Python : les collections

DAFS : Algorithmique et programmation

Xavier Crégut <prenom.nom@enseeiht.fr>

Motivation

Un programme manipule des données et généralement des **groupes de données**.
Il est donc important de disposer de groupes de données équipés des **opérations et** du **comportement appropriés** pour une classe de problèmes :

- connaître le nombre de caractères différents utilisés dans un texte
- connaître la fréquence (le nombre d'occurrences) des mots d'un texte.
- conserver les travaux qui devront être réalisés (par une imprimante, par un agent, etc.)
- pouvoir savoir quelles personnes occupent quels bureaux.
- et, bien sûr, faire toutes ces opérations de manière efficace !

Rappels

On a déjà vu :

- séquence immuable :
  - tuple (**tuple**)
  - chaîne (**str**)
- séquence modifiable :
  - liste (**list**)

Voir Data Structures in The Python Tutorial
Voir collections.abc – Abstract Base Classes for Containers : hiérarchie des types et opérations disponibles.

Pile (Stack)

Le type **Stack n'existe pas** en Python mais **list** permet de le réaliser simplement.

Caractéristiques d'une pile (LIFO : Last In First Out) :

- les éléments sont ajoutés en sommet de pile : **list.append()**
- on récupère et supprime l'élément au sommet de la pile : **list.pop()**

```
pile = [1, 2, 3]      # [1, 2, 3], 3 au sommet
pile.append(4)        # [1, 2, 3, 4] : 4 est le nouveau sommet
pile.append(5)        # [1, 2, 3, 4, 5] : 5 est le nouveau sommet
pile[-1]              # obtenir le sommet, la pile n'est pas modifiée
x = pile.pop()        # supprime l'élément en sommet et le retourne : 5
pile                  # [1, 2, 3, 4]
```

File (Queue)

Caractéristiques d'une file (FIFO : First In First Out), similaire à une file d'attente classique :

- les éléments sont ajoutés à la fin de la file
- les élément sont extraits par le début de la file

On pourrait par exemple réaliser une file avec une liste en utilisant :

- file.append(x) : pour ajouter à la fin de la liste
- x = file[0] : pour récupérer l'élément au début de la liste
- del file[0] : pour supprimer l'élément au début de la liste

Mais ce ne serait **pas efficace** (pour la suppression d'un élément)
Il existe le type **deque** (Double Ended QUEue).

```
from collections import deque
file = deque([1, 2, 3])      # 1 est en tête de file, 3 est en fin de file
file.append(4)               # deque([1, 2, 3, 4])
x = file.popleft()           # x == 1  et file == deque([2, 3, 4])
```

Remarques :

- On pourrait aussi utiliser **appendleft()** et **pop()**.
- Il existe aussi **extend()** et **extendleft()** qui ajoutent plusieurs éléments (itérable)

Voir la documentation de deque, généralisation des files et des piles.

Remarque : deque est préférable à list pour réaliser un pile.

Ensemble (set) : équivalent aux ensembles en math.

- Pas de numéro d'ordre sur les élements (et donc pas d'accès par indice)
- Pas de double : ajouter un élément déjà présent ne change pas l'ensemble
- **Principales opérations :** ajoute un élément, supprime un élément, appartenance, taille
- Relation d'ordre partielle : inclusion

```
s1 = {1, 2, 3, 1, 2}     # {1, 2, 3}      # Pas de double !
s2 = set([5, 4, 3])      # à partir d'une liste (un itérable), ordre sans importance
s3 = {3, 4}
s4 = set()               # Un ensemble vide
s5 = set('une chaine')   # à partir de tout itérable
```

op.	méthode	exemple
in		1 in s1 is True
	add(x)	ajoute l'élément dans l'ensemble
	pop()	supprime et retourne un élément au hasard
	discard(x)	supprime x de l'ensemble, rien si x not in set
	remove(x)	supprime x ou lève KeyError si x not in set
<=	issubset(other)	s3 <= s2 is True (ou s3.issubset(s2))
>=	issuperset(other)	s1 >= s2 is False and s2 >= 1 is False
|	union(other)	s1 | s2 == {1, 2, 3, 4, 5}
&	intersection(other)	s1 & s2 == {3}
-	difference(other)	s1 - s2 == {1, 2}
^	symetric_difference(other)	s1 ^ s2 == {1, 2, 4, 5}
|=	update(x)	s1.update(s2) -> s1 == {1, 2, 3, 4, 5}
	clear()	vide l'ensemble

Remarque : frozenset est un ensemble immuable

Dictionnaire (dict)

