Qualité de code et convention de codage

Règles de codage

Les règles de codage sont un ensemble de règles à suivre pour uniformiser les pratiques de développement logiciel, diffuser les bonnes pratiques de développement et éviter les erreurs de développement "classiques" au sein d'un groupe de développeurs.

Les règles de codage s'articulent autour de plusieurs thèmes, les plus courants étant :

- Le nommage et l'organisation des fichiers du code source
- le style d'indentation
- Les conventions de nommage, ou règles de nommage
- Les commentaires et documentation du code source
- Recommandations sur la déclaration des variables
- Recommandations sur l'écriture des instructions, des structures de contrôle et l'usage des parenthèses dans les expressions.

Afin d'associer règles et qualité de code, nous pouvons ajouter :

- Utiliser un gestionnaire de version come GitHub ou GitLab
 - o https://github.com/
 - o https://gitlab.com/gitlab-org
- Intégrer les tests tout au long du cycle de développement et utiliser un API de test.
 - o https://www.jetbrains.com/pycharm

Nommage

- les identificateurs des constantes doivent être composés de majuscules ou de tirets has.

les identificateurs des classes doivent être écrits en camel case (sans espace ni ponctuation, et en mettant en capitale la première lettre de chaque mot) en commençant par une lettre majuscule.

- les identificateurs des méthodes doivent être écrits en camel case en commençant par une lettre minuscule.

La notation hongroise est, en programmation informatique, une convention de nommage des variables et des fonctions qui met en avant soit leur usage, soit leur type. Cette notation utilisée couramment consiste à faire précéder le nom de la variable d'un préfixe reflétant son type. Ce préfixe est toujours écrit en minuscule puis la première lettre suivante en majuscule. Par exemple la variable booléenne danger est préfixée par un b pour indiquer un type booléen : bDanger.

Il y a des outils de vérification comme PyChecker http://pychecker.sourceforge.net/

En python

Ce site liste les règles stylistiques recommandées, invitant toute la communauté Python à écrire un code de la même façon (Le PEP8, en résumé) http://sametmax.com/le-pep8-en-resume/

Espaces

Les opérateurs doivent être entourés d'espaces.

Il faut faire ceci:

```
variable = 'valeur'
ceci == cela
1 + 2
Et non:
variable='valeur'
ceci==cela
1+2
```

Il y a deux exceptions notables.

La première étant qu'on groupe les opérateurs mathématiques ayant la priorité la plus haute pour distinguer les groupes :

```
a = x*2 - 1

b = x*x + y*y

c = (a+b) * (a-b)
```

La seconde est le signe = dans la déclaration d'arguments et le passage de paramètres :

```
def fonction(arg='valeur'): # ça c'est ok
    pass

resultat = fonction(arg='valeur') # ça aussi
```

On ne met pas d'espace à l'intérieur des parenthèses, crochets ou accolades.

Oui:

```
2 * (3 + 4)
def fonction(arg='valeur'):
{str(x): x for x in range(10)}
val = dico['key']
```

Non:

```
2 * ( 3 + 4 )
def fonction( arg='valeur' ):
{ str(x): x for x in range(10) }
val = dico[ 'key' ]
```

On ne met pas d'espace avant les deux points et les virgules, mais après oui.

Oui:

```
def fonction(arg1='valeur', arg2=None):
    dico = {'a': 1}

Non:
    def fonction(arg='valeur', arg2=None):
    dico = {'a': 1}
```

Lignes

Une ligne doit se limiter à 79 charactères. Cette limite, héritée des écrans touts petits, est toujours en vigeur car il est plus facile de scanner un code sur une courte colonne qu'en faisant des aller-retours constant

Si une ligne est trop longue, il existe plusieurs manières de la racourcir :

Ici l'indentation entre le nom de la fonction et des paramètres est légèrement différente pour mettre en avant la distinction.

