

Types de base

entier, flottant, booléen, chaîne, octets

```
int 783 0 -192 0b010 0o642 0xF33
      zéro binaire octal hexa
float 9.23 0.0 -1.7e-6
bool True False
str "Un\nDeux"
      retour à la ligne échappé
      'L\âme'
      échappé
bytes b"toto\xfe\x775"
      hexadécimal octal
```

Chaîne multiligne :
"""X\tY\tZ
1\t2\t3"""
tabulation échappée

☞ immutables

Types conteneurs

- séquences ordonnées, accès par index rapide, valeurs répétables
list [1, 5, 9] ["x", 11, 8.9] ["mot"]
tuple (1, 5, 9) 11, "y", 7.4 ("mot",)
Valeurs non modifiables (immutables) ☞ expression juste avec des virgules → **tuple**
str bytes (séquences ordonnées de caractères / d'octets)
- conteneurs clés, sans ordre a priori, accès par clé rapide, chaque clé unique
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 vide

Identificateurs

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

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
☉ 8y and for

Variables & affectation

☞ affectation ⇔ **association** d'un nom à une valeur

1) évaluation de la valeur de l'expression de droite
2) 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=9.2,-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
x-=2 décrémentation ⇔ x=x-2
x=None valeur constante « non défini »
del x suppression du nom x
```

Conversions

```
int("15") → 15
int("3f",16) → 63
int(15.56) → 15
float("-11.24e8") → -1124000000.0
round(15.56,1) → 15.6
bool(x) False pour x zéro, x conteneur vide, x None ou False ; True pour autres x
str(x) → "..." chaîne de représentation de x pour l'affichage (cf. Formatage au verso)
chr(64) → '@' ord('@') → 64
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]
```

Indexation conteneurs séquences

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

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
```

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

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

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.

Imports modules/noms

module **truc** ⇔ fichier **truc.py**

from monmod import nom1, nom2 as fct
→ accès direct aux noms, renommage avec **as**

import monmod → accès via **monmod.nom1** ...
☞ modules et packages cherchés dans le python path (cf. **sys.path**)

Instruction conditionnelle

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

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.

☞ avec une variable **x**:
if bool(x)==True: ⇔ **if x:**
if bool(x)==False: ⇔ **if not x:**

```
if age<=18:
    etat="Enfant"
elif age>65:
    etat="Retraité"
else:
    etat="Actif"
```

Maths

☞ nombres flottants... valeurs approchées !

Opérateurs: + - * / // % **
Priorités (...)

@ → × matricielle python 3.5+ 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 : **raise ExcClass(...)**

Traitement : **try:**
→ bloc traitement normal
except ExcClass as e:
→ bloc traitement erreur

☞ **finally** pour traitements finaux dans tous les cas.

