Listes, tuples et dictionnaires

Partie 1

PCSI - Lycées Condorcet / Jean Jaurès - mars 2024

1 Listes, tuples (rappels) et Dictionnaires

1.1 Listes et tuples

1.1.1 Définition

Les listes et les tuples sont variables pouvant en contenir d'autres.

- Les listes permettent de stocker dans une seule variable un ensemble **ordonné** de valeur, pouvant être de différents types. Les listes sont des objets modifiables, ou mutables : on peut changer la valeur d'un des objets de la liste après sa déclaration.
- Les tuples sont des listes non-modifiables. Un fois déclarés, ils ne peuvent plus être modifiés.

[1]:
```
L=[1,2]
L[0]=3
print("L =",L)

T=(1,2)
T[0]=3
print("T =",T)
```

```
L = [3, 2]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [1], in <cell line: 6>()
      3 print("L =",L)
      5 T=(1,2)
----> 6 T[0]=3
      7 print("T =",T)

TypeError: 'tuple' object does not support item assignment
```

Pour accéder à une des valeurs stockées dans une liste ou un tuple, il suffit de donner son **indice** entre crochet.

Attention : En Python, les indices des valeurs commencent à 0.

[2]:
```
L=[1,2,"coucou",(0,1)]
print(L[1])
T=(-5,12,"blablabla",[3,2])
print(T[1])
```

```
2
12
```

1.1.2 Slicing et concaténation

Slicing Si L et une liste et que i et j sont des indices :

- `L[i:j]` renvoie une nouvelle liste, contenant les valeurs de L allant de l'indice i à l'indice j-1
- `L[:j]` renvoie une nouvelle liste, contenant toutes les valeurs deL jusqu'à l'indice j-1
- `L[i :]` renvoie une nouvelle liste, contenant toutes les valeurs de L à partir de l'indice i
- `L[:]` renvoie une nouvelle liste, qui est une copie de la liste L.

On peut aussi préciser un « pas » :

- `L[i:j:p]` renvoie une nouvelle liste, contenant les valeurs de L allant de l'indice i à l'indice j-1 en avançant par pas de p

NB : On a exactement le même fonctionnement avec des tuples

[3]:
```
L=[10,20,30,40,50,60,70,80,90,100]
print(L[2:5])
print(L[:5])
print(L[5:])
print(L[2:5:2])
```

```
[30, 40, 50]
[10, 20, 30, 40, 50]
[60, 70, 80, 90, 100]
[30, 50]
```

Concaténation On peut concaténer deux listes ou deux tuples à l'aide de l'opération +:

[4]:
```
L=[2,3,4]+[7,8,9]
print(L)
```

```
[2, 3, 4, 7, 8, 9]
```

Exercice 1 Ecrire une fonction « concatene(L,M,N) » qui, pour 3 listes, ou 3 tuples, L, M, N renvoie la concaténation de ces trois listes.

[5]:
```

```

[6]:
```
#Test :
L1=[10,20,30]
M1=[30,40,50]
N1=[100]
L2=concatene(L1,M1,N1)
print(L2)
```

```
[10, 20, 30, 30, 40, 50, 100]
```

1.1.3 Effet de bord sur les listes.

Sans rentrer dans le détail, retenons que le caractère « mutable » des listes a pour conséquence que si on les modifie à l'intérieur d'une fonction, elles sont aussi modifiées à l'extérieur.

Dans l'exemple ci-dessous, on crée deux fonctions, l'une qui modifie la liste passée en argument et l'autre qui fait la même chose, avec un entier.

Lorsque l'on appelle la première des deux fonctions ci-dessous, la liste maListe est modifiée « en dehors » de la fonction. Alors que lorsqu'on appelle la deuxième fonction, l'entier monN n'est pas modifié par la fonction, ce qui est un comportement plus normal pour une fonction. Un tel effet sur les listes est appelé « effet de bord » et doit bien être pris en compte lors de l'utilisation des listes dans les fonctions.

[7]:
```
def AjouteUnListe(L):
    L[0]+=1
    return L

def AjouteUn(N):
    N+=1
    return N

maListe=[10,20,30,40,50,60,70,80,90,100]
monN = 10

nouvListe=AjouteUnListe(maListe)
print("nouvListe =",nouvListe)
print("maListe =",maListe)
nouvN=AjouteUn(monN)
print("NouvN =",nouvN)
print("monN =",monN)
```

```
nouvListe = [11, 20, 30, 40, 50, 60, 70, 80, 90, 100]
maListe = [11, 20, 30, 40, 50, 60, 70, 80, 90, 100]
NouvN = 11
monN = 10
```

Si on veut agir sur une liste dans une fonction sans la modifier en dehors de la fonction, il faut commencer par en créer une copie :

[8]:
```
def AjouteUnListeModif(L):
    L2=L[:]
    L2[0]+=1
    return L2
maListe=[10,20,30,40,50,60,70,80,90,100]
nouvListe=AjouteUnListeModif(maListe)
print("nouvListe =",nouvListe)
print("maListe =",maListe)
```

```
nouvListe = [11, 20, 30, 40, 50, 60, 70, 80, 90, 100]
maListe = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Exercice 2 Ecrire une fonction `enlever(L,i)` qui renvoie une copie de la liste (ou du tuple) L dont on a enlevé l'élément d'indice i.

[9]:
```
```

[10]:
```
#Test :
maListe=[10,20,30,40,50,60,70,80,90,100]
L2=enlever(maListe,3)
print(maListe)
print(L2)
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[10, 20, 40, 50, 60, 70, 80, 90, 100]
```

1.1.4 Parcours des éléments d'une liste ou d'un tuple.

On peut parcourir les éléments d'une liste ou d'un tuple à l'aide d'une boucle for de deux façons :

- Soit en parcourant les indices de la liste ou du tuple :

[11]:
```
L=[10,20,30,40,50]
for k in range(len(L)):
    print(L[k])
```

