La récursivité

I Introduction à la récursivité

Définition et exemples

Une fonction est dit **récursive** quand elle s'appelle elle-même. Sinon la fonction est dite **itérative**.

Exemples classiques :

- fonction qui calcule $n!$:

```
def factorielle(n):
    '''on suppose n>=0'''
    if n == 0:
        return 1                        #0!=1  [terminaison]
    else:
        return n * factorielle(n-1)    #n!=n*(n-1)!
```

- fonction qui calcule le n-ième terme de la suite de Fibonacci ($f_0 = f_1 = 1$ et $f_n = f_{n-1} + f_{n-2}$) :

```
def f(n):
    '''on traite le cas négatif en renvoyant 1'''
    if n <= 1:
        return 1 #[terminaison]
    else:
        return f(n-1) + f(n-2)
```

Programmation itérative ou programmation récursive ?

Les fonctions peuvent être souvent être écrites au choix sous forme itérative ou récursive.

Par exemple :

```
def suite(n,a):
    if n == 0:
        return a
    else:
        return 3 * suite(n-1,a) + 1

```

renvoie le n^{e} terme de la suite définie par $u_0 = a$ et $\forall n \in \mathbb{N}^*$, $u_n = 3u_{n-1} + 1$.

Son équivalent itératif est :

```
def suite(n,a):
    u = a
    for i in range(n):
        u = 3 * u + 1
    return u
```

On alors dit qu'on a « dérécursivé » la fonction.

II Exercices sur l'écriture récursive

Dans les exemples qui suivent, la récursivité n'apporte rien en terme d'efficacité d'exécution par rapport à l'écriture itérative. Le but est ici de s'entraîner à passer de l'une à l'autre de ces écritures pour constater que l'écriture du code est parfois plus simple dans un cas que dans l'autre.

Q1 Écrire une fonction récursive `reste(n,b)` (resp. `quotient(n,b)`, resp. `division(n,b)`) qui renvoie le reste (resp. le quotient, resp. la liste `[q,r]` formée du quotient et du reste) de la division euclidienne de l'entier naturel `n` par l'entier naturel non nul `b`.
On n'utilisera pas les fonctions `%, //` ni `/`. On utilisera le fait que si $0 \leq$`n`$<$`b`, le reste vaut `n` et le quotient vaut `0`, et que sinon ils s'obtiennent à partir de la division euclidienne de `n-b` par `b`.
Écrire une version itérative de ces fonctions.

Q2 Écrire une fonction récursive `nbDeChiffres(n)` qui renvoie le nombre de chiffres de l'écriture décimale de l'entier non nul `n`. On remarquera que tout entier `n`≥ 10 a un chiffre de plus que `n//10` ; on n'utilisera pas de chaîne de caractères.
Par ex. : `nbDeChiffres(563)`$\to$`3` ; `nbDeChiffres(12385)`$\to$`5`
Écrire une version itérative de la même fonction.

Q3 Si n est un nombre dont l'écriture binaire est $n = \overline{b_{p-1}b_{p-2}\dots b_1 b_0}$ alors $n = \overline{b_{p-1}b_{p-2}\dots b_1 0} + \overline{b_0} = 2 \times \overline{b_{p-1}b_{p-2}\dots b_1} + b_0$ donc b_0 est le reste dans la division euclidienne de n par 2 et b_1 est le chiffre des unités du quotient de n par 2 (et il suffit donc de diviser ce quotient par 2 pour déterminer b_1). Écrire une fonction récursive `binaire(n)` qui renvoie une liste correspondant à l'écriture binaire d'un entier n (elle renverra la liste vide pour $n = 0$).
Écrire une version itérative de la même fonction.

Q4 L'algorithme d'Euclide de calcul du PGCD (plus grand diviseur commun) de deux entiers naturels non nuls a et b est fondé sur le fait que si a n'est pas divisible par b, alors le PGCD de a et b est égal au PGCD de b et r, où r est le reste de la division euclidienne de a par b.
Écrire une fonction récursive `pgcd(a,b)` qui calcule le PGCD de a et b.
Écrire une version itérative de la même fonction.

Q5 Préciser ce que renvoie la fonction suivante :

```
    def S(n):
        S = 0
        for i in range(n):
            S = S + 1/(i+1)
        return S
```

Écrire version récursive de cette fonction.