Une variante:

Si c'est un appel chainé, on peut utiliser \ pour mettre à la ligne :

Si c'est une structure de données, on peut se la jouer langage fonctionel de la vibes du flex :

```
chiffres = [
    1, 2, 3,
    4, 5, 6,
]

contacts = {
   'Cleo': (),
   'Ramses': (
        ('maman', '0248163264'),
        ('papa', '01234567890'),
        ('mamie', '55555555'),
        ('momie', '066642424269')
   )
}
```

Séparer les fonctions et les classes à la racine d'un module par 2 lignes vides. Les méthodes par 1 ligne vide. Parfois je triche et je fais 3 et 2 au lieu de 2 et 1.

Les imports de plusieurs modules doivent être sur plusieurs lignes :

```
import sys
import os
Et non:
import sys, os
```

Bien sûr, ce n'est pas valable pour from x import z.

Souvenez-vous qu'on peut utiliser les parenthèses pour diviser de longues lignes. Par exemple :

Idem pour les chaînes très longues :

```
s = ("Les chaînes Python sont automatiquement"
    "concaténées par la VM si elles sont "
    "uniquement séparées par des espaces "
    "ou sauts de lignes.")
```

Pour en revenir aux lignes d'import, on doit les ordonner ainsi :

- Import de module > import du contenu du module
- Import de la lib standard > import de libs tierces parties > import de votre projet

Exemple:

```
import os # import module de la lib standard
import sys # on groupe car même type
```

```
from itertools import islice # import du contenu du module
from collections import namedtuple # import groupe car même type

import requests # import lib tierce partie
import arrow # on groupe car même type

from django.conf import settings # tierce partie, contenu du module
from django.shortcuts import redirect # on groupe car même type

from mon projet.mon module import ma bite # mon projet
```

Format du fichier

Encoding: UTF8 ou ASCII (ce qui est de l'UTF8 de toute façon).

Docstrings

```
(c.f. PEP257)
```

On utilise toujours des triples quotes :

```
def fonction_avec_docstring_courte():
    """Résumé en une ligne."""
    pass
```

Si la docstring est longue (elle peut être très très très longue si vous le souhaitez) :

```
def fonction():
    """Résumé en une ligne suivi d'une line vide.

    Description longue de la fonction qui
    se termine par une ligne vide puis une
    triple quotes sur sa propre ligne.
    """
```

Noms de variables

Lettres seules, en minuscule : pour les boucles et les indices.

Exemple:

```
for x in range(10):
    print(x)

i = get_index() + 12
print(ma liste[i])
```

Lettres minuscules + underscores : pour les modules, variables, fonctions et méthodes.

```
une_variable = 10

def une_fonction():
    return locals() or {}

class UneClasse:
    def une_methode_comme_une_autre(self):
        return globals()
```

Lettres majuscules + underscores : pour les (pseudo) constantes.

```
MAX_SIZE = 100000 # à mettre après les imports
```

Camel case: nom de classe.

```
class CeciEstUneClasse:
    def methodiquement(self):
        pass
```

Si le nom contient un acronyme, on fait une entorse à la règle :

```
class HTMLParserCQFDDDTCCMB:
    def methodiquement(self):
        pass
```

On n'utilise PAS le mixedCase.

Sources

- 1. https://fr.wikipedia.org/wiki/R%C3%A8gles de codage
- 2. https://fr.wikipedia.org/wiki/Notation hongroise
- 3. http://pychecker.sourceforge.net/
- 4. https://www.python.org/dev/peps/pep-0020/
- 5. https://www.pylint.org/
- 6. http://sametmax.com/le-pep8-en-resume/

