Les listes et les tableaux

Chapitre 4

Introduction

En python, les types prédéfinis tel que : **int, float, bool et str** restent limités pour résoudre des problèmes complexes. A titre d'exemple, nous citons le problème suivant : comment peut-on modéliser (représenter dans la RAM) des objets comme :

- Etudiant(nom,prénom,age,note,...)
- Voiture (marque,modèle, prix,couleur,...)
- Vecteur(abs, ord)
- etc

Pour résoudre ce problème, l'utilisation d'une structure données est désormais nécessaire.

Qu'est ce qu'une structure de données ?

- Une structure de données consiste à utiliser un ensemble de variables pour modéliser un tel objet.
- Python offre une diversité des structures de données comme :
  - listes ;
  - tuples ;
  - dictionnaires ;
  - etc.

Définition

Qu'est ce qu'une liste

- Une liste est une structure de données ordonnées particulièrement souple.
- Les listes peuvent contenir toutes sortes de données telles que : entiers, réels, complexes, caractères, logiques, chaînes de caractères, tuples, dictionnaires, ensemble, voire d'autres listes.
- Une liste est une collection ordonnée et modifiable : chaque élément est classé selon un index numérique et peut être librement redéfini.
- La numérotation des index commence à partir de zéro.

⇒ **Exemple**

```
>>> L = [1, 2, 3]
>>> type(L)
< class 'list' >
```

Déclaration d'une liste

Syntaxe

- On peut définir une liste comme une collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets
  $Nom\_Liste = [Val_1, Val_2, ..., Val_i, ..., Val_n]$
  chaque élément Val_i peut avoir n'importe quel type

Liste vide

Pour créer une liste vide il existe deux méthodes :

1. L=list() # cette fonction permet de convertir aussi n'importe quel type de données en liste
2. L= [],

Un accès partiel

Il existe deux types d'indices pour l'accès partiel à chaque élément d'une liste :

Indiçage positif

Soit la liste suivante : $L = [1, -6, 30, 26, 10]$ contenant n=5 valeurs

- l'accès au premier élément de la liste L se fait par L[0] (L[0]=1)
- l'accès au dernier élément de la liste se fait par L[n-1] (L[n-1]=10)
- les listes sont des séquences modifiables, on peut changer la valeur de chaque champ L[i]. Par exemple, L[0]=30

.

Indiçage négatif

Il existe un moyen de désigner les éléments d'une séquence en partant de la fin :

- L[-1] est le dernier élément de la séquence
- L[-2] est l'avant-dernier
- Si $k > 0$, L[-k] vaut L[len(s)-k], avec len retourne le nombre des éléments de L.

.

Un accès total

Copie superficielle (shallow copy)

- L'accès globale à une liste consiste à affecter une liste L1 à une autre liste L2, L2=L1
- Mais cette affectation une copie superficielle qui pose un problème.
- Le test donné dans l'exemple 1 permet de comprendre ce qui se passe :

Copie en profondeur (deep copy)

- Pour obtenir un comportement différent, on dispose d'un module copy qui offre une copie en profondeur :
- L'exemple 2 permet d'illustrer ce mécanisme :

Exemple 1

```
>>> L1 = [1, 2, [5, 6, 7]]
>>> L2 = L1; print(L1, L2)
[1, 2, [5, 6, 7]][1, 2, [5, 6, 7]]
>>> L2[2][0] = 10.5
>>> print(L1, L2)
[1, 2, [10.5, 6, 7]][1, 2, [10.5, 6, 7]]
```

Exemple 2

```
import copy
>>> L1 = [1, 2, [5, 6, 7]]
>>> L2 = copy.deepcopy(a)
>>> L2[2][0] = 3.2
>>> print(L1, L2)
[1, 2, [5, 6, 7]][1, 2, [3.2, 6, 7]]
```

Création d'une liste à l'aide de la fonction range()

Principe de la fonction range()

- La fonction range() génère par défaut une séquence de nombres entiers de valeurs croissantes ou décroissantes.
- L'appel de fonction range(n) génère les nombres de 0 à n-1.
- L'appel de fonction range(début,fin,pas) génère les nombres de début jusqu'à la valeur fin-1 et pas la valeur à sauter pour passer d'une valeur à la suivante.

