Devoir Surveillé 3 : Correction

1 Tracé de courbes représentatives de fonctions et calcul d'intégrales

1.1 Tracé de la courbe représentative d'une fonction

Question 1. Créer une fonction `f(x)` qui prend en argument un flottant `x` et renvoie le résultat de :

$$f(x) = 2 - \frac{1-x}{1+x^6}$$

```
def f(x):
      return 2-(1-x)/(1+x*6)
```

Question 2. Construire la liste `X=[0, 0.02, 0.04, 0.06 , ... , 2.98, 3]`. On pourra utiliser la méthode de son choix.

Une construction par compréhension est la plus simple et la plus claire ici :

```
X=[i*0.02 for i in range(151)]
```

On peut également partir d'une liste vide à laquelle on ajoute des éléments dans une boucle :

```
X=[]
for i in range(151):
      X.append(i*0.02)
```

Enfin, il est possible d'utiliser la commande `linspace` du module `numpy`. Attention toutefois à ne surtout pas utiliser des arguments non entiers dans un `range` usuel !

```
import numpy as np
X=np.linspace(0,3,151)
```

Question 3. Construire la liste `Y` des images par f des éléments de la liste `X`

Ici une construction par compréhension avec parcours de `X` par éléments est de loin la solution la plus simple :

```
Y=[f(x) for x in X]
```

Question 4. Quelle(s) commande(s) permettent d'afficher la courbe représentative de f sur $[0\,;3]$?

Question 5. Quelle(s) commande(s) permettent d'afficher la courbe représentative de f sur $[0\,;2]$?

```
import matplotlib.pyplot as plt
plt.plot(X,Y)
plt.show()
```

1.2 Méthodes des trapèzes

Question 6. Écrire une fonction `Trapeze(a,b,f,n)` qui prend en argument deux flottants `a` et `b` représentant les bornes d'intégration ; une fonction `f` et un entier strictement positif `n` représentant le nombre de pas d'intégration, et qui renvoie la valeur approchée de l'intégrale de f sur $[a;b]$ avec n pas, calculée par la méthode des trapèzes.

La question a été traitée en TP. On prendra garde à ne par compter les bornes plusieurs fois, ou les oublier. Le plus naturel :

```
def Trapeze(a,b,f,n):
      s=0
      pas=(b-a)/n
      for i in range(n-1):
            s=s+pas*(f(a+i*pas)+f(a+(i+1)*pas))/2
      return s
```

Question 7. Calculer avec la méthode des trapèzes, en utilisant $n = 10^5$ points l'intégrale suivante :

$$\int_0^3 2 - \frac{1-x}{1+x^6}\mathrm{d}x$$

On pourra utiliser toutes les fonctions de l'exercice ; y compris celles de la partie 1.1.

Une syntaxe du type `lambda x:` est maladroite ici, f étant déjà définie. Très simplement :

```
print(Trapeze(0,3,f,10**5))
```

2 Manipulation de listes

Question 8. Écrire une fonction `testpair(L)` qui prend en argument une liste `L` et qui renvoie un booléen indiquant si `L` est de longueur paire ou non.

On va se servir de lu reste de la division euclidienne de `len(L)` par 2 :

```
def testpair(L):
    return len(L)%2==0
```

Dans la suite de cette exercice, les listes manipulées seront supposées de longueur paire.

Question 9. Écrire une fonction `decoupe(L)` qui prend en argument une liste `L` de longueur paire et qui renvoie deux listes `Lg` et `Ld` de même longueur où les éléments de `Lg` sont les éléments situés au début de la liste `L` et les éléments de `Ld` ceux situés à la fin de la liste `L`. On utilisera les techniques de slicing.

L'exercice a été traité en cours, dans un cadre plus compliqué ... en utilisant les techniques de slicing, cela se fait en une ligne. On prendra garde au fait que `len(L)//2` est de type entier mais que `len(L)/2` est de type flottant.

```
def decoupe(L):
    return L[:len(L)//2], L[len(L)//2:]
```

Question 10. Écrire une fonction `compare(L)` qui prend en argument une liste `L` de longueur paire et qui vérifie si la somme des éléments de la première moitié des éléments de L est plus grande que la somme des éléments de la seconde moitié des éléments de L (comme définie dans la question précédente). Cette fonction renverra 0 si ce sont les éléments de la première moitié de `L` qui ont la somme la plus grande et 1 sinon.

