

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 précédemment 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.

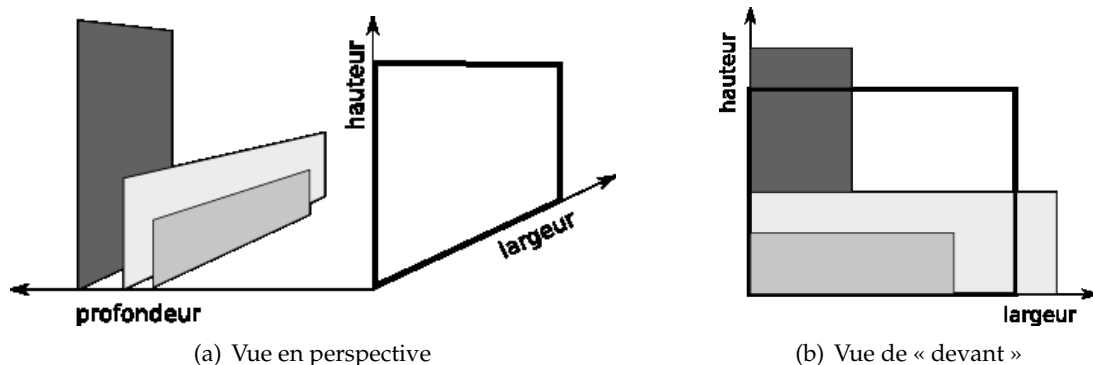


FIG. 1 – Exemple avec trois rectangles colorés

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 cela 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"			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'}]
id	couleur	forme	
132	bleu	rond	
456	rouge	carre	
234	rouge	triangle	
987	vert	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 (val 1, val 2, ..., val n)
```


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

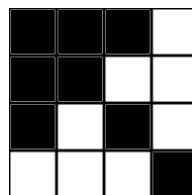
Sur la feuille de réponses jointe à l'énoncé, vous indiquerez les valeurs des listes obtenues.

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 paraît 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.

```

1  #!/usr/bin/env python

3  def n(i):
4      c = 0
5      for (x,l) in enumerate(i):
6          for (y,p) in enumerate(l):
7              if p:
8                  c+=1
9                  e(i,x,y)
10     return c

12 def e(i, x, y):
13     if (x < 0 or x >= len(i)
14         or y < 0 or y >= len(i [0])):
15         return
16     elif not i[x][y]:
17         return
18     else:
19         i[x][y] = False
20         for (dx,dy) in [(0,1), (1,0),
21                         (0,-1), (-1,0)]:
22             e(i, x+dx, y+dy)
23         if d:
24             for (dx,dy) in [(1,1), (1,-1),
25                             (-1,-1), (-1,1)]:
26                 e(i, x+dx, y+dy)

28 def q():
29     p = OptionParser()
30     p.add_option("-d", dest="d",
31                 action="store_true",
32                 default=False)
33     p.add_option("-f", dest="f")
34     o, args = p.parse_args()

36     if o.f and not args:
37         f = o.f
38     elif not o.f and len(args) == 1:
39         f = args[0]
40     else: p.error("Message d'erreur 1.")
41     return f, o.d

43 def l(f):
44     df = open(f)
45     i = []
46     for l in df:
47         i.append([])
48         for p in l.strip():
49             if int(p) == 1:
50                 i[-1].append(True)
51             elif int(p) == 0:
52                 i[-1].append(False)
53             else:
54                 return None
55         if len(i[-1]) != len(i [0]):
56             return None
57     return i

59 d = False
60 if __name__ == "__main__":
61     from optparse import OptionParser
62     f, d = q()
63     i = l(f)
64     if i == None:
65         print "Message d'erreur 2."
66     else:
67         print n(i)

```

FIG. 4 – Le script exo4. py