Collections d'objets sous Python Tuples, listes, dictionnaires

Ricco Rakotomalala

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_python.html

Tableau statique en lecture seule d'objets hétérogènes

LES TUPLES

Création des tuples et accès aux données

```
Les () sont importantes pour indiquer qu'il
#définition d'un tuple
t1 = (2,6,8,10,15,26) \leftarrow
                                        s'agit d'un tuple, «, » sépare les éléments.
print(t1) \leftarrow
                                                          (2,6,8,10,15,26)
#taille du tuple
                               6 éléments
print(len(t1)) \leftarrow
#accès indicé
                                1<sup>er</sup> élément, les indices vont de 0 à len(t1)-1
a = t1[0]
                                Remarque : a n'est pas un tuple
print(a)
#modification ?
                                           ERREUR
t1[2] = 3
#plage d'indices
                                 Attention : on récupère à partir du n°2 (inclus) au n°5
b = t1[2:5]
                                 (non-inclus) c.-à-d. les indices 2, 3, 4
print(b)
                                 Résultat : b est un tuple avec (8,10,15)
#autre plage
c = t1[:4]
                              Les 4 premiers éléments c.-à-d. les indices 0, 1, 2, 3 : nous
print(c)
                              obtenons le tuple (2, 6, 8, 10).
#indiçage négatif
d = t1[-1] \leftarrow
                                        Le 1<sup>er</sup> élément à partir de la fin : 26
print(d)
#indiçage négatif
e = t1[-3:]
                                           Les 3 derniers éléments : (10,15,26)
print(e)
```

Plus loin avec les tuples

```
#concaténation
                                            (2,6,8,10,15,26,7,9,31)
t2 = (7, 9, 31)
t3 = t1 + t2
print(t3) <
                                           (7,9,31,7,9,31)
#réplication
t4 = 2 * t2
print(t4)
                                                     Ca ne pose absolument aucun
                                                     problème.
#tuples d'objets hétérogènes
v1 = (3, 6, "toto", True, 34.1)
print(v1)
                                                    Sorte de tableau à 2 dimensions
#tuple de tuples
x = ((2,3,5), (6,7,9), (1,8))
print(x)
                                                x[0] \rightarrow (2,3,5)
                                                                  Organisation de
#accès indicé
                                                                  la structure
                                                x[1] \rightarrow (6,7,9)
print(x[2][1])
                           \rightarrow 8
                                                x[2] \rightarrow (1,8)
#accès aux tailles
                                              3 éléments sur la 1ère dimension
                           \rightarrow 3
print(len(x))
                                              2 éléments dans le tuple référencé par x[2]
print(len(x[2]))
                           \rightarrow 2
```

Bilan sur les tuples

- Type 'tuple' cf. appel de la fonction type()
- Collection d'objets de types hétérogènes
- Taille et contenu fixé lors de l'écriture du programme
- Impossible de modifier : objet non mutable
- La variable de type tuple est en réalité une référence (pointeur de pointeur)
- Bénéficie du mécanisme de ramasse miettes
- Accès indicé, plage d'indices possible, indices négatifs possibles aussi
- Structures complexes avec tuple de tuples, et même plus à voir plus tard

Tableau dynamique en lecture-écriture d'objets hétérogènes

LES LISTES

Liste – Le type list

Liste ≈ 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.

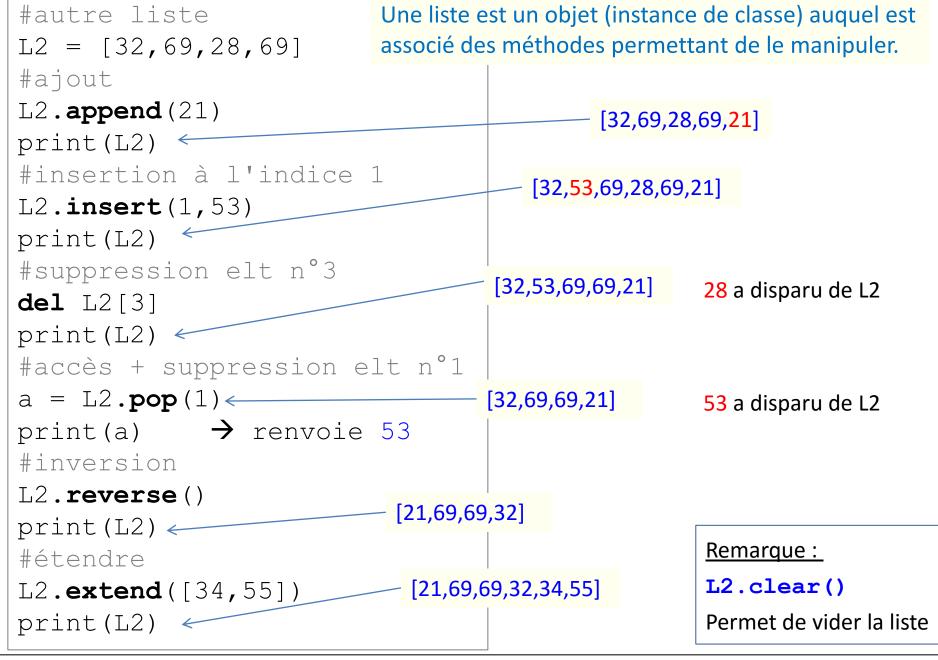
[2,6,8,10,15,26]

