

Programmation Python

Tuples Listes Dictionnaires

Dr. Yousfi Souheib

29 octobre 2020

Les tuples

```
#définition d'un tuple
```

```
t1 = (2,6,8,10,15,26)
```

Les `()` sont importantes pour indiquer qu'il s'agit d'un tuple, « , » sépare les éléments.

```
print(t1)
```

(2,6,8,10,15,26)

```
#taille du tuple
```

```
print(len(t1))
```

6 éléments

```
#accès indicé
```

```
a = t1[0]
```

1^{er} élément, les indices vont de 0 à `len(t1)-1`

```
print(a)
```

Remarque : `a` n'est pas un tuple

```
#modification ?
```

```
t1[2] = 3
```

ERREUR

```
#plage d'indices
```

```
b = t1[2:5]
```

Attention : on récupère à partir du n°2 (inclus) au n°5

```
print(b)
```

(non-inclus) c.-à-d. les indices 2, 3, 4

Résultat : `b` est un tuple avec (8,10,15)

```
#autre plage
```

```
c = t1[:4]
```

```
print(c)
```

Les 4 premiers éléments c.-à-d. les indices 0, 1, 2, 3 : nous obtenons le tuple (2, 6, 8, 10).

```
#indexage négatif
```

```
d = t1[-1]
```

```
print(d)
```

Le 1^{er} élément à partir de la fin : 26

```
#indexage négatif
```

```
e = t1[-3:]
```

```
print(e)
```

Les 3 derniers éléments : (10,15,26)

Les tuples

#concaténation

t2 = (7, 9,31)

t3 = t1 + t2

print(t3)

(2,6,8,10,15,26,7,9,31)

#réplication

t4 = 2 * t2

print(t4)

(7,9,31,7,9,31)

Ca ne pose absolument aucun problème.

#tuples d'objets hétérogènes

v1 = (3,6,"toto",True,34.1)

print(v1)

#tuple de tuples

x = ((2,3,5),(6,7,9),(1,8))

print(x)

Sorte de tableau à 2 dimensions

↓

x[0] → (2,3,5)

x[1] → (6,7,9)

x[2] → (1,8)

Organisation de la structure

#accès indicé

print(x[2][1])

→ 8

#accès aux tailles

print(len(x))

→ 3

print(len(x[2]))

→ 2

3 éléments sur la 1^{ère} dimension

2 éléments dans le tuple référencé par x[2]

Liste \approx tuple de **taille dynamique** et **modifiable**

```
#définition d'une liste
L1 = [2,6,8,10,15,26]
print(L1)

#taille de la tuple = 6
print(len(L1))

#accès indicé = 2
a = L1[0]
print(a)

#modification ! Possible !
L1[2] = 3
print(L1)
```

Les `[]` sont importantes pour indiquer qu'il s'agit d'une liste, « , » sépare les éléments.

`[2,6,8,10,15,26]`

`[2,6,3,10,15,26]`

Les autres mécanismes associés aux tuples sont transposables aux listes :

- plages d'indices
- indexages négatifs
- objets hétérogènes
- liste de listes (tableaux 2D ou +)
- concaténation, réplication

Les listes

```
#autre liste  
L2 = [32,69,28,69]
```

Une liste est un objet (instance de classe) auquel est associé des méthodes permettant de le manipuler.

```
#ajout
```

```
L2.append(21)
```

```
print(L2)
```

[32,69,28,69,21]

```
#insertion à l'indice 1
```

```
L2.insert(1,53)
```

```
print(L2)
```

[32,53,69,28,69,21]

```
#suppression elt n°3
```

```
del L2[3]
```

```
print(L2)
```

[32,53,69,69,21]

28 a disparu de L2

```
#accès + suppression elt n°1
```

```
a = L2.pop(1)
```

```
print(a)
```

→ renvoie 53

[32,69,69,21]

53 a disparu de L2

```
#inversion
```

```
L2.reverse()
```

```
print(L2)
```

[21,69,69,32]

```
#étendre
```

```
L2.extend([34,55])
```

```
print(L2)
```

[21,69,69,32,34,55]

Remarque :

L2.clear()

Permet de vider la liste

Générer une liste à partir d'une autre liste

Exemple 1 : Monter tous les chiffres au carré

```
source = [1,5,8,12,7]
resultat = []
for v in source:
    resultat.append(v**2)
print(resultat)
```



```
resultat = [v**2 for v in source]
print(resultat)
```

La convention d'écriture nous facilite la tâche !!!

