Laurent_LHERMET_Tableaux_Python

October 27, 2020

#

Tableaux

0.1 0. Remarque importante

- En python, ce que l'on désigne par le terme liste est en réalité ce que l'on appelle en algorithmique un tableau, c'est à dire une suite d'éléments contigus et ordonnés en mémoire. Ces éléments sont stockés en mémoire les uns à la suite des autres.

Exemple : La fonction `id` donne "l'identité" d'un objet python, c'est à dire l'adresse de l'objet en mémoire.

[4]:
```
T=[0,1,2,3,5]
print(id(T[0]))
print(id(T[1]))
print(id(T[2]))
print(id(T[3]))
print(id(T[4]))
```

```
140736399947520
140736399947552
140736399947584
140736399947616
140736399947680
```

- En terminale, on aborde le concept de liste chaînée, ce qui est source de confusions pour les élèves.
- Dans ce document, on utilisera abusivement le terme "liste" mais en gardant à l'esprit les remarques précédentes.

0.2 1. Le type list

Sous Python, on peut définir une liste comme une collection d'éléments séparés par une virgule, l'ensemble étant délimité par des crochets.

Exemples :

[5]:
```
#Exécuter le code ci-dessous :
semaine = ['Lun','Mar','Mer','Jeu','Ven']
```

```
jours =␣
 ↪[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31]
mois =␣
 ↪['Jan','Fév','Mars','Avr','Mai','Juin','Juil','Août','Sep','Oct','Nov','Déc']
temperatures = [-30.0,-20.0,-10.0,0.0,10.0,20.0,30.0,40.0]

ajd = [['Dim',21,'Juin'],20.0]

print (semaine, type(semaine))
print (ajd, type(ajd))
```

```
['Lun', 'Mar', 'Mer', 'Jeu', 'Ven'] <class 'list'>
[['Dim', 21, 'Juin'], 20.0] <class 'list'>
```

Les éléments individuels qui constituent une liste peuvent être de types variés : * Les éléments des listes `semaine` et `mois` sont des chaînes de caractères. * Les éléments de la liste `jours` sont des entiers. * Les éléments de la liste `temperatures` contient des nombres de type 'float'. * La liste `ajd` contient…2 éléments : une liste (elle-même composée de chaînes et d'un entier) et un nombre de type 'float'.

0.2.1 Similitudes avec les chaînes de caractères

Accès aux éléments : Comme les chaînes de caractères, les listes sont des collections ordonnées d'objets. On peut donc accéder à chaques élément individuellement si l'on connaît son index dans la liste . Comme pour les chaînes, l'index commence à partir de zéro.

Exemples:

[6]:
```
#Exécuter le code ci-dessous :
print(semaine[0])
print(jours[30])
print(mois[2])
```

```
Lun
31
Mars
```

Exercice 1 : Modifier le programme ci-dessus pour qu'il affiche la date d'aujourd'hui.

[7]:
```
# Réponse : à adapter en fonction de la date
print(semaine[0])
print(jours[25])
print(mois[9])
```

```
Lun
26
Oct
```

Longueur d'une liste La fonction `len`s'utilise aussi avec les listes. Exemples :

[8]:
```
print(len(jours)) #affiche la longueur de la liste 'jours', c'est à dire son␣
↪nombre d'éléments
print(len(ajd)) # la liste 'ajd' contient 2 éléments
```

```
31
2
```

Concaténation et répétition Comme pour les chaînes, on peut utiliser l'opérateur `+` (non commutatif) pour rassembler 2 listes par concanténattion et `*` pour assembler par répétition

Exemples :

[9]:
```
#Exécuter le code ci-dessous :
L1=[1,2,3]
L2=[4,5,6]
print(L1+L2)
print(L2+L1)

print(2*L1+3*L2)
```

```
[1, 2, 3, 4, 5, 6]
[4, 5, 6, 1, 2, 3]
[1, 2, 3, 1, 2, 3, 4, 5, 6, 4, 5, 6, 4, 5, 6]
```

0.2.2 Différences avec les chaînes

Modifier des éléments À la différence de ce qui se passe pour les chaînes, qui constituent un type de données non-modifiable (immuable), il est possible de modifier les éléments individuels d'une liste, elle est mutable :

[10]:
```
# Executer le code ci-dessous
semaine[0]='Dim'
print(semaine)
```

```
['Dim', 'Mar', 'Mer', 'Jeu', 'Ven']
```

Exercice 2: Annuler la modification précédente

[14]:
```
#Réponse:
semaine[0]='Lun'
print(semaine)
```

```
['Lun', 'Mar', 'Mer', 'Jeu', 'Ven']
```

Supprimer des éléments La fonction `del` permet de supprimer d'une liste un élément quelconque, à partir de son index.