Création d'une liste en utilisant range()

En Python, il existe deux méthodes pour générer les éléments d'une liste en utilisant la fonction range :

Méthode 1

```
>>> L = list(range(10))
>>> print(L)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(5, 13))
[5, 6, 7, 8, 9, 10, 11, 12]
```

Méthode 2

```
>>>L1=[i for i in range(1,10,2)]
>>> L1
[1, 3, 5, 7, 9]
>>> [i*i for i in range(10,1,-1)]
[100, 81, 64, 49, 36, 25, 16, 9, 4]
```

Parcours d'une liste

La boucle for

L'instruction for est l'instruction idéale pour parcourir une liste :

1. Exemple 1 :

```
>>> L = [1, 2, 5, 6, 7]
>>> for i in range(len(L)) :
... print(L[i],end='')
1 2 5 6 7
```

2. Exemple 2 :

```
>>> for n in range(1, 20, 3) :
... print(n, n**2, n**3)
1 1 1
4 16 64
7 49 343
10 100 1000
13 169 2197
16 256 4096
19 361 6859
```

Les tranches (slicing)

Il arrive fréquemment, lorsque l'on travaille avec des listes, que l'on souhaite extraire une sous séquence de la liste initiale. Python propose pour cela une technique simple que l'on appelle slicing (≪ découpage en tranches ≫). Elle consiste à indiquer entre crochets les indices correspondant au début et à la fin de la ≪ tranche ≫ que l'on souhaite extraire :
Soit lst une liste quelconque.

- lst[p] renvoie l'élément d'indice p de lst. centrale
- lst[p :n] renvoie une nouvelle liste constituée des éléments de lst d'indice p inclus à n exclu.
- lst[p :n :pas] renvoie une nouvelle liste constituée des éléments de lst d'indice p inclus à n exclu, tous les pas.
- lst[:] renvoie une nouvelle liste constituée de tous les éléments de lst.
- lst[p :] renvoie une nouvelle liste constituée de tous les éléments de lst à partir de l'élément d'indice p inclus.
- lst[:n] renvoie une nouvelle liste constituée de tous les éléments de lst depuis le premier jusqu'à l'élément d'indice n exclu.
- lst[: :pas] renvoie une nouvelle liste constituée des éléments de lst, tous les pas.

Insertion/Suppression d'un ou plusieurs éléments

Insertion dans une liste n'importe où

```
>>> L=[2, 5, 10, 89, 92] ;L
[2, 5, 10, 89, 92]
>>> L[3 :3]=[12,15] ; L
[2, 5, 10, 12, 15, 89, 92]
>>>L=[2, 5, 10, 89, 92]
>>>L[0 :0]=[10,20,30]] ; L
[10, 20, 30, 2, 5, 10, 89, 92]
```

Suppression/remplacement d'un ou plusieurs éléments

```
>>> L=[2, 5, 10, 89, 92] ;L
[2, 5, 10, 89, 92]
>>> L[4 :5]=[] ; L
[2, 5, 10, 89]
>>>L[2 :4]=[99] ; L #on utilise la liste initiale L=[2, 5, 10, 89, 92]
[2, 5, 99, 92]
```

Fonctions communes sur les listes

Soit la liste suivante : L=[6,2,7,-10,56,13]

Les fonctions : len, min, max, sum et del

- La fonction len() retourne le nombre des éléments d'une liste $len(L) = 6$
- La fonction min() retourne la valeur minimale d'une liste $min(L) = -10$
- La fonction max() retourne la valeur maximale d'une liste $max(L) = 56$
- La fonction sum() retourne la somme des éléments d'une liste $sum(L) = 74$
- La fonction del() permet de supprimer un élément d'une liste $del(L[0])$

⇒ **Exemples :**

```
>>> L=[6,2,7,-10,56,13]
>>>len(L)
6
>>>min(L)
-10
>>>sum(L)
74
```

Fonctions communes sur les listes

Les opérateurs : +, * et in

- L1+L2 l'opérateur + permet de concaténer les deux listes L1 et L2.
- L*i l'opérateur * permet de retourner une nouvelle liste contenant i copies de L.
- x in L : retourne True si x est dans L et False sinon.