Q6 Écrire deux fonctions récursives `somme(L)` qui renvoient la somme des éléments de la liste `L` (l'une ramenant la somme d'une liste de longueur n à la somme d'une liste de longueur $n-1$ et l'autre procédant par dichotomie).
Écrire une version itérative de la première de ces deux fonctions.

Q7 Écrire une fonction récursive `decomposition(n)` qui renvoie la décomposition de `n` en facteurs premiers, rangés dans l'ordre croissant. On rappelle qu'un nombre `n` est premier si et seulement si, il n'est divisible par aucun nombre `d` tel que `2`$\leq$ `d`$\leq \sqrt{}$`n`.
Par exemple : `decomposition(12)` $\to$ `[2,2,3]` ; `decomposition(13)` $\to$ `[13]`
Écrire une version itérative de la même fonction.

III Maniement d'une structure de donnée récursive : listes imbriquées

Les listes `[[1, 2], [9, 10]]`, `[[1], 2]`, `[[1], 2, [3, [[4]]]]` ou plus simplement `[1, 2, 3]` sont des listes imbriquées d'entiers. On voudrait écrire une fonction `S(L)` qui renvoie la somme des entiers contenus dans une liste imbriquée d'entiers `L`.
Par exemple : `S([[5], 2, [10, 8, [[7]]]])` → `32`

Les listes imbriquées ont une structure naturellement récursive : une liste imbriquée est soit nombre entier, soit une liste dont chacun des éléments est un nombre entier ou une liste imbriquée. La somme d'une liste imbriquée `L` peut aussi être définie récursivement de la manière suivante : si `L` est un entier, la somme de `L` est égale à l'entier `L` ; sinon `L` est une liste dont la somme est par définition (récursive) la somme de chaque élément (liste ou entier) de cette liste.
Par exemple pour calculer `S([[5], 2, [10, 8, [[7]]]])` :

- `S([[5], 2, [10, 8, [[7]]]])` est égal à `S([5]) + 2 + S( [10, 8, [[7]]])` ;
- `S([5])` est égal à `5` ;
- `S([10, 8, [[7]]])` est égal à `10 + 8 + S([[7]])` ;
- `S([[7]])` est égal à `S([7])` lui même égal à `7` ;
- on en déduit que `S([10, 8, [[7]]])` est égal à `10 + 8 + 7` et donc que `S([[5], 2, [10, 8, [[7]]]])` est égal à `5 + 2 + 10 + 8 + 7 = 32`.

Lorsqu'on applique récursivement cette définition de la somme, on aboutit en fin de compte à des sommes d'entiers.

Q8 Écrire une fonction `S(L)` qui renvoie la somme d'une liste imbriquée (on utilisera l'instruction `str(x).isdigit()` qui renvoie `True` si la variable `x` est de type entier et `False` sinon).

Q9 Écrire une fonction `contient(L,x)` d'argument une liste imbriquée `L` et un entier `x` qui renvoie `True` si `x` est élément d'une quelconque des listes imbriquées dans `L` et `False` sinon.

IV Décompte du nombre d'appels récursifs et dérécursivation

La récursivité est particulièrement intéressante lorsque la dérécursivation engendrerait l'écriture d'une fonction beaucoup plus compliquée. Cela se produit lorsque le nombre de variables stockées dans les appels récursifs est important. Dans ce cas, la fonction itérative correspondante devra définir des structures de données permettant de stocker les valeurs de toutes ces variables, ce qui est fait automatiquement pour une fonction récursive. Une version itérative est par contre préférable lorsque le nombre d'appels récursifs est très grand.

Q10 En utilisant uniquement la formule du triangle de PASCAL et le fait que $\binom{n}{0} = 1 = \binom{n}{n} = 1$, écrire une fonction récursive `binom(n,p)` qui renvoie le coefficient binomial $\binom{n}{p}$.

Combien faut-il d'appels récursifs pour calculer $\binom{5}{3}$? (Penser à les représenter sous la forme d'un arbre).

Q11 Écrire une fonction `somme(L1,L2)` qui prend en argument deux listes de nombres de même taille et renvoie leur somme au sens de l'addition des vecteurs (somme coordonnée par coordonnée), sous forme de liste.

Utiliser la fonction précédente pour écrire une fonction itérative `binom2(n,p)` qui renvoie le coefficient binomial $\binom{n}{p}$.

