Examen de 1ère session de Langages de scripts (LS4)

Jeudi 28 mai 2009

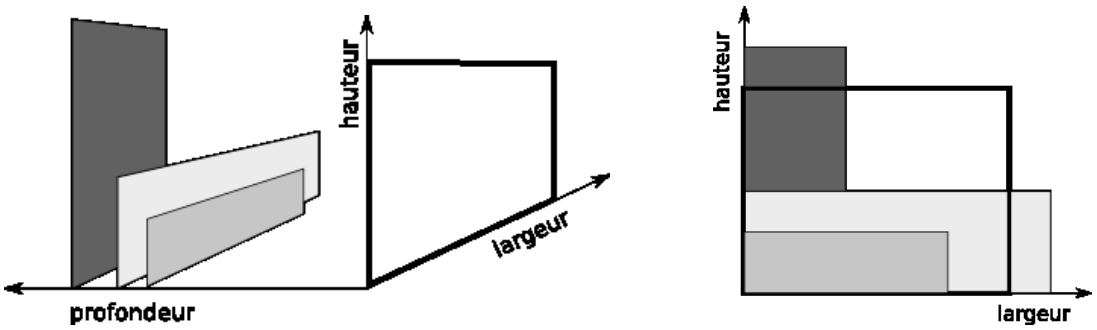
Durée : 3h

Attention : une feuille de réponses est jointe à cet énoncé, pour l'exercice 3. Vous devez l'agrafer à votre copie et inscrire le code qui y figure sur toutes vos copies.

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

Exercice 1 : Algorithme du peintre

On suppose qu'un observateur regarde un ensemble de rectangles colorés à travers une fenêtre (voir figure 1). Pour simplifier, on supposera d'une part que tous les rectangles ont une profondeur négligeable (comme des feuilles de papier) et des côtés parallèles aux bords de la fenêtre, et d'autre part que l'observateur se situe infiniment loin, ce qui évite d'avoir à gérer les problèmes de perspective.



(a) Vue en perspective (b) Vue de « devant »

FIG. 1 – Exemple avec trois rectangles colorés

On définit les structures de données suivantes. On utilisera le mot *point* pour désigner un point sur l'un des rectangles et le mot *pixel* pour désigner un point de la fenêtre.

- Un pixel est donné par un couple de coordonnées entières correspondant aux axes de la fenêtre.
- Un point est donné par un triplet d'entiers, les deux premières coordonnées correspondent aux axes de la fenêtre d'observation, la troisième à la profondeur à laquelle se situe le point (voir figure 1).
- Un rectangle est donné par un couple de points de même profondeur (qui représentent deux coins opposés du rectangle).
- Un rectangle coloré est un couple dont la première composante est un rectangle et la deuxième une couleur représentée par un triplet d'entiers.

Au début du problème, on dispose d'une liste de rectangles colorés et des dimensions de la fenêtre d'observation. On veut déterminer la couleur de chaque pixel vu par la fenêtre. Pour celà nous vous proposons deux méthodes.

Première méthode : On boucle sur les pixels.

1. On dit qu'un rectangle contient un pixel si, en oubliant la profondeur du rectangle, le pixel se situe à l'intérieur de celui-ci. Comment peut-on récupérer la liste des rectangles qui contiennent un pixel donné ? (on demande du code)
2. Pour chaque pixel on récupère la liste des rectangles qui ont une intersection non vide avec la fenêtre d'observation et qui contiennent ce pixel. Puis on prend le minimum de hauteur sur cette liste de rectangles, afin de récupérer la couleur correspondante.
   Ecrire la fonction `peintre_pixels` correspondante.

Seconde méthode : On boucle sur les rectangles.

3. Comment peut-on récupérer la liste des rectangles classée du plus éloigné (profondeur maximale) au plus proche (profondeur minimale) ? (on demande du code)
4. On parcourt les rectangles du plus éloigné au plus proche. Pour chaque rectangle on colorie les pixels qu'il contient, sans se soucier de savoir s'il est déjà coloré ou non.
   Ecrire la fonction `peintre_rectangles` correspondante.