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

Les autres mécanismes associés aux tuples sont transposables aux listes :

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

Plus loin avec les listes : modification de taille et de contenu



Les « List Comprehensions » (traduite en « listes en intension »)

Objectif : un mécanisme simple (et concis) pour 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)

Pour aller plus loin: https://docs.python.org/3.4/tutorial/datastructures.html#list-comprehensions-https://sametmax.com/python-love-les-listes-en-intention-partie/

Plus loin avec les listes : traitement par le contenu

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

```
#recherche d'élément
                                          Renvoie True puisque la valeur
trouve = 32 in L2
                                          32 se trouve dans la liste
print(trouve)
#index
                                     Renvoie 4 puisque la valeur 34 apparaît
id = L2.index(34)
                                     à l'indice n°4 (indice du 1er trouvé)
print(id)
#comptage
                                       Renvoie 2 puisque la valeur 69 apparaît
nb = L2.count(69)
                                       2 fois dans la liste
print(nb)
                                    Retire la valeur 69 de la liste, la
#retrait par valeur
                                    première que la méthode trouvera
L2.remove(69)
print(L2) ←
                                  [21,69,32,34,55]
```

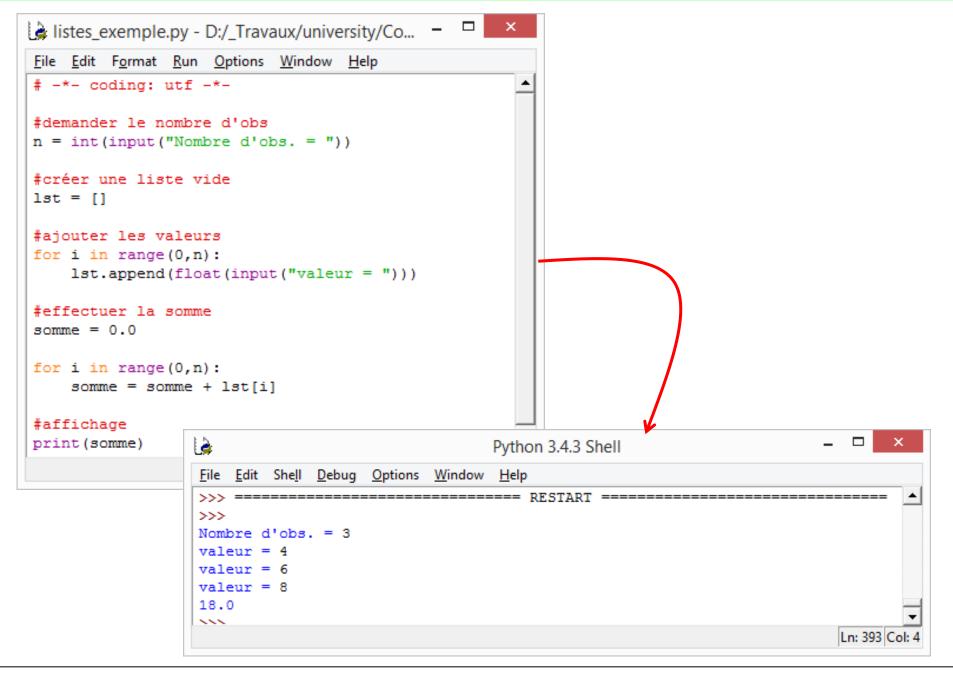


Ce mécanisme fonctionne avec tout type d'objet pourvu qu'une comparaison soit possible (ex. chaîne, etc.)

