

Mémo Python Lycée

Types de base *entier, flottant, booléen, chaîne*

int 783 0 -192
float 9.23 0.0 -1.7e-6
bool True False
str "Un \n Deux" 'L \ 'âme'
 retour à la ligne échappe
 Multiligne possible {
 "X\tY\tZ
 1\t2 \t3"
 tabulation

Types Conteneurs

■ séquences ordonnées
list [1,5,9] ["x",11,8.9] ["mot"]
str "159" "x,11,8.9" "mot"
Comme séquence ordonnée de caractères
tuple (1,5,9) 11,"y",7.4 ("mot",)
 expression juste avec des virgules
 ■ Dictionnaire : accès par couple clé/valeur ; clés = types de base ou tuples
dict {"clé": "valeur"} {1:"un",2:"deux",3.14:"π"}
 sans ordre *a priori*,

Identificateurs

noms de variables, fonctions, modules, ...

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

Affectation de variables

x = 1.2+8+sin(0)
 ↑ valeur ou expression de calcul
 nom de variable (identificateur)
y,z,r = 9.2,-7.6,"bad"
 ↑ noms de variables conteneur de plusieurs valeurs (ici un tuple)

Conversions *type (expression)*

int("15") on peut spécifier la base du nombre entier en 2nd paramètre
int(15.56) tronque la partie décimale. **round**(15.56) donne l'arrondi
float(" -11.24e8")
str(78.3) et pour avoir la représentation littérale → **repr**("Texte")
voir au verso le formatage de chaînes, qui permet un contrôle fin
bool → utiliser des comparateurs (avec ==, !=, <, >, ...), résultat logique booléen
list("abc") → utilise chaque élément de la séquence en paramètre → ['a','b','c']
dict([(3,"trois"),(1,"un")]) → {1:'un',3:'trois'}
 "des mots espacés".**split**() → ['des','mots','espacés']
 "1,4,8,2".**split**(",") → ['1','4','8','2']
 chaîne de séparation

Indexation des séquences *pour les listes ou chaînes, ...*

lst=[A, B, C, D, E, F]
 0 1 2 3 4 5
len(**lst**) → 6
 Accès aux éléments par [index]
lst[0]→A **lst**[1]→B
 Accès à des sous-séquences par [tranche début : tranche fin : pas]
lst[1:3]→[B,C]
lst[:3]→[A,B,C] **lst**[4:]→[E,F]
lst[:]→[A,B,C,D,E,F] **lst**::2→[A,C,E]
 Modifications : uniquement sur les listes (ou séquences modifiables donc pas les chaînes).
del **lst**[3:5] Suppression
lst[1:4]=['hop',9] ou affectations

Logique booléenne

Comparateurs: < > <= >= == !=
 ≤ ≥ = ≠
a and b et logique
 les deux en même temps
a or b ou logique
 l'un ou l'autre ou les deux
not a non logique
True valeur constante vrai
False valeur constante faux

Blocs d'instructions

instruction parente:
 → bloc d'instructions 1...
 :
 instruction parente:
 → bloc d'instructions 2...
 :
 instruction suivante après bloc 1

bloc d'instructions exécuté **Instruction conditionnelle**
 uniquement si une condition est vraie

if expression logique:
 → bloc d'instructions
 combinable avec des sinon si, sinon si... et un seul sinon final, exemple :
if x==42:
 # bloc si expression logique x==42 vraie
 print("vérité vraie")
elif x>0:
 # bloc sinon si expression logique x>0 vraie
 print("positivons")
elif bTermine:
 # bloc sinon si variable booléenne bTermine vraie
 print("ah, c'est fini")
else:
 # bloc sinon des autres cas restants
 print("ça veut pas")

☞ nombres flottants... valeurs approchées !

Opérateurs: + - * / // % **
 × ÷ ↑ ↑
 ÷ entière reste ÷
 (1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6

angles en radians

Maths

from math import sin,pi...
sin(pi/4)→0.707...
cos(2*pi/3)→-0.4999...
acos(0.5)→1.0471...
sqrt(81)→9.0
log(e**2)→2.0 etc. (cf doc)

bloc d'instructions exécuté tant que la condition est vraie

Instruction boucle conditionnelle

while expression logique:

→ bloc d'instructions

s = 0
i = 1 initialisations avant la boucle

condition avec au moins une valeur variable (ici i)

while i <= 100:

bloc exécuté tant que $i \leq 100$

s = s + i2** faire varier la variable de condition !
i = i + 1

résultat de calcul après la boucle

print("somme:", s) attention aux boucles sans fin !

