Algorithmique et Programmation 1

TD4 : N-uplets et Listes

1 Les n-uplets (tuples)

On représente les couleurs par un triplet d'entiers compris entre 0 et 255. Ces entiers représentent les composantes rouge, verte et bleue de la couleur. Par exemple (0,0,0) représente le noir et (255,255,255) représente le blanc. Voici un programme qui demande à l'utilisateur d'entrer le nom d'une couleur (on ne connaît que les couleurs blanc, noir, rouge, vert et bleu) et qui définit le triplet correspondant :

```
nomc = input("Entrer le nom d'une couleur : ")
if nomc=="noir":
   c = (0,0,0)
elif nomc=="blanc" :
   c = (255,255,255)
elif nomc=="rouge" :
   c = (255,0,0)
elif nomc=="vert" :
   c = (0,255,0)
elif nomc=="bleu" :
   c = (0,0,255)
else:
   print("Nous ne connaissons pas la couleur " + nomc)
```

Retenez simplement que ce programme fournit un triplet `c` dont les composantes ont été choisies par l'utilisateur.

1. On a une idée de l'intensité de la couleur en faisant la moyenne de ses composantes. Définir une variable entière `I` représentant l'intensité de la couleur `c` ;
2. Définir une nouvelle couleur `sombre` (un triplet donc) dont les composantes sont celles de `c` divisées par 2. Les composantes obtenues doivent toujours être entières (on pourra prendre la partie entière de la division).
3. Soit `d` une nouvelle couleur saisie par l'utilisateur (de la même façon que `c`). Définir une nouvelle couleur `cd` dont chacune des composantes est la somme de la composante correspondante de `c` et de `d`. Par exemple si `c=(255,0,0)` et `d=(0,127,0)`, alors `cd` devrait valoir `(255,127,0)`.

2 Les listes

Dans cette section `lst` désignera toujours une liste d'entiers entrée par l'utilisateur dont vous ne connaissez ni la longueur ni les éléments.

2.1 Lecture simple et modification en place

1. Écrire une boucle `for` qui parcourt la liste `lst` et afficher les éléments de cette liste multiplés par deux, sans modifier la valeur des éléments de la liste.
2. Écrire une boucle `for` qui parcourt la liste `lst` et qui double la valeur de chaque élément de la liste. Autrement dit, ce programme devra modifier `lst` en remplaçant chaque élément par une valeur deux fois plus grande.
3. Écrire une boucle `for` qui parcourt la liste `lst` et qui convertit tous les entiers en chaînes de caractères. Par exemple si, au départ, `lst=[3,5,4]` alors, après cette boucle, on devrait avoir `lst` devrait être la liste `["3","5","4"]`.

Variable solution

Lorsqu'on cherche un algorithme itératif pour résoudre un problème lié aux listes, il est souvent utile de définir une variable représentant la solution recherchée avec une valeur par défaut correspondant à la solution pour la liste vide. Votre algorithme devra alors parcourir la liste en modifiant la solution à chaque tour jusqu'à ce qu'elle ait la valeur souhaitée. Dans le cours, vous avez un programme qui calcule la somme des éléments d'une liste :

```
somme = 0 # voici la variable solution avec la valeur par defaut (0)
          # correspondant a la somme des elements d'une liste vide.
for e in lst :
   somme += e # somme est augmentee a chaque iteration

print(somme)
```

2.2 Algorithmes nécessitant une simple lecture

Pour les algorithmes nécessitant une simple lecture, lorsqu'on parcourt une liste, on n'a pas nécessairement besoin de connaître l'indice de chaque élément. Il suffit de connaître sa valeur.

1. Écrire une boucle `for` qui détermine si `lst` contient ou non l'entier 3. Dans cette question, vous n'avez pas le droit d'utiliser l'opérateur `in`. *Indication :* Vous pouvez par exemple définir une variable logique appelée `il_y_est` valant initialement `False` (puisque la liste vide ne contient pas 3). Ensuite, vous pouvez parcourir la liste et, le cas échéant, modifier la valeur de cette variable.
2. Écrire une boucle `for` qui détermine l'élément le plus petit de `lst`. La liste vide n'a pas d'élément minimal. Votre programme devra détecter lorsque la liste est vide et afficher un message d'erreur.
3. Est-ce que, suivant le choix que vous avez fait pour initialiser la variable solution, votre algorithme peut échouer à trouver cette solution ? Si oui, faire de façon à ce que votre algorithme n'échoue jamais pour une liste non-vide.
4. Écrire une boucle `for` qui détermine l'indice `n` du dernier élément de la liste pour lequel `lst[n]` vaut 3. Si aucun élément de la liste ne vaut 3, il faudra afficher un message d'erreur. Par exemple pour `lst=[8,1,3,7,3,0]`, le dernier élément valant 3 est l'élément d'indice 4. Si la liste ne contient pas de 3, le programme devra afficher un message d'erreur. *Indication :* vous pouvez définir une variable `n`. En parcourant la liste, dès que vous trouvez un élément valant 3, vous donnerez à `n` la valeur de l'indice de cet élément. Quelle devrait être la valeur initiale de `n` pour qu'à la fin de la boucle, vous sachiez si un élément 3 a été trouvé ?
5. Question difficile (à aborder si vous avez fini tout le reste). Écrire un programme déterminant si `lst` est croissant. On suppose que la liste vide et les singletons (listes contenant un seul élément) sont croissants. On peut définir une variable logique `est_croissant` et l'initialiser à `True`. À la différence des questions précédentes, cette question nécessite de connaître les indices des éléments parcourus.

2.3 Algorithmes nécessitant de construire une nouvelle liste

Dans cette section, on suppose que la variable `l` (`L` minuscule) représente un entier positif saisi par l'utilisateur. Il est inutile d'écrire l'instruction chargée de lire cette saisie. Comme dans la section précédente, vous pouvez créer une variable solution représentant la liste à construire et initialiser cette variable à la liste vide.

1. Écrire un programme qui crée une liste de `l` entiers tous égaux à 5.
2. Écrire un programme qui, au moyen d'une boucle `for`, crée une liste de `l` chaînes de caractères égaux aux premiers entiers. Par exemple si `l=3`, ce programme devra construire la liste `["1","2","3"]`.
3. Écrire un programme qui, crée une liste de `l` entiers choisis au hasard entre 0 et 255.

2.4 Algorithmes portant sur les listes de listes

Dans cette section, on suppose que `img` est une liste de listes d'entiers représentant une image. On suppose que chaque liste appartenant à `img` représente une ligne contenant le même nombre d'entiers et que chaque entier représente un pixel.

1. Définir, en fonction de `img`, deux variables `lg` et `ht` représentant respectivement la largeur et la hauteur de l'image en nombre de pixels.
2. Écrire, au moyen de boucles `for`, un programme qui modifie les valeurs de `img` de façon à ce que la première ligne soit uniquement constituée de 1, la deuxième ligne de 2 et la `n`-ième ligne de `n`. Par exemple si `img` est une liste de 4 listes de 3 entiers, on devrait obtenir la liste :

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

3. Dans cette question, on suppose que les variables `l` et `h` représentent deux entiers positifs saisis par l'utilisateur. Il est inutile d'écrire l'instruction chargée de lire ces saisies. Écrire un programme qui génère une liste de `h` listes, chacune de ces listes ayant `l` entiers égaux au numéro de la ligne (comme la liste `img` décrite ci-dessus).