Combien de sommes ont été effectuées pour calculer $\binom{5}{3}$

Q12 Dans la fonction récursive `division` de la question 1, pourquoi l'écriture suivante :

```
    ...
    else:
        [q,r]=division(n-b,b)
        return [1+q,r]
```

est-elle de très loin préférable (indépendamment de la lisibilité du code) à :

```
    ...
    else:
        return [1+division(n-b,b)[0],division(n-b,b)[1]]
```

V Pour ceux qui ont déjà fini

On représente un damier dont les cases sont noires ou blanches par une liste `L` de listes dont les éléments valent 0 (pour noir) ou 1 (pour blanc).

Q13 Écrire une procédure `coloriage(L,i,j)` qui, si la case `L[i][j]` contient 1 (blanc), modifie cette case en la coloriant en gris (i.e. avec la valeur 2) ainsi que toutes les cases du damier qui lui sont connexes. Cette procédure sera récursive, et agira par « contamination », en examinant toutes les cases voisines (i.e. séparées par un côté) d'une case déjà grisée.
Par exemple :

```
L=[[1,0,1,1,1],
   [1,1,0,1,0],
   [1,0,0,0,1],
   [1,1,1,0,1]]
coloriage(L,1,0)
print(L)
>>> [[2,0,1,1,1],
     [2,2,0,1,0],
     [2,0,0,0,1],
     [2,2,2,0,1]]
```

Q14 Un commerçant doit rendre la somme de `n` € (`n` entier naturel non nul) à un client. Pour cela, il dispose de stocks illimités de pièces de 1 €, 2 €, 5 €. Par exemple, il y a 7 manières de rendre 8 € :

8×1 €	4×2 €	6×1 € $+ 1 \times 2$ €	3×1 € $+ 1 \times 5$ €
4×1 € $+ 2 \times 2$ €	1×1 € $+ 1 \times 2$ € $+ 1 \times 5$ €	2×1 € $+ 3 \times 2$ €	

À l'aide d'une fonction récursive auxiliaire, écrire une fonction `monnaie(n)` qui calcule le nombre de manières différentes de rendre `n` € de monnaie.

Q15 Écrire une fonction récursive `chemins(x,y)` qui renvoie le nombre de chemins allant de l'origine, de coordonnées `0,0`, au point de coordonnées entières positives `x,y`, où les chemins considérés sont les lignes brisées reliant des points de coordonnnées entières telles que l'on passe d'un point au suivant par une translation de vecteur $(1,0)$ (vers la droite), $(0,1)$ (vers le haut) ou $(1,1)$ (en diagonale vers le haut et à droite).
Par exemple : `chemin(1,1)`→ `3` ; `chemin(1,2)`→ `5`

Q16 Écrire une fonction récursive `nb(L,N)` où `L` est une liste de chiffres distincts (de 0 à 9) qui renvoie la liste de tout les nombres inférieurs ou égaux à `N` dont tous les chiffres appartiennent à `L`.
Par exemple : `nb([1,3],32)`→`[1,3,11,13,31]`

Q17 Écrire une fonction récursive `supprimeDoublon(L)` où `L` est une liste, et qui renvoie la liste `L` où l'on a supprimé les doublons.
Par exemple : `supprimeDoublon([1,2,5,6,6,2,4,1,5])`→`[1,2,5,6,4]`

Q18 Écrire une fonction `tours(n)` qui renvoie toutes les configurations possibles de placements de `n` tours, numérotées de `1` à `n`, sur un échiquier `n`×`n` de sorte qu'aucune ne soit en prise avec une autre (i.e. placée sur une même ligne ou une même colonne).
On représentera un échiquier sous la forme d'une liste de listes où les cases vides sont codées par 0 et chaque tour par son numéro.
Par exemple : `tour(2)`→ `[[[1,0],[0,2]],[[2,0],[0,1]],[[0,2],[1,0]],[[0,1],[2,0]]]`

Corrigé

Q1

```
#versions récursives
def reste(n,b):
    '''On suppose n >= 0'''
    if n < b:
        return n
    else:
        return reste(n-b,b)

def quotient(n,b):
    '''On suppose n >= 0'''
    if n < b:
        return 0
    else:
        return 1 + quotient(n-b,b)

def division(n,b):
    '''On suppose n >= 0'''
    if n < b:
        return [0,n]
    else:
        [q,r]= division (n-b,b)
        return [1+q,r]

#versions itératives
def resteIt(n,b):
    '''On suppose n >= 0'''
    r = n
    while r >= b:
        r = r - b
    return r

def quotientIt(n,b):
    '''On suppose n >= 0'''
    r = n
    q = 0
    while r >= b:
        r = r - b
        q = q + 1
    return q

def divisionIt(n,b):
    '''On suppose n >= 0'''
    r = n
    q = 0
    while r >= b:
        r = r - b
        q = q + 1
    return [q,r]
```

