IE Informatique et Société Numérique 2

FC, AS, EUR, AMER

Avril 2024

Durée: 1h30
Documents et calculatrice interdits

Un programme **mal indenté**, ou avec de **mauvais choix de noms de variables** sera **sanctionné**.

Exercice 1 Lecture de code (3.5pts)

(Q1.1) Qu'affiche le code suivant ?

```
t1 = []
n = 2
l = []
for i in range(n):
    for j in range(n):
        l.append(i*j)
    t1.append(l)
t1[0][1]=5
print(t1)
```

(Q1.2) Qu'affiche le code suivant ?

```
t2 = [[1,2,3],[8,9,3]]
n = 3
for i in range(n):
    for j in range(n):
        print(t2[i][j])
print(t2)
```

(Q1.3) Qu'affiche le code suivant ?

```
def decrement(a):
    a = a -1

def sub(l):
    for i in range(len(l)):
       decrement(l[i])

def my_function(l,i):
    l[i] = decrement(l[i])

x = 2
print(x)
l = [6, 7, 8]
sub(l)
print(l)
my_function(l,1)
print(l)
```

Exercice 2 Correction de code (2,5pts)

(Q2.1) Identifiez les quelques bugs, ainsi que les 2 erreurs de documentation du programme suivant.

```
1  def recherche_places(grille, valeur) :
2  """Recherche la position de chacune des occurences de 'valeur' dans grille.
3     Paramètres :
4         grille : une liste d'entiers
5         valeur : l'entier cherché
6     Renvoie :
7         une liste 2D d'entiers, chaque sous-liste est de taille 2 et correspond
8         aux coordonnées d'une valeur sous la forme [ligne, colonne]"""
9      res = []
10     for num_ligne in range(len(grille[0])) :
11         for num_col in range(len(grille)) :
12             if grille[num_ligne][num_col] == valeur:
13                 res.append(num_ligne)
14                 res.append(num_col)
15     return res
16
17 def remplace(grille, les_pos, val) :
18     """Stocke val dans grille à chacune des positions indiquées dans les_pos.
19        Paramètres :
20            grille : liste 2D d'entiers
21            les_pos : liste 2D d'entiers (une liste de positions [ligne, colonne]
22                      valides dans grille)
23            val : entier à stocker
24        Renvoie : la grille modifiée """
25     for pos in les_pos :
26         grille[pos[1]][pos[0]] = val
27
28 liste = [[1,2,5,9],[9,5,1,3],[8,7,4,1],[1,4,9,1]]
29 pos = recherche_places(liste, 1)
30 remplace(liste, pos, 42)
31 print(liste)
```

Exercice 3 Problème - tri de nombres complexes (14 pts)

Important : les questions de la fin de l'exercice (notamment 3.4 à 3.6) peuvent être faites même si les premières questions ont été ignorées. Vous pouvez utiliser une fonction même si vous n'avez fourni ni son algorithme ni son code (mais son en-tête doit avoir été définie).

On se propose d'étudier un algorithme de tri qui travaille sur des nombres complexes plutôt que sur des nombres entiers.

Comparaison et ordre Pour comparer et ordonner les nombres complexes, on utilisera leur norme (ou module). Par exemple, si on a les nombres $c_1 = -2 + 2i$ et $c_2 = 1 + i$, alors leur normes sont

$$|c_1| = \sqrt{(-2)^2 + (2)^2} = 2\sqrt{2}$$

et

$$|c_2| = \sqrt{1^2 + 1^2} = \sqrt{2}$$

Ainsi, comme :

$$|c_2| < |c_1|$$

on considérera que :

$$c_2 < c_1$$

Notez que votre tri considérera donc que 2 complexes sont égaux s'ils ont la même norme.

Choix de représentation des complexes Pour représenter les nombres complexes on utilisera des listes avec deux éléments, le premier élément représentant la partie réelle, le deuxième élément la partie imaginaire. Par exemple les nombres c_1 et c_2 seront représentés par :

```
c1 = [-2,2]
c2 = [1,1]
```

Tri par sélection Pour l'algorithme de tri, vous utiliserez le tri par sélection sur les entiers dont l'algorithme est rappelé ci-dessous :

```
# Fonction tri_selection(liste de valeurs) :
#     Pour chaque indice i_pivot de la liste
#         Chercher l'indice i_min du plus petit élément a partir de l'indice i_pivot
#         Si nécessaire, échanger les éléments situés aux indices i_pivot et i_min
```

(Q3.1) Proposez une fonction `affiche_liste_complexes` prenant en paramètre une liste de nombres complexes et qui l'affiche exactement au format précisé ci-dessous (1pt)

L'appel `affiche_liste_complexes([[-2,2],[1,-1]])` devra afficher exactement :

```
[ (-2 + 2i) (1 + -1i) ]
```

(Q3.2) En vous aidant du rappel ci-dessous, proposez un découpage fonctionnel de l'algorithme de tri qui prend en entrée une liste de nombres complexes et la trie. Cette décomposition devra inclure au minimum 3 fonctions (sans compter `tri`). (3pts)

Rappel : Pour effectuer un découpage fonctionnel, il faut :
- identifier les fonctions présentes dans l'algorithme fourni ;
- pour chacune fournir une **définition**, une **description** (docstring) et un **algorithme** ;
- appliquer le découpage fonctionnel à ces nouveaux algorithmes jusqu'à ce que l'algo de chaque fonction soit trivial.

(Q3.3) Implémentez la fonction `tri` de l'algorithme fourni conformément à votre découpage fonctionnel. Implémentez aussi toutes ses sous-fonctions. Vous n'avez pas besoin de gérer les imports, ni d'écrire les docstring qui auraient déjà été écrites précédemment. (6pts)

(Q3.4) Écrivez une fonction qui permet de tester si une liste de complexes est triée. (1 pt)

(Q3.5) Proposez un code python qui effectue les opérations suivantes à l'aide des fonctions précédentes. (1 pt)

- Définir la liste `l` contenant les complexes suivants : $1 + 0i$, $-2 + 1i$, $6 - 2i$ et $2 - 1i$;
- Afficher la liste `l`
- Tester si `l` est triée
- Si `l` n'est pas triée, trier la liste `l`
- Afficher la liste `l`

(Q3.6) Pour l'une de vos fonctions, proposez un jeu de tests couvrant 4 cas. Expliquez pourquoi ces cas sont pertinents. (2pts)