

Base Types

integer, float, boolean, string, bytes

```
int 783 0 -192 0b010 0o642 0xF3
float 9.23 0.0 -1.7e-6
bool True False
str "One\nTwo"
bytes b"toto\xfe\775"
```

Multiline string:
"X\tY\tZ
1\t2\t3"

hexadecimal octal

immutables

Container Types

ordered sequences, fast index access, repeatable values

```
list [1,5,9] ["x",11,8.9] ["mot"]
tuple (1,5,9) 11,"y",7.4 ("mot",)
```

Non modifiable values (immutables) expression with only commas → tuple

key containers, no a priori order, fast key access, each key is unique

```
dict {"key": "value"} dict(a=3,b=4,k="v")
collection set {"key1", "key2"} {1,9,3,0} set {}
frozenset immutable set empty
```

Identifiers

for variables, functions, modules, classes... names

a...zA...Z_ followed by a...zA...Z_0...9

- diacritics allowed but should be avoided
- language keywords forbidden
- lower/UPPER case discrimination

⊙ a toto x7 y_max BigOne
⊙ 8y and for

Variables assignment

assignment ⇔ binding of a name with a value

- evaluation of right side expression value
- assignment in order with left side names

```
x=1.2+8+sin(y)
a=b=c=0 assignment to same value
y,z,r=9.2,-7.6,0 multiple assignments
a,b=b,a values swap
a,*b=seq unpacking of sequence in
*a,b=seq item and list
x+=3 increment ⇔ x=x+3
x-=2 decrement ⇔ x=x-2
x=None « undefined » constant value
del x remove name x
```

Conversions

type (expression)

can specify integer number base in 2nd parameter

truncate decimal part

rounding to 1 decimal (0 decimal → integer number)

bool(x) False for null x, empty container x, None or False x; True for other x

str(x) → "..." representation string of x for display (cf. formatting on the back)

chr(64) → '@' ord('@') → 64 code → char

repr(x) → "..." literal representation string of x

bytes([72,9,64]) → b'H\t@'

list("abc") → ['a','b','c']

dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}

set(["one","two"]) → {'one','two'}

separator str and sequence of str → assembled str

':'.join(['toto','12','pswd']) → 'toto:12:pswd'

str splitted on whitespaces → list of str

"words with spaces".split() → ['words','with','spaces']

str splitted on separator str → list of str

"1,4,8,2".split(",") → ['1','4','8','2']

sequence of one type → list of another type (via list comprehension)

[int(x) for x in ('1','29','-3')] → [1,29,-3]

Sequence Containers Indexing

for lists, tuples, strings, bytes...

| | | | | | |
|----------------|----|----|----|----|----|
| negative index | -5 | -4 | -3 | -2 | -1 |
| positive index | 0 | 1 | 2 | 3 | 4 |

```
lst=[10,20,30,40,50]
```

| | | | | | | |
|----------------|----|----|----|----|----|---|
| positive slice | 0 | 1 | 2 | 3 | 4 | 5 |
| negative slice | -5 | -4 | -3 | -2 | -1 | |

Access to sub-sequences via lst[start slice:end slice:step]

```
lst[:-1] → [10,20,30,40] lst[::-1] → [50,40,30,20,10] lst[1:3] → [20,30] lst[:3] → [10,20,30]
lst[1:-1] → [20,30,40] lst[::-2] → [50,30,10] lst[-3:-1] → [30,40] lst[3:] → [40,50]
lst[:2] → [10,30,50] lst[:] → [10,20,30,40,50] shallow copy of sequence
```

Missing slice indication → from start / up to end.

On mutable sequences (list), remove with del lst[3:5] and modify with assignment lst[1:4]=[15,25]

Items count

len(lst) → 5

index from 0 (here from 0 to 4)

Individual access to items via lst[index]

```
lst[0] → 10 ⇒ first one lst[1] → 20
lst[-1] → 50 ⇒ last one lst[-2] → 40
```

On mutable sequences (list), remove with del lst[3] and modify with assignment lst[4]=25

Boolean Logic

Comparisons : < > <= >= == != (boolean results)

a and b logical and both simultaneously

a or b logical or one or other or both

pitfall : and and or return value of a or of b (under shortcut evaluation).
⇒ ensure that a and b are booleans.

not a logical not

True False } True and False constants

Statements Blocks

```
parent statement:
statement block 1...
...
parent statement:
statement block 2...
...
next statement after block 1
```

configure editor to insert 4 spaces in place of an indentation tab.

