Récursivité

1 Contenu du document et travail à réaliser

- Le document présente des exemples de fonctions définies de façon récursive.
- L'objectif est de comprendre la programmation récursive de quelques fonctions puis de chercher à en écrire par vous-mêmes quelques-unes.
- Trois exercices sont à rendre (exercices dont le titre est sur fond jaune) dans les casiers numériques de vos enseignants pour le 20/01.

fonction récursive

Une fonction récursive est une fonction qui s'appelle elle-même.

2 Récursivité et listes

Exercice 1 (une fonction récursive déjà rencontrée)
On définit la fonction python ci-dessous dans laquelle L est une liste.

Python

```
def r(x,deb,fin,L):
    if deb>fin : return "fini"
    t=(deb+fin)//2
    if x==L[t] : return t
    if x<L[t] : return r(x,deb,t-1,L)
    else : return r(x,t+1,fin,L)
```

Décrire sur papier les étapes lors de l'appel ci-dessous. Quel est le nom de l'algorithme utilisé ici ?

Python

```
L=[2,3,4,7,11,15,17]
print recherche(4,0,len(L)-1,L)
```

Exercice 2
On définit la fonction ci-dessous :

Python

```python
# -*- coding: utf-8 -*-

def ed(L,M=[]):
    """ L est une liste """
    if not(L):return M
    a=L.pop(0)
    if a not in M : M.append(a)
    return ed(L,M)
```

On rappelle que L.pop(i) supprime l'élément d'indice i de la liste L.
Exemple : si L=[5,7,8,15], après l'instruction L.pop(1), la liste L est égale à [5,8,15].

Que renverra l'appel ci-dessous :

Python

```python
L=[2,3,2,6,8,9,9,10,9,3,6,7,8,8,9]
M=ed(L)
print M
```

Exercice 3

On définit la fonction ci-dessous où le paramètre L est une liste :

Python

```python
def pp(L):
    if len(L)==1 : return L[0]
    if L[0]<L[1] : L.pop(1)
    else : L.pop(0)
    return pp(L)
```

Que renverra l'appel ci-dessous :

Python

```python
L=[24,45,2,3,2,6,8,9,9,10,9,3,6,7,8,8,9]
print pp(L)
```

3 Récursivité et travail sur les chaînes de caractères

Exercice à rendre 1

La fonction python compte_a ci-dessous est telle que :
– input : une chaîne de caractères.
– output : le nombre de lettres a dans la chaîne.
Retrouver ce qui a été effacé (pointillés) :

Python

```python
# -*- coding: utf-8 -*-


def compte_a(chaine):
    if len(chaine)==1:
        if chaine[0]=='a' :return 1
        else:return 0
    else:
        if chaine[0]=='a':  ..................................
        else :  ....................................

print compte_a('blabla')  # affiche 2

print compte_a('dur')   # affiche 0
```

4 Récursivité et opérations sur les entiers

Exercice 4

Un programme en langage python :

Python

```python
# -*- coding: utf-8 -*-

def p(a,b):
    if b==0 : ..................
    return p(a+1,b-1)
```

$p(u,v)$ doit renvoyer la somme $u+v$. Compléter le cas de base (pointillés).

"Structure" d'une fonction récursive

– Dans le programme précédent, le cas « b==0 » est le "cas de base". C'est un cas pour lequel l'image à retourner ne nécessite pas d'appel à la fonction p.
– Dans les appels $p(a+1,b-1)$, le second argument est décrémenté d'une unité à chaque appel et finira par être nul (b est supposé être un entier positif dans l'appel initial). C'est ce qui garantit que le programme s'achèvera.

Exercice 5

Un programme en langage python :

Python

```
# -*- coding: utf-8 -*-

def f(a,b):
    """ a et b sont deux entiers naturels non nuls."""
    if b==1 : return a
    return a+f(a,b-1)

print f(3,5)
```

1. Quel est le résultat affiché par ce programme ?
2. Quel est le cas de base dans cette fonction récursive ?
3. Qu'est ce qui garantit dans les appels récursifs que le programme finira par s'arrêter ?
4. Que retourne $f(a,b)$ (a et b étant des entiers naturels non nuls) ?

