

# Mémento Python 3 (partie 1)

## entier, flottant, booléen, chaîne, octets Types de base

```
int 783 0 -192 0b010 0o642 0xF3
    nul      binaire octal hexa
float 9.23 0.0 -1.7e-6
bool True False
str "Un\nDeux" Chaîne multiligne :
    retour à la ligne échappé "'X\tY\tZ'
    'L'\âme' 1\t2\t3""
    ' échappé tabulation échappée
bytes b"toto\xfe\775"
      hexadecimal octal
      immutables
```

## séquences ordonnées, accès par index rapide, valeurs répétables Types conteneurs

```
list [1,5,9] ["x",11,8.9] ["mot"]
tuple (1,5,9) 11,"y",7.4 ("mot",)
str bytes
dictionnaire dict {"clé": "valeur"} dict(a=3,b=4,k="v")
(couples clé/valeur) {1:"un",3:"trois",2:"deux",3.14:"pi"}
ensemble set {"clé1","clé2"} {1,9,3,0} set()
clés=valeurs hachables (types base, immutables...) frozenset ensemble immuable vides
```

## pour noms de variables, fonctions, modules, classes... Identificateurs

a...zA...Z suivi de a...zA...Z\_0...9  
accents possibles mais à éviter  
mots clés du langage interdits  
distinction casse min/MAJ  
a toto x7 y\_max BigOne  
by and for

## Affectation de variables

association d'un nom à une valeur  
évaluation de la valeur de l'expression de droite  
affectation dans l'ordre avec les noms de gauche  
x=1.2+8+sin(y)  
a=b=c=0 affectation à la même valeur  
y,z,r=9,-7.6,0 affectations multiples  
a,b=b,a échange de valeurs  
a,\*b=seq dépaquetage de séquence en  
\*a,b=seq élément et liste  
x+=3 incrémentation ⇔ x=x+3 et  
x-=2 décrémentation ⇔ x=x-2 \*=  
x=None valeur constante « non défini » /=  
del x suppression du nom x %=  
...

## int("15") → 15 type(expression) Conversions

int("3f",16) → 63 spécification de la base du nombre entier en 2<sup>nd</sup> paramètre  
int(15.56) → 15 troncature de la partie décimale  
float("-11.24e8") → -1124000000.0  
round(15.56,1) → 15.6 arrondi à 1 décimale (0 décimale → nb entier)  
bool(x) False pour x nul, x conteneur vide, x None ou False ; True pour autres x  
str(x) → "..." chaîne de représentation de x pour l'affichage (cf. Formattage, en partie 2)  
chr(64) → '@' ord('@') → 64 code ↔ caractère  
repr(x) → "..." chaîne de représentation littérale de x  
bytes([72,9,64]) → b'H\t@'  
list("abc") → ['a','b','c']  
dict([(3,"trois"),(1,"un")]) → {1:'un',3:'trois'}  
set(["un","deux"]) → {'un','deux'}  
str de jointure et séquence de str → str assemblée  
'.'.join(['toto','12','pswd']) → 'toto:12:pswd'  
str découpée sur les blancs → list de str  
"des mots espacés".split() → ['des','mots','espacés']  
str découpée sur str séparateur → list de str  
"1,4,8,2".split(",") → ['1','4','8','2']  
séquence d'un type → list d'un autre type (par liste en compréhension)  
[int(x) for x in ('1','29','-3')] → [1,29,-3]

pour les listes, tuples, chaînes de caractères, bytes...

## Indexation des conteneurs séquences

index négatif	-5	-4	-3	-2	-1
index positif	0	1	2	3	4

lst=[10, 20, 30, 40, 50]  
tranche positive 0 1 2 3 4 5  
tranche négative -5 -4 -3 -2 -1

Nombre d'éléments  
len(lst) → 5  
Accès individuel aux éléments par lst[index]  
lst[0] → 10 ⇒ le premier lst[1] → 20  
lst[-1] → 50 ⇒ le dernier lst[-2] → 40  
Sur les séquences modifiables (list), suppression avec del lst[3] et modification par affectation lst[4]=25

Accès à des sous-séquences par lst[tranche début:tranche fin:pas]

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] copie superficielle de la séquence

Indication de tranche manquante → à partir du début / jusqu'à la fin.

Sur les séquences modifiables (list), suppression avec del lst[3:5] et modification par affectation lst[1:4]=[15,25]

## Logique booléenne

Comparateurs : < > <= >= == != (résultats booléens) ≤ ≥ = ≠  
a and b et logique les deux en même temps  
a or b ou logique l'un ou l'autre ou les deux  
piège : and et or retournent la valeur de a ou de b (selon l'évaluation au plus court).  
⇒ s'assurer que a et b sont booléens.  
not a non logique  
True False constantes Vrai/Faux

## Blocs d'instructions

instruction parente :  
bloc d'instructions 1...  
instruction parente :  
bloc d'instructions 2...  
instruction suivante après bloc 1  
régler l'éditeur pour insérer 4 espaces à la place d'une tabulation d'indentation.

## module truc ⇔ fichier truc.py Modules & Imports

from monmod import nom1,nom2 as fct  
→ accès direct à nom1, renommage nom2 en fct avec as  
import monmod → accès via monmod.nom1...  
modules et packages recherchés dans le python path (cf. sys.path)

## un bloc d'instructions exécuté, uniquement si sa condition est vraie Instruction conditionnelle

if condition logique :  
bloc d'instructions  
Combinable avec des sinon si, sinon si... et un seul sinon final. Seul le bloc de la première condition trouvée vraie est exécuté.  
if age<=18: etat="Enfant"  
elif age>65: etat="Retraité"  
else: etat="Actif"

## Maths

Opérateurs : + - \* / // % \*\*  
Priorités (...) × ÷ ↑ ↓ a<sup>b</sup>  
@ → × matricielle (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  
priorités usuelles

## angles en 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.

## Exceptions sur erreurs

Signalisation sur détection :  
raise ExcClass(...)  
Traitement :  
try :  
→ bloc traitement normal  
except ExcClass as e :  
→ bloc traitement erreur  
bloc finally pour traitements finaux dans tous les cas.

