Récursivité
Exercices corrigés

Exercice 1: L'exponentiation rapide

Écrire une fonction récursive d'entête **puissance(x, n)** qui retourne la valeur de x^n en utilisant la formule suivante:

$$puissance(x,n) = \begin{cases} 1 & si \quad n = 0 \\ x & si \quad n = 1 \\ puissance(x^2, n//2) & si\ n\ est\ pair \\ x * puissance(x^2, n//2) & si\ n\ est\ impair \end{cases}$$

```
def puissance(x,n):
     if n==0: return 1
     if n==1: return x
     if n%2==0 : return puissance(x*x,n//2)
     return x*puissance(x*x,(n-1)/2)
```

Exercice 2: Conversion binaire <---->décimal

Écrire une fonction récursive d'entête **Base2(n)** qui permet de convertir un entier naturel n dans le système binaire. La suite binaire est stockée dans une liste.

Exemple :	
`>>> base2 (6)` `[1 , 1 , 0]`	`>>> base2 (15)` `[1 , 1 , 1,1]`

Écrire une fonction récursive d'entête **def Base10(a)** qui retourne la valeur décimale d'une suite binaire passée comme argument (on traite deux cas a est un entier et a est une liste) .

Exemple :	
`>>> base10 ([1,1,0])` `6`	`>>> base10 (1111)` `15`

```
def base2(n):
     if n==0 or n==1:
     return[n]
     else:
          return base2(n//2)+[n%2]
```

```
def base10(a):
     if type(a) is not list: a=list(str(a))
     if len(a)==1:
          return int(a[0])
     else:
          return int(a[0])*2**(len(a)-1)+base10(a[1:])
```

Version2 (sans utilisation de l'exponentiation)

```
def base10(a):
     if type(a) != list: a=[int(e) for e in str(a)]
     if len(a)==1: return a[0]
     return 2*base10(a[:-1])+a[-1]
```

Exercice 3: Opérations sur les listes

1. Ecrire une fonction récursive de nom **TailleListe** qui prend en argument une liste et retourne le nombre d'éléments qu'elle contient.
2. Ecrire une fonction de nom **AffListe** qui affiche les éléments de la liste passée en paramètre.
3. Ecrire une fonction récursive de nom **AffListeInv** qui affiche, dans l'ordre inverse, les éléments de la liste passée en paramètre.
4. Ecrire une fonction **InsOrd** qui retourne une liste triée obtenue par l'insertion d'un nouvel élément à une liste triée.

Remarque:
On s'interdira d'utiliser les méthodes prédéfinies sur liste: append, sort, len, reverse, ...

```
def TailleListe(L):
     if L==[]:
          return 0
     else:
          return 1+TailleListe(L[1:])
```

```
def afficheliste(L):
     if L==[]:
          print()
     else:
          print(L[0],end='')
          afficheliste(L[1:])
```

```
def AffListeInv(L):
     if L!=[]:
          AffListeInv(L[1:])
          print(L[0],end='\t')
```

```
def InsOrd_V1 (l,v):
     if l==[] or v>l[-1]:
          return l+[v]
     else:
          return InsOrd1(l[:-1],v) + l[-1:]
```

```
def InsOrd_V2 (l,v):
     if l==[] or v<l[0]:
          return [v]+l
     else:
          return [l[0]]+InsOrd2(l[1:],v)
```

Exercice 4: Opérations sur les chaînes de caractères

1. Ecrire une fonction récursive **strcpy(src,dest)** qui retourne une copie de la chaine de caractères ch (copiée caractère par caracère).

.

2. Ecrire une fonction récursive **strcpm(ch1,ch2)** qui permet de comparer deux chaînes de caractères. Cette fonction retourne 1 si ch1 >ch2, retourne 0 si ch1=ch2 et retourne -1 si ch1<ch2.
3. Ecrire une fonction récursive **anagramme(ch1,ch2)** qui retouren **True** si une chaîne **ch1** donnée est anagramme d'une autre chaîne **ch2** ou **False** sinon. **Par exemple** : 'algorithme' est anagramme de 'logarithme' et 'tanger' est anagramme de 'argent'

```
def strcpy(src):
    if src=='':
        return ''
    else:
        return src[0]+strcpy(src[1:])
```

```
def strcmp(ch1,ch2):
    if ch1=='':
        if ch2 == '':
            return 0
        else:
            return -1
    else:
        if ch2 == '':
            return 1
        else:
            if ch1[0]>ch2[0]:
                return 1
            else:
                if ch1[0]<ch2[0]:
                    return -1
                else:
                    return strcmp(ch1[1:],ch2[1:])
```

```
def anagramme1(ch1,ch2):
    if len(ch1)!= len(ch2): return False
    if ch1=='' and ch2=='':
        return True
    else:
        if ch1[0] not in ch2:
            return False
        else:
            return anagramme(ch1[1:],ch2.replace(ch1[0],''))
```

```
def anagramme2(ch1,ch2):
  if len(ch1)!= len(ch2): return False
  if ch1=='' and ch2=='':     return True
  if ch1[0] not in ch2  :     return False
  p=ch2.find(ch1[0])    #chercher la position de premier caractère de ch1 dans ch2
  return anagramme(ch1[1:], ch2[:p]+ch2[p+1:])
```

