evaluated, e.g. `2*5` yields `10`

Argument ("Parameter"): A value that is passed to a command, e.g. `print("hi")`

String ("Zeichenkette"): A value consisting of charakters, e.g. `"Hello"`

Literal: a hard-coded value, e.g. `5` or `"Hello"`

Variables: Referenzes to values in memory

Operator precedence:

Paranthesis	(...)	(2+2)*3 == 12
Power	**	2**4 == 16
Multiplication	*	2 * 4 == 8
Division	/ //	13/7==1,85 13//7==1
Modulo	%	13%7 == 6
Addition	+	13+7 == 20
Substraction	-	13-7 == 6
Comparision	== > >= is != < <= is not	
NOT	not	
AND	and	
OR	or	

```
# one-line comment
''' ... ''' multi-line comment
```

Python everything is an object.

Immutable: bool (True,False), int(-7, 42), float(-7.2, 42.0), string('hi', "7"), tuple ((42,42.0))

Mutable: list ([-7,'hi',42.0]), set, dictionary ({"first":1, "second":'2'})

String:
- `str1 + str2` → connect strings
- `n * str1` → n-times str 1
- `str[n]` → (n-1) Element (zero-based)
- `str[start,end,step]` → Elements from start to (end-1) in step-size
- `Len('Hello') = 5` → #Elements in string
- `'x' in str1` → is Element x in string
- `str1.split(delimiter)` → split the string at the delimiter. (list) (default = ' ')
- `chr(N)` → `chr(65)='A'` ⇒ ASCII

Bool:
- 0 and '' (empty string) → False
- all others → True
- True or <something> → <something> wird nicht ausgewertet.
- and returns first False value
- or returns first True value
  - ↳ otherwise returns last value
- and, or are no bool! → return actual value they're comparing
  - ↳ cast to bool

and-or-Trick: <expr> and <TruePart> or <FalsePart>

`0 < x < 10 == (0<x) and (x<10)`

None:
- No Value
- return → returns None
  - ↳ kann weggelassen werden

List:
- `lst1[n]` → Element (n-1) (zero-based)
- `lst1[-n]` → n-tes Element from the back
- `lst1[start:end:step]` → Elements from start to (end-1) in step-size
- `lst1.append(x)` → add x at the end of the string
- `lst1.remove(x)` → remove x from the list
- `del(lst1[x])` → remove Element at index x of the list
- `len(lst1)` → #Elements in the list
- `lst1.index(x)` → returns the index of the Element x in the list.
- `lst1=[[s,t],[u,v],...]` → nested-list → list of lists
  - ↳ `lst1[row][column]`
- `list(range(start, end, step))`

`[<expression> for <item> in <list> if <cond.>]`
↳ List comprehension

`import copy` → way to copy lists
`<new_list> = copy.deepcopy(some_list)`

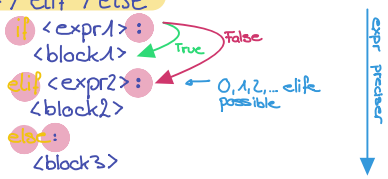
`var1, var2,...,vark = expr.` → all K Variables equal expr.
`var1, var2,... = expr1, expr2, ...`
↳ `var1=expr1 ; var2=expr2 ; ...`

`print([expression [, expression [, ....]]]`
`input(<something>)`
→ [...] optional , < > necessary

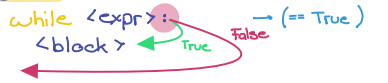
tuples:
`x,y = 2,3` (pack) → `z=(x,y)=(2,3)` (unpack) → `x,y=z`
`z[1] = 10` does not work
`len(z) = 2` → #Elements in tuple

float:
`z = 42.0` `x = 42.`
Limited precision ⇒ `z == x` → doesn't work
↳ `Math.isclose(z,x)`

if / elif / else



while:



infinit-loop → <expr> == True → always True
Continue → go to next iteration (reenters the loop by evaluating expr.)
break → exits current loop immediately

for:
length is known
```
for <items> in <sequence>:
    <block>
```
→ don't change <items> in <block>
nested for-loop → x/y-coordinates
```
for x in rows:
    for y in columns:
        <block>
```

```
for i, item in enumerate(list):
```
i ↳ index, item ↳ value

Functions:
Function header: `def <valid_name>([<param1>[,...]]):`
Function body:
```
    ["""docstring"""]
    <block>
```
(→ pass, to do nothing)

`<some_var> = <valid_name(<p1>, ...)`

`def foo(x, y=10, z=0):`
default values, fill up from right/behind
↳ `var = foo(5)` ; `var = foo(5,z=1)`

immutable param → creates new object
↳ only valid in scope/function

Mutable param → same object is changed
↳ (list, sequences)

Recursive:
1) base-case and it's solution
2) smaller problem → Assume recursion works
3) Solution of smaller problem, to solve the whole problem

File I/O:
`with open(filename1,'r') as f:`
'r' = read 'w' = write
↳ closes file after block!

Json:
`json.dumps(str1, indent=4, separators=(',', '-->'))`
without → if with like
`json.loads(str2)`

OS: `import os`
`os.getcwd()` → current directory
`os.path.join(<direc>,<subdirec>)` → Compose path
Like→ `os.listdir(<path>)` → Lists files in directory.

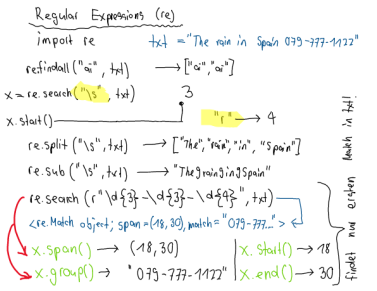
```
try:
    <block>   ← code that can fail
except SomeError:
    <block>   ← ToDo, if ther is SomeError
finally:
    <block>   ← Allwayse executed
```

Dictionary:
`<my_dict> = {<a_key>:<a_value>, ...}` (mutable, not unique)
↳ immutable and unique
- `my_dict[new_key] = value` → Add new entry.
- `my_dict[existing_key] = value` → Change value of a key
- `del my_dict[existing_key]` → Remove a entry.
- `x = my_dict.pop(ex_key)` → Get and Delete
- `key in my_dict` → Check if key is present
- `x = my_dict.get(key,'n/a')` → Get value of key
- `keys = my_dict.keys()` → same with .values()
- `for key in keys:` → Iterate through dict.

Set:
collection of unique, unordered Elements
`set([9,2,3,2,7,1]) = {9,2,3,7,1}`
Operations: union (|), intersection (&), difference (-), symmetric diff (^)

`str.strip()` → deletes leading and trailing characters
`str.strip("foo")`

Regex:



\d	"\d"	looks for Numbers
\s	"\s"	looks for whitespace
\A	"\AThe"	looks if "The" = x[0],[1][2]
\b	r"\bain"	für jedes r"The\b" am Ende schauen (Wort schauen)
\B	r"\Bain"	nicht am Anfang von Wort r"ain\B" → Ende
\D	"\D"	alles ausser Zahlen ["T","h","e"," ",...]
\S	"\S"	alles ausser whitespace ["T","h","e","r"...]
\w	"\w"	Zahlen+ Buchstaben+ underscore (ex. kein minus "-")
\W	"\W"	keine (Zahlen+ Buchstaben)
\2	"Spain\2"	looks if "Spain" = x[-5][-4][-3][-2][-1]

```
Match = re.search( r'...', str )
Phone = r'(\+41\s|0) \d\d \d{3} \d{4}'
Mail = r' \b[\w.%+-]+@[\w.-]+\.[a-zA-Z]{2,6}\b
```