Informatique — MPSI/PCSI

Correction du TP n° 7

1 Fonction récursive

Exercice 7.1
Version itérative de l'algorithme d'Euclide.

```
def euclide(a,b):
    while b!=0:
        r=a%b
        a=b
        b=r
    return(a)
```

où l'on renvoie forcément a car la condition de terminaison de la boucle `while` est $b = 0$.

Sachant que pour tout $(a, b) \in \mathbb{N}_*^2$, $a > b$, on a :

$$\text{PGCD}(a, b) = \text{PGCD}(a - b, b) = \text{PGCD}(b, a\%b)$$

où $a\%b$ désigne le reste de la division de a par b, il vient alors la version récursive de l'algortithme d'Euclide :

$$\text{PGCD}(a, b) = \begin{cases} a & \text{si } b = 0 \\ \text{PGCD}(b, a\%b) & \text{sinon} \end{cases}$$

que l'on peut implémenter sous la forme :

```
def PGCD(a,b):
    if b==0:
        return(a)
    else:
        return(PGCD(b,a%b))
```

Exercice 7.2
Le seul point auquel il faut être attentif est l'ordre des instructions : on appelle la fonction à l'indice $n - 1$ puis on affiche une nouvelle ligne dans le premier cas, on affiche une nouvelle ligne puis on affiche la fonction dans le second.

1. Pour afficher des étoiles dans l'ordre décroissant, il suffit

$$\forall n \in \mathbb{N}_*, \quad \text{étoiles}(n) = \begin{cases} \text{Afficher } \texttt{"*"} & \text{si } n = 1 \\ \text{Afficher } n \texttt{ "*" puis appeler étoiles}(n-1) & \text{sinon} \end{cases}$$

avec le variant n qui décroit de n à 1. ce que l'on implémente sous la

```
def etoiles(n):
    print(n*"*")
    if n>1:
        etoiles(n-1)
```

La pile d'éxécution pour $n = 4$ conduit à :

```
etoile(4)
    ****
    etoiles(3)
        ***
        etoiles(2)
            **
            etoiles(1)
                *
```

2. En remarquant qu'en appelant la fonction avant d'afficher, soit avec :

$$\forall n \in \mathbb{N}_*, \quad \text{étoiles}(n) = \begin{cases} \text{Afficher } \texttt{"*"} & \text{si } n = 1 \\ \text{appeler étoiles}(n-1) \text{ puis Afficher } n \texttt{"*"} & \text{sinon} \end{cases}$$

avec le même variant, alors la pile d'exécution devient :

```
etoile(4)
    etoiles(3)
        etoiles(2)
            etoiles(1)
                *
            **
        ***
    ****
```

et produit bien le nombre croissant d'étoiles attendu.

Exercice 7.3
Soient a et b deux entiers non nuls avec $a > b$. Alors :

$$\exists q \in \mathbb{N}, \exists r \in [\![0, b-1]\!], a = qb + r$$

d'où :

$$\begin{aligned} a &= qb + r \\ a \leftarrow a - b &= (q-1)\,b + r \\ &\vdots \\ a \leftarrow a - b &= (q-(q-1))\,b + r \geqslant b \\ a \leftarrow a - qb &= (q-q)\,b + r = r < b \end{aligned}$$

1. On en déduit que le reste de la division euclidienne de a par $b \in [\![1, a-1]\!]$ est défini de façon récursive par :

$$R(a,b) = \begin{cases} a & \text{si } a < b \\ R(a-b,b) & \text{sinon} \end{cases}$$

avec le variant « a » qui décroit de a à r par pas de $-b$.

```
def R(a,b):
    if a<b:
        return(a)
    else:
        return(R(a-b,b))
```

Pour $a = 11$ et $b = 3$, la pile d'exécution est :

```
R(11,3)
    R(8,3)
        R(5,3)
            R(2,3)
                2
            2
        2
    2
```

2. En observant que le nombre d'appels récursifs correspond au quotient q, il suffit pour définir le quotient d'utiliser avec $a = 11$ et $b = 3$, une pile d'exécution de la forme :

```
Q(11,3)
    1+Q(8,3)
```

```
        1+Q(5,3)
            1+Q(2,3)
                0
            1
        2
    3
```

telle que la fonction quotient soit alors définie par :

$$Q(a,b) = \begin{cases} 0 & \text{si } a < b \\ 1 + Q(a-b,b) & \text{sinon} \end{cases}$$

que l'on implémente sous la forme :

```
def Q(a,b):
    if a<b:
        return(0)
    else:
        return(1+Q(a-b,b))
```