x+=3

x - = 2

del x

 $increment \Leftrightarrow x=x+3$

 $decrement \Leftrightarrow x=x-2$

x=None « undefined » constant value

remove name x

Python 3 Cheat Sheet

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

```
Base Types
                                                                                                              Container Types
integer, float, boolean, string, bytes
                                                   • ordered sequences, fast index access, repeatable values
                                                            list [1,5,9]
                                                                                 ["x",11,8.9]
                                                                                                          ["mot"]
                                                                                                                             int 783 0 -192
                          0b010 0o642 0xF3
float 9.23 0.0
                           binary
                                  octal
                                          hexa
                                                         ,tuple (1,5,9)
                                                                                   11, "y", 7.4
                                                                                                          ("mot",)
                                                                                                                             ()
                      -1.7e-6
                                                   Non modifiable values (immutables)
                                                                                  bool True False
                            ×10<sup>-6</sup>
                                                         * str bytes (ordered sequences of chars / bytes)
   str "One\nTwo"
                                                                                                                           b""
                            Multiline string:
                                                   • key containers, no a priori order, fast key access, each key is unique
       escaped new line
                               """X\tY\tZ
                               1\t2\t3"""
                                                  dictionary dict {"key":"value"}
                                                                                              dict(a=3,b=4,k="v")
                                                                                                                            { }
         'I<u>\</u>m'
         escaped '
                                                  (key/value associations) {1:"one", 3:"three", 2:"two", 3.14:"π"}
                                 escaped tab
bytes b"toto\xfe\775"
                                                              set {"key1", "key2"}
                                                                                                                        set()
                                                                                              {1,9,3,0}
                                     ₫ immutables
             hexadecimal octal

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

                                                                                              frozenset immutable set
                                                                                                                           empty
for variables, functions,
                             Identifiers
```

```
modules, classes... names
                                                int("3f",16) \rightarrow 63
a...zA...Z_ followed by a...zA...Z_0...9
                                                int (15.56) \rightarrow 15
diacritics allowed but should be avoided

    language keywords forbidden

                                                round (15.56, 1) \rightarrow 15.6
□ lower/UPPER case discrimination
      © a toto x7 y_max BigOne
      8 8y and for
                                                chr(64) \rightarrow '@' \quad ord('@') \rightarrow 64
                   Variables assignment
 assignment ⇔ binding of a name with a value
                                                bytes([72,9,64]) \rightarrow b'H\t@'
 1) evaluation of right side expression value
                                                list("abc") \rightarrow ['a', 'b', 'c']
 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
                                                 str splitted on whitespaces \rightarrow list of str
a, *b=seq \ unpacking of sequence in
*a, b=seq | item and list
```

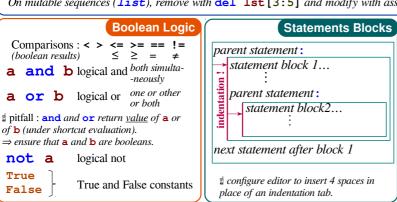
and

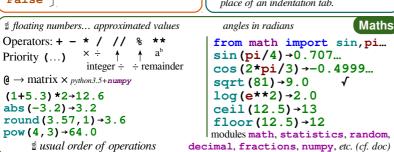
/=

용=

```
type (expression)
                                                                         Conversions
int ("15") \rightarrow 15
                                  can specify integer number base in 2^{nd} parameter
                                  truncate decimal part
float ("-11.24e8") \rightarrow -1124000000.0
                                  rounding to 1 decimal (0 decimal \rightarrow 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)
                                           code \leftrightarrow char
repr (x) \rightarrow "..." literal representation string of x
dict([(3,"three"),(1,"one")]) \rightarrow \{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'
    "words with spaces".split() → ['words', 'with', 'spaces']
\mathtt{str} splitted on separator \mathtt{str} \to \mathtt{list} of \mathtt{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]
```

```
Sequence Containers Indexing
                                        for lists, tuples, strings, bytes...
                    -5
                           -4
                                   -3
                                           -2
                                                   -1
                                                                Items count
                                                                                      Individual access to items via lst [index]
  negative index
                    0
                            1
                                    2
                                            3
   positive index
                                                            len (lst) \rightarrow 5
                                                                                      lst[0]→10
                                                                                                         \Rightarrow first one
                                                                                                                           1st[1]→20
          lst=[10,
                           20,
                                   30,
                                                   50]
                                           40
                                                                                      1st [-1] → 50 \Rightarrow last one
                                                                                                                           1st [-2] \rightarrow 40
                                                               positive slice
                  0
                         1
                                        3
                                               4
                                                                                      On mutable sequences (list), remove with
                                                              (here from 0 to 4)
                               -3
   negative slice
                                                                                      del 1st[3] and modify with assignment
                                                                                      1st[4]=25
Access to sub-sequences via lst [start slice: end slice: step]
                                                                                                                lst[:3] \rightarrow [10, 20, 30]
lst[:-1] \rightarrow [10,20,30,40] lst[::-1] \rightarrow [50,40,30,20,10] lst[1:3] \rightarrow [20,30]
                                                                                 lst[-3:-1] \rightarrow [30,40] lst[3:] \rightarrow [40,50]
lst[1:-1] \rightarrow [20,30,40]
                                     lst[::-2] \rightarrow [50, 30, 10]
                                     lst[:] \rightarrow [10, 20, 30, 40, 50] shallow copy of sequence
lst[::2] \rightarrow [10, 30, 50]
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]
```

