PYTHON CHEAT SHEET

Base Types

integer, float, boolean, string, bytes

```
int 783  0 -192  0b010 0o642 0xF3
                 null  binary octal hexa
float 9.23 0.0  -1.7e-6
                        ×10^-6
bool True  False
str "One\nTwo"
      escaped new line
    'I\'m'
      escaped '
bytes b"toto\xfe\775"
          hexadecimal  octal
```

Multiline string:

```
"""X\tY\tZ
1\t2\t3"""
   escaped tab
```

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 just comas →* `tuple`

```
str bytes  (ordered sequences of chars / bytes)   "" b""
```

- **key containers**, no *a priori* order, fast key acces, each key is unique

```
dictionary  dict {"key":"value"}   dict(a=3,b=4,k="v")   {}
(key/value associations) {1:"one",3:"three",2:"two",3.14:"π"}
collection  set {"key1","key2"}   {1,9,3,0}   set()
```

keys=hashable values (base types, immutables…) `frozenset` immutable set *empty*

Boolean Logic

Comparators: `< > <= >= == !=` (boolean results) ≤ ≥ = ≠

`a and b` logical and — *both simulta-neously*

`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 block2…
    ⋮
next statement after block 1
```

indentation !

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

Modules/Names Imports

module `truc` ⇔ *file* `truc.py`

```
from monmod import nom1,nom2 as fct
```
→direct acces to names, renaming with `as`

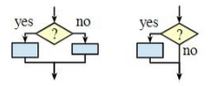
```
import monmod
```
→acces via `monmod.nom1` …

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

Conditional Statement

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 boolean var `x`:
`if x==True:` ⇔ `if x:`
`if x==False:` ⇔ `if not x:`

Maths

floating numbers… approximated values *angles in radians*

Operators: `+ - * / // % **`
Priority `(…)` × ÷ ↑ ↑ a^b
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 priorities

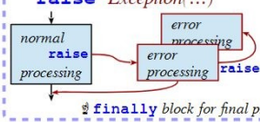
```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0   √
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```

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

Exceptions on Errors

Signaling an error:

```
raise Exception(…)
```



Errors processing:

```
try:
    normal procesising block
except Exception as e:
    error processing block
```

`finally` *block for final processing in all cases.*

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

`=`

1) evaluation of right side expression value
2) assignment in order with left side names

assignment ⇔ **binding** of a *name* with a *value*

```
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
*a,b=seq   unpacking of sequence in 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
```

and `*= /= %=` …

Conversions

`type(`*expression*`)`

```
int("15") → 15
int("3f",16) → 63   can specify integer number base in 2nd parameter
int(15.56) → 15   truncate decimal part
float("-11.24e8") → -1124000000.0
round(15.56,1)→ 15.6   rounding to 1 decimal (0 decimal → integer number)
bool(x)  False for null x, empty container x, None x or False x ; True for other x
str(x)→ "…"   representation string of x for display (cf. formating 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 separation str → list of str
    "1,4,8,2".split(",") → ['1','4','8','2']
sequence of one type → list of another type (via comprehension list)
    [int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Generic Operations on Containers

```
len(c)→ items count
min(c)  max(c)  sum(c)
sorted(c)→ list sorted copy
val in c → boolean, membership operator in (absence not in)
enumerate(c)→ iterator on (index, value)
zip(c1,c2…)→ iterator on tuples containing ci items at same index
all(c)→ True if all c items evaluated to true, else False
any(c)→ True if at least one item of c evaluated true, else False
```

Note: For dictionaries and sets, these operations use ***keys****.*

Specific to ordered sequences containers (lists, tuples, strings, bytes…)

```
reversed(c)→ inversed iterator   c*5→ duplicate   c+c2→ concatenate
c.index(val)→ position   c.count(val)→ events count
```

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

Opérations on Lists

modify original list

```
lst.append(val)   add item at end
lst.extend(seq)   add sequence of items at end
lst.insert(idx,val)   insert item at index
lst.remove(val)   remove first item with value val
lst.pop([idx])→value   remove & return item at index idx (default last)
lst.sort()  lst.reverse()   sort / reverse liste in place
```

Go simultaneously on sequence's **index** and **values**:

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

Integers Sequences

`range([`*start*`,] `*end*` [,`*step*`])`

start default 0, *fin* not included in sequence, *pas* signed default 1

```
range(5)→ 0 1 2 3 4   range(2,12,3)→ 2 5 8 11
range(3,8)→ 3 4 5 6 7   range(20,5,-5)→ 20 15 10
range(len(seq))→ sequence of index of values in seq
```

range provides an immutable sequence of int constructed as needed

Function Definition

function name (identifier) — named parameters

```
def fct(x,y,z):
    """documentation"""
    # statements block, res computation, etc.
    return res
```

`fct`

result value of the call, if no computed result to return: `return None`

parameters and all variables of this block exist only *in* the block and *during* the function call (think of a "black box")

Advanced: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`

**args variable positional arguments (→`tuple`), default values,*
***kwargs variable named arguments (→`dict`)*