Q2

```
#version récursive
def nbDeChiffres(n):
    '''renvoie le nombre de chiffres de n ; on suppose n >= 0'''
    if n < 10 :
        return 1
    else:
        return 1 + nbDeChiffres(n//10)
```

```python

#versions itératives
def nbDeChiffresIt(n):
    '''renvoie le nombre de chiffres de n ; on suppose n >= 0; à la fin de la fonction, la valeur de n a changé'''
    nb_chiffres = 1
    while n >= 10:
        n = n//10
        nb_chiffres += 1
    return nb_chiffres

def nbDeChiffresIt2(n):
    '''renvoie le nombre de chiffres de n ; on suppose n >= 0'''
    nb_chiffres = 0 #ou nb_chiffres = 1 et on élimine la ligne *
    nombre = n
    while nombre >= 10:
        nombre = nombre//10
        nb_chiffres += 1
    nb_chiffres += 1  #ligne *
    return nb_chiffres
```

Q3

```python
#version récursive
def binaire(n):
    if n < 2:
        return [n]
    else:
        return binaire(n//2)+[n%2]

#version itérative
def binaireIt(n):
    binaire = []
    nombre = n
    while nombre >= 2:
        binaire = [nombre%2] + binaire
        nombre = nombre//2 #avec nombre = nombre/2 ça ne marche pas
    binaire = [nombre] + binaire
    return binaire
```

Q4

```python
#version récursive
def pgcd(a,b):
    if a%b == 0:
        return b
    else:
        return pgcd(b,a%b)

#version itérative
def pgcd(a,b):
    while a%b != 0:
        a, b = b, a%b
    return b

#ou une version itérative proche du cours
def pgcdIt(a,b):
    r0,r1 = a,b
    r2 = r0%r1
    while r2 > 0:
        r0,r1 = r1,r2 #on translate les valeurs
        r2 = r0%r1
    return r1
```

Q5 La fontion `S` renvoie la somme partielle des n premiers termes de la série harmonique, à savoir $S_0 = 0$ et pour tout $n \geq 1$, $S_n = \sum_{k=1}^{n} \frac{1}{k} = \sum_{k=0}^{n-1} \frac{1}{k+1}$.

Une version récursive de `S` *est :*

```
121 def S(n):
122     if n == 0:
123         return 0
124     else:
125         return S(n-1) + 1/n
```

Q6

```
128 #version récursive 1
129 def somme(L):
130     if L == []:
131         return 0
132     else:
133         return L[0] + somme(L[1:])
134 
135 #version récursive 2
136 def somme(L):
137     if L == []:
138         return 0
139     elif len(L) == 1:
140         return L[0]
141     else:
142         return somme(L[:len(L)//2]) + somme(L[len(L)//2:])
143 
144 #version itérative relative à la récursive 1
145 def sommeIt(L):
146     s = 0
147     while L != []:
148         s = L[0] + s
149         L = L[1:]
150     return s
151 
152 #ou de façon classique
153 def sommeIt2(L):
154     s = 0
155     for i in range(len(L)):
156         s += L[i]
157     return s
```

Q7

```
160 #version récursive
161 from math import sqrt
162 
163 def decomposition(n):
164     for i in range (2, int(sqrt(n))+1): #test des diviseurs de n
165         if n%i == 0:
166             return [i] + decomposition(n//i) #on continue la recherche
    avec n/i; quand la liste est produite, on s'arrête
167     return [n] #on ajoute n >= int(sqrt(n))+1
```

```python
#version complètement récursive
def decomposition(n, i):
    if i == int(sqrt(n))+1:
        return [n]
    else:
        if n%i == 0:
            return [i] + decomposition(n//i, i)
        else:
            return decomposition(n, i+1)

#version_itérative
from math import sqrt

def decompositionIt(n):
    M = []
    for i in range (2,int(sqrt(n))+1): #test des diviseurs de n
        while n%i == 0:
            M.append(i)
            n = n // i #on continue la recherche avec n/i
    M.append(n)
    return M
```