Exercice 2 : SQL de base

On veut simuler un système de gestion de base de données relationnelles et écrire quelques commandes SQL de base. L'énoncé comportera pour chaque question un exemple de requête SQL correspondant à ce qu'on veut obtenir ainsi que l'explication de cette requête. La question 5 fait exclusivement référence aux requêtes ayant la forme de celles données dans l'énoncé.

On rappelle qu'on peut voir une table dans une base de données comme un tableau bidimensionnel. Chaque ligne correspond à une entrée de la table, chaque colonne à un attribut (voir figure 2). Le langage de requête SQL permet entre autres d'ajouter et de supprimer des lignes ou de sélectionner des lignes et des colonnes d'une table donnée.

Une entrée dans une table sera représentée par un dictionnaire dont les clefs (au sens python du terme) sont les attributs de la table. Une table est donc une liste de dictionnaires (voir figure 2). Par abus de langage, dans un contexte python le mot *table* désignera une telle liste.

Les requêtes SQL qui renvoient un résultat (les sélections) le renvoie sous forme de table, vos fonctions devront donc renvoyer leurs résultats sous forme de liste de dictionnaires tels que décrite ci-dessus.

table "empilements"

id	couleur	forme
132	bleu	rond
456	rouge	carre
234	rouge	triangle
987	vert	triangle

```
empilements =
[{'id':132,'couleur':'bleu','forme':'rond'},
{'id':456,'couleur':'rouge','forme':'carre'},
{'id':234,'couleur':'rouge','forme':'triangle'},
{'id':987,'couleur':'vert','forme':'triangle'}]
```

FIG. 2 – exemple de table dans une base de données et sa traduction en Python

1. Lire une base dans un fichier.
2. Ajouter des données :

```
INSERT INTO table ('attr1','attr2',...,'attrn')
VALUES (val1,val2,...,valn)
```

Ecrivez une fonction `insert` qui prenne en argument une table et un tuple de chaînes de caractères de la forme `'attribut=valeur'` et qui ajoute cette entrée dans la table. Un exemple d'appel de cette fonction est donné par
`insert(empilements, ('id=321', 'couleur=violet', 'forme=hexagone'))`

3. Sélectionner des données.
   (a) Sélectionner un attribut dans une table : `SELECT attribut FROM table`
   Ecrire une fonction `select_1x1` qui permet de sélectionner un attribut dans une table, l'attribut étant donné sous forme de chaîne de caractères. Vous pouvez voir un exemple figure 3.

requête SQL : `SELECT couleur FROM empilements`

résultat de la requête (sur la table `empilements` initiale) :

couleur
bleu
rouge
rouge
vert

appel correspondant en python : `select_1x1(empilements, 'couleur')`

FIG. 3 – exemple de sélection d'un attribut dans une table

   (b) Sélectionner plusieurs attributs dans une table :
   `SELECT attribut1,...,attributn FROM table`
   Ecrire une fonction `select_nx1` qui permet de sélectionner plusieurs attributs dans une table, les attributs étant donnés sous forme de tuple de chaînes de caractères.
   (c) Ecrire une fonction `select` dont le comportement est d'appeler une des deux fonctions `select_1x1` ou `select_nx1` écrites aux questions précédentes en fonction de la nature des arguments donnés.

4. Supprimer des données :

```
DELETE * FROM table
WHERE condition
```

Ecrivez une fonction `delete` qui prenne en argument une table et une condition qui permet de supprimer des entrées de la table en fonction des valeurs de leurs attributs (colonnes). On se limitera aux conditions de comparaison entre la valeur d'un attribut et une constante ou entre les valeurs de deux attributs. Les conditions seront exprimées sous forme de chaînes de caractères faisant apparaître des noms d'attributs, des signes de comparaison (<, = ou >) et des constantes.
La fonction *builtin* `eval` dont voici un extrait de l'aide permet d'évaluer une expression donnée par une chaîne de caractères syntaxiquement correcte.

```
Help on built-in function eval in module __builtin__:

eval(...)
    eval(source) -> value

    Evaluate the source.
    The source may be a string representing a Python expression.
```

5. Ecrire une requête.
   Ecrire une fonction `requete` qui prendra comme argument une chaîne de caractères représentant une requête `sql` simple (c'est-à-dire telle que rencontrée dans les questions précédentes) et une table et appellera la fonction correspondante avec les bons arguments, étant entendu que la valeur de la table dont le nom apparaît dans la requête est donnée par le deuxième argument.