Exemple 2 : Actions conditionnelles

```
source = [1,5,8,12,7]
resultat = []
for v in source:
    if (v % 2 == 0):
        resultat.append(v**2)
print(resultat)
```



```
resultat = [v**2 for v in source if (v % 2 == 0)]
print(resultat)
```

traitement par le contenu dans une liste

L2 = [21,69,69,32,34,55]

```
#recherche d'élément  
trouve = 32 in L2  
print(trouve)
```

Renvoie True puisque la valeur
32 se trouve dans la liste

```
#index  
id = L2.index(34)  
print(id)
```

Renvoie 4 puisque la valeur 34 apparaît
à l'indice n°4 (indice du 1^{er} trouvé)

```
#comptage  
nb = L2.count(69)  
print(nb)
```

Renvoie 2 puisque la valeur 69 apparaît
2 fois dans la liste

```
#retrait par valeur  
L2.remove(69)  
print(L2)
```

Retire la valeur 69 de la liste, la
première que la méthode trouvera

[21,69,32,34,55]



Ce mécanisme fonctionne avec tout type d'objet pourvu
qu'une comparaison soit possible (ex. chaîne, etc.)

Les affectations dans une liste

```
#L3
L3 = [61,92,17]
print(L3)
#affectation ?
L4 = L3
print(L4)
#modification d'une valeur
L4[1] = 55
#répercussions
print(L4)      → [61,55,17]
#mais aussi sur L3
print(L3)      → [61,55,17] ???
```

En réalité, c'est la référence qui est copiée.

L3 et L4 « pointent » au même endroit.

```
#L3
L3 = [61,92,17]

#copie des valeurs
L4 = L3.copy()
print(L4)

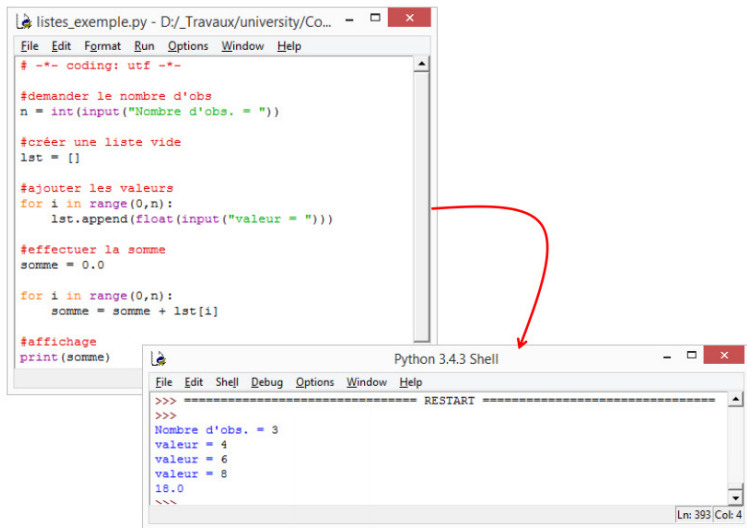
L4[1] = 55
print(L4)      → [61,55,17]

print(L3)      → [61,92,17] !!!
```

L4 référence une nouvelle zone mémoire,
et les données de L3 y sont recopiées.

L3 n'est pas impacté.

Exemple



The image shows a Python script named `listes_exemple.py` and its execution in a `Python 3.4.3 Shell`. A red arrow points from the script to the shell output.