[15]:
```
del(semaine[4])
print(semaine)
```

```
['Lun', 'Mar', 'Mer', 'Jeu']
```

Ajouter des éléments On peut ajouter un élément à une liste. Pour cela, il faut considérer que la liste est un objet, dont on va utiliser l'une des méthodes, ici la méthode `append`.

[16]:
```
#Exécuter le code ci-dessous
semaine.append('Ven')
print(semaine)
```

```
['Lun', 'Mar', 'Mer', 'Jeu', 'Ven']
```

Remarques : * "append" en anglais veut dire "ajouter". * L'élément est ajouté à la fin de la liste. * On applique une méthode à un objet en reliant les deux à l'aide d'un point.

Exercice 3: Ajouter le week-end à la liste `semaine` et l'afficher

[18]:
```
# Réponse :
semaine.append('Sam')
semaine.append('Dim')
print(semaine)
```

```
['Lun', 'Mar', 'Mer', 'Jeu', 'Ven', 'Sam', 'Dim']
```

Autres méthodes

- Il existe bien sûr de nombreuses méthodes permettant d'agir sur les listes python(trier, renverser, etc..)
- Documentation : https://docs.python.org/fr/3/tutorial/datastructures.html

0.3 2. Points de vigilance

0.3.1 La copie de tableau

Considérons une liste `L1` à copier dans une autre variable `L2`.On pourrait penser procéder ainsi :

[19]:
```
L1=[1,1,2,3,5]
L2=L1
```

Modifions le contenu de `L2`

[20]:
```
L2.append(8)
print(L2)
```

```
[1, 1, 2, 3, 5, 8]
```

Mais...

[21]:
```
print(L1)
```

```
[1, 1, 2, 3, 5, 8]
```

Remarque : * En procédant ainsi, on ne crée pas une véritable copie. À la suite de l'instruction `L1=L2`, il n'existe toujours qu'un seul objet dans la mémoire de l'ordinateur.Une nouvelle référence vers cette liste est créée (un alias) mais toute modification souhaitée utilisant les alias `L1` ou `L2` implique la modification de l'objet.

Exercice 4 : Ecrire la fonction `copy(L)` qui prend en paramètre une liste et qui renvoie une "véritable" copie de la liste L. On pourra ainsi agir sur cette copie sans modifier `L`.

[28]:
```
def copy(L) :
    '''Crée une copie de la liste L'''
    bis=[]                          #Création d'une liste vide
    for i in range (len(L)):    # Répetition autant de fois que de termes dans␣
 ↪la liste L transmise
        bis.append(L[i])        #remplissage de la liste bis par le contenu de␣
 ↪la liste L
    return bis                  #retour de la liste bis copiée

L1=[1,1,2,3,5,8]
L2=copy(L1)
L2.append(13)
print(L1)
print(L2)
```

```
[1, 1, 2, 3, 5, 8]
[1, 1, 2, 3, 5, 8, 13]
```

0.3.2 A propos de l'insertion et de la suppression

Exercice 5: On considère le tableau ci-dessous :

| 1 | 1 | 3 | 5 | 8 |

On souhaite insérer la valeur 0 en première position et obtenir le tableau suivant:

| 0 | 1 | 1 | 3 | 5 | 8 |

- On commence par agrandir le tableau, en ajoutant un nouvel emplacement à la fin :

| 1 | 1 | 3 | 5 | 8 | |

- Il faut ensuite décaler tous les élements du tableau initial vers la droite en commençant par le dernier :

| 1 | 1 | 1 | 3 | 5 | 8 |

- Enfin, on écrit la valeur souhaitée dans la première case du tableau :

| 0 | 1 | 1 | 3 | 5 | 8 |

Questions :

Considérons maintenant un tableau qui comporte 1 millions d'éléments. 1. Combien de déplacements seront nécéssaires pour insérer une valeur en première position ? 2. Et si l'on souhaite supprimer le premier élément ?

Réponses : 1. Si 1 000 000 d'élements, il faudra faire 1 000 000 de déplacements après avoir rajouté un nouvel emplacement à la fin. 2. Si l'on souhaite supprimer le premier élement, le recours à del(L1[0]) permet de supprimer le premier élement directement en une opération. La liste est automatiquement raccourcie d'un élément.