Modules/Names Imports

module true ⇒ file true.py

```
from monmod import nom1,nom2 as fct
→ direct access to names, renaming with as
import monmod → access via monmod.nom1...
```

modules and packages searched in python path (cf sys.path)

statement block executed only if a condition is true

if logical condition: statements block

Can go with several elif, elif... and only one final else. Only the block of first true condition is executed.

```
if age<=18: state="Kid"
elif age>65: state="Retired"
else: state="Active"
```

with a var x:

```
if bool(x)==True: ⇔ if x:
if bool(x)==False: ⇔ if not x:
```

Signaling an error: raise ExcClass(...)

Errors processing: try: normal processing block except Exception as e: error processing block

Exceptions on Errors

```
normal raise X()
error processing
error raise processing
finally block for final processing in all cases.
```

Maths

angles in radians

```
from math import sin,pi...
sin(pi/4) → 0.707...
cos(2*pi/3) → -0.4999...
sqrt(81) → 9.0
log(e**2) → 2.0
ceil(12.5) → 13
floor(12.5) → 12
```

modules math, statistics, random, decimal, fractions, numpy, etc. (cf. doc)

Operators

floating numbers... approximated values

Operators: + - * / // % **

Priority (...)

integer ÷ ÷ remainder

@ → matrix × python3.5+numpy

```
(1+5.3)*2 → 12.6
abs(-3.2) → 3.2
round(3.57,1) → 3.6
pow(4,3) → 64.0
```

usual order of operations

statements block executed as long as condition is true

☞ *beware of inflation*

| | |
|-----------------|------------------------------------|
| break | <i>immediate exit</i> |
| continue | <i>next iteration</i> |
| else | <i>block for normal loop exit.</i> |

Algo:

$$S = \sum_{i=1}^{i=100} i^2$$

*statements block executed **for each** item of a container or iterator*

s = "Some text" } *initializations before the loop*

Algo: count number of e in the string.

Go over sequence's **index**

- ```
lst = [11, 18, 9, 12, 23, 4, 17]
lost = []
```

*Algo: limit values greater than 15, memorizing of lost values.*

Go simultaneously over sequence's **index** and **value**:

```
for idx, val in enumerate(lst):
```

| Generic Operations on Containers                                                                            |                                                                                                          |
|-------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <code>len(c)</code> → items count                                                                           | <i>Note: For dictionaries and sets, these operations use keys.</i><br><i>in</i> (absence <b>not in</b> ) |
| <code>min(c)</code> <code>max(c)</code> <code>sum(c)</code>                                                 |                                                                                                          |
| <code>sorted(c)</code> → <b>list</b> sorted <i>copy</i>                                                     |                                                                                                          |
| <b>val in c</b> → boolean, membership operator                                                              |                                                                                                          |
| <code>enumerate(c)</code> → <i>iterator</i> on (index, value)                                               |                                                                                                          |
| <code>zip(c1, c2...)</code> → <i>iterator</i> on tuples containing <b>c<sub>i</sub></b> items at same index |                                                                                                          |
| <b>all(c)</b> → <b>True</b> if <b>all c</b> items evaluated to true, else <b>False</b>                      |                                                                                                          |
| <b>any(c)</b> → <b>True</b> if <b>at least one</b> item of <b>c</b> evaluated true, else <b>False</b>       |                                                                                                          |

```
import copy
copy.copy(c) → shallow copy of container
copy.deepcopy(c) → deep copy of container
```

|                                                      | Operations on <b>list</b>                               |
|------------------------------------------------------|---------------------------------------------------------|
| <code>list</code>                                    | modify original list                                    |
| <code>list.append(val)</code>                        | add item at end                                         |
| <code>list.extend(seq)</code>                        | add sequence of items at end                            |
| <code>list.insert(idx, val)</code>                   | insert item at index                                    |
| <code>list.remove(val)</code>                        | remove first item with value <i>val</i>                 |
| <code>list.pop([idx]) → value</code>                 | remove & return item at index <i>idx</i> (default last) |
| <code>list.sort()</code> <code>list.reverse()</code> | sort / reverse list <i>in place</i>                     |

```

d[key]=value d.clear()
d[key] → value del d[key]
d.update(d2) { update/add
 { associations
d.keys()
d.values() } →iterable views on
d.items() { keys/values/associations
d.pop(key[,default]) → value
d.popitem() → (key,value)
d.get(key[,default]) → value
d.setdefault(key[,default]) →value

