Examen de 1re session de Langages de script

15 mai 2008

Durée : 3 heures

Documents autorisés : notes personnelles

Exercice 1 : compréhension

L'objectif de cet exercice est de comprendre un script existant. Commencer par lire attentivement le code donné dans le listing 1 (page 4 du sujet), puis répondre aux questions.

1. À quoi sert le code à la première ligne du script (`from random import choice`) ?
2. Quel est le type de la valeur contenue dans la variable `stats_lettres` ? Donner une explication détaillée.
3. Même question pour la variable `questions`.
4. Quelle est la signification de la valeur `stats_lettres['A']` ?
5. Le listing contient des commentaires numérotés de 1 à 12. Proposer une ou deux lignes de commentaire à placer à chacun de ces emplacements pour rendre le code plus clair.
6. Décrire *brièvement* (5 à 10 lignes au *grand maximum* !) ce que fait ce script, puis en donner un exemple d'utilisation.
7. Modifier le script pour qu'il affiche le message *"Cette question n'a servi à rien !"* si la question ne change pas le nombre de possibilités restantes, et *"Bonne question !"* si au contraire elle a permis de le réduire de moitié au moins.
   Note : *il est inutile de recopier le code du script en entier, servez-vous des numéros de ligne pour indiquer où il faut insérer du code.*
8. Modifier le script pour qu'il affiche le message *"Cette question a déjà été posée !"* si l'utilisateur pose une question qu'il a déjà posée auparavant.

Exercice 2 : « `tr.py` »

La commande « `tr` » est un filtre réalisant des modifications lettre à lettre de son entrée standard, le résultat étant écrit sur la sortie standard. Ces modifications peuvent être soit des remplacements (comportement par défaut), soit des suppressions (option `-d`). Dans son comportement par défaut, « `tr` » prend en argument deux chaînes de caractères de même longueur, qui sont interprétées comme des listes de caractères, le i^e de la première devant être remplacé par le i^e de la deuxième. Par exemple :

```
$ echo 'aaabaabb' | tr a c # remplace tous les a par des c
cccbccbb
$ echo 'aaabaabb' | tr abc def # remplace a par d, b par e et c par f
dddeddee
$ echo 'aaabaabb' | tr ab bc # remplace a par b et b par c (mais pas a par c !)
bbbcbbcc
```

Avec l'option `-d`, « `tr` » prend un seul argument, à nouveau interprété comme la liste des caractères concernés.

```
$ echo 'aaabaabccb' | tr -d ac # supprime les a et les c
bbb
```

On souhaite écrire un script PYTHON se comportant de manière analogue, et acceptant en outre les options `-i` et `-o` modifiant les canaux d'entrée et de sortie. La gestion des options sera réalisée de la manière suivante :

```
from optparse import OptionParser
parser = OptionParser()
parser.add_option('-d', '--delete', action = 'store_true', dest = 'delete')
parser.add_option('-i', '--input', action = 'store', dest = 'input')
parser.add_option('-o', '--output', action = 'store', dest = 'output')
parser.set_defaults(delete = False, squeeze = False, input = '', output = '')
opts, args = parser.parse_args()
```

1. Écrire une fonction `verif_arguments` renvoyant `True` si le nombre d'arguments est correct.
2. Écrire une fonction `remplacements` renvoyant un *dictionnaire* représentant les couples (*motif, rempl*) déduits de l'interprétation de la ligne de commande.
3. Écrire une fonction `ouvrir_entree_sortie` renvoyant un couple de fichiers ouverts correspondant aux canaux d'entrée et de sortie. On rappelle que `sys.stdin` et `sys.stdout` correspondent à l'entrée et à la sortie standard et se manipulent comme des fichiers ordinaires *déjà ouverts*.
4. Écrire une fonction `remplacer(dico, input, output)` réalisant les remplacements codés dans `dico`, à partir de `input`, et écrivant le résultat sur `output`.

Exercice 3 : classes

L'objectif de cet exercice est de manipuler des fichiers correspondant à des photos de manière à pouvoir constituer des albums, les stocker, etc.

On suppose que les fichiers photos sont les fichiers ayant le suffixe "`jpg`".

Un album photo est constitué d'un ensemble de photos. Plusieurs albums différents peuvent avoir des photos en commun et afin d'éviter de dupliquer ces fichiers (qui peuvent être de grande taille !), on va constituer une base de données sous la forme d'un *dictionnaire* contenant une *fiche* par photo. Chaque fiche comportera un numéro unique et différentes informations sur le fichier photo associé et un *album* sera représenté par un ensemble de numéros de fiche.

Dans cet exercice, on va d'abord définir la classe `FichePhoto`, construire une photothèque (la base de données contenant toutes les fiches), puis on abordera la classe `Album`.

La classe `FichePhoto` :

1. Définir une classe `FichePhoto` avec un constructeur à 1 argument (le nom complet du fichier correspondant à la fiche en construction) et initialisant les champs suivants :
   - un champ `numero` pour stocker le numéro de la fiche (on utilisera le numéro d'inode du fichier associé, cf. ci-dessous) ;
   - un champ `nom` (une chaîne de caractères) pour représenter le nom *absolu* du fichier ;
   - un champ `taille` pour stocker la taille du fichier ;
   - un champ `date` (de type entier) pour stocker la date de la création de la photo (c.-à-d. du fichier associé).

   Précisions : On peut obtenir la taille d'un fichier `f` avec la fonction `os.path.getsize(f)`, sa date de création avec `os.path.getctime(f)` et son numéro d'inode avec `os.stat(f)[1]`.