Savoir Faire

Définir une fonction de façon récursive.
- Prévoir les "cas de base", c'est à dire ceux qui ne nécessitent pas d'appel récursif de la fonction.
- S'assurer que, dans les appels récursifs, les arguments sont plus "simples" que ceux avec lesquels la fonction a été appelée (ce qui signifie essentiellement qu'ils « évoluent vers le cas de base »)
- Reconstituer correctement la valeur de retour de la fonction à partir du résultat du ou des appels récursifs.

5 Quelques exercices liés à l'arithmétique

Exercice 6 (grille de sudoku)
Pour écrire un programme de résolution d'un sudoku, on utilise une liste de listes pour définir la grille. Par exemple :

Python

```
G=[[3,0,0,4,1,0,0,8,7]
 ,[0,0,9,0,0,5,0,6,0]
 ,[4,0,0,7,9,0,5,0,3]
 ,[0,7,3,2,4,0,0,0,0]
 ,[0,0,0,0,0,0,0,0,0]
 ,[0,0,0,0,7,8,2,4,0]
 ,[6,0,2,0,8,3,0,0,5]
 ,[0,5,0,1,0,0,3,0,0]
 ,[1,3,0,0,2,4,0,9,6]]
```

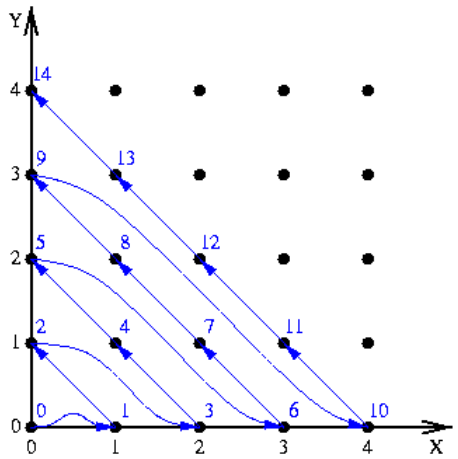
Les éléments de la grille sont les éléments :
– ligne 0 : G[0][0], G[0][1], G[0][2], …, G[0][8]
– ligne 1 : G[1][0], G[1][1], G[1][2], …, G[1][8]
– …
– ligne 8 : G[8][0], G[8][1], G[8][2], …, G[8][8]
On décide également de repérer chaque cellule par un entier.
La cellule de coordonnées (i, j) selon la numérotation ci-dessus ($0 \leqslant i \leqslant 8$, $0 \leqslant j \leqslant 8$) sera également repérée par $f(i, j)$ avec la définition ci-dessous :

Python

```
# -*- coding: utf-8 -*-


def f(i,j):
    if i==0 and j==0:return 0
    if j>0: return f(i,j-1)+1
    if j==0:return f(i-1,8)+1
```

Donner une définition non récursive de $f(i, j)$.

Exercice à rendre 2 (diagonale de Cantor)
On numérote chaque point du plan de coordonnées $(x; y)$ (où x et y sont des entiers naturels) par le procédé suggéré sur la figure ci-dessous :



Écrire une fonction numero(x,y), définie de façon récursive, qui retourne le numéro du point de coordonnées $(x; y)$.

Exercice à rendre 3 (nombre de chiffres d'un entier)
On rappelle que le quotient de la division euclidienne d'un entier n par 10 donne le nombre de dizaines de cet entier. Le quotient de la division euclidienne de $n = 5478$ par 10 est par exemple 547.

En déduire une fonction NbChiffres(n) prenant en paramètre un entier naturel n (écrit en décimal) et retournant le nombre de chiffres de cet entier n en base 10. Cette fonction sera définie récursivement, en langage python.

Exercice 7
Écrire une fonction python récursive reste(a,b) prenant en arguments deux entiers naturels non nuls a et b et retournant le reste de la division euclidienne de a par b.

Exercice 8 (Algorithme d'Euclide)
A l'aide des deux propriétés suivantes :
– pour tous entiers a et b, on a $\text{pgcd}(a;b) = \text{pgcd}(a-b;b)$.
– pour tout entier a, on a $\text{pgcd}(a;0) = a$.
Écrire une fonction python récursive pgcd(a,b) retournant le pgcd des entiers naturels a et b.

