Examen de 2nde session de Langages de scripts (LS4) n°

Documents autorisés : notes personnelles (hors ouvrages / polycopiés).

Vous pouvez à tout moment sauter une question et supposer écrite une fonction précédement demandée, à condition de l'indiquer clairement dans votre copie et de donner son prototype.

Les questions ne sont pas classées par ordre de difficulté !

Exercice 1 : Chiffrement d'images

On souhaite crypter une image appelée *secret* en la "cachant" dans une autre image appelée *clef*.

Chaque pixel a une couleur déterminée par trois nuances (rouge, vert et bleu). Chacune de ces nuances est codée sur 1 octet, soit 8 bits. On considère que les bits de poids fort, c'est-à-dire ceux qui compte vraiment, correspondent aux grandes puissances de 2 dans le codage de l'octet et que les bits de poids faible correspondent aux petites puissances de 2. Ainsi les nuances `11110000` et `11111111` (en binaire) sont proches alors que les nuances `11110000` et `00000000` sont très éloignées.

La fonction `que_fort` donnée ci-dessous permet d'"oublier" les bits de poids faible.

```
def que_fort(couleur):
    """renvoie la couleur avec uniquement les bits de poids fort"""
    return (couleur>>4)<<4
```

On mélange alors la nuance correspondant à la clef et au secret en fabriquant une nouvelle nuance dont les bits de poids fort sont donnés par les bits de poids fort de la clef et les bits de poids faible par les bits de poids fort du secret, comme expliqué à la figure 1, ce que fait la fonction `melange_nuance` donnée ci-dessous. On obtient le chiffrement de la nuance `secret` par la nuance `clef`.

```
def melange_nuance(clef, secret):
    """renvoie une nuance composee ainsi :
    - les bits de poids fort de la clef en poids fort
    - les bits de poids fort du secret en poids faible
    une nuance est un entier correspondant a rgb"""
    return que_fort(clef) + (que_fort(secret)>>4)
```

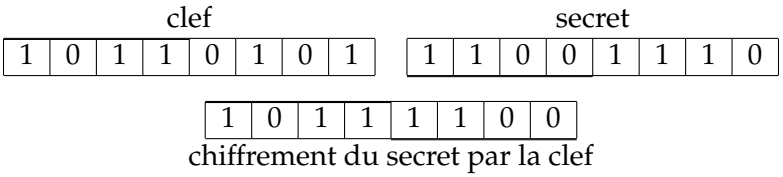

FIG. 1 – mélange de nuances

On appelle *couleur* un triplet de trois entiers, chacun de ces entiers correspondant à une nuance de rouge, vert ou bleu. Une *image* est une matrice bidimensionnelle de couleurs.

1. Ecrivez la fonction `melange_couleurs` qui prend en argument deux couleurs, `clef` et `secret`, et renvoie une couleur composée pour chaque nuance du chiffrement de la nuance correspondante du `secret` par la `clef`.

2. Ecrivez la fonction `melange_lignes` qui prend en argument deux listes de couleurs, `clef` et `secret`, supposées de même dimension et renvoie une ligne composée de couleurs correspondant au chiffrement des couleurs du `secret` par la `clef`.
3. Ecrivez la fonction `melange_images` qui prend en argument deux matrices bidimensionnelles de couleurs, `clef` et `secret`, supposées de mêmes dimensions et renvoie une matrice composée de couleurs correspondant au chiffrement des couleurs du `secret` par la `clef`.
4. Ecrivez une fonction `sauvegarde_image` qui permet de sauvegarder une image en mémoire telle que définie ci-dessus dans un fichier `PPM P3` (*ascii*) dont la profondeur de couleur est 255.

Exercice 2 : Manipulation de listes

Le but de cet exercice est d'utiliser au mieux les possibilités apportées par python. On attend du code en réponse.

1. On dispose d'une liste d'éléments quelconques `L` dans laquelle certains éléments peuvent apparaître plusieurs fois. Comment obtenir une liste contenant les mêmes éléments sans répétition ?
2. On dispose d'une liste de listes `L`. Comment obtenir une liste qui contient tous les éléments des sous-listes dans leur ordre d'apparition ?
   exemple : `[[1,2,3],[4,5],[],[4,3,2],[1]]` $\longrightarrow$ `[1,2,3,4,5,4,3,2,1]`
3. On dispose d'une liste `L` et on veut la partager en sous-listes de longueur un entier `n` donné. Comment faire ?

Exercice 3 : Listes d'amis

On souhaite programmer un script de gestion de réseau social du style de certains sites Internet à succès (« réseau d'amis »). On va écrire un script permettant de gérer un certain nombre de personnes, et pour simplifier on ne s'intéresse qu'à une seule information, à savoir la liste d'amis de chaque personne.