Exercice 5: Recherche dichotomique

On rappelle que l'idée de la recherche par dichotomie d'un élément x dans un tableau trié pour l'ordre croissant est la suivante :

On compare x avec l'élément m du milieu du tableau,

- si x = m : on a trouvé une solution ;
- si x < m : x ne peut pas se trouver après m dans le tableau ; il suffit alors de le rechercher dans la partie à gauche de m;
- si x > m : x ne peut pas se trouver avant m dans le tableau ; il suffit alors de le rechercher dans la partie à droite de m;

b) Donner une fonction itérative **dico_iteratif** qui traduit cet algorithme?

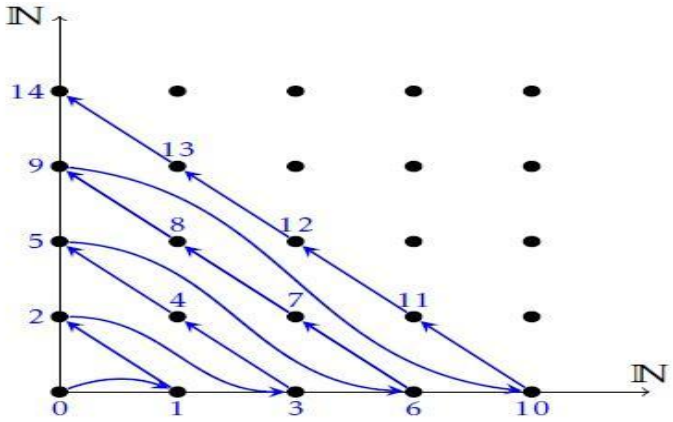
a) Donner une fonction récursive **dico_recusif** qui traduit cet algorithme ?

```
def dico_iteratif (x, T):
    g =0
    d = len(T) - 1
    while g<=d :
        m = (g + d) // 2
        if T[m] == x :
            return m
        elif T[m] < x :
            g = m + 1
        else:
            d = m - 1
    return None
```

```
def dico_recursif(x, T, g, d):
    if g>d : return None
    m = (g+d) // 2
    if T[m] == x :
        return m
    elif T[m] < x :
        return dico_recursif(x,T,m+1,d)
    else:
        return dico_recursif(x,T,g,m-1)
```

Exercice 6: (diagonale de Cantor)

L'ensemble $\mathbb{N}\,\mathbb{N}$ x $\mathbb{N}\,\mathbb{N}$ est dénombrable en numérotant chaque couple (x, y) $\in \mathbb{N} \in \mathbb{N}^2$ suivant le procédé suggéré par la figure ci-dessous :



1- Ecrire une fonction récursive qui retourne le numéro du point de coordonnées (x, y).
2- Ecrire une fonction récursive qui retourne coordonnées (x, y) correspondant un numéro de point donné.

```
def numerote(x,y):
     if x==0 and y==0: return 0
     if y>0:
          return 1+numerote(x+1,y-1)
     else:
          return 1+numerote(0,x-1)
```

#version directe (plus rapide)

```
def numerote(x, y):
     return x + (x+y)*(x+y+1)//2
```

```
def reciproque(n):
     if n==0:
          return (0,0)
     else:
          (x,y)=reciproque(n-1)
          if x>0:
               return (x-1,y+1)
          else:
               return (y+1,0)
```

Exercice 7:

On suppose disposer d'une fonction cercle(x, y, r) qui trace à l'écran un cercle de centre (x, y) de rayon r.