2. Définir une méthode `ajouter_theme` prenant en argument une chaîne de caractères `s` et qui ajoute `s` à un champ `themes` (de type liste) de la fiche concernée. On pourra ainsi associer à chaque photo une liste de thèmes différents.
3. Écrire une méthode `__str__` pour afficher les informations contenues dans une fiche.

Construction de la photothèque : on va écrire une fonction `phototheque` qui prend en argument un répertoire `rep` et qui renvoie un *dictionnaire* contenant les fiches de tous les fichiers photos présents dans le répertoire `rep` ou l'un de ses sous-répertoires. On rappelle qu'étant donné un nom de fichier `f` relatif au répertoire courant, on peut obtenir son nom absolu avec la méthode `os.path.abspath(f)`. On utilisera le numéro de chaque fiche comme clé dans le dictionnaire renvoyé par la fonction `phototheque`.

4. Écrire la fonction `phototheque`.

La classe `Album` : un album photo sera représenté par un *ensemble* de numéros de fiches photos d'une photothèque. A partir de ces numéros, il sera facile de retrouver les informations relatives aux photos correspondantes grâce à la photothèque.

5. Définir une classe `Album` avec un constructeur à 2 arguments : un dictionnaire `D` et une liste `L`. Le dictionnaire `D` correspond à une photothèque et servira à initialiser un champ `phototeq` de l'objet `Album` en construction. La liste `L` correspond à une liste de numéros de fiches photos de la photothèque. Le constructeur devra aussi initialiser un champ `contenu` de type *ensemble* contenant les numéros de `L` après avoir vérifié que chaque numéro correspond à une fiche.
6. Écrire une méthode `ajouter` qui prend un numéro et ajoute la fiche photo correspondante (si une telle fiche existe dans la photothèque).
7. Écrire une méthode `selection_theme` qui prend une chaîne de caractères `s` et ajoute à l'album toutes les photos de la photothèque dont un thème est `s`.
8. Écrire une méthode `__str__` pour afficher toutes les fiches contenues dans un album.
9. Écrire une méthode `taille` qui renvoie la somme de toutes tailles des photos contenues dans l'album.
10. Écrire une méthode `derniere_photo` renvoyant la photo la plus récente de l'album.
11. Écrire une méthode `sauvegarde` prenant en argument un nom de répertoire et qui copie les fichiers photos de l'album dans ce répertoire. On pourra utiliser la fonction `copy(source,dest)` du module `shutil`.

```
1   from random import choice

3   stats_lettres={'A': (0, 2, 2, 1, 0), 'B':(2, 0, 0, 3, 1), 'C': (1, 2, 0, 0, 0), 'D':(1, 0, 0, 0, 1),
4                  'E': (0, 3, 0, 3, 1), 'F':(0, 3, 0, 2, 1), 'G': (1, 2, 0, 1, 1), 'H':(0, 4, 0, 1, 2),
5                  'I': (0, 2, 0, 0, 1), 'J':(1, 2, 0, 0, 1), 'K': (0, 4, 2, 0, 1), 'L':(0, 2, 0, 1, 1),
6                  'M': (0, 2, 2, 0, 2), 'N':(0, 2, 1, 0, 2), 'O': (1, 0, 0, 0, 0), 'P':(1, 1, 0, 2, 1),
7                  'Q': (1, 2, 1, 0, 0), 'R':(1, 2, 1, 0, 1), 'S': (1, 2, 0, 0, 0), 'T':(0, 3, 0, 1, 1),
8                  'U': (1, 2, 0, 0, 2), 'V':(0, 2, 2, 0, 0), 'W': (0, 2, 4, 0, 0), 'X':(0, 4, 2, 0, 0),
9                  'Y': (0, 3, 2, 0, 1), 'Z':(0, 2, 1, 2, 0)}

11  questions={'courbes': 0, 'extremites' : 1, 'obliques' : 2, 'horizontales' : 3, 'verticales' : 4}

13  def une_partie():
14      lettres       = stats_lettres.keys()
15      choix         = choice( lettres )                          # commentaire 1
16      possibilites  = lettres [:]                                # commentaire 2

18      while (True):
19          try:
20              entree = raw_input('Ensuite ?')
21          except (EOFError, KeyboardInterrupt):                  # commentaire 3
22              print '\nOK, abandonne si tu veux.'
23              return

25          if len(entree)==1:                                     # commentaire 4

27              if entree not in possibilites :                    # commentaire 5
28                  print ("Faux ! Cette lettre n'est pas compatible "
29                         "avec les informations que je t'ai donnée.")

31              elif len( possibilites )>1:                        # commentaire 6
32                  temp = [lettre for lettre in possibilites
33                          if lettre != entree]
34                  contre_exemple = choice(temp)
35                  print "Tu n'as pas encore assez d'information."
36                  print "Comment sais-tu que ce n'est pas un",
37                  print contre_exemple + ", par exemple ?"

39              else:                                              # commentaire 7
40                  print "Oui, c'est ça. Bien joué !" ; return

42          elif questions.has_key(entree):                        # commentaire 8

44              champ = questions[entree]                          # commentaire 9
45              reponse = stats_lettres[choix][champ]
46              print "Il y a", reponse, entree, "dans la lettre cherchée."

48              possibilites  = [lettre                            # commentaire 10
49                               for lettre in possibilites
50                               if stats_lettres [ lettre ][champ] == reponse]

52          else:                                                  # commentaire 11
53              print "Je ne comprends pas la question."

55  if __name__ == "__main__":                                     # commentaire 12
56      une_partie()
```

Listing 1 – Le script `lettres.py`.