RECURSIVITE - initiation

La programmation des fonctions personnalisées a donné lieu à l'essor d'une logique un peu particulière, adaptée en particulier au traitement de certains problèmes mathématiques: la programmation récursive.

Pour vous expliquer de quoi il retourne, nous allons reprendre un exemple: le calcul d'une factorielle
Rappelez-vous : la formule de calcul de la factorielle d'un nombre n s'écrit :

N ! = 1 x 2 x 3 x … x n
Exemple : 5 ! = 1 x 2 x 3 x 4 x 5 = 120

Nous pouvons programmer cela avec une boucle Pour :

POUR N allant de 1 à n
    FactN = FactN * N
FIN POUR

Mais une autre manière de voir les choses, ni plus juste, ni moins juste, serait de dire que quel que

n ! = n x (n-1)!
Exemple : 5 ! = 5 x 4 ! ; 4 ! = 4 x 3 ! ; 3 ! = 3 x 2 ! ; 2 ! = 2 x 1 !

La factorielle d'un nombre, c'est ce nombre multiplié par la factorielle du nombre précédent. Encore une fois, c'est une manière ni plus juste ni moins juste de présenter les choses ; c'est simplement une manière différente.

Si l'on doit programmer cela, on peut alors imaginer une fonction Fact, chargée de calculer la factorielle. Cette fonction effectue la multiplication du nombre passé en argument par la factorielle du nombre précédent. Et cette factorielle du nombre précédent va bien entendu être elle-même calculée par la fonction Fact.
Autrement dit, on va créer une fonction qui pour fournir son résultat, **va s'appeler elle-même un certain nombre de fois**. C'est cela, la récursivité.

Toutefois, il nous manque une chose pour finir : quand ces auto-appels de la fonction Fact vont-ils s'arrêter ? On s'arrête quand on arrive au nombre 1, pour lequel la factorielle est par définition 1.
Cela produit l'écriture suivante, un peu déconcertante certes, mais parfois très pratique :

Fonction *Fact (N en Numérique)*
Si N <= 1
    Renvoyer 1
Sinon
    Renvoyer *Fact(N-1) * N*
Finsi
Fin Fonction

En Python :
```
def factorial(N):
  if(N <= 1):
    return 1
  else:
    return  factorial(n-1) * N
```

Le processus récursif remplace en quelque sorte la boucle, c'est-à-dire un processus itératif.
Vous remarquerez aussi qu'on traite le problème à l'envers : on part du nombre, et on remonte à rebours jusqu'à 1 pour pouvoir calculer la factorielle. Cet effet de rebours est caractéristique de la programmation récursive.

Deux remarques fondamentales :

- la programmation récursive, pour traiter certains problèmes, est **très économique pour le programmeur** ; elle permet de faire les choses correctement, en très peu d'instructions.
- **tout problème formulé en termes récursifs peut également être formulé en termes itératifs !** Donc, si la programmation récursive peut faciliter la vie du programmeur, elle n'est jamais indispensable.

Comment reconnaitre une fonction récursive :

Rien de plus simple : si vous voyez une fonction qui s'appelle elle-même, bingo, c'est une fonction récursive.

Exemple ci contre : *fibonacci* s'appelle elle-même ! C'est une fonction récursive

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

Pourquoi écrire une fonction récursive ?

A quels problèmes les fonctions récursives sont-elles adaptées ?

Les fonctions récursives sont des fonctions qui s'appellent elles-mêmes. Elles doivent donc résoudre des problèmes qui "s'appellent eux-mêmes". Dans certains d'entre eux, la solution du problème général demande la résolution de plusieurs sous-problèmes particuliers, qui sont semblables au premier problème. Par exemple, on peut dire que pour résoudre le problème « combien vaut la factorielle de 4 ? » , il faut résoudre le problème « combien vaut la factorielle de 3 ? » .

Autre exemple : inversion d'une chaîne de caractère

Nous allons étudier une fonction qui prend une chaîne en entrée et renvoie l'inverse de cette chaîne (« bonjour » doit devenir « ruojnob »

La première étape est de définir notre scénario de base, qui vérifiera si la chaîne est égale à 0 et, si oui, retourne la chaîne elle-même.

La deuxième étape est d'appeler de manière récursif la fonction d'inversion afin d'extraire le premier caractère et ensuite l'ajouter à la fin de la chaîne.

La fonction (qu'on nommera ***rev***) qui en résulte est :

FONCTION rev (*mot* en chaine de caractères)
SI longueur (*mot*) == 1
RENVOYER *mot*
SINON
RENVOYER rev (*mot* sauf le premier caractère) + premier caractère de mot
FIN SI
FIN FONCTION

En Python :

```
def reverse(a):
    if len(a) == 0:
        return a
    else:
        return reverse(a[1:]) + a[0]

print(reverse("NSI"))
# result
#ISN
```

Aide (rappel) :

-Une chaine de caractère peut se manipuler comme une liste.

-Si on considère une liste *liste = [0,2,3,56,22]* :

```
print (liste[2:]) # imprime tout sauf les 2 premiers ; [3, 56, 22]
print (liste[1:]) # imprime tout sauf le premier ; [2,3, 56, 22]
print (liste[:2])   # imprime que les 2 premiers ; [0, 2]
print (liste[:2] + liste[:2]) # concatène les listes ; [0, 2, 0, 2]
print (liste[:]) # imprime tout
```

Exercice : écrire l'algorithme d'une fonction récursive ***som*** qui renvoie la somme des éléments d'une liste puis l'implanter en Python.

Algorithme :

En Python :