Script Content:

```
# -*- coding: utf -*-

#demander le nombre d'obs
n = int(input("Nombre d'obs. = "))

#créer une liste vide
lst = []

#ajouter les valeurs
for i in range(0,n):
    lst.append(float(input("valeur = ")))

#effectuer la somme
somme = 0.0

for i in range(0,n):
    somme = somme + lst[i]

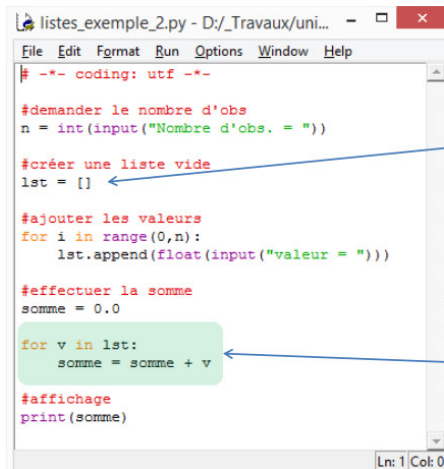
#affichage
print(somme)
```

Shell Output:

```
>>> ===== RESTART =====
>>>
Nombre d'obs. = 3
valeur = 4
valeur = 6
valeur = 8
18.0
>>>
```

The shell output shows the user inputting 3 for the number of observations, followed by three values (4, 6, 8) for the list. The final output is 18.0, which is the sum of the list elements.

Exemple amélioré



```
listes_exemple_2.py - D:/Travaux/uni...  
File Edit Format Run Options Window Help  
# -*- coding: utf -*-  
  
#demander le nombre d'obs  
n = int(input("Nombre d'obs. = "))  
  
#créer une liste vide  
lst = []  
  
#ajouter les valeurs  
for i in range(0,n):  
    lst.append(float(input("valeur = ")))  
  
#effectuer la somme  
somme = 0.0  
  
for v in lst:  
    somme = somme + v  
  
#affichage  
print(somme)  
Ln: 1 Col: 0
```

Permet de définir une liste initialement vide.

Une liste est directement « itérable », il n'est pas nécessaire de passer par un indice (un peu comme le foreach de certains langages de prog.).

Les chaînes de caractères

```
#définir une chaîne
s1 = "bonjour le monde"
print(s1)
#longueur
long = len(s1)
print(long)
#accès indicé
s2 = s1[:7]
print(s2)
#non modifiable
s1[0] = "B"
#méthodes associées
S = s1.upper()
print(S)
#recherche d'une sous-chaîne
id = S.find("JO")
print(id) → 3 (1ère occurrence si plusieurs)
#nb d'occurrences
nb = S.count("ON")
print(nb) → 2
#remplacement de « O » par « A »
SA = S.replace("O", "A")
print(SA)
```

Guillemets pour délimiter une chaîne

Mécanisme identique aux
tuples et listes

ERREUR. Une chaîne n'est pas modifiable. Il faut mettre
le résultat d'une manipulation dans une autre chaîne.

Des méthodes spécifiques permettent
de manipuler les chaînes. Cf.
<https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str>

Les chaînes de caractères

Une chaîne peut être transformée en liste pour réaliser des traitements sophistiqués. L'outil est très souple.

```
#transf. en liste  
liste = list(S)  
print(liste)
```

`['B','O','N','J','O','U','R',' ','L','E',' ','M','O','N','D','E']`
Toutes les opérations sur les listes sont possibles par la suite.

```
#découpage par séparateur  
decoupe = S.split(" ")  
print(decoupe)
```

`['BONJOUR', 'LE', 'MONDE']`

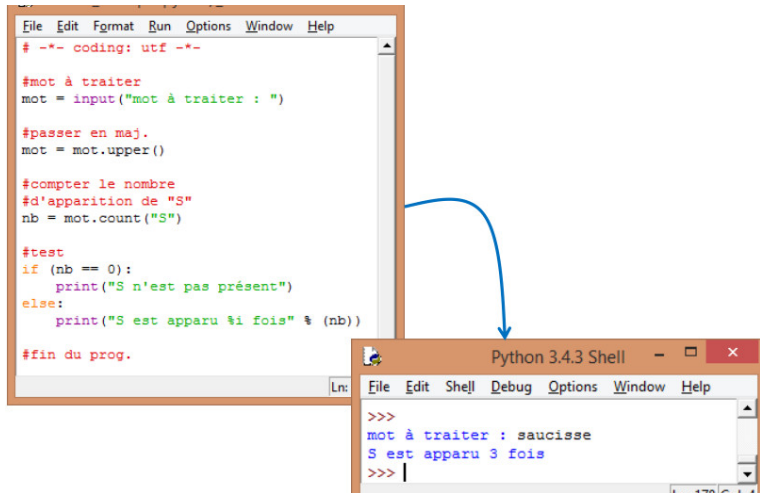
Espace est utilisé comme séparateur ici, mais ça peut être tout autre carac., y compris un caractère spécial (ex. \t pour tabulation)

```
#former une chaîne à  
#partir d'une liste  
SB = "-".join(decoupe)  
print(SB)
```

`"BONJOUR-LE-MONDE"`

Les mots de la liste ont été fusionnés avec le séparateur "-". Tout séparateur est possible, y compris la chaîne vide.

