TP noté : corrigé

PTSI Lycée Eiffel

17 décembre 2015

I. Quelques algorithmes élémentaires sur les listes.

Question 1 : Écrire une fonction Python **egalite(l,m)** déterminant si deux listes l et m contiennent les mêmes éléments (dans le même ordre). On supposera que les listes sont de même longueur (mais vous pouvez mettre un test à l'intérieur de la fonction pour vérifier que c'est bien le cas).

```
def egalite(l,m) :
    if len(l) !=len(m) :
        return False
    for i in range(len(l)) :
        if l[i] !=m[i] :
            return False
    return True
```

Dans la mesure où un return arrête immédiatement l'exécution de la fonction, le programme va commencer par tester l'égalité des longueurs (et s'arrête si elle ne sont pas égales), puis celle des éléments si besoin, et ne renverra True que s'il est arrivé jusqu'au bout de la liste sans trouver d'éléments différents.

Question 2 : Écrire une fonction Python **supprimenegatifs(l)** qui retourne une liste où tous les éléments négatifs de l ont été remplacés par des zéros.

```
def supprimenegatifs(l) :
    m=[]
    for i in l :
        if i>=0 :
            m.append(i)
        else :
            m.append(0)
    return m
```

Il est plus simple ici de faire directement parcourir à la variable i les éléments de la liste.

Question 3 : Écrire une fonction Python **positifs(l)** qui retourne une liste ne contenant que les éléments positifs (ou nuls) de la liste l (et supprimant les négatifs).

```
def positifs(l) :
    m=[]
    for i in l :
        if i>=0 :
```

```
            m.append(i)
    return m
```

Question 4 : Écrire une fonction Python **indices(l,a)** déterminant les indices de la liste où apparait la valeur a. La fonction renverra une liste, vide dans le cas où a n'apparait jamais dans la liste.

```
def indices(l,a) :
    m=[]
    for i in range(len(l)) :
        if l[i]==a :
            m.append(i)
    return m
```

II. Quelques fonctions sur les couples d'entiers.

Question 5 : Écrire une fonction Python **evalue(l)** qui retourne la valeur donnée par la fonction evalue définie ci-dessus. On supposera que l est une liste de deux nombres entiers, et on gèrera les cas particuliers pour $q = 0$.

```
def evalue(l) :
    p,q=l[0],l[1]
    if q==0 :
        if p==0 :
            return 1
        else :
            return 'infini'
    return float(p)/q
```

Le float indiqué à la dernière ligne est simplement là pour éviter que Python ne fasse une division entière (ce qu'il fait pas défaut dans certaines versions).

Question 6 : Écrire une fonction Python **median(l,m)** qui retourne la liste de deux entiers correspondant à la définition ci-dessus.

```
def median(l,m) :
    return [l[0]+m[0],l[1]+m[1]]
```

Question 7 : Écrire une fonction Python **egalitex(x,l)** qui s'applique à un nombre flottant x et à une liste de deux entiers [p,q], et qui renvoie True si $evalue(p, q) = x$ (et False sinon, y compris dans les cas où evalue donne la valeur infini).

```
def egalitex(x,l) :
    if x==evalue(l) :
        return True
    return False
```

Question 8 : Écrire une fonction Python **compare(x,l)** sur le même modèle que la précédente et qui renvoie True si $x \leqslant evalue(l)$, en considérant que tout flottant est plus petit que infini.

```
def compare(x,l) :
    if evalue(l)=='infini' :
        return True
    if x<= evalue(l) :
        return True
    return False
```

III. Suites de Farey.

Question 9 : Écrire une fonction Python **inseremedians(l)** qui prend comme argument une liste l de couples d'entiers (soit $l = [c_1, \ldots, c_p]$), et renvoit la liste de couples $[c_1, m_1, \ldots, m_{p-1}, c_p]$, où, comme décrit ci-dessus, m_i est le médian de c_i et c_{i+1}. En déduire une fonction **Farey(n)** qui prend comme argument un entier n et renvoit la suite de Farey F_n.

```
def inseremedians(l) :
    m=[l[0]]
    for i in range(len(l)-1) :
        m.append(median(l[i],l[i+1]))
        m.append(l[i+1])
    return m

def Farey(n) :
    l=[[0,1],[1,0]]
    for i in range(n) :
        l=inseremedians(l)
    return l
```