Q8

```python
#version récursive partielle
def SIt(L):
    somme = 0
    for i in range(len(L)):
        if not str(L[i]).isdigit(): #si L[0] est une liste
            somme += SIt(L[i]) #somme avec désimbrication
        else :
            somme += L[i] #somme sans désimbrication
    return somme

#version complètement récursive
def S(L):
    if L == []:
        return 0
    elif not str(L[0]).isdigit(): #si L[0] est une liste
        return S(L[0]) + S(L[1:]) #somme avec désimbrication
    else:
        return L[0] + S(L[1:]) #somme sans désimbrication
```

Q9

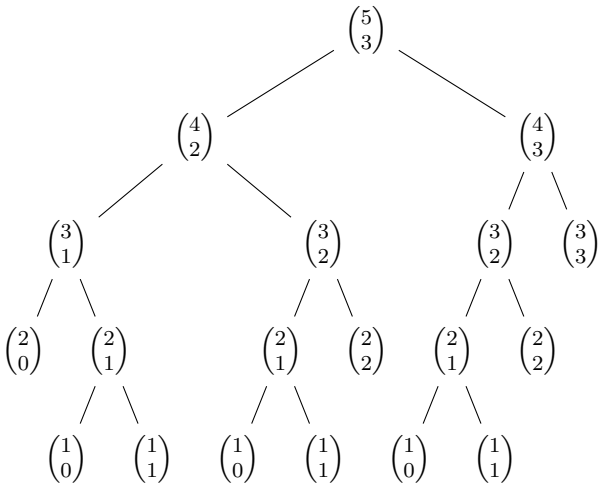
```python
#version complètement récursive
def contient(L,x):
    if L == []:
        return False
    else:
        if L[0] == x:
            return True
        elif not str(L[0]).isdigit(): #si L[0] est une liste
            return contient(L[0],x) or contient(L[1:],x)
        else:
            return contient(L[1:],x)
```

Q10

```
225 def binom(n,p):
226     if p < 0 or p > n : #traitement des cas de p hors [0,n]
227        return 0
228     elif p == 0 or p == n:
229         return 1
230     else: #n!=p and p!=0 and n!=0
231         return binom(n-1,p-1) + binom(n-1,p)
```

Le calcul de $\binom{5}{3}$ fait appel 19 fois à `binom(n,p)`.

L'arbre des appels est détaillé en page suivante.



Q11

```
234 def somme(L1,L2):
235     M = []
236     for i in range(len(L1)):
237         M.append(L1[i]+L2[i])
238     return M
239 
240 #ou
241 def somme2(L1,L2):
242     return [L1[i]+L2[i] for i in range(len(L1))]
243 
244 
245 def binom2a(n,p):
246     if p < 0 or p > n : #traitement des cas de p hors [0,n]
247        return 0
248     elif p == 0 or p == n:
249         return 1
250     else:
251         L = [1]
252         for i in range(n):
253             L = somme(L+[0],[0]+L)
254         return L[p]
255 
256 #ou
257 def binom2b(n,p):
258     if p < 0 or p > n : #traitement des cas de p hors [0,n]
259        return 0
260     elif p == 0 or p == n:
261         return 1
```

```
    else: #n!=p and p!=0 and n!=0
        L1,L2 = [1],[1] #cas n==1
        for i in range(n-1):
            S = somme(L1,L2)
            L1 = [1] + S
            L2 = S + [1]
        return L1[p]
```

Le calcul de $\binom{5}{3}$ avec `binom2a(5,3)` nécessite 5 appels de `somme(L1,L2)`. Chacun de ces appels amène à effectuer de 2 à 6 sommes, càd. au total $\sum_{k=2}^{6} k = 20$ sommes.
Avec `binom2b(5,3)`, on effectue 4 appels de `somme(L1,L2)`. Chacun de ces appels amène à effectuer de 1 à 4 sommes, càd. au total $\sum_{k=1}^{4} k = 10$ sommes.
Notons cependant que la complexité des fonctions est la même, en $O(n^2)$.

Q12 Dans la version conseillée du code, l'appel récursif `division(n-b,b)` n'a lieu qu'un seule fois, alors qu'il a lieu deux fois avec la seconde version (espace mémoire doublée).