Remarques :

- Le coût en temps de l'insertion d'un élément en première position est considérable puisqu'il faut décaler tous les éléments déjà présent.
- Il en est de même pour la suppression.
- L'utilisation des méthodes `insert` et `pop` sont donc à utiliser avec précaution.

A voir avec les formateurs : Suppression du 1er élement avec une seule opération non ? Je ne comprends pas l'enchainement entre les questions préalables et la remarque sur les méthodes insert et pop.

`complément perso`

```
list.append(x)

  # Ajoute un élément à la fin de la liste. Équivalent à a[len(a):] = [x].


list.insert(i, x)

   # Insère un élément à la position indiquée. Le premier argument est la position de l'élémen

list.pop([i])

    # Enlève de la liste l'élément situé à la position indiquée et le renvoie en valeur de ret
```

[30]:
```
#ajout perso

L1=[1,1,2,3,5,8]
L2=[]
#L1.insert(2,'L')
L2.append(L1.pop(2))
#del(L1[2])
print(L1)
print(L2)
```

```
[1, 1, 3, 5, 8]
[2]
```

0.4 3. Générer des tableaux

0.4.1 Listes en compréhension

Voici un programme qui ajoute des éléments dans une liste façon successive:

[31]:
```
L=[]
for i in range(20):
    if i%2==0:
        L.append(i)
print(L)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Il existe une manière plus concise de procéder, la définition en compréhension :

[32]:
```
# Une liste en compréhension
L=[i for i in range(20) if i%2==0]
print(L)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Syntaxes Voici des types d'instructions possibles pour générer des listes en compréhension:

- [*fonction de x* **for** *x in ensemble*]
- [*fonction de x* **if** *condition sur x* **else** *autre fonction de x* **for** *x in ensemble*]
- [*fonction de x* **for** *x in ensemble* **if** *condition sur x*]

Exemples:

[33]:
```
#1
L=[3*i for i in range(11)]
print(L)

#2
from random import *
n=10
Des=[randint(1,6) for i in range(n)]
print(Des)

#3
pileface=[choice(['P','F']) for i in range(20)]
print(pileface)
```

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
[2, 4, 4, 1, 5, 6, 2, 5, 6, 6]
['F', 'F', 'F', 'F', 'P', 'P', 'F', 'P', 'P', 'F', 'F', 'F', 'P', 'P', 'P', 'F',
'F', 'F', 'P', 'P']
```

Exercice 6: En utilisant la syntaxe des listes en compréhension : 1. Générer les entiers naturels de 1 à 100. 2. Générer les multiples de 5 inférieurs ou égaux à 100. 3. Générer une liste des entiers naturels de 1 à 100 dans laquelle les multiples de 5 seront remplacées par le caractère *

[35]:
```
#1.
L=[i for i in range (1,101)]
print(L)
M=[i+1 for i in range (100)] #2ème version
print (M)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

[37]:
```
#2.
L=[i for i in range (101) if i%5==0] #inférieurs ou EGAUX à 100
print (L)
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95,
100]
```

[52]:
```
#3.
L=[i for i in range (101)]
print(L)
Lf = [n if n % 5 != 0 else '*' for n in L]
print(Lf)
# solution alternative
M = [n if n % 5 != 0 else '*' for n in range(101)]
print(M)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81,
82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
['*', 1, 2, 3, 4, '*', 6, 7, 8, 9, '*', 11, 12, 13, 14, '*', 16, 17, 18, 19,
'*', 21, 22, 23, 24, '*', 26, 27, 28, 29, '*', 31, 32, 33, 34, '*', 36, 37, 38,
39, '*', 41, 42, 43, 44, '*', 46, 47, 48, 49, '*', 51, 52, 53, 54, '*', 56, 57,
58, 59, '*', 61, 62, 63, 64, '*', 66, 67, 68, 69, '*', 71, 72, 73, 74, '*', 76,
77, 78, 79, '*', 81, 82, 83, 84, '*', 86, 87, 88, 89, '*', 91, 92, 93, 94, '*',
96, 97, 98, 99, '*']
['*', 1, 2, 3, 4, '*', 6, 7, 8, 9, '*', 11, 12, 13, 14, '*', 16, 17, 18, 19,
'*', 21, 22, 23, 24, '*', 26, 27, 28, 29, '*', 31, 32, 33, 34, '*', 36, 37, 38,
39, '*', 41, 42, 43, 44, '*', 46, 47, 48, 49, '*', 51, 52, 53, 54, '*', 56, 57,
58, 59, '*', 61, 62, 63, 64, '*', 66, 67, 68, 69, '*', 71, 72, 73, 74, '*', 76,
77, 78, 79, '*', 81, 82, 83, 84, '*', 86, 87, 88, 89, '*', 91, 92, 93, 94, '*',
```

```
96, 97, 98, 99, '*']
```