La stratégie est ici simple : on rappelle la fonction précédente, on somme les termes des deux listes puis on compare les sommes calculées.

```
def compare(L):
    Lg, Ld = decoupe(L)
    sg=0
    sd=0
    for i in range(len(Lg)):
        sg=sg+Lg[i]
        sd=sd+Ld[i]
    if sg>sd:
        return 0
    else:
        return 1
```

Question 11. Quelle est la complexité de l'algorithme ainsi écrit ?

On pose n=`len(L)`.
La fonction `decoupe(L)` nécessite n affectations, les lignes 3 et 4 de `compare` nécessitent une affectation. Les lignes 6 et 7 coûtent une affectation, une somme et un accès, répétés n/2 fois pour un coût total de 3n. La ligne 8 coûte une comparaison.
On a ainsi un coût total de 4n+3 opérations élémentaires, donc $\mathcal{O}(n)$ opérations.

3 Chaînes de caractères

Question 12. (Question de cours)
Écrire une fonction `motdschaine(s,mot)` qui prend en argument deux chaînes de caractères `s` et `mot` et qui renvoie un booléen indiquant si la chaîne de caractères `mot` est présente dans la chaîne de caractères `s`. Il est interdit d'utiliser directement la syntaxe `mot in s`. Il est par contre possible d'utiliser les techniques de slicing.

La version la plus performante de l'algorithme "naïf" a été vue en cours. Ici l'énoncé admettait l'usage du slicing, qui s'écrit :

```
def motdschaine(s,mot):
      for i in range(len(s)-len(mot)):
            if s[i:i+len(mot)]==mot:
            ind return True
      return False
```

Question 13. Écrire une fonction `inverse(s)` qui prend en argument une chaîne de caractère `s` et qui renvoie la chaîne de caractères inversée. Ainsi `inverse("tom")` renverra `"mot"`

Pas moins de 6 corrections différentes ont été vues en TP. La plus élégante conceptuellement :

```
def inverse(s):
      ch=""
      for x in s:
            ch=x+ch
      return ch
```

Question 14. Écrire une fonction `verifACGT(s)` qui prend en argument une chaîne de caractères `s` et qui renvoie un booléen qui indique si cette chaîne de caractères n'est constituée que des caractères A, C, G et T.

L'usage d'une expression booléenne est nettement meilleur qu'une série d'instructions conditionnelles imbriquées :

```
def verifACGT(s):
      for x in s:
            if x!='A' and x!='C' and x!='G' and x!='T':
                  return False
      return True
```

Les brins d'ADN sont codées avec des caractères 'A', 'C', 'G' et 'T'. À un brin d'ADN correspond un brin d'ADN complémentaire où le 'A' est substitué par un 'T'; le 'T' par un 'A', le 'C' par un 'G' et le 'G' par un 'T'. Ce brin complémentaire se lit par ailleurs en sens inverse.

Ainsi le brin d'ADN "AGTTGCAG" est substitué par le brin "TCAACGTC"; brin qui est lu à l'envers comme "CTGCAACT"

Question 15. Écrire une fonction `substitue(ADN)` qui prend en argument une chaîne de caractères `ADN` ne contenant que des caractères 'A', 'C', 'G' et 'T' et qui renvoie la chaîne de caractères substituée et inversée comme décrit ci-haut. On se servira de la fonction précédente.

```
def substitue(ADN):
      ch=""
      for x in ADN:
      if x=='A':
            ch=ch+'T'
      if x=='T':
            ch=ch+'A'
      if x=='C':
            ch=ch+'G'
      if x=='G':
            ch=ch+'C'
      return inverse(ch)
```

Une solution plus avancée, pour les plus ambitieux d'entre vous :

```
def substitue(ADN):
      ch=""
      L=['A','G','C','T']
      for x in ADN:
            ch=ch+L[3-L.index(x)]
      return inverse(ch)
```