Exercice 3 : Mutation de listes

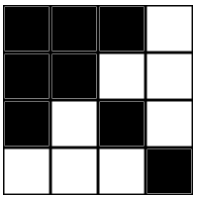
Sur la feuille de réponses jointe à l'énoncé, vous indiquerez les valeurs des listes obtenues.

Exercice 4 : Compréhension

Votre binôme préféré vous a envoyé un script `exo4.py` (figure 4) et vous demande de le vérifier avant de le renvoyer au prof pour correction. Malheureusement, il n'a pas du tout commenté son code et a utilisé des noms de variables incompréhensibles. Pour rendre les choses encore plus difficiles, vous avez perdu l'énoncé du TP et ne savez plus du tout ce que le script devait faire. Vous vous souvenez juste qu'il était question d'**images en noir et blanc**.

Indications : Il est recommandé de lire l'énoncé de l'exercice en entier (listing inclus) **avant** de commencer. Le principal travail ici est de comprendre le script fourni. Vous pouvez répondre aux questions dans l'ordre que vous souhaitez. La question 1 notamment est transversale : complétez-la au fur et à mesure de votre avancée dans l'exercice.

1. Proposez des noms plus explicites pour chacune des variables et des fonctions du script, ainsi que des noms longs pour les options `-f` et `-d`. Vous pouvez éventuellement garder le nom existant d'une variable s'il vous parait déjà adapté.
2. La fonction `l` manipule des fichiers représentant des images en noir et blanc. Décrivez rapidement le format de ces fichiers, et donnez le contenu du fichier permettant de représenter l'image 4×4 suivante :



3. Proposez un meilleur message d'erreur à la ligne 65 du listing. Dans quel(s) cas ce message est-il affiché (ne pas répondre « si `i` vaut `None` » !) ?
4. Expliquez de quelle manière ce script doit être utilisé, en détaillant en particulier ce qui se passe quand on l'importe dans l'interpréteur Python et quand on le lance directement depuis le shell. Dans les deux cas, donnez des exemples d'utilisation.
5. Quel est l'effet de la méthode `error` appelée à la ligne 40 ? Proposez un meilleur message d'erreur que le message actuel. Donnez un exemple d'utilisation du script qui entraînera l'affichage de cette erreur.
6. Déroulez l'exécution de la dernière ligne du script sur l'image donnée à la question 2, en supposant que la variable `d` vaut `False`. Quelle valeur est affichée à la fin du calcul ?
7. Même question si `d` vaut `True`.
8. Expliquez en quelques mots ce que calcule ce script, et quelle est l'idée de l'algorithme utilisé.

```
#!/usr/bin/env python

def n(i):
    c = 0
    for (x,l) in enumerate(i):
        for (y,p) in enumerate(l):
            if p:
                c+=1
                e(i,x,y)
    return c

def e(i, x, y):
    if (x < 0 or x >= len(i)
        or y < 0 or y >= len(i [0])):
        return
    elif not i[x][y]:
        return
    else:
        i[x][y] = False
        for (dx,dy) in [(0,1),  (1,0),
                        (0,-1), (-1,0)]:
            e(i, x+dx, y+dy)
        if d:
            for (dx,dy) in [(1,1),  (1,-1),
                            (-1,-1), (-1,1)]:
                e(i, x+dx, y+dy)

def q():
    p = OptionParser()
    p.add_option("-d", dest="d",
                 action="store_true",
                 default=False)
    p.add_option("-f", dest="f")
    o, args = p.parse_args()
    if o.f and not args:
        f = o.f
    elif not o.f and len(args) == 1:
        f = args[0]
    else: p.error("Message d'erreur 1.")
    return f, o.d

def l(f):
    df = open(f)
    i = []
    for l in df:
        i.append([])
        for p in l.strip ():
            if int(p) == 1:
                i[-1].append(True)
            elif int(p) == 0:
                i[-1].append(False)
            else:
                return None
        if len(i[-1]) != len(i [0]):
            return None
    return i

d = False
if __name__ == "__main__":
    from optparse import OptionParser
    f, d = q()
    i = l(f)
    if i == None:
        print "Message d'erreur 2."
    else:
        print n(i)
```

FIG. 4 – Le script `exo4.py`