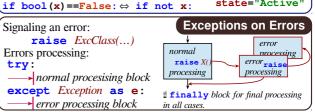




Modules/Names Imports module truc⇔file truc.py from monmod import nom1, nom2 as fct →direct access to names, renaming with as import monmod →access via monmod.nom1 ...



if age<=18: final else. Only the block of first true state="Kid" condition is executed. elif age>65: state="Retired" if bool(x)==True: ⇔ if x: else: state="Active"



```
Conditional Loop Statement | statements block executed for each | Iterative Loop Statement
   statements block executed as long as
                                                                                 item of a container or iterator
   condition is true
infinite loops:
      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
                                                               ₫ else block for normal
ф
  i = 1 condition with a least one variable value (here i)
                                                               loop exit.
                                                                                          s = "Some text" initializations before the loop
beware
                                                                                          cnt = 0
                                                                Algo:
                                                                                                                                                    good habit : don't modify loop variable
   while i <= 100:
                                                                      i = 100
                                                                                            loop variable, assignment managed by for statement or c in s:
                                                                       \sum_{i}^{2} i^{2}
        s = s + i**2
                                                                                          for
                                                                                                if c == "e":
        i = i + 1
                           print("sum:",s)
                                                                                                     cnt = cnt + 1
                                                                                                                                  number of e
                                                                                          print("found", cnt, "'e'")
                                                                                                                                  in the string.
                                                                     Display
                                                                                  loop on dict/set ⇔ loop on keys sequences
 print("v=", 3, "cm : ", x, ", ", y+4)
                                                                                  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)):
                                                                                       val = lst[idx]
                                                                                                                            than 15, memorizing
                                                                        Input
 s = input("Instructions:")
                                                                                       if val > 15:
                                                                                                                            of lost values.
                                                                                            lost.append(val)
    input always returns a string, convert it to required type
                                                                                  lst[idx] = 15
print("modif:",lst,"-lost:",lost)
        (cf. boxed Conversions on the other side).
len (c) → items count
                                    Generic Operations on Containers
                                                                                  Go simultaneously over sequence's index and values:
min(c) max(c) sum(c)
                                             Note: For dictionaries and sets, these
                                                                                  for idx,val in enumerate(lst):
sorted(c) → list sorted copy
                                              operations use keys.
val in c \rightarrow boolean, membership operator in (absence not in)
                                                                                                                              Integer Sequences
                                                                                    range ([start,] end [,step])
enumerate (\mathbf{c}) \rightarrow iterator on (index, value)
                                                                                  ₫ start default 0, end not included in sequence, step signed, default 1
zip (c1, c2...) \rightarrow iterator on tuples containing c<sub>i</sub> 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 3 4 5 6 7
                                                                                                                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
Specific to ordered sequences containers (lists, tuples, strings, bytes...)
                                                                                  reversed (c) \rightarrow inversed iterator c*5\rightarrow duplicate
                                                         c+c2→ concatenate
                                                                                                                              Function Definition
                                     c. count (val) \rightarrow events count
                                                                                  function name (identifier)
c.index (val) \rightarrow position
import copy
                                                                                              named parameters
copy.copy (c) → shallow copy of container
                                                                                   def fct(x, y, z):
                                                                                                                                            fct
copy . deepcopy (c) → deep copy of container
                                                                                          """documentation"""
                                                                                          # statements block, res computation, etc.
                                                      Operations on Lists
return res ← result value of the call, if no computed
lst.append(val)
                               add item at end
                                                                                                               result to return: return None
                               add sequence of items at end
lst.extend(seq)
                                                                                   lst.insert(idx, val)
                              insert item at index
                                                                                   variables of this block exist only in the block and during the function
lst.remove(val)
                               remove first item with value val
                                                                                   call (think of a "black box")
                                                                                   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 (\rightarrow tuple), default values,
                                                                                     **kwares variable named arguments (\rightarrow dict)
     Operations on Dictionaries
                                                       Operations on Sets
