for variables, functions,

modules, classes... names

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

language keywords forbidden

lower/UPPER case discrimination

diacritics allowed but should be avoided

a toto x7 y_max BigOne

Python 3 Cheat Sheet

Latest version on: https://perso.limsi.fr/pointal/python:memento

```
Base Types
integer, float, boolean, string, bytes
   int 783 0 -192
                         0b010 0o642 0xF3
                                        hexa
                         binary
                                 octal
float 9.23 0.0
                      -1.7e-6
 bool True False
   str "One\nTwo"
                           Multiline string:
                             """X\tY\tZ
       escaped new line
                             1\t2\t3"""
         'I<u>\</u>m'
                               escaped tab
         escaped '
bytes b"toto\xfe\775"
                                    immutables
            hexadecimal octal
```

Identifiers

```
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",)
*str bytes (ordered sequences of chars / bytes)
                                                              b""
• key containers, no a priori order, fast key access, 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:"n"}
         set {"key1", "key2"} {1,9,3,0}
collection
                                                            set (i)

    ★ keys=hashable values (base types, immutables...) frozenset immutable set

                                                              empty
```

```
8 8y and for
                   Variables assignment
 d assignment ⇔ binding of a name with a value
 1) evaluation of right side expression value
 2) 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
                                          and
x+=3
           increment \Leftrightarrow x=x+3
                                          *=
x=2
           decrement \Leftrightarrow x=x-2
                                          /=
                                          ક=
x=None « undefined » constant value
del x
```

remove name x

```
Conversions
                                           type (expression)
int("15") \rightarrow 15
int("3f", 16) \rightarrow 63
                                 can specify integer number base in 2<sup>nd</sup> parameter
int(15.56) \rightarrow 15
                                 truncate decimal part
float ("-11.24e8") \rightarrow -1124000000.0
round (15.56, 1) \rightarrow 15.6 rounding to 1 decimal (0 \text{ decimal} \rightarrow \text{ integer number})
bool (x) False for null x, empty container x, None or False x; True for other x
str(x) \rightarrow "..." representation string of x for display (cf. formatting on the back)
chr(64) \rightarrow '0' \quad ord('0') \rightarrow 64
                                       code \leftrightarrow char
repr(x) \rightarrow "..." literal representation string of x
bytes([72,9,64]) \rightarrow b'H\t@'
list("abc") \rightarrow ['a', 'b', 'c']
dict([(3, "three"), (1, "one")]) → {1: 'one', 3: 'three'}
set(["one","two"]) → {'one','two'}
separator str and sequence of str \rightarrow 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(",") \rightarrow ['1','4','8','2']
sequence of one type \rightarrow list of another type (via list comprehension)
    [int(x) for x in ('1', '29', '-3')] \rightarrow [1,29,-3]
```

```
for lists, tuples, strings, bytes...
                   -5
  negative index
   positive index
                  0
          lst=[10, 20, 30, 40,
   positive slice
  negative slice
Access to sub-sequences via 1st [start slice: end slice: step]
```

Items count $len(lst) \rightarrow 5$ **d** index from 0 (here from 0 to 4)

Individual access to items via lst [index] $lst[0] \rightarrow 10 \Rightarrow first one \qquad lst[1] \rightarrow 20$ $1st[-1] \Rightarrow 50 \Rightarrow last one$ $1st[-2] \rightarrow 40$ On mutable sequences (list), remove with del 1st[3] and modify with assignment 1st[4]=25

Sequence Containers Indexing

Boolean Logic

 $lst[:-1] \rightarrow [10,20,30,40]$ $lst[::-1] \rightarrow [50,40,30,20,10]$ $lst[1:3] \rightarrow [20,30]$ $lst[:3] \rightarrow [10,20,30]$ 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 \rightarrow from start / up to end.

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

Comparisons : < > <= >= != (boolean results) $\leq \geq = \neq$ a and b logical and both simulta--neously one or other a or b logical or or both i 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 True and False constants False

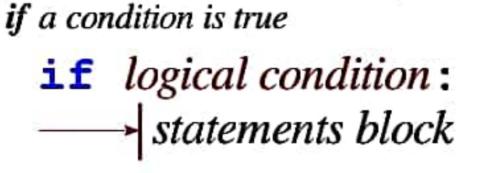
Statements Blocks parent statement: statement block 1... parent statement: statement block2... next statement after block 1

place of an indentation tab. angles in radians

desconfigure editor to insert 4 spaces in

Maths floating numbers... approximated values Operators: + - * / // % from math import sin, pi... ×÷ 🛕 $\sin(pi/4) \to 0.707...$ Priority (...) integer ÷ ÷ remainder $\cos(2*pi/3) \rightarrow -0.4999...$ @ → matrix × python3.5+numpy sqrt (81) →9.0 $log(e**2) \rightarrow 2.0$ $(1+5.3)*2\rightarrow12.6$ abs $(-3.2) \rightarrow 3.2$ $ceil(12.5) \rightarrow 13$ round $(3.57, 1) \rightarrow 3.6$ floor (12.5) →12 $pow(4,3) \rightarrow 64.0$ modules math, statistics, random, **■** usual order of operations decimal, fractions, numpy, etc. (cf. doc)