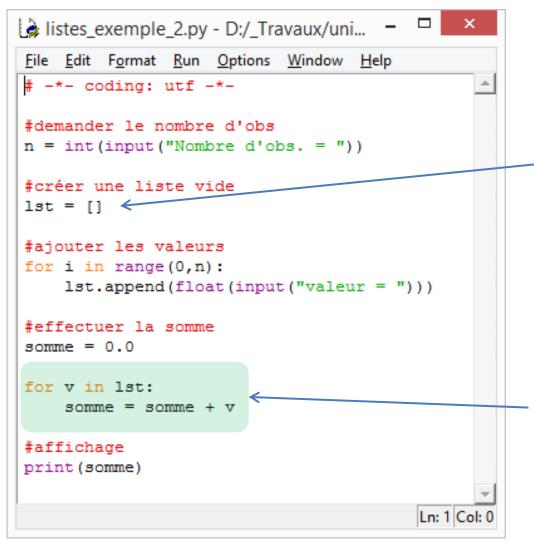
Une variable de type liste est une référence

```
#L3
L3 = [61, 92, 17]
print(L3)
                                 En réalité, c'est la référence qui est copiée.
#affectation ?
L4 = L3
                                 L3 et L4 « pointent » au même endroit.
print(L4)
#modification d'une valeur
L4[1] = 55
#répercussions
print(L4) \rightarrow [61,55,17]
#mais aussi sur L3
print(L3) \rightarrow [61,55,17] ???
#L3
L3 = [61, 92, 17]
#copie des valeurs
                              L4 référence une nouvelle zone mémoire,
L4 = L3.copy()
                              et les données de L3 y sont recopiées.
print(L4)
L4[1] = 55
print (L4) \rightarrow [61,55,17]
                                               L3 n'est pas impacté.
print(L3) \rightarrow [61,92,17] !!!
```

Un exemple : somme de valeurs saisies par l'utilisateur



Un exemple(bis) : boucle directe sur les éléments de la liste



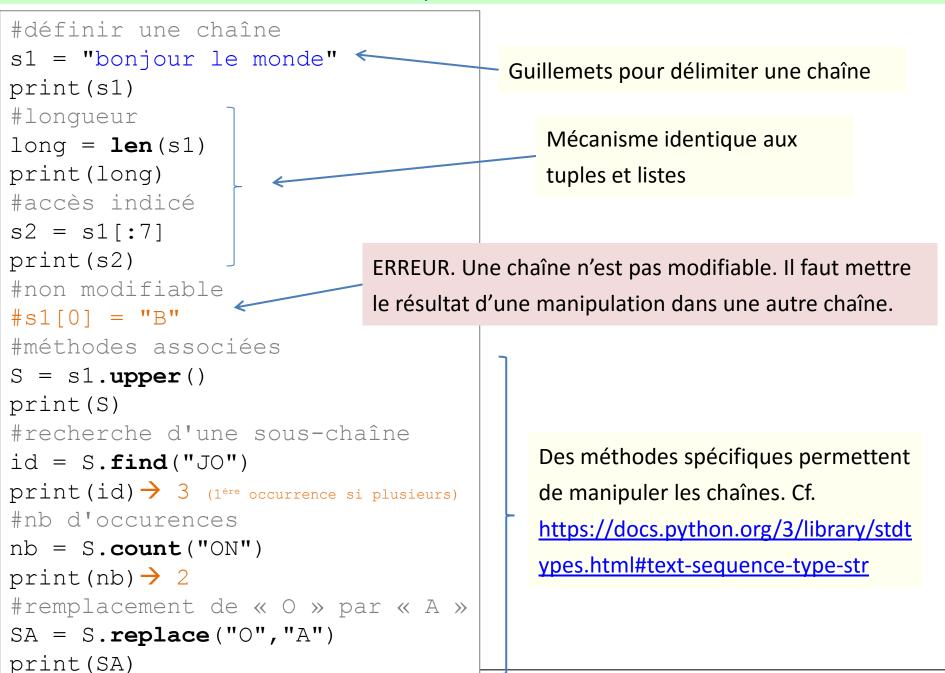
Permet de définir une liste initialement vide.

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.).

Un cas particulier de liste

CHAÎNE DE CARACTÈRES

Une chaîne de caractères est une liste particulière avec des méthodes associées



15

Transformation explicite en liste (pour traitements)

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) <</pre>
#découpage par séparateur
decoupe = S.split(" ")
print(decoupe)
#former une chaîne à
#partir d'une liste
SB = "-".join(decoupe)
print(SB)
```

```
['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.