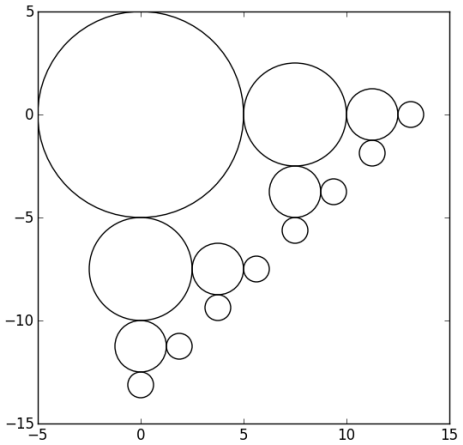
```
import pylab
F = pylab.gca() # F est une 'figure'
```

```
def cercle (x ,y , r ) :
    cir = pylab . Circle ([ x , y] , radius=r , fill =False )
    F. add_patch(cir)
```

```
cercle(0,0,4)
pylab.axis('scaled')
pylab .show()
```

1- Ecrire la fonction récursive **CerclesRec(x, y, r)** permettant de tracer le dessin ci-dessous :

```
CerclesRec(0,0,5)
pylab.axis('scaled')
pylab .show()
```

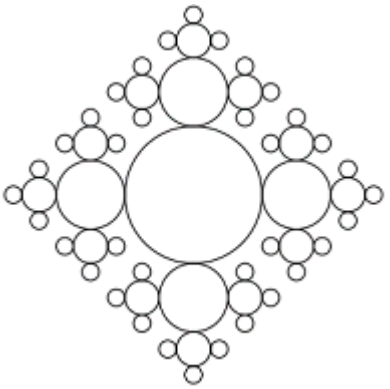


(Chaque cercle est de rayon moitié moindre qu'à la génération précédente).

```
def CerclesRec(x ,y , r ) :
     cercle (x ,y , r )
     if r >1:
          CerclesRec(x+3*r/2 ,y , r /2)
          CerclesRec(x ,y-3*r/2 , r /2)
```

2- Modifier le programme python précédent pour obtenir cette figure.

On pourra utiliser un paramètre supplémentaire pour définir la fonction récursive (x, y, y, position) où position (position du voisin) sera l'une des chaînes de caractères: haut, bas, droite, gauche.

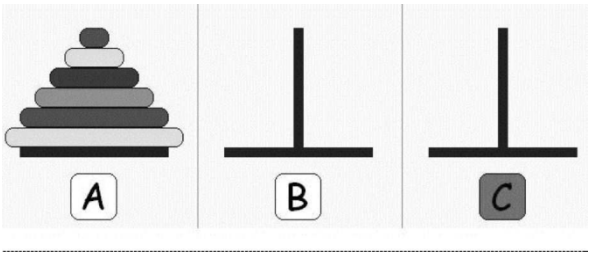


```
def CerclesRec2(x ,y , r,pos='' ) :
     cercle (x ,y , r )
     if r >1:
          if pos!='gauche' : CerclesRec2(x+3*r/2 ,y , r /2, 'droit')
          if pos!='droit'  : CerclesRec2(x-3*r/2 ,y , r /2,
'gauche')
          if pos!='bas'    : CerclesRec2(x ,y+3*r/2 , r /2, 'haut')
          if pos!='haut'   : CerclesRec2(x ,y-3*r/2 , r /2, 'bas')
```

Exercice 8 : Tour de Hanoï

Le problème des **tours de Hanoï** est un jeu de réflexion imaginé par le mathématicien français Édouard Lucas, et consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :

- on ne peut déplacer plus d'un disque à la fois,
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.



Analyse de la solution :

En analysant les solutions pour $n = 2$ et $n = 3$ et en numérotant les n disques de 1 à n dans l'ordre croissant des diamètres, on peut résumer les opérations ainsi : pour déplacer les n disques de la tour A à la tour C, il faut déplacer n - 1 disques de la tour A à la tour B, le disque n de la tour A à la tour C puis déplacer les n - 1 disques de la tour B à la tour C. L'algorithme reprend mot pour mot ce principe.

Le nombre de déplacements de disques vérifie donc la relation de récurrence :

$$\begin{cases} x_0 &= 0 \\ x_n &= 2x_{n-1} + 1 \quad \text{si } n \geq 1 \end{cases}$$

ce qui donne bien

$$x_n = 2^n - 1$$

#Tour de Hanoï

```
def hanoi(n, A=1, B=2, C=3):
    if n == 0: return
    hanoi(n-1, A, C, B)
    print("Déplacer le disque %d de la tige %d vers la tige %d."% (n,A,C))
    hanoi(n-1, B, A, C)
```