Les tuples et les listes avec Python

Table des matières

1) Introduction

Il est possible de "stocker" plusieurs grandeurs dans une même structure, ce type de structure est appelé une séquence : ensemble fini et ordonné d'éléments indicés de 0 à $n-1$ (si la séquence comporte n éléments).
Rappel : nous avons vu dans les chapitres précédents qu'il est possible d'accéder à n'importe quel caractère d'une chaîne de caractères à l'aide de son indice de position (le premier caractère étant indexé 0).

```
chaine="numerique"
print("Le premier caractère est ",chaine[0])
print("Le cinquième caractère est",chaine[4])
```

Une chaîne de caractère est donc bien un type de structure fini et ordonné d'éléments indexés de 0 à `len(chaine)-1`, c'est à dire une séquence.

```
chaine=input("Chaine de caractères =")
for i in range (len(chaine))
        print(chaine[i],end=".")
```

Il existe d'autres types de séquences que nous allons découvrir ici, notamment les n-uplets (encore appelés tuples) et les listes.

2) Les tuples

2.1) Définition

Un tuple est une collection d'éléments séparés par des virgules et généralement entouré de parenthèses même si ce n'est pas une obligation. Cela permet toutefois d'améliorer la lisibilité du code. Le tuple est une séquence immuable (une fois déclaré, le tuple ne peut plus être modifié) d'objets indicés qui peuvent être de tout type : entiers, flottants, chaînes de caractères...

```
tup=("mon age est",18,"ans")
print("tuple",tup)
```

2.2) Afficher les éléments d'un tuple

Comme pour une chaîne de caractères, il est possible de parcourir directement les éléments du tuple à l'aide de l'instruction `for i in tup`

```
tup=("mon age est",18,"ans")
print("tuple",tup)

for i in tup
        print i
```

Et également par un parcours des indices :

```
tup=("mon age est",18,"ans")
print("tuple",tup)

for i in range(len(tup))
        print (tup[i])
```

Comme avec une chaine de caractères, la fonction len() prenant un tuple comme argument renvoie le nombre d'éléments (longueur) du tuple.

Autre possibilité :

```
tup=("mon age est",18,"ans")
print("tuple",tup)

a,b,c=tup
```

```
print(a)
print(b)
print(c)
```

2.3) Immuabilité d'un tuple

Essayons de modifier un tuple prédéfini :

```
tup=("mon age est",18,"ans")
print("tuple",tup)

tup[2]=19
print(tup)
```

On constate que python refuse effectivement la modification d'un élément du tuple :

```
File "/Cours/NSI/Premiere/6-Tuples et listes/pg/tuples_erreurs.py", line 12, in <module>
tup[2]=19

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

2.4) Quel peut être l'intérêt d'un tuple ?

Nous souhaitons écrire une fonction qui prend en argument deux nombres entiers et nous renvoie le quotient et le reste de la division euclidienne du premier nombre par le second. Or nus avons vu qu'une fonction ne retourne qu'un seul élément ! Il est possible de solutionner ce problème en demandant à la fonction de renvoyer le résultat sous forme d'un tuple (quotient, reste).

```
def divEuclidienne(a:int,b:int)->tuple:
        tup=(a//b,a%b)
        return tup


a=100
y=9
quotient,reste=divEuclidienne(a,y)
print(quotient,reste)
```

3) Les listes

3.1) Définition

Tout comme un tuple, une liste est une collection d'informations qui peuvent être de même type ou de type différent. Les éléments de la liste sont séparés par des virgules et placés entre des crochets.

Création de listes par affectation :

```
list_int=[-12,-6,9,78]
list_float=[12.7,9.8,37.2]
list_string=["Janvier","Février","Mars","Avril","Mai","Juin"]
list_liste=[[9,8,7],[9,7,2]] # c'est une matrice
list_mixt=['Hubert',1.75,8,70e-3,[3,5,9],True]
list_vide=[]

#comme pour les chaînes de caractèreset tuples
#pour afficher les différents éléments de la liste
print(list_mixt)
print(list_liste)

#pour afficher le type de la liste
print(type(list_int))

#pour afficher la longueur de la liste (nombre d'éléments)
print(len(list_int))
print(len(list_float))
```

La grande différence entre une liste et un tuple, c'est que la liste est modifiable.

3.2) Affichage des divers éléments de la liste

Comme pour les tuples, chaque élément est repéré dans la liste par un indice. Là aussi, l'indexation commence à 0 et non pas à 1.