Q13

```
def coloriage(L,i,j):
    if L[i][j] == 1:
        L[i][j] = 2
        if i != 0:
            coloriage(L,i-1,j)
        if j != 0:
            coloriage(L,i,j-1)
        if i != len(L)-1:
            coloriage(L,i+1,j)
        if j != len(L[0])-1:
            coloriage(L,i,j+1)

#ou avec moins d'appels récursifs
def indicesVoisins(n,p,i,j):
    '''renvoie la liste des indices des voisins de (i,j) qui sont blancs'''
    orientation = {(0,-1),(-1,0),(1,0),(0,1)} #ensemble des directions autorisées
    listeVoisins = []
    #on ne considère que les indices de la grille
    if i >= 0 and i <= n-1 and j >= 0 and j <= p-1:
        for direction in orientation:
            I = i + direction[0]
            J = j + direction[1]
            #on ne considère que les indices des voisins qui sont blancs
            if I >= 0 and I <= n-1 and J >= 0 and J <= p-1 and L[i][j] == 1:
                listeVoisins.append([I,J])
    return listeVoisins

def coloriage2(L,i,j):
    '''L est supposé non vide'''
    n,p = len(L),len(L[0])
    if L[i][j] == 1:
        L[i][j] = 2
        V = indicesVoisins(n,p,i,j)
        for point in V: #2<=k<=4
            I,J = point[0],point[1]
            if L[I][J] == 1:
                coloriage2(L,I,J)
```

Q14

```
312 def monnaie2(n,euros):
313     '''renvoie le nombre de manières de constituer n euros avec des pièces à valeurs dans la liste "euros", rangée dans l'ordre décroissant; on suppose que "euros" contient la valeur 1'''
314     p = euros[0]
315     if p == 1:
316         return 1 #une seule manière de rendre la monnaie avec uniquement des pièces de 1
317     else:
318         compteur = 0
319         for i in range(n//p+1):  #n//p pièces de p euros rendues
320             compteur = compteur + monnaie2(n-p*i,euros[1:])
321             #pour chaque (p,i) on a une solution (car n-p*i>=0) et on complète avec des 1
322         return compteur
323 
324 def monnaie(n): #fonction principale
325     return monnaie2(n,[5,2,1])
```

Q15

```
328 def chemins(x,y):
329     if x == 0 and y == 0:
330         return 1
331     else:
332         if x != 0 and y != 0:
333             return chemins(x-1,y-1) + chemins(x-1,y) + chemins(x,y-1)
334         if x == 0 and y != 0:
335             return chemins(x,y-1)
336         if x != 0 and y == 0:
337             return chemins(x-1,y)
```

Q16

```
340 def nb(L,N):
341     if N == 0:
342         if 0 in L:
343             return [N]
344         else:
345             return []
346     else:
347         n = str(N)
348         for i in range(len(n)):
349             if int(n[i]) not in L:
350                 return nb(L,N-1)
351         return nb(L,N-1) + [N]
```

Q17

```
354 def supprimeDoublon(L):
355     if len(L) == 0 or len(L) == 1:
356         return L
357     else: #le choix de L[-1] plutôt que L[0] permet de garder l'ordre de lecture de gauche à droite de L sans doublons
358         if L[-1] not in L[:-1]:
359             return supprimeDoublon(L[:-1]) + [L[-1]]
360         else:
361             return supprimeDoublon(L[:-1])
```

Q18

```
364 def fusion_ligne(tableau,i):
365     ligne = [[0 for i in range(len(tableau[0]))]] #liste de listes de zé
    ros
366     return tableau[:i]+ligne+tableau[i:]
367 
368 def fusion_colonne(tableau,j):
369     c = []
370     for k in range(len(tableau)):
371         c.append(tableau[k][:j]+[0]+tableau[k][j:])
372     return c
373 
374 def tours(n):
375     '''renvoie la liste des configurations de n tours sur un échiquier
    de taille n avec n>=1'''
376     if n == 1:
377         return [[[1]]] #liste des configurations : liste de listes de
    listes
378     else:
379         liste = []
380         for i in range(n):
381             for j in range(n):
382                 l = tours(n-1) #liste de configurations
383                 for k in range (len(l)):
384                     t = fusion_colonne(l[k],j)
385                     t = fusion_ligne(t,i)
386                     t[i][j] = n
387                     liste.append(t)
388         return liste
```