Exercice 7.4
Le cas terminal est ici associé aux chaînes de une caractère qui n'ont qu'une seule permutation. Partant d'une chaîne `c` de n caractères, il s'agît donc de diminuer successivement le nombre de caractères de 1. En isolsant le premier carcatère `c[0]`, on peut alors rechercher les permutations des $(n-1)$ lettres suivantes (la coupe `c[1:]`), puis, pour chaque permutation $e \in p(\texttt{c[1:]})$ insérer la lettre `c[0]` à chacune des n positions, soit :

$$\{\texttt{e[:k]+c[0]+e[k:]} \mid \forall k \in [\![0, n-1]\!]\}$$

ou `e[:0]` et `e[n-1::]` sont des chaînes vides. Il vient alors la définition récursive :

$$p(c) = \begin{cases} [c] & \text{si } \texttt{len}(c) = 1 \\ \{\texttt{e[:k]+c[0]+e[k:]} \mid \forall k \in [\![0, n-1]\!], \quad \forall e \in p(\texttt{c[1:]})\} & \text{sinon} \end{cases}$$

ce que l'on peut écrire :

```
def permut(c):
    if len(c)==1:
        return([c])
    else:
        return([e[:k]+c[0]+e[k:] for k in range(len(c)) for e in permut(c[1:])])
```

ou, en détaillant les étapes :

```
def permut(chaine):
    if l==1: # Dans le cas ou la chaine est de longuer 1, une seule permutation
        return [chaine]
    else:
        L=[]
        c=chaine[0]
        Li=permut(chaine[1:]) # genere toute les permuations des n-1 derniers caracteres
        for p in Li: # parcourt la liste des permuations precedentes
            for i in range(l): # insere le premier caractere a chacune des places possibles
                L.append(p[:i]+c+p[i:])
        return L
```

2 Généralisation

Exercice 7.5
Pour $n = 13$, on appelle la fonction successivement pour $n = 6$, $n = 3$, $n = 1$ et $n = 0$. Il faut deux multiplications pour calculer `exp_rapide(x,n)` à partir de `exp_rapide(x,m)` (où `m=n//2`) si n est impair et une multiplication si n est pair. On effectue donc au total $2 + 1 + 2 + 2 = 7$ multiplications.

En comparaison, l'algorithme de multiplication naïf demanderait à effectuer 12 multiplications.

Exercice 7.6
Le principe de la recherche dichotomique est de diviser l'intervalle $[\![0, n-1]\!]$ en 2 et de regarder dans lequel se situe l'élément recherché. En notant $m = \left\lfloor \frac{n-1}{2} \right\rfloor$ l'indice du milieu, il vient trois cas :

— si $L_m = x$, il faut renvoyer « vrai » ;
— sinon si $L_m > x$, il faut garder l'intervalle $[\![0, m-1]\!]$;
— sinon, $L_m < x$, il faut garder l'intervalle $[\![m+1, n-1]\!]$.

```
def rech_dicho(x,L):
    """Cherche si x appartient a L par dichotomie

    Parametres
    x (float)
    L (list) : liste de réels tries par ordre croissant

    Retour
    Boolean
    True si x appartient a L, False sinon
    """
    n=len(L)
    if n==0:
        return False
    else:
        m=n//2
        if L[m]==x:
            return True
        if L[m-1]>=x:
            return rech_dicho(x,L[:m])
        else:
            return rech_dicho(x,L[m+1:])
```

On notera que dans le cas d'une liste L de longueur $n = 2^m$, on fait au pire m appels récursifs et on utilise une place mémoire de $2L$ (le L initial, puis $L/2$, puis $L/4$, etc.) Pour limiter l'espace mémoire, il est préférable de faire des appels avec indices, sous la forme :

```
def dichotomie(L,x,i=0,j=None):
    if j==None:
        j=len(L)-1
    if i==j:
        return(L[i]==x)
    else:
        m=(i+j)//2
        if L[m]==x:
            return(True)
        elif L[m]>x:
            return(dichotomie(L,x,i,m-1))
        else:
            return(dichotomie(L,x,m+1,j))
```

3 Similitudes et fractale

Exercice 7.7 1. Si N est l'image de M par la rotation de centre I et d'angle t, les coordonnées de l'image M' de M par la similitude de centre I, d'angle t et de rapport k sont obtenues en ajoutant à I les coordonées du vecteur $k\overrightarrow{IN}$.
La fonction suivante exprime ce calcul.

```
def simil(I,M,k=1,t=0):
    N=rotation(I,M,t)
    return [I[i]+k*(N[i]-I[i]) for i in range(2)]
```

ou, comme les transformations commutent

```
def simil(I,M,k=1,t=0):
    return(rotation(I,[I[i]+k*(M[i]-I[i]) for i in range(2)],t))
```

2. Il suffit d'appliquer la fonction précédente à tous les élélments de L en parcourant cette liste par valeurs.

```
def simil_poly(I,L,k=1,theta=0):
    return [simil(I,e,k,theta) for e in L]
```

Exercice 7.8 1. Si A et B désignent les deux extrémités d'un segment rejoignant deux points successifs de L et $n > 0$ la courbe apparaissant après une nouvelle étape de construction rejoints les points A, C, E et D définis comme suit :

— C et D verifient $\overrightarrow{AC} = \frac{1}{3}\overrightarrow{AB}$ et $\overrightarrow{AE} = \frac{2}{3}\overrightarrow{AB}$
— E est l'image du point D par la rotation de centre C et d'angle $\frac{\pi}{3}$.

