## **Mémento Python 3**

Version originale sous licence CC4 https://perso.limsi.fr/pointal/python:memento

```
entier, flottant, booléen, chaîne
                           Types de base
   int 783 0 -192
                       0b010
                                0xF3
             zéro
                       binaire
                                 hexa
float 9.23 0.0
                    -1.7e-6
 bool True False
   str "Un\nDeux"
                               retour à la ligne
```

```
type (expression) Conversions
int("15") \rightarrow 15
str(12) \rightarrow "12"
int(15.56) \rightarrow 15
                                 troncature de la partie décimale
list("abc") → ["a", "b", "c"]
":".join(["toto","12","pswd"]) \rightarrow "toto:12:pswd"
chr(64) \rightarrow "@" ord("@") \rightarrow 64
                                             code ↔ caractère
```

```
pour noms de variables, Identificateurs
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
```

```
Aide
help(help)
affiche aussi votre
propre documenta-
tion
dir (truc)
```

```
• séquences ordonnées, accès par index
                                                             Types conteneurs
tuple (1,5,9) Valeurs non modifiables (immutables)
list [1,5,9] Valeurs modifiables (mutables)
                                                                      [] liste vide
 liste en compréhension :
 [i ** 2 for i in range(5)] \rightarrow [0,1,4,9,16]
 [i for i in range(7) if i % 2 == 0] \rightarrow [0,2,4,6]
■ conteneurs clés, sans ordre à priori, accès par clé rapide, chaque clé unique dictionnaire dict {"clé":"valeur"} Mutables {} dictionnaire v
                                                               { } dictionnaire vide
```

```
1) évaluation de la valeur de l'expression de droite
 2) affectation dans l'ordre avec les noms de gauche
\mathbf{x} = 1.2 + 8 * \mathbf{y}

\mathbf{y}, \mathbf{z}, \mathbf{r} = 9.2, -7, 0 affectations multiples
a, b = b, a échange de valeurs
```

2 range fournit une séquence immutable d'entiers construits au besoin

affectation ⇔ association d'un nom à une valeur

Variables & affectation

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

range  $(2, 12, 3) \rightarrow 25811$ 

range (20, 5, -5)  $\rightarrow$  20 15 10

```
pour les listes, chaînes de caractères, tuples ... Indexation conteneurs séquences
                                   3
 index positif
                                                  Sur les séquences modifiables
       lst=[10, 20, 30, 40, 50] (list) modification par
                                                  affectation lst[4]=25
Accès individuel aux éléments par 1st [index]
                    Nombre d'éléments
lst[0] \rightarrow 10
                                         1st.append (val) ajout d'un élément à
                                                              la fin
                    len(1st) \rightarrow 5
1st[1]→20
```

chaque élément d'un conteneur ou d'un itérateur

print("trouvé", cpt, "'e'")

uniquement si sa condition est vraie **if** condition logique:

nom de la fonction (identificateur)

def fct(x, y, z):

paramètres nommés

"""documentation"""

# bloc instructions, calcul de res, etc.

```
nombres flottants... valeurs approchées! Maths
Opérateurs : + - * / // % **
                   \times \div \qquad \uparrow \qquad \uparrow \qquad a^b
Priorités (...)
                     ÷ entière reste ÷
(1+5.3)*2\rightarrow12.6 abs(-3.2)\rightarrow3.2
from math import sin, pi...
\sin(pi/4) \to 0.707...
\cos(2*pi/3) \rightarrow -0.4999...
sqrt(81) \rightarrow 9.0
```

range ([début,] fin [,pas])

range (5)  $\rightarrow$  0 1 2 3 4

range  $(3, 8) \rightarrow 34567$ 

## Op. sur dictionnaires

Séquences d'entiers

```
d[cl\acute{e}] = valeur
d[cl\acute{e}] \rightarrow valeur
                          vues itérables sur
d.keys() les clés
d.values() valeurs
d.items() couples
d.get(clé[,défaut]) \rightarrow valeur
```

suivant . for var in séquence: bloc d'instructions Parcours des valeurs d'un conteneur s = "Du texte" | initialisations avant la boucle variable de boucle, affectation gérée par l'instruction for for c in s: if c == "e": Algo: comptage cpt = cpt + 1

bloc d'instructions exécuté pour Instruction boucle itérative

```
Instruction boucle conditionnelle
   bloc d'instructions exécuté
   tant que la condition est vraie
boucles sans
      while condition logique:
             bloc d'instructions
  s = 0
i = 1 initialisations avant la boucle
condition avec au moins une valeur variable (ici i)
  while i <= 100:
```

```
bonne habitude : ne pas modifier la variable de boucle
                              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)
```

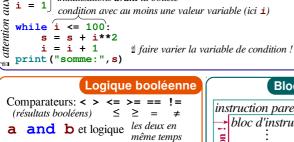
un bloc d'instructions exécuté, Instruction conditionnelle

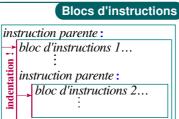
du nombre de  $\in$ 

dans la chaîne.

Définition de fonction

fct

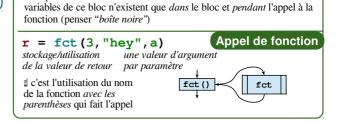




→ bloc d'instructions Combinable avec des sinon si, sinon si... et un seul sinon final. Seul le bloc de la if age<=18: première condition trouvée vraie est etat="Enfant" exécuté. elif age>65: etat="Retraité" ₫ avec une variable x: else: if bool(x)==True: ⇔ if x: etat="Actif" if bool(x) == False:  $\Leftrightarrow$  if not x:

```
a or b ou logique l'un ou l'autre ou les deux
g piège : and et or retournent la valeur de
a ou de b (selon l'évaluation au plus court).
\Rightarrow s'assurer que a et b sont booléens.
not a
                 non logique
 True
                 constantes Vrai/Faux
False
```

```
instruction suivante après bloc 1
🖠 régler l'éditeur pour insérer 4 espaces à
la place d'une tabulation d'indentation.
```



return res← valeur résultat de l'appel, si rien

à retourner return None