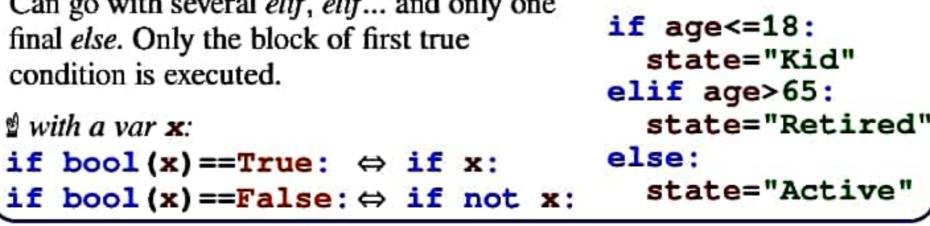
Modules/Names Imports module truc⇔file truc.py from monmod import nom1, nom2 as fct →direct access to names, renaming with as import monmod → access via monmod. nom1 ... modules and packages searched in python path (cf sys.path)



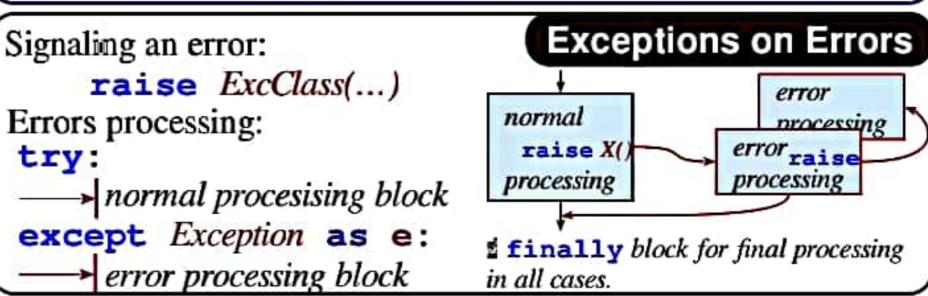
statement block executed only

with a var x:

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



Conditional Statement



Conditional Loop Statement Iterative Loop Statement statements block executed for each statements block executed as long as item of a container or iterator condition is true beware of infinite loops: yes ? while logical condition: for var in sequence: Loop Control finish statements block immediate exit statements block break continue next iteration Go over sequence's values s = 0 initializations before the loop **telse** block for normal s = "Some text" initializations before the loop loop exit. i = 1 condition with a least one variable value (here i) cnt = 0Algo: good habit : don't modi∉y loop variable while i <= 100: i = 100loop, variable, assignment managed by for statement s + i**2for c in s: i = i + 1make condition variable change! Algo: count if c == "e": print("sum:",s) i=1number of e cnt = cnt + 1print ("found", cnt, "'e'") in the string. Display print("v=",3,"cm :",x,",",y+4) loop on dict/set ⇔ loop on keys sequences use slices to loop on a subset of a sequence Go over sequence's index items to display: literal values, variables, expressions □ modify item at index print options: □ access items around index (before / after) □ sep=" " items separator, default space lst = [11, 18, 9, 12, 23, 4, 17]end="\n" end of print, default new line lost = [] - file=sys.stdout print to file, default standard output Algo: limit values greater for idx in range(len(lst)): than 15, memorizing val = lst[idx] Input = input("Instructions:") if val > 15: of lost values. BI lost.append(val) input always returns a string, convert it to required type lst[idx] = 15(cf. boxed *Conversions* on the other side). print("modif:",lst,"-lost:",lost) Generic Operations on Containers len (c) \rightarrow items count Go simultaneously over sequence's index and values: sum (c) min(c) max(c) Note: For dictionaries and sets, these for idx, val in enumerate(lst): $sorted(c) \rightarrow list sorted copy$ operations use keys. val in c → boolean, membership operator in (absence not in) Integer Sequences range ([start,] end [,step]) enumerate (c) \rightarrow iterator on (index, value) start default 0, end not included in sequence, step signed, default 1 zip (c1, c2...) → iterator on tuples containing c, items at same index range (5) \rightarrow 0 1 2 3 4 range $(2, 12, 3) \rightarrow 25811$ all (c) → True if all c items evaluated to true, else False range $(3, 8) \rightarrow 34567$ range $(20, 5, -5) \rightarrow 20 \ 15 \ 10$ any (c) → True if at least one item of c evaluated true, else False range (len (seq)) \rightarrow sequence of index of values in seq range provides an immutable sequence of int constructed as needed Specific to ordered sequences containers (lists, tuples, strings, bytes...) reversed (c) \rightarrow inversed iterator $c*5 \rightarrow$ duplicate **c+c2**→ concatenate **Function Definition** function name (identifier) **c.index** (val) \rightarrow position c.count (val) → events count named parameters import copy copy.copy(c) → shallow copy of container def fct(x,y,z): fct copy.deepcopy(c) → deep copy of container """documentation""" Operations on Lists → # statements block, res computation, etc. return res