Contrôle de boucle

break sortie immédiate

continue itération suivante

$$s = \sum_{i=1}^{100} i^2$$

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

Instruction boucle itérative

for variable in séquence:

→ bloc d'instructions

Parcours des valeurs de la séquence

s = "Du texte" initialisations avant la boucle
cpt = 0 variable de boucle, valeur gérée par l'instruction **for**

for c in s:

if c == "e":

cpt = cpt + 1

Comptage du nombre de e dans la chaîne.

print("trouvé", cpt, "e")

boucle sur dict et sur séquence des clés

utilisation des tranches pour parcourir un sous-ensemble de la séquence

Parcours des index de la séquence

□ changement de l'élément à la position

□ accès aux éléments autour de la position (avant/après)

lst = [11,18,9,12,23,4,17]

perdu = []

for idx in range(len(lst)):

val = lst[idx]

if val > 15:

perdu.append(val)

lst[idx] = 15

Bornage des valeurs supérieures à 15, mémorisation des valeurs perdues.

print("modif:", lst, "modif:", perdu)

Parcours simultané index et valeur de la séquence:

for idx, val in enumerate(lst):

Affichage / Saisie

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

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=f** (print vers fichier, défaut sortie standard)

s = input("Directives:")

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

len(c) → nb d'éléments

min(c) **max(c)** **sum(c)**

sorted(c) → copie triée

val in c → booléen, opérateur in de test de présence (not in d'absence)

enumerate(c) → itérateur sur (index, valeur)

Spécifique aux conteneurs de séquences (listes, tuples, chaînes):

reversed(c) → itérateur inversé **c*5** → duplication **c+c2** → concaténation

c.index(val) → position **c.count(val)** → nb d'occurrences

Opérations sur conteneurs

Note: Pour dictionnaires et ensembles, ces opérations travaillent sur les clés.

modification de la liste originale

lst.append(item)

lst.extend(seq)

lst.insert(idx, val)

lst.remove(val)

lst.pop(idx) suppression de l'élément à une position et retour de la valeur

lst.sort() **lst.reverse()** tri / inversion de la liste sur place

ajout d'un élément à la fin

ajout d'une séquence d'éléments à la fin

insertion d'un élément à une position

suppression d'un élément à partir de sa valeur

Opérations sur listes

Opérations sur dictionnaires

d[clé]=valeur **d.clear()**

d[clé]→valeur **del d[clé]**

d.update(d2) mise à jour/ajout des couples

d.keys() vues sur les clés,

d.items() valeurs, couples

d.pop(clé)

Opérations sur ensembles

Opérateurs:

| → union (caractère barre verticale)

& → intersection

- ^ → différence/diff symétrique

< <= > >= → relations d'inclusion

s.update(s2)

s.add(clé) **s.remove(clé)**

s.discard(clé)

stockage de données sur disque, et lecture

Fichiers

f = open("fic.txt", "w", encoding="utf8")

variable nom du fichier mode d'ouverture encodage des

fichier pour sur le disque □ 'r' lecture (read) caractères pour les

les opérations (+chemin...) □ 'w' écriture (write) fichiers textes:

cf fonctions des modules os et os.path □ 'a' ajout (append)... utf8 ascii

latin1 ...

en écriture chaîne vide si fin de fichier en lecture

f.write("coucou") **s = f.read(4)** si nb de caractères

fichier texte → lecture / écriture de chaînes uniquement, convertir de/vers le type désiré

lecture ligne suivante

s = f.readline()

ne pas oublier de refermer le fichier après son utilisation !

Fermeture automatique Pythonnesque : **with open(...)** as **f:**

très courant : boucle itérative de lecture des lignes d'un fichier texte :

for ligne in f:

→ bloc de traitement de la ligne

Génération de séquences d'entiers

très utilisé pour les boucles itératives for par défaut 0 non compris

range([début,] fin [,pas])

range(5) → 0 1 2 3 4

range(3, 8) → 3 4 5 6 7

range(2, 12, 3) → 2 5 8 11

range retourne un « générateur », faire une conversion

en liste pour voir les valeurs, par exemple:

print(list(range(4)))

Définition de fonction

nom de la fonction (identificateur) paramètres nommés

def nomfct(p_x, p_y, p_z):

"""documentation"""

bloc instructions, calcul de res, etc.

return res valeur résultat de l'appel.

les paramètres et toutes les variables de ce bloc n'existent si pas de résultat calculé à retourner : **return None**

que dans le bloc et pendant l'appel à la fonction (« boîte noire »)

Appel de fonction

r = nomfct(3, i+2, 2*i)

un argument par paramètre

récupération du résultat retourné (si nécessaire)

Formatage de chaînes

directives de formatage valeurs à formater

"modele{} {} {}".format(x, y, r) → **str**

"{sélection:formatage!conversion}"

□ Sélection :

2 → **45.727**

x → **toto**

0.nom → **toto**

4[clé] → **toto**

0[2] → **toto**

□ Formatage :

car-repl. alignement signe larg.mini.précision-larg.max type

<>^+ - espace 0 au début pour remplissage avec des 0

entiers: **b** binaire, **c** caractère, **d** décimal (défaut), **o** octal, **x** ou **X** hexa...

flottant: **e** ou **E** exponentielle, **f** ou **F** point fixe, **g** ou **G** approprié (défaut)

% pourcentage

chaîne: **s** ...

□ Conversion : **s** (texte lisible) ou **r** (représentation littérale)