```
10
20
30
40
50
```

- Soit en parcourant directement les éléments de la liste ou du tuple :

[12]:
```
L=[10,20,30,40,50]
for l in L:
    print(l)
```

```
10
20
30
40
50
```

Exercice 3 Ecrire une fonction `maxi(L)` qui renvoie la valeur maximale de la liste L. On donnera deux versions, l'une qui parcours les indices de la liste, l'autre qui parcourt les éléments de la liste.

[22]:
```
#Version parcourant les indices de la liste :



#Version parcourant les éléments de la liste :
```

[23]:
```
# Test :
maListe=[1, 4, 2, 3, 1, 3, 3, 6, 1, 3]
M=maxi(maListe)
print(M)
```

6

Exercice 4 Ecrire une fonction `indMaxi(L)` qui renvoie l'indice de la valeur maximale de la liste L.

[24]:
```

```

[25]:
```
# Test :
maListe=[1, 4, 2, 3, 1, 3, 3, 6, 1, 3]
imax=indMaxi(maListe)
print(imax)
```

9

Exercice 5 Ecrire une fonction `moyenne(L)` qui renvoie la moyenne de la liste L.

[57]:

[46]:
```
# Test :
N = 1000000
from random import uniform as rand
L = [rand(0,10**(-15)) for i in range(N)]
from statistics import mean as moy_python
from statistics import pvariance as var_python
from time import perf_counter as tps
def T_exec(f,L):
    '''f est une fonction prenant en argument une liste'''
    tic = tps()
    m=f(L)
    toc = tps()
    return m,toc - tic

def compare_fonctions(f1,f2,L,nom1,nom2):
    x1,t1=T_exec(f1,L)
    x2,t2=T_exec(f2,L)
    print("Moyenne obtenue avec " + nom1 + " :",x1)
    print("Temps d'exécution : ",t1)
    print("Moyenne obtenue avec " + nom2 + " :",x2)
    print("Temps d'exécution : ",t2)


compare_fonctions(Moyenne,moy_python,L,"moyenne perso","moy_python")
```

```
Moyenne obtenue avec moyenne perso : 4.999815489592829e-16
Temps d'exécution :  0.10869819999993524
Moyenne obtenue avec moy_python : 4.999815489592944e-16
Temps d'exécution :  2.2646718999999393
```

1.1.5 Ajouter ou enlever un élément à une liste : append() et pop()

- La méthode append() permet d'ajouter un élément à une liste.
- La méthode pop() permet d'enlever un élément d'une liste et, si besoin, de récupérer sa valeur.

Exercice 6 Ecrire une fonction `Enlever(L,a)` qui renvoie une copie de la liste L dont on a enlevé toutes les occurrences de la variable a. Elle ne doit pas modifier la liste L initiale.

[58]:

[60]:
```
# Test :
maListe=[1, 4, 2, 3, 1, 3, 3, 6, 1, 3]
L2=Enlever(maListe,1)
print(maListe)
print(L2)
```

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

Exercice 7 A l'aide de la fonction `randint(a,b)` du module random, qui permet de générer un nombre aléatoire entre les entiers a et b inclus, écrire une fonction `lancersDes(N)` qui renvoie une liste de N lancers de dés.

On importera le module random avec l'alias rd.

[66]:
```
```

[67]:
```
# Test :
mesLancers=lancersDes2(20)# On génère 20 lancers de dés.
print(mesLancers)
```

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

1.1.6 Listes de listes ou tuples de tuples

Les éléments d'une liste peuvent être des listes qui peuvent elle-même contenir des listes, etc. on peut donc avoir des listes imbriquées.

Par exemple, voici une liste qui contient 3 listes contenant chacune deux listes à deux éléments, et comment on accède aux différents éléments.

[77]:
```
L=[[[1,2],[3,4]],[[5,6],[7,8]],[[9,10],[11,12]]]
print("L = ", L)
print("L[1] = ",L[1])
print("L[1][0] = ",L[1][0])
print("L[1][0][1] = ",L[1][0][1])
```

```
L =  [[[1, 2], [3, 4]], [[5, 6], [7, 8]], [[9, 10], [11, 12]]]
L[1] =  [[5, 6], [7, 8]]
L[1][0] =  [5, 6]
L[1][0][1] =  6
```

Exercice 8 Ecrire une fonction `TrianglePascal(N)` qui renvoie une liste contenant les n premières lignes du triangle de Pascal. Vous la construirez en utilisant l'identité de Pascal.

[83]:
```
```

[97]:
```
# Test :
B=TrianglePascal(10)
print(B)
print("")
for L in B:
    print(L)
print("")
print("B[4][2]=",B[4][2])
```

```
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1],
[1, 6, 15, 20, 15, 6, 1], [1, 7, 21, 35, 35, 21, 7, 1], [1, 8, 28, 56, 70, 56,
28, 8, 1], [1, 9, 36, 84, 126, 126, 84, 36, 9, 1], [1, 10, 45, 120, 210, 252,
210, 120, 45, 10, 1]]

[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
[1, 7, 21, 35, 35, 21, 7, 1]
[1, 8, 28, 56, 70, 56, 28, 8, 1]
[1, 9, 36, 84, 126, 126, 84, 36, 9, 1]
[1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]

B[4][2]= 6
```