TD4: Récursivité
QQ éléments de réponse

PC0, L2 Informatique UBO

26 février 2022

But : Récursivité et classe

1 Classe ListeSuite

Définir la classe **ListeSuite** avec deux variables d'instance : la taille et la liste des valeurs. Pour cette classe, définir les méthodes suivantes :

- méthode de construction/initialisation qui permet de créer une liste vide de taille 0.
- les accesseurs aux variables d'instance
- une méthode d'affichage pour afficher la liste des valeurs

Pour une taille donnée, la liste de valeurs suit une loi qui s'exprime sous forme d'une suite. La n^{eme} valeur prend la valeur u_n définie de la manière suite :

- la valeur initiale u_0 est une valeur sélectionnée de manière aléatoire entre 0 et 10,
- si $u_n < 25$: $u_{n+1} = 2 * u_n - 3$,
- si $u_n >= 25$: $u_{n+1} = 20 - u_n$,

1. Définir une méthode dans la classe **ListeSuite** permettant de définir les valeurs de la liste à partir de la définition de la suite u_n. Cette méthode peut être appelée lors de l'intialisation de la liste de l'objet quand la taille est fixée.
2. Donner une instance de ListeSuite pour une taille égale à 20

Pour la sélection d'une valeur de manière aléatoire, on pourra utiliser le code suivant :

```
import random as rd
rd.randint(0,20)
```

Indications :

— Définir la suite forme de fonction récursive Un(n)
— Tester la fonction
— Definir la classe ListeSuite avec 2 attributs : taille et liste
— Tester les methodes de la classe

```
def Un(n):
  ....

for i in range(0,20):
     print(i, Un(i))

class ListeSuite(object):
    def __init__(self):
   .....
    def get_taille(self):
   ....
    def get_liste(self):
   .....
    def set_taille(self, taille):
   .....
    def set_liste(self, liste):
   .....
    def __str__(self):
   .....
    def suite(self):
     .....

maListe=ListeSuite()
print(maListe)
maListe.set_taille(20)
print(maListe)
```

L'utilisation de *randint* dans la fonction amène à révaluer la valeur de U0 plusieurs fois et donc peut donner une suite de valeurs erronées

```
def Un(n):

     if n==0:
          return U0
     elif (Un(n-1))<25:
          return Un(n-1)*2 -3
     else:
          return 20 -Un(n-1)

import random as rd
U0= rd.randint(0,10)   #on évalue U0 qu'une seule fois à l'extérieur de la fonction
for i in range(0,20):
      print(i, Un(i))

class ListeSuite(object):
    def __init__(self):
        self.taille=0
        self.liste=[]
    def get_taille(self):
        return self.taille
    def get_liste(self):
        return self.liste
    def set_taille(self, taille):
        self.taille=taille
        self.suite()
    def set_liste(self, liste):
        self.liste=liste
    def __str__(self):
        return "taille=" + str(self.taille) + "," + str(self.liste)

    def suite(self):
        for i in range(self.taille):
             self.liste.append(Un(i))

maListe=ListeSuite()
print(maListe)
maListe.set_taille(20)
print(maListe)
```

2 Tri fusion

2.1 Version récursive

Définir une fonction qui effectue un tri fusion sur des listes d'éléments de type simple (ex : liste d'entiers) Rappel du fonctionnement du tri fusion :

- diviser la liste de n éléments à trier en deux sous-listes de $n/2$ éléments
- trier les deux sous-listes récursivement à l'aide du tri par fusion
- fusionner les deux sous-listes triées pour produire la réponse triée

Ecrire le code correspondant au tri fusion et tester ce tri sur un tirage aléatoire.

2.2 Tri fusion appliqué à un objet de la classe ListeSuite

Pour la classe **ListeSuite** :

- Rédéfinir la méthode $fusion$ dans cette classe pour fusionner deux objets de classe *ListeSuite*. Cette méthode renvoie un nouvel objet *ListeSuite*.
- Redéfinir la méthode de tri_fusion pour ce type d'objet. Cette méthode trie les valeurs de la liste appartenant à l'objet **ListeSuite** et renvoie un nouvel objet *ListeSuite*.

```
def fusion(L1,L2):
        .......
def tri_fusion(L):
        ..........#ecrire une fonction récursive pour le tri fusion

class ListeSuite(object):
    def __init__(self):
         self.taille=0
         self.liste=[]
    def get_taille(self):
         return self.taille
    def get_liste(self):
        return self.liste
    def set_taille(self, taille):
        self.taille=taille
        self.suite()
    def set_liste(self, liste):
        self.liste=liste
    def __str__(self):
        return "taille=" + str(self.taille) + "," + str(self.liste)

    def suite(self):
         for i in range(self.taille):
               self.liste.append(Un(i))

    def fusion(self, other):
        ..........
    def tri_fusion(self):
        ..........

maListe=ListeSuite()
print(maListe)
maListe.set_taille(20)
print(maListe)
```

```
class ListeSuite(object):
     ............
    def fusion(self, other):
         lnew= ListeSuite()
         lnew.taille=self.taille + other.taille
         lnew.liste=fusion(self.liste, other.liste)
         return lnew

     def tri_fusion(self):
         lnew= ListeSuite()
         lnew.taille=self.taille
         lnew.liste= tri_fusion(self.liste)
         return lnew
```

Remarque : les méthodes $fusion$ et tri_fusion ne sont pas récursives, elles font juste appel aux fonctions $fusion$ et tri_fusion qui portent les mêmes noms

3 Gestion des segments de droites

Vous pouvez reprendre pour cet exercice la classe **Segment** définit au TD2. La classe **Segment** définit un segment de droite par ses deux points extrémités.

On rappelle que pour afficher des segments de droite, on peut utiliser le code suivant :

```
import pylab
pylab.figure(1) #crée la figure 1
pylab.plot([1,2,3,4], [1,7,3,5]) #dessine sur la figure 1
pylab.show() #affiche
```

1. Définir une méthode permettant de translater un segment de droite.
2. Définir la liste des segments de droite translatés après avoir appliqué 10 fois la translation. Les afficher.
3. A partir d'un segment dont une extrémité du segment est le point de coordonnées $(0, 0)$, on définit tous les segments de droite en divisant par 2 la longueur. On effectue cette opération tant que la longueur est supérieure à un certain seuil. Définir une méthode récursive dans la classe **Segment** qui gére la nouvelle liste de segments.
4. (optionnel) Visualiser les segments de droite de la question précédente après avoir effectué une translation sur les segments de droite afin de les rendre visibles.

```
class Segment(object):
 .....

     def divide(self):
           if self.longueur() < 10:
               return None
          else:
              l=Segment(self.origin, self.target.div(2))
              print(l,l.longueur())
              return l.divide()
```

Dans ce code, vous avez un exemple de méthode écrite de manière récursive avec la méthode *divide*.