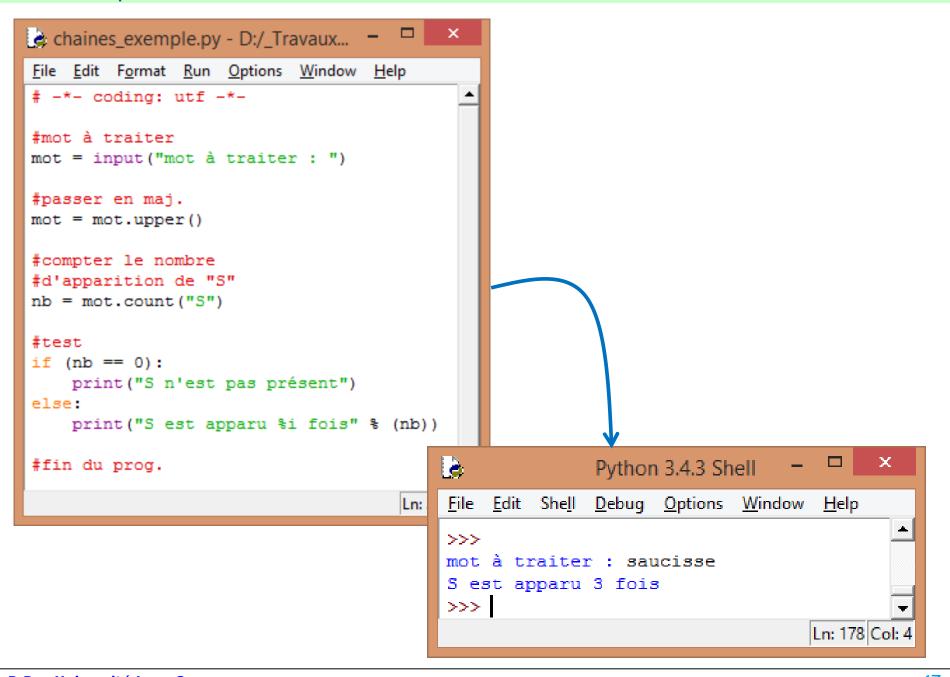
```
['BONJOUR','LE,'MONDE']
```

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

```
"BONJOUR-LE-MONDE"
```

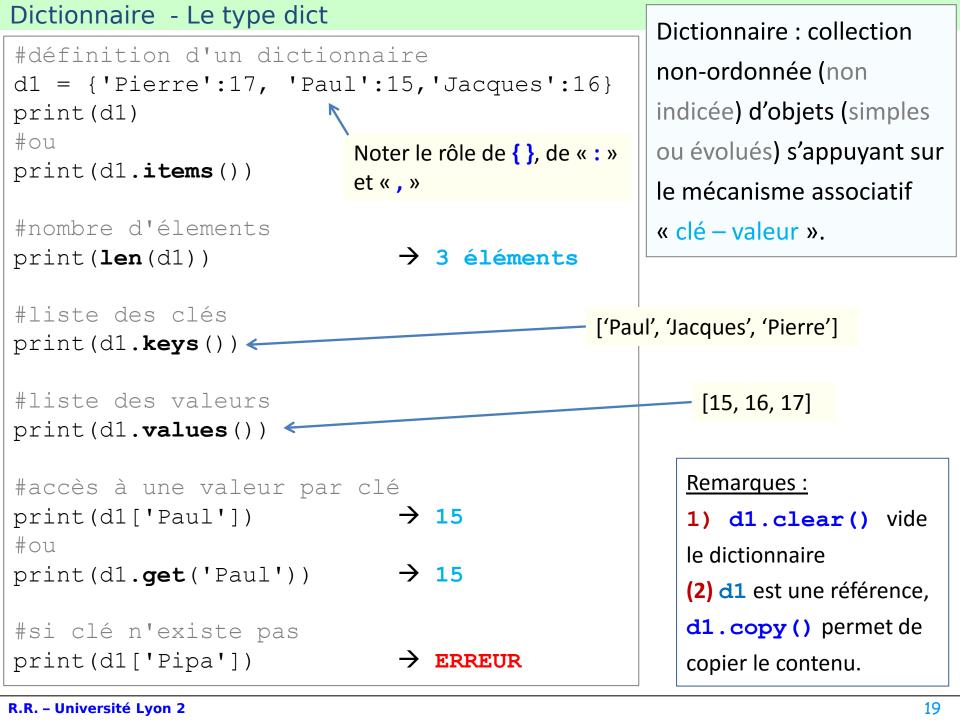
Les mots de la liste ont été fusionnés avec le séparateur "-". Tout séparateur est possible, y compris la chaîne vide.