Question 10 : Écrire une fonction Python **evalfarey(n)** qui retourne la liste des valeurs $eval(c_i)$ pour tous les couples c_i constituant la suite de Farey F_n. En déduire une fonction **conjecture(n)** qui renvoie la valeur True si notre conjecture est vérifiée sur la suite de Farey F_n.

```
def evalfarey(n) :
    return [evalue(i) for i in Farey(n)]

def conjecture(n) :
    l=evalfarey(n)
    for i in range(len(l)-1) :
        if l[i]>l[i+1] :
            return False
    return True
```

Le comportement du programme ne sera pas perturbé par le fait que la dernière évaluation donnera systématiquement la valeur infini (qui n'est pas numérique), elle ne sera de toute pas considérée comme inférieure à la précédente.

Question 11 : Sur le modèle de la question précédente, écrire un programme permettant de tester cette conjecture sur la suite de Farey F_n.

```
def conjecturebis(n) :
```

```
    l=Farey(n)
    for i in range(len(l)-1) :
        if l[i][0]*l[i+1][1]-l[i][1]*l[i+1][0] !=-1 :
            return False
    return True
```

IV. Rercherche dichotomique.

Question 12 : Écrire une fonction Python **appartientfarey(x,n)** qui détermine si le nombre flottant x correspond à la valeur $evalue(c_i)$ pour un des couples d'entiers c_i de la suite de Farey F_n. La fonction renverra bien sûr True si c'est le cas, False sinon.

```
def appartientfarey(x,n) :
    l=evalfarey(n)
    for i in l :
        if x==i :
            return True
    return False
```

Question 13 : Écrire une fonction Python **dichofarey(x,n)** qui effectue le calcul de g et de d après n étapes de cet algorithme. En déduire une fonction **appartientfareybis(x,n)** qui affiche tous les médians calculés lors des étapes de cet algorithme, et qui renvoie à la fin True seulement si $evalue(g) = x$ ou $evalue(d) = x$ (ce qui revient à dire que x appartient à F_n).

```
def dichofarey(x,n) :
    g,d=[0,1],[1,0]
    for i in range(n) :
        m=median(g,d)
        if evalue(m)<x :
            g=m
        else :
            d=m
    return g,d

def appartientfareybis(x,n) :
    g,d=[0,1],[1,0]
    for i in range(n) :
        m=median(g,d)
        print(m)
        if evalue(m)<x :
            g=m
        elif evalue(m)>x :
            d=m
        else :
            return True
    return False
```

Le programme est très légèrement différent de ce qui était demandé : à chaque étape de la dichotomie, il affiche le médian (comme demandé), puis remplace g ou d par le médian seulement si

ce médian n'est pas égal à x. Dans le cas contraire, on vient de trouver la valeur de x à l'intérieur de la suite de Farey à laquelle appartient le médian, on arrête tout.

Question 14 : Écrire une fonction Python **approxfarey(x,e)** qui retourne le plus petit entier n pour lequel la distance entre $evalue(g)$ et $evalue(d)$ est inférieure à e (précision de l'approximation souhaitée), ainsi que les deux couples g et d correspondants.

```
def approxfarey(x,e) :
    g,d=[0,1],[1,0]
    n=0
    ecart=e+1
    while ecart >e :
        m=median(g,d)
        print(m)
        if evalue(m)<x :
            g=m
        else :
            d=m
        n=n+1
        if evalue(d) !='infini' :
        ecart=evalue(d)-evalue(g)
    return g,d,n
```

On obtient ainsi, en calculant $approxfarey(sqrt(2), 0.00001)$, après 16 étapes de l'algorithme, un encadrement de $\sqrt{2}$ par les deux fractions $\frac{816}{577}$ et $\frac{577}{408}$. Pour les plus curieux, on peut pousser jusqu'à une précision de 10^{-10} et obtenir au bout de 29 étapes un encadrement par les fractions $\frac{275\ 807}{195\ 025}$ et $\frac{114\ 243}{80\ 782}$. Si on essaye avec le nombre π, il faut 322 étapes pour obtenir la même précision (c'est normal, plus le nombre est éloigné de 1, plus ça prend du temps).