Il suffit donc de calculer les coordonnées de chacun des ces trois points, (en utilisant la fonction `simil` pour celles de E, puis d'appeller récursivement la fonction à l'ordre $n-1$ pour chacun des segments $[AC]$, $[CE]$, $[ED]$ et $[DB]$. On concatène ensuite les listes obtenues, en tenant compte du fait qu'on peut ne pas tenir compte du premier élément de chaque liste (à part pour la première) qui est le dernier élément de la liste précédente.
Dans le cas $n = 0$, la liste L est laissée inchangée.
La fonction suivante réalise cette construction en faisant appel à la fonction `simil` pour déterminer les coordonnées de C, D et E.

```
def von_koch_cote(L,n):
    if n==0:
        return L
    else:
        A,B = L # unpack
        C = simil(A,B,1/3,0)
        D = simil(A,B,2/3,0)
        E = simil(C,D,1,ma.pi/3)
        # On appelle récursivement la fonction sur chaque segment
        # et on reconstitue la liste des sommets obtenus
        M = []
        for x,y,z in [(A,C,0), (C,E,1), (E,D,1), (D,B,1)]:
            M+=von_koch_cote([x,y],n-1)[z:]
        return M
```

où la coupe d'indice `z` permet d'éviter de renvoyer le premier point pour les segments 2 à 4 car leur premier coïncide avec le dernier point du segment précédent.

2. On peut utiliser par exemple comme sommets du triangle équilatéral le points $A(1;0)$ et les points B et C obtenus à partir de A par rotations successives de centre O et d'angle $-\frac{2\pi}{3}$ (attention à bien prendre un angle de rotation négatif pour que les nouveaux triangles soient ajoutés à l'extérieur du triangle initial). Il suffit ensuite d'appeler pour chaque côté la fonction de la question précédente pour obtenir la liste despoints L à relier.
On extrait ensuite les listes des coordonnées des points de L et on utilise le module `mathplotlib.pyplot` (renommé ici `plt`).

```
def von_koch(n):
    # Création des sommets du triangle
    O=[0,0]
    A=[1,0]
    B=simil(O,A,1,-2*ma.pi/3)
    C=simil(O,B,1,-2*ma.pi/3)
    # Liste des points à tracer pour chaque côté
    L=[]
    for x,y in [(A,B), (B,C), (C,A)]:
        L+=von_koch_cote([x,y],n)
    # Affichage
    X,Y=[e[0] for e in L],[e[1] for e in L]
    plt.figure()
    plt.plot(X,Y)
    plt.axis('equal')
    plt.show()
```

Exercice 7.9

1. Si $n = 0$, on retourne L. Sinon, on appelle récursivement la fonction à l'ordre $n-1$ sur l'image de L par la première similitude données par l'énoncée, puis on fait une rotation de la figure obtenue. Il faut prendre garde à inverser l'ordre des sommets de $L2$ pour bien parcourir les sommets de celle-ci en partant de l'extrémité de $L1$. Il est par ailleurs inutile de conserver le premier point de $L2$ qui coïncide avec le dernier de $L1$.

```
def points_dragon(L,n):
    if n==0:
        return L
    else :
        L1=(points_dragon(simil_poly(L[0],L,ma.sqrt(2)/2,ma.pi/4),n-1))
        L2=simil_poly(L1[-1],L1,1,ma.pi/2)
        L2.reverse()
        return L1+L2[1:]
```

2. Il reste à appeler la fonction précédente en partant des points $A(0;0)$ et $B(1;0)$ et afficher le résultat à l'aide du module `mathplotlib.pyplot`.

```
def dragon(n):
    L=[[0,0],[1,0]]
    L=points_dragon(L,n)
    X,Y=[e[0] for e in L],[e[1] for e in L]
    plt.figure()
    plt.plot(X,Y)
    plt.show()
```

4 Appels récursifs multiples

Exercice 7.10

Si $n = 0$ ou $n = 1$, il y a zéro découpe possible. Si $n = 2$ ou $n = 3$, une seule. Sinon, on considère le morceau le plus à droite. Sa longueur est 2 ou 3. Dans le premier cas, il y a $d(n-2)$ découpes possibles pour la partie à gauche, et dans le deuxième cas $d(n-3)$, donc $d(n) = d(n-2) + d(n-3)$.

```
def nb_decoupes(n):
    if n<=1:
        return 0
    elif n<=3:
        return 1
    else:
        return nb_decoupes(n-2)+nb_decoupes(n-3)
```

Exercice 7.11

```
def binom(n,k):
    if k>n:
        return 0
    elif k==0 or k==n:
        return 1
    else:
        return binom(n-1,k)+binom(n-1,k-1)
```