6 Exercices liés à la notion de suite

Exercice 9
n étant un entier naturel non nul, on définit $n!$ (lire "factoriel n") par :

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

Définir une fonction python récursive calculant $n!$.

Exercice 10
La suite de Fibonacci est définie par $f_0 = f_1 = 1$ et pour tout naturel $n \geqslant 2$ par :

$$f_n = f_{n-1} + f_{n-2}$$

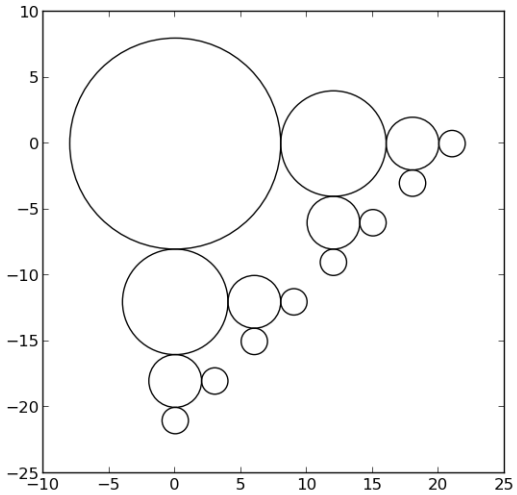
1. Écrire une fonction python récursive f(n) calculant le terme d'indice n de cette suite.
2. Combien d'appels à la fonction f sont-ils faits pour calculer $f(5)$?
3. Écrire une version récursive du calcul de $f(n)$ ne nécessitant qu'au plus n appels à elle-même pour calculer $f(n)$ (pour $n \geqslant 2$). On pourra chercher sur internet la notion d' « accumulateur ».

Efficacité d'un algorithme

Pour bien comprendre la différence entre les deux programmations proposées de la suite de Fibonacci, calculer f(50) avec chacune des deux programmations et mesurer le temps d'exécution : le temps d'exécution de f(n) est en fait exponentiel en n pour la première programmation, alors qu'il est linéaire en n pour la seconde programmation. Ce problème de temps d'exécution est essentiel en programmation : il est clair que la première programmation est inacceptable pour un programme d'usage quotidien alors que le temps d'exécution de la seconde est tout à fait raisonnable. Les deux fonctions déterminent pourtant les mêmes nombres ! Nous reviendrons plus tard sur ce problème de « complexité en temps » d'un programme.

7 Une figure définie de façon récursive

On peut décrire la figure suivante de façon récursive :



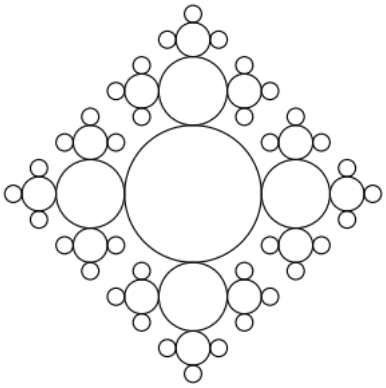
La figure est formée d'un cercle et de deux copies de ce cercle ayant subies une réduction d'un facteur 2, ces deux petits cercles étant tangents extérieurement au cercle initial et tels que les lignes des centres sont parallèles aux axes du repère.
Ces deux petits cercles deviennent à leur tour "cercle initial" pour poursuivre la figure.
On peut traduire avec python ce descriptif récursif de la façon suivante :

Python

```python
# -*- coding: utf-8 -*-

# pour en savoir plus sur pylab, chercher pylab
# ou matplotlib sur la toile
import pylab

F=pylab.gca() # F peut être vue comme un objet 'figure'

def cercle(x,y,r):
    """cercle de centre (x,y) et de rayon r"""
    # création du cercle :
    cir = pylab.Circle([x,y], radius=r, fill=False)
    # ajout du cercle à la figure :
    F.add_patch(cir)

def CerclesRec(x,y,r):
    """ construction récursive de la figure"""
    cercle(x,y,r)
    if r>1:
        CerclesRec(x+3*r/2,y,r/2)
        CerclesRec(x,y-3*r/2,r/2)

# appel de la fonction CerclesRec
CerclesRec(0,0,8)

# pour placer toute la figure dans un repère orthonormé :
pylab.axis('scaled')
# affichage de la figure :
pylab.show()
```

1. Qu'est ce qui garantit que cette fonction ne s'appellera qu'un nombre fini de fois ?
2. Dans l'appel initial, si l'on change CerclesRec(0,0,8) par CerclesRec(0,0,64), qu'obtiendra-t-on ?

