Examen Programmation ENSAE rattrapage

première année 2009

Examen écrit (1 heure)

Tous documents autorisés.

1 Mise à l'échelle

Une rue vient d'être ajoutée à la ville de Paris. Elle comporte 3 numéros impairs et 5 numéros pairs.

1) On souhaite écrire une fonction qui retourne séparément la liste des numéros pairs et la liste des numéros impairs. Compléter la fonction suivante en remplaçant ..... par ce qu'il faut. `nbimpair` est le nombre de numéros impairs, `nbpair` est le nombre de numéros pairs. Pour 3 numéros impairs et 5 numéros pairs, on devrait obtenir $[1, 3, 5]$ et $[2, 4, 6, 8, 10]$. (1 point)

```
def numero_simple (nbimpair, nbpair) :
    im = [..... for i in range (1, nbimpair+1) ]
    pa = [..... for i in range (1, nbpair  +1) ]
    return im,pa
```

2) Dans l'exemple précédent, le dernier numéro impair est 5 alors que le dernier numéro pair est 10. On voudrait que ces derniers numéros soient plus proches. On décide alors qu'un immeuble pourra recevoir un, deux ou plusieurs numéros consécutifs (pairs ou impairs) s'il est grand. C'est l'objectif de la fonction `numero_deux`. `impair` représente la liste des longueurs de chaque immeuble du côté pair. `pair` représente la liste des longueurs du côté pair.

```
def numerotation (longueur, debut, moy) :
    res = []
    for li in range (0, len (longueur)) :
        if len (res) == 0 : x1 = debut
        else :              x1 = res [-1][1]+2

        nb = longueur [li] / moy                # c'est une division entière
                                                # ligne importante
                                                # pour la dernière question
        x2 = x1 + nb*2
        res.append ( (x1,x2) )
        debut = x2 + 2
    return res

def numero_deux (impair, pair) :
    moypair   = sum (pair)   / len (pair)       # vaut 25
    moyimpair = sum (impair) / len (impair)     # vaut 41
    im        = numerotation (impair, 1, moyimpair)
    pa        = numerotation (pair,   2, moypair)
    return im,pa

impair = [20,70,35]
pair   = [20,20,40,35,10]
im,pa  = numero_deux (impair, pair)
print im
print pa
```

Ce programme contient une ligne inutile : si elle est retirée, le résultat ne change pas. Quelle est-elle ? (1 point)

3) Le programme affiche cela :

```
[(1, 1), (3, 5), (7, 7)]
[(2, 2), (4, 4), (6, 8), (10, 12), (14, 14)]
```

Ce résultat surprend beaucoup celui qui a conçu le programme, ce n'est pas celui attendu. Le dernier numéro pair et le dernier numéro impair devraient être beaucoup plus proches. Où est cette erreur qu'il faut chercher dans la fonction `numero_deux` et qu'est-ce que le programme affichera une fois cette erreur corrigée ?

4) Ce programme est encore imparfait. Pourriez-vous trouver une situation dans laquelle le dernier numéro pair est éloigné du dernier numéro impair ? On pourra se pencher sur une rue dont les longueurs paires sont toutes égales et différentes des longueurs impaires elles-mêmes toutes égales. (1 point)

5) Question facultative **(ne rapporte pas de point)** : la remarque à la question précédente vous suggère-t-elle une solution ? Connaissez-vous le nom d'un algorithme utilisé en graphisme qui pourrait être adapté à ce problème ce problème ?

2 Listes récursives

Cet exercice concerne les fonctions `copy` et `deepcopy` du module `copy`. Chaque réponse attendue est une suite de trois chiffres compris entre 0 et 9. Parmi les trois programmes, les deux derniers sont presque identiques au premier à quelques lignes près : celle-ci sont indiquées en commentaires.

1) Qu'affiche le programme suivant ? (1 point)

```
a = [ None, [0] ]
b = [ None, [0] ]
c = [ None, [0] ]

a [0] = b
b [0] = c
c [0] = a

x = a
for i in range (0, 9) :
    x [1][0] += 1
    x = x [0]

print "a [1][0] =", a [1][0]
print "b [1][0] =", b [1][0]
print "c [1][0] =", c [1][0]
```

2) Et celui-ci ? (1 point)

```
a = [ None, [0] ]
b = [ None, [0] ]
c = [ None, [0] ]

a [0] = b
b [0] = c
```

```
c [0] = a

import copy             # ligne ajoutée
a = copy.copy (a)       # ligne ajoutée

x = a
for i in range (0, 9) :
    x [1][0] += 1
    x = x [0]

print "a [1][0] =", a [1][0]
print "b [1][0] =", b [1][0]
print "c [1][0] =", c [1][0]
```

3) Et celui-là ? (1 point)

```
a = [ None, [0] ]
b = [ None, [0] ]
c = [ None, [0] ]

a [0] = b
b [0] = c
c [0] = a

import copy
a = copy.deepcopy (a)     # ligne modifiée

x = a
for i in range (0, 9) :
    x [1][0] += 1
    x = x [0]

print "a [1][0] =", a [1][0]
print "b [1][0] =", b [1][0]
print "c [1][0] =", c [1][0]
```