Exercice 7: En utilisant les fonctions `randint()` ou `choice()` du module `random` et la syntaxe des listes en compréhension: 1. Générer une liste de 20 entiers naturels aléatoires entre 0 et 255. 2. Générer 100 caractères au hasard parmi `a`,`b`,`c`

[53]:
```
#1.
from random import *
n=20
Des=[randint(0,255) for i in range(n)]
print(Des)
```

```
[243, 17, 120, 249, 244, 10, 106, 18, 47, 115, 223, 110, 157, 69, 188, 102, 164,
178, 244, 35]
```

[56]:
```
#2.
Lis=[choice(['a','b','c']) for i in range(100)]
print(Lis)
```

```
['b', 'b', 'a', 'b', 'b', 'c', 'c', 'b', 'b', 'c', 'c', 'a', 'b', 'b', 'b', 'a',
'c', 'c', 'a', 'b', 'c', 'c', 'b', 'c', 'b', 'a', 'b', 'c', 'b', 'a', 'a', 'c',
'b', 'a', 'b', 'c', 'b', 'a', 'a', 'c', 'c', 'a', 'b', 'b', 'c', 'c', 'b', 'a',
'b', 'a', 'c', 'c', 'b', 'b', 'b', 'b', 'a', 'a', 'b', 'a', 'a', 'b', 'a', 'b',
'c', 'c', 'c', 'c', 'a', 'a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'a', 'c', 'a',
'b', 'a', 'b', 'b', 'b', 'c', 'b', 'a', 'c', 'b', 'c', 'b', 'b', 'c', 'c', 'a',
'a', 'c', 'a', 'a']
```

Exercice 8 : Synthèse Le saviez-vous ?

Un nombre premier est un nombre qui n'est divisible que par un et par lui-même.

Écrivez la fonction `premier(n)` qui renvoie la liste de tous les nombres premiers compris entre 1 et `n`, en utilisant la méthode du crible d'Eratosthène : * Créez une liste de 1000 éléments, chacun initialisé à la valeur 1. * Parcourez cette liste à partir de l'élément d'indice 2 : si l'élément analysé possède la valeur 1, mettez à zéro tous les autres éléments de la liste, dont les indices sont des multiples entiers de l'indice auquel vous êtes arrivé. Lorsque vous aurez parcouru ainsi toute la liste, les indices des éléments restés à 1 sont des nombres premiers.

En effet :

A partir de l'indice 2, vous annulez tous les éléments d'indices pairs : 4, 6, 8, 10, etc… Avec l'indice 3, vous annulez les éléments d'indices 6, 9, 12, 15, etc., et ainsi de suite. Seuls resteront à 1 les éléments dont les indices sont effectivement des nombres premiers.

- Il reste enfin à reconstituer la liste des nombres premiers entre 1 et `n` à partir des indices des éléments restés à 1 (on pourra générer cette liste en compréhension)

[124]:
```
def premier(n) :
    '''renvoie la liste des nombres premiers
    inférieurs ou égaux à n'''
    n +=1
```

```
    Linit = [1 for i in range (n)]
    # Linit = [1,1]+[1 for i in range(2,n)]
    # print(Linit)     #affichage pour debuggage

    for indice in range(2,n):
        if Linit[indice]==1:
            #on va effacer tous les indices multiples de indice jusqu'à n
            for j in range(indice*2,n,indice):
                Linit[j]=0
            # print (Linit,'avec comme indice', indice) #affichage pour␣
↪debuggage
    Lfin = [p for p in range(2,n) if Linit[p]!=0]
    return Lfin

print(premier(1000))
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163,
167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251,
257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557,
563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647,
653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757,
761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863,
877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983,
991, 997]
```

[125]:
```
#Solution Web

def eratosthene(n):
    """retourne la tableau des nombres premiers <= n (crible d'eratosthene)"""
    n += 1
    tableau = [0,0] + [i for i in range(2, n)]
    for i in range(2, n):
        if tableau[i] != 0:
            # c'est un nombre 1er: on garde, mais on neutralise ses multiples
            for j in range(i*2, n, i):
                tableau[j] = 0
    return [p for p in tableau if p!=0]

print (eratosthene(1000))
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163,
167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251,
257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
```

```
449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557,
563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647,
653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757,
761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863,
877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983,
991, 997]
```