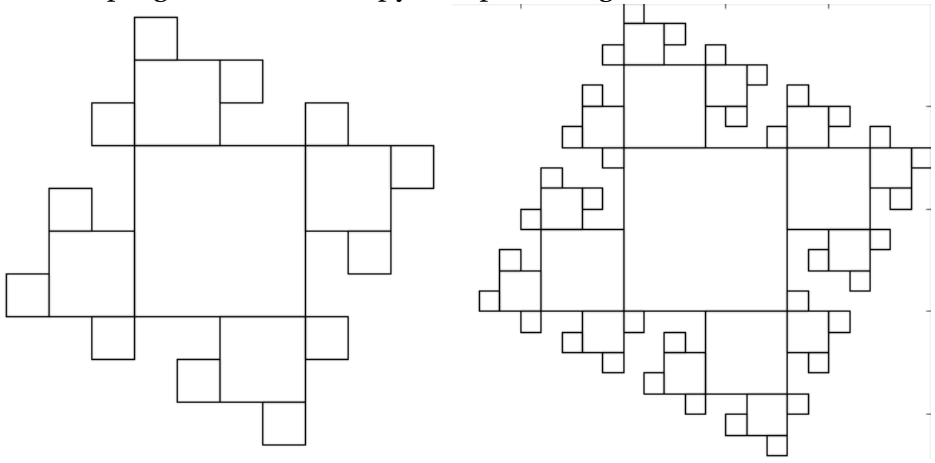
Exercice 11 (modification de la figure)
On modifie la figure précédente pour obtenir :



Modifier le programme python précédent pour obtenir cette figure.
On pourra utiliser un paramètre supplémentaire pour définir la fonction récursive (abscisse du centre, ordonnée du centre, rayon du cercle, position du voisin) où position sera l'une des chaînes de caractères : haut, bas, droite, gauche.

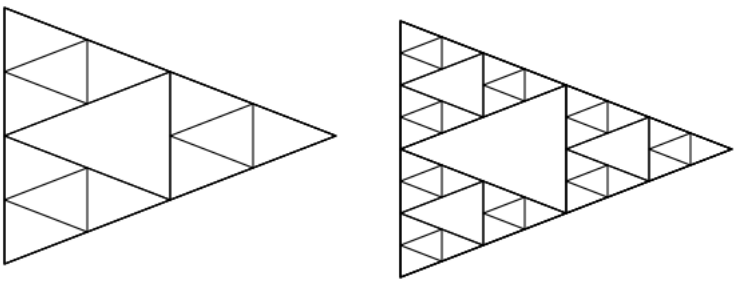
Exercice 12 (modification 2 de la figure)

pylab.Rectangle((x,y),a,b, fill =False) permet de définir un rectangle dont les côtés, de longueurs a et b, sont parallèles aux axes et dont le sommet "sud-ouest" a pour coordonnées (x, y).
Écrire un programme récursif python pour la figure dont on donne deux étapes ci-dessous :



Exercice 13

pylab.Polygon([(xa,ya),(xb,yb),(xc,yc)], fill =False) permet de définir un triangle dont les sommets ont pour coordonnées (xa, ya), (xb, yb), (xc, yc).
Écrire un programme récursif python pour la figure dont on donne deux étapes ci-dessous :



8 Pour info.

En vrac, quelques problèmes qui se prêtent bien à une programmation récursive.
– Solveur de grille de sudoku.
– Afficher les étapes pour déplacer les anneaux dans le jeu des tours de Hanoï.
– Déterminer comment placer 8 reines sur un échiquier 8 sur 8 de façon à ce qu'aucune ne soit en prise avec une autre.
– Déterminer un parcours d'un cavalier sur un échiquier passant par chacune des cases de l'échiquier une unique fois.
– Algorithme du tri par fusion, algorithme du tri rapide.
– Représentation de fractales.