```
list_mois=["Janvier","Février","Mars","Avril","Mai","Juin"]
print("le premier mois est ",list_mois[0])
```

Il est donc possible d'afficher un par un l'ensemble des éléments d'une liste à l'aide d'une boucle `for` :

```
list_mois=["Janvier","Février","Mars","Avril","Mai","Juin"]
numero=["premier","second","troisième","quatrième","cinquième","sixième"]

for i in range(len(list_mois)):
        print("Le",numero[i],"mois de l'année indéxé",i,"est :",list_mois[i])

#qui peut s'écrire aussi
for i in range(len(list_mois)):
        print("Le {} mois de l'année indéxé {} est : {}".format(numero[i],i,list_mois[i] ))
```

Il est possible d'itérer directement les éléments de la liste (comme pour les chaînes de caractères) :

```
list_mois=["Janvier","Février","Mars","Avril","Mai","Juin"]
numero=["premier","second","troisième","quatrième","cinquième","sixième"]

for i in list_mois:
        print(i)
```

3.3) Ajout d'éléments dans une liste

Pour rajouter un élément dans une liste, par exemple 'juillet' à notre liste `liste_mois`, il est possible d'utiliser la méthode `append()` (to append signifie « ajouter »). L'élément est ajouté en fin de liste. La méthode `append()` ne permet d'ajouter qu'un seul élément à la fois.

```
list_mois=["Janvier","Février","Mars","Avril","Mai","Juin"]

list_mois.append("Juilllet")
print(list_mois)
```

Une autre méthode consiste à concaténer notre liste existante avec une autre liste. L'intérêt est que plusieurs éléments peuvent être rajoutés en même temps :

```
list_mois=list_mois+["Août","Septembre"]
print(list_mois)
```

Attention, dans l'exemple qui suit le mois d'octobre sera rajouté en début de liste et non en fin de liste :

```
list_mois=["Octobre"]+list_mois
print(list_mois)
```

Il est également possible d'utiliser la méthode extend() qui permet de concaténer non pas un seul élément mais une seconde liste.

```
list_mois.extend(["Novembre","Décembre"])
print(list_mois)
```

3.4) Suppression d'éléments dans une liste

Pour supprimer un élément dans une liste, par exemple l'élément de rang i, il est possible d'utiliser la commande `del` (`del` signifie « delete »).

```
del list_mois[0]
print(list_mois)
```

Il est possible de supprimer un élément dans un tableau à partir non pas de son indice, mais de sa valeur. La méthode `remove()` supprime uniquement la première occurrence trouvée.

```
list=[2,5,3,8,5,5]
list.remove(5)
print(list) #seul le premier 5 a disparu
```

Si l'élément n'est pas trouvé, un message d'erreur est retourné et le programme s'arrête, donc bien s'assurer auparavant que l'élément à supprimer est présent dans la liste.

```
ValueError:list.remove(x: x not in list)
```

```
list=[2,5,3,8,5,5]

while 5 in list:
        list.remove(5)
print(list)
```

3.5) Construction d'une liste par compréhension

Les compréhensions de listes fournissent un moyen de construire des listes de manière très concise.

Une compréhension de liste consiste à placer entre crochets une expression suivie par une boucle `for` (ou plusieurs boucles imbriquées) et éventuellement un test conditionnel pour filtrer.

Exemple 1 : supposons que l'on souhaite réaliser une liste contenant le carré des nombres de 1 à 10 :

Méthode classique

```
carre=[]
for i in range (1,11):
        carre.append(i**2)
print(carre)
```

Méthode par compréhension

```
carre=[i**2 for i in range(1,11)]
```

Il est possible de filtrer la liste précédente en ne prenant que le carré des nombres paires :

Méthode classique

```
carre=[]
for i in range (1,11):
        if i%2==0:
                carre.append(i**2)
print(carre)
```

Méthode par compréhension

```
carre=[i**2 for i in range(1,11) if i%2==0]
```

Exemple 2 création de la liste des caractères qui se trouvent dans une chaîne de caractère (itérable), dans l'ordre où ils sont rencontrés.

Méthode classique

```
mot="anticonstitutionnellement"
lst=[]
for i in range (len(mot)):
        lst.append(mot[i])
print(lst)
```

Méthode par compréhension

```
mot="anticonstitutionnellement"
lst=[i for i in mot]
print(lst)
```

Création de la liste des voyelles qui se trouvent dans une chaîne de caractère (itérable)

Méthode classique

```
mot="anticonstitutionnellement"
voyelle=["a","e","i","o","u","y"]
lst=[]
for i in mot:
        for j in voyelle:
                if i==j:
                        lst.append(j)
print(lst)
```

Méthode par compréhension

```
mot="anticonstitutionnellement"
voyelle=["a","e","i","o","u","y"]
lst=[i for i in mot for j in voyelle if i==j]
print(lst)
```

3.6) Des listes de listes pour représenter des matrices

Voici un exemple de liste de listes :

```
matrice=[[1,2,3],[4,5,6],[7,8,9]]
print(matrice)
```

Pour des raisons de lisibilité, il est cependant plus commode de la représenter de la manière suivante, sous forme matricielle :

```
matrice=[[1,2,3],
        [4,5,6],
        [7,8,9]]
print(matrice)
```

Pour cibler un élément de la matrice, on peut utiliser la notation avec des doubles crochets : matrice[ligne][colonne] Attention : les numéros de ligne et colonne démarrent à 0.