                                          Operators:
                                                                                   \mathbf{r} = \mathbf{fct}(3, \mathbf{i} + 2, 2 * \mathbf{i})
                                                                                                                                     Function Call
                       d.clear()
d[key] = value
                                            I → union (vertical bar char)
                                                                                   storage/use of
                                                                                                        one argument per
                       del d[key]
d[key] \rightarrow value
                                                                                   returned value
                                                                                                        parameter
                                               → intersection
d. update (d2) { update/add associations

    - ^ → difference/symmetric diff.

                                                                                                                                               fct
                                                                                  this is the use of function
                                                                                                                Advanced:
                                            < <= > >= → inclusion relations
d.keys()
                                                                                  name with parentheses
                                                                                                                 *sequence
d.values() 

d.items() 

→iterable views on 
keys/values/associations
                 →iterable views on
                                          Operators also exist as methods.
                                                                                  which does the call
                                                                                                                **dict
                                          s.update(s2) s.copy()
d. pop (key[,default]) \rightarrow value
                                                                                                                         Operations on Strings
                                                                                  s.startswith(prefix[,start[,end]])
d.popitem() \rightarrow (key, value) d.get(key[, default]) \rightarrow value
                                          s.add(key) s.remove(key)
                                                                                  s.endswith(suffix[,start[,end]]) s.strip([chars])
                                          s.discard(key) s.clear()
                                          s.pop()
                                                                                  s.count(sub[,start[,end]]) s.partition(sep) \rightarrow (before,sep,after)
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()
     f = open("file.txt", "w", encoding="utf8")
                                                                                  s.casefold()
                                                                                                    s.capitalize() s.center([width,fill])
file variable
                name of file
                                  opening mode
                                                                                  s.ljust([width,fill]) s.rjust([width,fill]) s.zfill([width])
                                                            encoding of
for operations
                on disk
                                    'r' read
                                                            chars for text
                                                                                                          s.split([sep]) s.join(seq)
                                                                                  s.encode (encoding)
                                  □ 'w' write
                                                            files:
                (+path...)
cf. modules os, os.path and pathlib ....'+' 'x'
                                                                                     formating directives
                                                                                                                   values to format
                                                            utf8
                                                                    ascii
                                                                                                                                       Formatting
                                                'b' 't' latin1 ...
                                                                                   "modele{} {} {}".format(x,y,r)—
                                 🖆 read empty string if end of file
                                                                      reading
                                                                                   "{selection: formatting!conversion}"
 f.write("coucou")
                                 f.read([n])
                                                       \rightarrow next chars
                                                                                  □ Selection :
                                                                                                               "{:+2.3f}".format(45.72793)
                                      if n not specified, read up to end!
 f.writelines (list of lines)
                                 f.readlines ([n]) \rightarrow list of next lines f.readline () \rightarrow next line
                                                                                     2
                                                                                                              →'+45.728'
                                                                                                              "{1:>10s}".format(8,"toto")

→' toto'
                                                                                     nom
                                f.readline()
                                                                                     0.nom
          🖠 text mode t by default (read/write str), possible binary
                                                                                     4 [key]
                                                                                                              "{x!r}".format(x="I'm")
          mode b (read/write bytes). Convert from/to required type!
                                                                                     0[2]
                                                                                                              \rightarrow'"I\'m"'
                    dont forget to close the file after use!
f.close()
                                                                                   □ Formatting :
                                    f.truncate([size]) resize
f.flush() write cache
                                                                                   fill char alignment sign mini width precision~maxwidth type
                                                                                   <> ^ = + - space
reading/writing progress sequentially in the file, modifiable with:
                                                                                                           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 ..
 of a text file:
                                                       # processing of line
                                                                                   □ Conversion: s (readable text) or r (literal representation)
```