```

Operators:

- |  $\rightarrow$  union (vertical bar char)
- &  $\rightarrow$  intersection
- ^  $\rightarrow$  difference/symmetric diff.
- < <= > >=  $\rightarrow$  inclusion relations

*Operators also exist as methods.*

```
s.update(s2) s.copy()
s.add(key) s.remove(key)
s.discard(key) s.clear()
s.pop()
```

```
function name (identifier)
 named parameters
```

|     |
|-----|
| fct |
|-----|

Advanced: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`  
`*args` variable positional arguments ( $\rightarrow$  tuple), default values,  
`**kwargs` variable named arguments ( $\rightarrow$  dict)

```
r = fct(3, i+2, 2*i)
```

storage/use of  
returned value

one argument per  
parameter

**Operations on Strings**

```
s.startswith(prefix[,start[,end]])
s.endswith(suffix[,start[,end]]) s.strip([chars])
s.count(sub[,start[,end]]) s.partition(sep) → (before,sep,after)
s.index(sub[,start[,end]]) s.find(sub[,start[,end]])
s.is...() tests on chars categories (ex. s.isalpha())
s.upper() s.lower() s.title() s.swapcase()
s.casefold() s.capitalize() s.center([width,fill])
s.ljust([width,fill]) s.rjust([width,fill]) s.zfill([width])
s.encode(encoding) s.split([sep]) s.join(seq)
```

formatting directives      values to format      **Format**

`"model{e{} {} {} }".format(x, y, r) → str`

```
"{selection:formatting!conversion}"
```

- Selection :
  - 2
  - nom
  - 0.nom
  - 4[key]
  - 0[2]

Examples

```
"{:+.2.3f}".format(45.72793)
→ '+45.728'
"{1:>10s}".format(8,"toto")
→ ' toto'
"{x!r}".format(x="I'm")
→ "'I'm'"

```

- **Formatting :**  
fill char alignment sign mini width . precision~maxwidth type  
 <> ^ = + - space 0 at start for sign with 0  
 integer: **b** binary, **c** char, **d** decimal (default), **o** octal, **x** or **X** hexa...  
 float: **e** or **E** exponential, **f** or **F** fixed point, **g** or **G** appropriate (default),  
 string: **s** ... % percent
- **Conversion :** **s** (readable text) or **r** (literal representation)

storing data on disk, and reading it back

**Files**

```
f = open("file.txt", "w", encoding="utf8")
```

file variable  
for operations

name of file  
on disk  
(+path...)

opening mode

- 'r' read
- 'w' write
- 'a' append

encoding of  
chars for text  
files:  
utf8    ascii  
latin1    ...

cf. modules **os**, **os.path** and **pathlib**

```
... '+' 'x' 'b' 't'
```

| writing                             |                          | reading                              |
|-------------------------------------|--------------------------|--------------------------------------|
| <b>f.write</b> ("coucou")           | <b>f.read</b> ({n})      | → next chars                         |
| <b>f.writelines</b> (list of lines) |                          | if n not specified, read up to end ! |
|                                     | <b>f.readlines</b> ({n}) | → <b>list</b> of next lines          |
|                                     | <b>f.readline</b> ()     | → next line                          |

**f.close()** ⚠ dont forget to **close the file** after use !

**f.flush()** write cache      **f.truncate([size])** resize  
reading/writing progress sequentially in the file, modifiable with:  
**f.tell()** → position      **f.seek(position[,origin])**

Very common: opening with a guarded block (automatic closing) and reading loop on lines of a text file:

```
with open(...) as f:
 for line in f:
 # processing of line
```