```
print(matrice[1][2]) # affiche l'élément en 2eme ligne et 3ème colonne
```

Pour affecter à cet élément la valeur 10 :

```
print(matrice[1][2])=10
print(matrice) #la valeur de l'élément a bien été modifié
```

Pour parcourir l'ensemble des éléments de la matrice, il faut utiliser 2 boucles for imbriquées :

```
matrice=[[1,2,3],
[4,5,6],
[7,8,9]]

nbLignes=3
nbColonnes=3

for i in range(nbLignes):
        for j in range(nbColonnes):
                print(matrice[i][j])
```

3.7) Pour aller plus loin (non exigible)

a) Trancher des listes (Slices)

On peut extraire d'un seul coup toute une partie de la liste : liste[a :b] renvoie la sous-liste des éléments de rang a à b – 1 (l'élément de rang b est exclu). List[:a] renvoie la sous-liste des éléments de rang 0 à a – 1 (l'élément de rang a est exclu). List [a :] renvoie la sous-liste des éléments de rang a inclus jusqu'au dernier.

```
list_nb=[0,1,2,3,4,5,6,7,8,9]

list_centre=list_nb[3:8]
print(list_centre)

liste_gauche=list_nb[:5]
print(list_gauche)

liste_droite=list_nb(5:)
print(liste_droite)
```

b) Des méthodes pour inverser et trier des listes

La méthode `reverse()` permet d'inverser une liste et la méthode `sort()` permet de trier la liste.

```
list=["z","d","g","f","a"]

list.reverse()
print(list)

list.sort()
print(list)
```

Documentation en ligne python : https://docs.python.org/fr/3/tutorial/datastructures.html

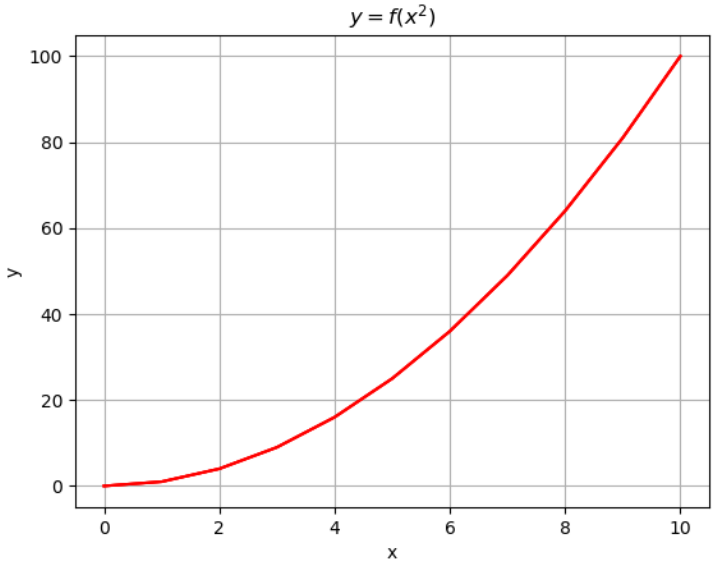
4) Exemple de trace d'une courbe en utilisant des listes et la bibliothèque-matplotlib.pyplot

Le programme suivant trace la courbe y=f(x2) en utilisant la bibliothèque matplolib.pyplot. Pour cela il est nécessaire de remplir une liste des abscisses des points de la courbe nommée x et une liste des ordonnées nommée y :

```
from matplotlib.pyplot import *

x=[]
y=[]

for i in range(11):
        x.append(i)
        y.append(i**2)
```

```
title("$y=f(x^2)$")
xlabel=("x")
ylabel=("y")

plot(x,y,color="red")
grid()
show()
```

Courbe obtenue :



5) Exercices

Exercice 1 :

Écrire une fonction `genereTab` qui prend en argument 2 entiers et retourne un tableau de longueur égale au 1er argument et dont chaque élément est un nombre entier aléatoire compris entre 0 et le 2 nd argument (inclus)

Exercice 2 :

Écrire une fonction `genereTabComp` identique à la précédente à l'exception que le tableau retourné est généré par compréhension

Exercice 3 :

Écrire une fonction `occurence` qui prend en argument un tableau et 1 valeur de type quelconque et retourne le nombre d'occurrences du 2nd argument dans le tableau.

Exercice 4 :

Écrire une fonction `extremes` qui prend en argument un tableau d'entiers et renvoie un tuple comportant la valeur minimale, moyenne et maximale du tableau.

Exercice 5 :

Écrire une fonction `decalDroite` qui prend en argument un tableau et un entier n, retourne le tableau avec tous les éléments décalés de n rangs vers la droite. Par souci d'optimisation de la mémoire, on ne créera pas un 2nd tableau.

Exercice 6 :

Écrire une fonction `inverse` qui prend en argument un tableau et retourne le tableau inversé. Par souci d'optimisation de la mémoire, on ne créera pas un 2nd tableau.