Un exemple



Liste avec accès par clé

LES DICTIONNAIRES



Dictionnaire – Modifications, ajouts et suppressions

```
#modification
d1 [ 'Jacques' ] = 18 \checkmark {'Pierre':17, 'Paul':15,'Jacques':16} \rightarrow {'Pierre':17, 'Paul':15,'Jacques':18}
print(d1)
                                          Ajout par définition d'une nouvelle paire « clé –
                                          valeur ». N.B.: Si 'Henri' existe déjà, son ancienne
                                          valeur sera écrasée.
#ajouter un élément
d1['Henri'] = 22
                                        {'Pierre':17, 'Paul':15, 'Jacques':18, 'Henri':22}
print(d1)
#ajout d'un bloc d'éléments
d1.update({'Monica':36,'Bill':49})
print(d1) ←
                              {'Pierre':17, 'Paul':15,'Jacques':18, 'Henri':22, 'Monica':36, 'Bill': 49}
#détecter présence clé
test = 'Pierre' in d1
print(test)
                                    → True
#suppression par clé
del d1['Monica']
print (d1) < \(\frac{\text{Pierre}':17, 'Paul':15, 'Jacques':18, 'Henri':22, 'Bill': 49}\)
```

Plus loin avec les clés

Les clés ne sont pas forcément des chaînes de caractères. L'outil est très souple mais, attention, autant de liberté peut être aussi préjudiciable. Il faut être très rigoureux.

```
#autre type de clé
d2 = {('Pierre',56):['Directeur',1253,True],('Paul',55):['Employé',100,False]}
print(d2.keys())
print(d2.values())
```

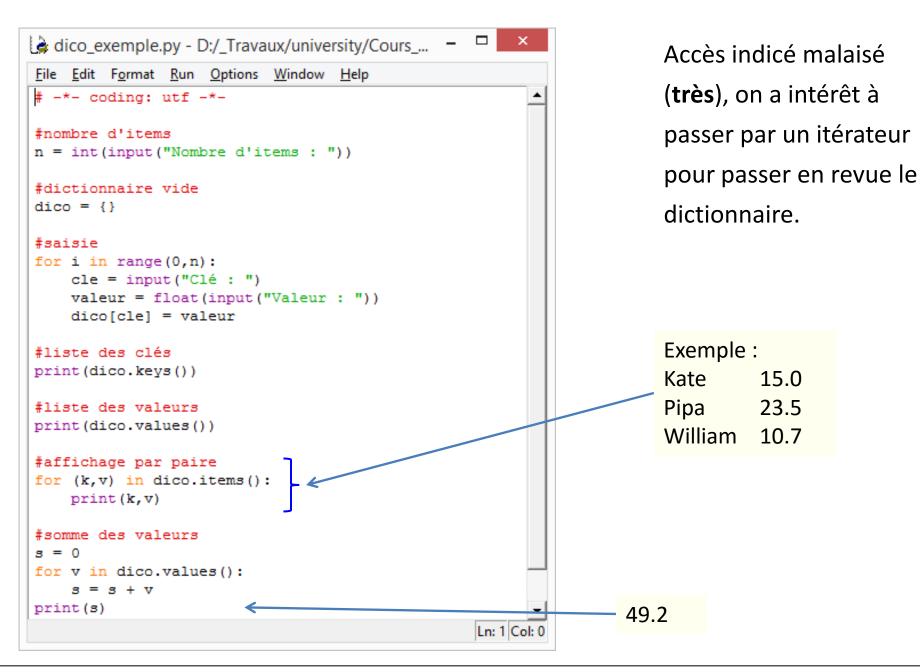
Dans cet exemple:



- clé est un tuple ;
- valeur est une liste.

D'autres types sont possibles (ex. instances de classes – à voir plus loin).

Exemple



Références

De la documentation à profusion (n'achetez pas des livres sur Python)

Site du cours

http://eric.univ-lyon2.fr/~ricco/cours/cours programmation python.html

Site de Python

Welcome to Python - https://www.python.org/

Python 3.4.3 documentation - https://docs.python.org/3/index.html

Portail Python

Page Python de <u>Developpez.com</u>

Quelques cours en ligne

P. Fuchs, P. Poulain, « Cours de Python » sur Developpez.com

G. Swinnen, « Apprendre à programmer avec Python » sur Developpez.com

« Python », Cours interactif sur Codecademy

POLLS (KDnuggets)

Data Mining / Analytics Tools Used

Python, 4^{ème} en 2015

What languages you used for data mining / data science?

Python, 3^{ème} en 2014 (derrière R et SAS)