

**Types de base**

entier, flottant, booléen, chaîne

```
int 783 0 -192 0b010 0o642 0xF3
float 9.23 0.0 -1.7e-6
bool True False
str "Un Deux"
```

hexa  
octal  
 $\times 10^{-6}$

**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  $\Leftrightarrow$  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

x+=3 incrémentation  $\Leftrightarrow$  x=x+3

x-=2 décrémentation  $\Leftrightarrow$  x=x-2

x=None valeur constante « non défini »

del x suppression du nom x

et  
\*=  
/=

**Imports modules/noms**

module truc  $\Leftrightarrow$  fichier truc.py

from monmod import nom1 as fct

→ accès direct aux noms, renommage avec as

import monmod

→ accès via monmod.nom1...

**Exceptions sur erreurs**

Signalisation : `raise ExcClass(...)`

Traitement :

```
try:
    # bloc traitement normal
except ExcClass as e:
    # bloc traitement erreur
```

traitement normal

traitement erreur

traitement erreur

traitement normal

Ⓢ bloc **finally** pour traitements finaux dans tous les cas.

**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:  $\Leftrightarrow$  if x:
if bool(x) == False:  $\Leftrightarrow$  if not x:
```

if age<=18: etat="Enfant"

elif age>65: etat="Retraité"

else: etat="Actif"

**Instruction boucle conditionnelle**

bloc d'instructions exécuté tant que la condition est vraie

**while condition logique :**

→ bloc d'instructions

s = 0 initialisations avant la boucle

i = 1 condition avec au moins une valeur variable (ici i)

```
while i <= 100:
    s = s + i**2
    i = i + 1
print("somme:", s)
```

Ⓢ faire varier la variable de condition !

Algo :  $i=100$

$S = \sum_{i=1}^n i^2$

attention aux boucles sans fin !

**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.

**Conversions**

type (expression)

```
int("15") → 15
int("3f", 16) → 63 spécification de la base du nombre entier en 2nd 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 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 code  $\Leftrightarrow$  caractère
repr(x) → "..." chaîne de représentation littérale de x
list("abc") → ['a', 'b', 'c']
```

**Maths**

angles en radians

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

modules math, random, decimal, fractions, etc.

Opérateurs : + - \* / // % \*\*

Priorités (...)

× ÷ ↑ ↑ a<sup>b</sup>

÷ entière reste ÷

(1+5.3)\*2 → 12.6

abs(-3.2) → 3.2

round(3.57, 1) → 3.6

pow(4, 3) → 64.0

Ⓢ priorités usuelles

**Définition de fonction**

nom de la fonction (identificateur)

paramètres nommés

```
def fct(x, y, z):
    """documentation"""
    # bloc instructions, calcul de res, etc.
    return res
```

fct

→ valeur résultat de l'appel, si pas de résultat calculé à retourner : **return None**

Ⓢ les paramètres et toutes les variables de ce bloc n'existent que dans le bloc et pendant l'appel à la fonction (penser "boîte noire")

**Appel de fonction**

stockage/utilisation de la valeur de retour

Une valeur d'argument par paramètre

```
r = fct(3, i+2, 2*i)
```

Ⓢ c'est l'utilisation du nom de la fonction avec les parenthèses qui fait l'appel

**Instruction boucle itérative**

bloc d'instructions exécuté pour chaque élément d'un conteneur ou d'un itérateur

**for var in séquence :**

→ bloc d'instructions

Parcours des valeurs d'un conteneur