Se souvenir de ses amis. On se demande comment représenter un réseau d'amis en Python.

La relation d'amitié n'est pas supposée symétrique : si Théodore considère Iphigénie comme son amie, cela ne veut pas dire qu'Iphigénie considère Théodore comme son ami. Par contre, tous les amis d'une personne doivent être des utilisateurs connus du système. Enfin, on ne peut pas être son propre ami.

1. Choisir une structure de données adaptée pour représenter un réseau d'amis, et donner un exemple avec trois utilisateurs ayant respectivement zéro, un et deux amis.

Chargement et sauvegarde. On souhaite pouvoir charger et enregistrer les réseaux d'amis dans des fichiers texte. Pour cela, chaque ligne représentera un utilisateur et sa liste d'amis, le tout séparé par des points d'exclamation. Par exemple, si les amis de Théodore sont Iphigénie et Maxence, le fichier contiendra la ligne :

```
Théodore!Iphigénie!Maxence
```

Un utilisateur peut avoir une liste d'amis vide. On suppose dans ce cas-là que la ligne ne se termine *pas* par un point d'exclamation.

2. Écrire la fonction `sauver_amis` prenant comme argument un réseau d'amis et un fichier et sauvegardant le réseau d'amis dans ce fichier au format indiqué ci-dessus.
3. Écrire la fonction `charger_amis` prenant comme argument un nom de fichier et renvoyant un réseau d'amis comme défini à la question précédente.

   Si deux lignes commencent par le même nom d'utilisateur, on affichera un message d'erreur à l'écran, et la fonction renverra la valeur `None`.

   Remarque : Si le fichier contient des lignes vides (blanches), elles doivent être ignorées. Les noms d'utilisateurs peuvent contenir des espaces.

Vérification d'un réseau d'amis. On souhaite éviter des problèmes pendant la manipulation du réseau d'amis en vérifiant que chaque nom mentionné dans une la liste d'amis d'un des utilisateurs correspond bien à un utilisateur. D'autre part, on interdit qu'un utilisateur apparaisse dans sa propre liste d'amis. Ces deux cas peuvent se produire si on charge un fichier incorrect.

4. Écrire la fonction `verif_amis` prenant comme argument un réseau d'amis et renvoyant `True` si tous les noms apparaissant dans le réseau sont des utilisateurs connus (c'est-à-dire possédant eux-même une liste d'amis) et si aucun utilisateur n'est son propre ami.

Amis communs. On souhaite pouvoir déterminer si deux utilisateurs ont des amis communs.

5. Écrire la fonction `amis_communs` prenant comme argument un réseau d'amis et deux noms d'utilisateurs et renvoyant la liste des personnes qui apparaissent dans leurs deux listes d'amis.

De qui suis-je l'ami ? On souhaite pouvoir indiquer à chaque utilisateur quelles autres personnes le ou la considèrent comme leur ami(e).

6. Écrire la fonction `ami_de_qui` prenant comme argument un réseau d'amis et un nom d'utilisateur et renvoyant la liste des personnes ayant cet utilisateur dans leur liste d'amis.

Amis réciproques. On souhaite pouvoir indiquer à chaque utilisateur lesquels de ses amis l'ont inscrit sur leur liste d'amis.

7. Écrire la fonction `amis_reciproques` prenant comme argument un réseau d'amis et un nom d'utilisateur et renvoyant la liste des amis de cet utilisateur ayant eux aussi cet utilisateur dans leur liste d'amis.

Les amis de mes amis... Lorsqu'on rencontre une nouvelle personne, il est intéressant de savoir si on a des amis en commun. Pour cela, on se propose de réaliser un ensemble de fonctions indiquant à un utilisateur s'il a un lien avec une autre personne du réseau.

8. Écrire une fonction `amis_indirects_2` prenant comme argument un réseau et un nom d'utilisateur, et renvoyant la liste des personnes qui sont des amis d'amis de cet utilisateur (attention, cette liste ne doit pas contenir les amis directs de l'utilisateur, ni bien sûr l'utilisateur lui-même !).
9. Écrire une fonction `amis_indirects_n` prenant comme argument un réseau, un nom d'utilisateur et un entier `n`, et renvoyant la liste des personnes qui sont des amis indirects de « distance » n de cet utilisateur (attention, cette liste ne doit pas contenir les amis de distance strictement inférieure à `n` de l'utilisateur !).

10. Écrire une fonction `ami_eloigne` prenant comme argument un réseau et deux noms d'utilisateurs et renvoyant un entier `n` si le second utilisateur est un ami indirect (de distance `n`) du premier utilisateur, et -1 sinon.