Introduction à la récursivité

Prérequis : pour comprendre ce cours, il faut avoir une expérience minimale de la programmation en Python et dominer le concept de récursivité.

1. Principe

Un programme récursif est un programme qui s'appelle lui-même. Cela permet (parfois) de racourcir le code de manière importante, mais surtout de programmer plus naturellement car certains algorithmes sont intrinsèquement récursifs - les tours de Hanoï, que l'on verra plus loin, en sont un bon exemple. Toutefois, il est bon de savoir, même si le mode opératoire sous-jacent est hors programme, que tout algorithme récursif peut être dérécursifié.

Par ailleurs, la récursivité n'est pas sans danger : les appels récursifs sont par nature conditionnels et peuvent donc, si l'on s'y prend mal, ne pas se terminer. Plus insidieux, le programme peut calculer de multiples fois la même chose, ce qui représente une inflation de la complexité en temps. Ce défaut s'accompagne en général d'une inflation concomitante de la complexité en mémoire car un programme récursif stocke les calculs intermédiaires jusqu'à la fin - sur ce point, on verra la génération de la suite de Fibonacci, qui en est un exemple typique.

Deux exemples simples de comparaison entre programmation itérative et programmation récursive :

`def factorielle(n):`
`  f = 1`
`  for k in range(1,n+1):`
`  f *= k`
`  return f`
	`def factorielle_rec(n):`
	`  if n <= 1:`
	`  return 1`
	`  else:`
	`  return n*factorielle_rec(n-1)`

`def rev1(C):`
`  K = ''`
`  for c in C:`
`  K = c + K`
`  return K`
	`def rev2(C):`
	`  if C == '':`
	`  return C`
	`  else:`
	`  return (rev2(C[1:])) + C[0]`

Complément. Pour comparer la vitesse d'exécution des formes récursive et itérative, on peut utiliser la bibliothèque `time`. Dans le cas du deuxième exemple, cela donnerait (sans chercher à moyenner) :

```
import time as tm
def comp_rev(C):
    t = tm.time()
    rev1(C)
    u = tm.time()
    rev2(C)
    v = tm.time()
    return u-t,v-u
M = [chr(65+i) for i in range(26)]
C = ''.join(M)
print(comp_rev(C))
```

On voit que les fonctions récursives comportent un « point d'arrêt » indiquant la fin des appels et la sortie du programme. Dans l'ensemble, la structure est la suivante :

recursive (paramètres)
  si test d'arrêt
    retourner une valeur initiale
  sinon
    instructions ;
    recursive (paramètres changés).

La liste des paramètres d'appel de la fonction récursive fait office de fonction de terminaison (variant de boucle *while*). Les résultats des appels sont stockés au fur et à mesure dans une pile dite *pile d'exécution*, qui est désempilée à l'arrêt du programme. Ce qui illustre l'étymologie de la notion, qui vient du verbe latin *recurrere* signifiant « revenir en arrière ». En Python, le nombre d'appels est limité par le système à 1000 pour éviter les explosions en vol. Si l'on veut augmenter ce nombre (on l'utilisera dans le TP sur les tris), on tape ce qui suit - évidemment, la valeur `n=1000` n'a pas été chosie au hasard et il est déconseillé de s'en éloigner trop fortement :

```
import sys
sys.setrecursionlimit(n)
```

Voici un autre exemple : dans le TP d'analyse de Fourier d'un signal en créneau, on a utilisé une fonction `separetab` pour récupérer les coefficients de Fourier du signal dans un fichier texte. La $n$ième ligne du fichier contenait les coefficients a_n et b_n, séparés par une tabulation. Voici deux versions de cette récupération différentes de celle proposée en TP.

```
fichier = open('/chemin/.../an-et-bn.txt','r')
coeffs = fichier.read()
fichier.close()
```

```
lignes = coeffs.split('\n')
a = []; b = []
for k in lignes:
    couple = k.split('\t')
    if k != '':
        a.append(float(couple[0]))
        b.append(float(couple[1]))
```

```
cliste = list(coeffs)
aa = []; bb = []
def recup(L):
    if len(L)<=1:
        return (aa,bb)
    else:
        i = L.index('\t'); j = L.index('\n')
        aa.append(float(''.join(L[:i])))
        bb.append(float(''.join(L[i+1:j])))
    return recup(L[j+1:])
```

Exercice 1. Écrire une version récursive du programme suivant :

```
def produit(a,b):
    p=0
    for k in range(b):
        p+=a
    return p
```

Exercice 2. Écrire un programme récursif qui calcule le pgcd de deux entiers strictement positifs par soustractions successives.

Exercice 3. Écrire une fonction récursive testant si une chaîne de caractère donnée en entrée est un palindrome.

Exercice 4. Écrire une version récursive de l'algorithme d'exponentiation rapide, dont on rappelle ci-dessous une version itérative.

```
def Puissance(a,n):
    A = a; N = n; R = 1
    while N > 0 :
        if N%2 == 0 :
            A = A**2; N /= 2
        else:
            R*= A; N -= 1
    return R
```

2. Quelques problèmes récursifs historiques

Exercice 5. La légende dit que dans un temple de Hanoï, à l'origine du monde, 64 disques de diamètre croissant étaient empilés sur un taquet. Deux autres taquets étaient disponibles, utilisés pour déplacer les disques, la condition étant qu'un disque d'un certain diamètre ne peut pas être placé au dessus d'un disque de diamètre inférieur. Les disques sont donc toujours empilés dans un certain ordre, les plus grands figurant toujours en bas du taquet. La légende dit aussi que des moines sont en train de déplacer les 64 disques du premier vers le troisième taquet, au rythme de un par seconde et que quand ils auront terminé, ce sera la fin du monde.

Modélisation : les disques sont numérotés selon leur diamètre, de 1 à n (cela ne change pas grand chose de prendre n ou 64). Les taquets sont numérotés de A à C.

1. Résoudre le problème pour $n = 2$, puis $n = 3$.
2. Expliquer comment, si l'on sait déplacer $n - 1$ disques d'un taquet à l'autre, on peut en déplacer n.
3. Écrire un programme récursif donnant les déplacements pour déplacer une tour de hauteur n du taquet A au taquet C en utilisant le taquet B. Les déplacements seront listés sous la forme « déplacement du disque i de T vers T' » avec $T \in \{A, B, C\}$ et l'on définira une fonction `Hanoi(U,V,W,n)` pour déplacer n disques du disque U au disque W en passant par le disque V.

Exercice 6. Compléter la fonction suivante de manière récursive pour qu'elle convertisse un nombre en chiffres romain en chiffres arabes.

```
def romain(C):
    R = 'IVXLCDM'
    N = [1,5,10,50,100,500,1000]
    if len(C) == 1:
        return N[R.index(C)]
```

Exercice 7. L'Egypte antique connaissait partiellement l'usage des fractions, sous la forme des inverses d'entiers. On peut démontrer que tout nombre $x \in \mathbb{Q}_+^*$ peut se décomposer de manière unique en une somme d'inverses distincts. Autrement dit :

$$\forall x = \frac{p}{q} \in]0,1[\mathbb{Q}_+^*,\ \exists ! n \in \mathbb{N}^*,\ \exists !\, b_1 > \cdots > b_n > 0 \colon \frac{p}{q} = \sum_{j=1}^{n} \frac{1}{b_j}.$$

Écrire un programme récursif calculant $(b_1, \ldots, b_n)$ (on commencera par trouver une formule donnant b_1).