```
s = "Du texte" } initialisations avant la boucle
cpt = 0
for c in s:
    if c == "e":
        cpt = cpt + 1
print("trouvé", cpt, "e")
```

Algo : comptage du nombre de e dans la chaîne.

boucle sur dict/set  $\Leftrightarrow$  boucle sur séquence des clés

utilisation des tranches pour parcourir un sous-ensemble d'une séquence

index négatif	-5	-4	-3	-2	-1
index positif	0	1	2	3	4
	10	20	30	40	50
tranche positive	0	1	2	3	4
tranche négative	-5	-4	-3	-2	-1

Nombre d'éléments

`len(lst) → 5`index à partir de 0  
(de 0 à 4 ici)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]`

■ 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", )`  
Valeurs non modifiables (immuables)    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, immuables...)    `frozenset` ensemble immuable    vide

Séquences d'entiers

`range([début,] fin [,pas])`

`début` défaut 0, `fin` non compris dans la séquence  
`pas` signé et défaut 1

`range(5) → 0 1 2 3 4`  
`range(3, 8) → 3 4 5 6 7`  
`range(2, 12, 3) → 2 5 8 11`  
`Range(20, 5, -5) → 20 15 10`

■ `range` fournit une séquence d'entiers construits au besoin

Opérations sur listes

`len(lst) →` nb d'éléments  
`min(lst) →` Valeur mini de la liste  
`max(lst) →` Valeur maxi de la liste  
`sum(lst) →` Somme des éléments de la liste  
`sorted(lst) →` liste triée  
`val in c →` booléen  
`In :` test de présence  
`not in :` test d'absence

`lst*5 →` duplication  
`lst+lst2 →` concaténation  
`lst.index(val) →` position  
`lst.count(val) →` nb d'occurrences

`lst.append(val)` ajout d'un élément à la fin  
`lst.extend(seq)` ajout d'une séquence d'éléments à la fin  
`lst.insert(idx, val)` insertion d'un élément à une position  
`lst.remove(val)` suppression du premier élément de valeur `val`  
`lst.pop([idx]) → valeur` supp. & retourne l'item d'index `idx` (défaut le dernier)  
`lst.sort()`    `lst.reverse()` tri / inversion de la liste sur place

Opérations sur chaînes

`s.startswith(prefix[, début[, fin]]) →` Booléen : Test commence par `prefix`  
`s.endswith(suffix[, début[, fin]]) →` Booléen : Test termine par `suffix`  
`s.strip([caractères])` Retire du texte les caractères spécifiés  
`s.count(sub[, début[, fin]]) →` Nombre d'occurrences `sub`  
`s.find(sub[, début[, fin]]) →` Position de la chaîne `sub`  
`s.is_*` tests sur les catégories de caractères (ex. `s.isalpha()`)  
`s.upper()` → Texte avec lettres en majuscules  
`s.lower()` → Texte avec lettres en minuscules  
`s.title()` → Texte avec 1<sup>re</sup> lettre de chaque mot en majuscules  
`s.swapcase()` → Texte avec la case inversée (maj ↔ min)  
`s.capitalize()`  
`s.split([sep])` → Liste avec les éléments du texte localisés avec `sep`  
`str` découpée sur les blancs  
`"des mots espacés".split() → ['des', 'mots', 'espacés']`  
`str` découpée sur séparateur  
`"1,4,8,2".split(",") → ['1', '4', '8', '2']`

`s.join(seq) →` jonction de séquences

stockage de données sur disque, et lecture

Fichiers

`f = open("fic.txt", "w", encoding="utf8")`  
variable    nom du fichier    mode d'ouverture    encodage des caractères pour les fichiers textes:  
fichier pour les opérations    sur le disque (+chemin...)    `'r'` lecture (read)    utf8    ascii  
cf modules `os`, `os.path` et `pathlib`    `'w'` écriture (write)    latin1 ...  
`'a'` ajout (append)    `'+' 'x' 'b' 't'`

en écriture    en lecture  
`f.write("coucou")`    `f.read([n])` → caractères suivants  
`f.writelines(list de lignes)`    si `n` non spécifié, lit jusqu'à la fin !  
`f.readlines([n])` → `list` lignes suivantes  
`f.readline()` → ligne suivante

par défaut mode texte `t` (lit/écrit `str`), mode binaire `b`  
possible (lit/écrit `bytes`). Convertir de/vers le type désiré !

`f.close()` ne pas oublier de refermer le fichier après son utilisation !`f.flush()` écriture du cache    `f.truncate([taille])` retaillage

lecture/écriture progressent séquentiellement dans le fichier, modifiable avec :

`f.tell() → position`    `f.seek(position[, origine])`

Très courant : ouverture en bloc gardé (fermeture automatique) et boucle de lecture des lignes d'un fichier texte.  
`with open(...) as f:`  
`for ligne in f:`  
# traitement de ligne

`print("v=" 3, "cm :", x, ", ", y+4)` Affichage

éléments à afficher : valeurs littérales, variables, expressions

Options de `print`:

`sep=" "` séparateur d'éléments, défaut espace  
`end="\n"` fin d'affichage, défaut fin de ligne  
`file=sys.stdout` print vers fichier, défaut sortie standard

`s = input("Directives:")` Saisie

`input` retourne toujours une chaîne, la convertir vers le type désiré (cf. encadré Conversions au recto).