- **Définition :** Permet d'utiliser une clé (key) pour enregistrer et récupérer une information.
- **Remarque :** Un genre de généralisation des listes où l'indice devient n'importe quel type.
- **Principales opérations :** ajouter une valeur avec sa clé, récupérer une valeur grâce à une clé
- **Synonymes :** Tableaux associatifs, *Map*
- Pas de numéro d'ordre sur les élements (et donc pas d'accès par indice)
- **Attention :** La clé doit être hashable

```
d1 = {}          # ou d1 = dict() :  un dictionnaire vide
d1 = {'un': 1, 'deux': 2, 3: 'trois'}   # dictionnaire avec des couples clé:valeur
d2 = dict(un=1, deux=2)      # {'deux': 2, 'un': 1}, si les clés sont des chaînes

print(d1['un'], d1[3])       # 1 'trois'                                (accès)
d1['un'] = 'I'               # {3: 'trois', 'deux': 2, 'un': 'I'}       (modification)
del d1['un']                 # {3: 'trois', 'deux': 2}                  (suppression)
'un' in d1                   # False : est-ce que 'un' est une clé de d1 ?
x = d1.get('un', 'oups!')        # x == 'oups!'   get(clé [, defaut])
i = d1.items()               # dict_items([(3, 'trois'), ('deux', 2)])
k = d1.keys()                # dict_keys([3, 'deux'])
v = d1.values()              # dict_values(['trois', 2])
# Remarque : items(), keys(), values() sont des vues sur le dictionnaire.
d1[10] = 'X'                 # {10: 'X', 3: 'trois', 'deux': 2}
print(k, v)                  # dict_keys([10, 3, 'deux']) dict_values(['X', 'trois', 2])
```

Listes et autres types en compréhension

Principe

L'idée est d'indiquer comment on définit une nouvelle collection par filtrage des éléments d'une autre collection.

Exemples

```
cubes = [x ** 3 for x in range(6)]  # [0, 1, 8, 27, 64, 125]
    #  [0 ** 3, 1 ** 3, 2 ** 3, 3 ** 3, 4 ** 3, 5 ** 3]

pairs = [x for x in cubes if x % 2 == 0]    # [0, 8, 64]

d = { k: v for k, v in zip(['I', 'II', 'III', 'IV'], range(1, 10)) }
    # {'III': 3, 'IV': 4, 'I': 1, 'II': 2}
    # zip : construit des couples (ou tuples) en prennant successivement
    # un élément de chaque itérable.  S'arrête sur le premier fini.

cubesTuple = tuple(x ** 3 for x in range(6))
    # (0, 1, 8, 27, 64, 125)

quoi = (x ** 3 for x in range(6))
    # <generator object <genexpr> at 0x7f0582892780>

type(quoi)  # <class 'generator'>
```

Générateur : cas particulier d'itérateur

Définition : Itérateur

Un itérateur est un objet qui permet de parcourir successivement tous les éléments d'un conteneur. next(it) permet de récupérer l'élément suivant de l'itérateur (lève StopIteration si plus d'éléments)

Exemples

```
g = (x ** 2 for x in range(0, 2))
print(next(g))  # 0
print(next(g))  # 1
print(next(g))  # Exception StopIteration

g = (x ** 2 for x in range(0, 2))
for n in g: # for utilise next(g) pour initialiser n
    print(n, end=' ')
# affiche : 0 1

l = [1, 'A']
next(l)         # TypeError: 'list' object is not an iterator
it = iter(l)    # retourne un itérateur sur la liste l
print(next(it)) # 1
print(next(it)) # 'A'
print(next(it)) # Exception StopIteration
iter(1)     # TypeError: 'int' object is not iterable
```

Générateurs

Motivation

Mécanisme qui permet d'écrire simplement un **iterable** (sur lequel on peut faire un **for**).

Moyen : **yield** expression

Une fonction qui contient l'instruction *yield* retourne un **générateur**.
Chaque sollicitation exécute la fonction jusqu'au **yield** suivant et retourne l'expression associée.

Exemple : où comment refaire son *range*

```
def my_range(n):
    c = 0
    while c < n:
        yield c
        c = c + 1

for i in my_range(10):
    print(i, end=', ')
type(my_range(10))
```

affiche : `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, <class 'generator'>`

Exercices

1. Supprimer d'une chaîne tous les caractères qui ne sont ni lettre, ni chiffre.
2. Remplacer dans une chaîne tous les caractères qui ne sont ni lettre, ni chiffre.
3. Obtenir la première lettre minuscule d'une chaîne.
4. Calculer le nombre d'éléments positifs d'une liste.
5. Déterminer si tous les éléments d'une liste sont positifs (on pourra utiliser la fonction prédéfinie `all`). Efficacité ?
6. Comment traiter les exercices proposées sur le slide Motivation ?