Les chaînes de caractères



The image shows a Python IDE window with a menu bar (File, Edit, Format, Run, Options, Window, Help) and a code editor. The code in the editor is as follows:

```
# -*- coding: utf -*-  
  
#mot à traiter  
mot = input("mot à traiter : ")  
  
#passer en maj.  
mot = mot.upper()  
  
#compter le nombre  
#d'apparition de "S"  
nb = mot.count("S")  
  
#test  
if (nb == 0):  
    print("S n'est pas présent")  
else:  
    print("S est apparu %i fois" % (nb))  
  
#fin du prog.
```

A blue arrow points from the code editor to a "Python 3.4.3 Shell" window. The shell window has a menu bar (File, Edit, Shell, Debug, Options, Window, Help) and shows the execution of the script:

```
>>>  
mot à traiter : saucisse  
S est apparu 3 fois  
>>> |
```

Les dictionnaires

```
#définition d'un dictionnaire
d1 = {'Pierre':17, 'Paul':15, 'Jacques':16}
print(d1)
#ou
print(d1.items())

#nombre d'éléments
print(len(d1))

#liste des clés
print(d1.keys())

#liste des valeurs
print(d1.values())

#accès à une valeur par clé
print(d1['Paul'])
#ou
print(d1.get('Paul'))

#si clé n'existe pas
print(d1['Pipa'])
```

Noter le rôle de {}, de « : »
et « , »

→ 3 éléments

['Paul', 'Jacques', 'Pierre']

[15, 16, 17]

→ 15

→ 15

→ ERREUR

Dictionnaire : collection
non-ordonnée (non
indicée) d'objets (simples
ou évolués) s'appuyant sur
le mécanisme associatif
« clé – valeur ».

Remarques :

- 1) **d1.clear()** vide
le dictionnaire
- (2) **d1** est une référence,
d1.copy() permet de
copier le contenu.

Les dictionnaires

```
#modification
d1['Jacques'] = 18
print(d1)

#ajouter un élément
d1['Henri'] = 22
print(d1)

#ajout d'un bloc d'éléments
d1.update({'Monica':36,'Bill':49})
print(d1)

#détecter présence clé
test = 'Pierre' in d1
print(test)

#suppression par clé
del d1['Monica']
print(d1)
```

← {'Pierre':17, 'Paul':15, 'Jacques':16} → {'Pierre':17, 'Paul':15, 'Jacques':18}

Ajout par définition d'une nouvelle paire « clé – valeur ». N.B. : Si 'Henri' existe déjà, son ancienne valeur sera écrasée.

← {'Pierre':17, 'Paul':15, 'Jacques':18, 'Henri':22}

← {'Pierre':17, 'Paul':15, 'Jacques':18, 'Henri':22, 'Monica':36, 'Bill':49}

→ True

← {'Pierre':17, 'Paul':15, 'Jacques':18, 'Henri':22, 'Bill':49}

Les dictionnaires Exemple

```
dico_exemple.py - D:/Travaux/university/Cours_...
File Edit Format Run Options Window Help

# -*- coding: utf -*-

#nombre d'items
n = int(input("Nombre d'items : "))

#dictionnaire vide
dico = {}

#saisie
for i in range(0,n):
    cle = input("Clé : ")
    valeur = float(input("Valeur : "))
    dico[cle] = valeur

#liste des clés
print(dico.keys())

#liste des valeurs
print(dico.values())

#affichage par paire
for (k,v) in dico.items():
    print(k,v)

#somme des valeurs
s = 0
for v in dico.values():
    s = s + v
print(s)
```

Accès indicé malaisé
(très), on a intérêt à
passer par un itérateur
pour passer en revue le
dictionnaire.

Exemple :

Kate	15.0
Pipa	23.5
William	10.7

49.2