result value of the call, if no computed add item at end 1st.append(val) result to return: return None lst.extend(seq) add sequence of items at end parameters and all insert item at index 1st.insert(idx, val) variables of this block exist only in the block and during the function call (think of a "black box") remove first item with value val 1st.remove(val) Advanced: def fct(x,y,z,*args,a=3,b=5,**kwargs): 1st.pop([idx]) $\rightarrow value$ remove & return item at index idx (default last) lst.sort() lst.reverse() sort / reverse liste in place *args variable positional arguments (→tuple), default values, **kwargs variable named arguments (→dict) Operations on Dictionaries Operations on Sets **Function Call** r = fct(3,i+2,2*i)Operators: d.clear() d[key] = value→ union (vertical bar char) storage/use of one argument per del d[key] $d[key] \rightarrow value$ returned value → intersection parameter update/add d.update(d2)^ → difference/symmetric diff. # this is the use of function fct () fct Advanced: associations < <= >= \rightarrow inclusion relations d.keys() name with parentheses *sequence →iterable views on Operators also exist as methods. d.values() which does the call **dict keys/values/associations d.items() s.update(s2) s.copy() **d.** pop $(key[,default]) \rightarrow value$ Operations on Strings s.startswith(prefix[,start[,end]]) s.add(key) s.remove(key) **d.popitem()** \rightarrow (key, value) s.endswith(suffix[,start[,end]]) s.strip([chars]) s.discard(key) s.clear() $d.get(key[,default]) \rightarrow value$ s.count(sub[,start[,end]]) s.partition(sep) → (before,sep,after) s.pop() $d.setdefault(key[,default]) \rightarrow value$ s.index(sub[,start[,end]]) s.find(sub[,start[,end]]) Files s.is...() tests on chars categories (ex. s.isalpha()) storing data on disk, and reading it back s.upper() s.lower() s.title() s.swapcase() = open("file.txt", "w", encoding="utf8") s.casefold() s.capitalize() s.center([width,fill]) namé of file opening mode file variable s.ljust([width,fill]) encoding of s.rjust([width,fill]) s.zfill([width]) □ 'r' read chars for text on disk for operations s.encode (encoding) s.split([sep]) s.join(seq) □ 'w' write files: (+path...) formating directives values to format Formatting □ 'a' append utf8 ascii cf. modules os, os. path and pathlib ... '+' 'x' 'b' 't' latin1 "modele{} {} ".format(x,y,r)writing reading read empty string if end of file " { selection : formatting ! conversion } " f.read([n])→ next chars f.write("coucou") □ Selection : "{:+2.3f}".format(45.72793) if n not specified, read up to end! f.writelines (list of lines) \rightarrow '+45.728' **f.readlines** $([n]) \rightarrow list of next lines$ → next line nom "{1:>10s}".format(8, "toto") f.readline() 0.nom text mode t by default (read/write str), possible binary toto' 4 [key] "{x!r}".format(x="I'm") mode b (read/write bytes). Convert from/to required type! 0[2] →'"I\'m"' f.close() dont forget to close the file after use! Formatting: f.truncate([size]) resize fill char alignment sign mini width . precision~maxwidth type f.flush() write cache reading/writing progress sequentially in the file, modifiable with: <> ^= + - space 0 at start for filling with 0 f.tell() \rightarrow position f.seek (position[,origin]) integer: b binary, c char, d decimal (default), o octal, x or X hexa... Very common: opening with a guarded block with open (...) as f: float: e or E exponential, f or F fixed point, g or G appropriate (default), (automatic closing) and reading loop on lines for line in f : string: s ... % percent of a text file: # processing of line Conversion: s (readable text) or r (literal representation)