⇒ **Exemples :**

```
>>>L1=[0,2,4],L2=[1,3,5]
>>>L3=L1+L2
>>>print(L3)
[0, 2, 4, 1, 3, 5]
>>>L4=L1*3
>>>print(L4)
[0, 2, 4, 0, 2, 4, 0, 2, 4]
>>>2 in L1
True
>>>print(L1[5])
IndexError : list index out of range
```

Méthodes pour les listes

Méthode	Type de retour	Déscription
list()	list	Renvoie une liste vide
L.append(x)	None	ajoute un élément x (nombre, chaîne, liste, tuple, dictionnaire) à la fin d'une liste
L.insert(i,x)	None	Modifie L en insérant x dans la position de l'indice i
L.extend(seq)	None	Modifie L en lui ajoutant les éléments de l'itérable seq.
L.count(x)	int	retourne le nombre d'occurrences de x dans la liste L
L.index(x)	int	retourne l'indice du premier élément de L égale à x (ValueError si x n'existe pas)
L.index(x,n)	int	retourne l'indice de la nième occurrence de x dans L
L.pop()	item	renvoie le dernier élément et le supprime de la liste
L.remove(x)	None	supprime la première occurrence de x dans L
L.reverse()	None	Modifie L en renversant l'ordre de ses éléments
L.sort()	None	Modifie L en triant par ordre croissant ses éléments

Implémentation avec le module array

Syntaxe

Pour déclarer un tableau en utilisant le module array, il faut tout d'abord charger ce module dans la mémoire
>>>from array import *

- $Nom\_Tab = array(typecode, [initialisateur])$ avec
  $initialisateur = [Val_1, Val_2, ..., Val_n]$ ou $initialisateur = range(n)$
- Exemple
  $>>> T1 = array('i', [-1, 2, 3])$ # T1 contient 3 valeurs de type entier.
  $>>> type(T1)$ # pour savoir le type de T1
  $< type\ 'array.array' >$
  $>>> T2 = array('f', range(10))$ # T2 contient dix valeurs de type réel.

.

Remarque

- Avec l'utilisation du module array, nous pouvons assurer d'avoir le même type pour tous les élément d'un tableau.

.

Implémentation avec le module array

Remplir et afficher un tableau

On applique le même principe que les listes.

Les opérateurs : +,*,-,/ et in

c'est identique que l'implémentation d'un tableau avec une liste.

Les fonctions prédéfinies et originales

c'est identique que l'implémentation d'un tableau avec une liste.

Implémentation avec le module numpy

Principe

- NumPy est un package pour le calcul scientifique dans Python.
- Géneralment ce module n'est pas intégré par défaut dans Python, on peut le telecharger via ce lien *http* : *//numpy.scipy.org*
- NumPy se spécialise dans le calcul matriciel, des tableaux multidimensionnels, l'algèbre linéaire, des fonctions de nombre aléatoire et l'anlyse numérique.

.

Syntaxe

Pour déclarer un tableau en utilisant le module numpy, il faut tout d'abord charger ce module dans la mémoire
>>>from numpy import *

- $Nom\_Tab = array([initialisateur], dtype)$
- Exemple
  $>>> T1 = array([-1, 2, 3], dtype = int)$ # T1 contient 3 valeurs entières.
  $>>> T2 = array(range(10), dtype = float)$ # T2 contient 10 valeurs réelles.

.