Mathématiques spéciales le 8 Septembre 2017

# TP n°1 - Récursivité - Correction

Dans ce TP, nous allons mettre en pratique la notion de récursivité sur des exemples. On rappelle la définition de fonction récursive.

**Définition 1.** *Fonction récursive*

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

On peut bien-sûr faire le parallèle entre les fonctions récursives en Informatique et les suites récurrentes en Mathématiques.

On commence par traiter un exemple classique et fondamental d'utilisation de la récursivité :

## 1. La factorielle

On rappelle la notation, pour $n \in \mathbb{N}^*$,

$$n! = \prod_{i=1}^{n} i = 1 \times 2 \times ... \times n \text{ et } 0! = 1,$$

et on appelle **factorielle** $n$, la quantité $n!$.

**Question 1.**

Comment calculer numériquement de manière efficace la valeur de $n!$ pour un entier donné ?

Allons-y pas à pas :

### a. Méthode itérative

**Exercice 1.**

Écrire une fonction `factoIt(n)` qui retourne la factorielle de son argument (où `n` est censé être un entier naturel). On utilisera un méthode itérative (boucle **`for`** ou **`while`**).
Tester sur quelques valeurs $(1, 2, 3, 4, 5$ par exemple) pour vérifier qu'on obtient bien le bon résultat.
Chronométrer le temps de calcul de `factoIt(900)` et `factoIt(1000)`.

On rappelle que pour chronométrer une fonction (disons `maFonction`) en python, on importe le module time et on utilise la méthode `time.time()` qui renvoie le nombre de secondes écoulé depuis le 1er janvier 1970 à minuit.

```python
import time
t0=time.time() #temps avant l'exécution de la fonction
maFonction()
t1=time.time() #temps après l'exécution de la fonction
print(t1-t0,'secondes de temps de calcul') #on affiche la différence des deux
    temps.
```

Correction.

Correction avec un boucle `for`

```python
def factoIt(n):
    """Calcul de factorielle n où n est un entier naturel. Fonction itérative (
        boucle for)"""
    k=1
    for i in range(1,n+1):
        k=k*i
    return k
```

ou

Correction avec un boucle `while`

```python
def factoIt(n):
    """Calcul de factorielle n où n est un entier naturel. Fonction itérative (
        boucle while)"""
    k,i=1,1
    while i <=5
        k*=i
        i+=1
    return k
```

### b. Méthode récursive naïve

**Exercice 2.**

Écrire une fonction `factoRec(n)` qui retourne la factorielle de son argument (où `n` est censé être un entier naturel). On utilisera un méthode récursive en utilisant l'initialisation $0! = 1$ et la relation $n! = n \times (n-1)!$ si $n \neq 0$.
Tester sur quelques valeurs $(1, 2, 3, 4, 5$ par exemple) pour vérifier qu'on obtient bien le bon résultat.
Chronométrer le temps de calcul de `factoRec(900)`.

Que se passe-t-il quand on veut calculer`factoIt(1000)` ?
En fait, Python impose une limitation du nombre d'appels récursifs (pour éviter les temps de calcul trop longs). Par défaut, la limitation est de 1000 (ou un peu moins). Pour modifier cette limite, on peut utiliser le module `sys` :

Modification du nombre maximal d'appels récursifs

```
import sys
sys.setrecursionlimit(100000) #on fixe la limite du nombre d'appels récursifs à
    100000
```

Correction.

Correction factorielle récursive naïve

```
def factoRec(n):
    """Calcul de factorielle n où n est un entier naturel. Fonction récursive"""
    if n==0:
        return 1
    return n*factoRec(n-1)
```

### c. Méthode récursive améliorée

On considère la fonction $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ définie, pour $a, b \in \mathbb{N}$, par :

$$\pi(a,b) = \prod_{k=a}^{b-1} k.$$

Montrer les égalités suivantes :

$$\pi(a,b) = \begin{cases} 1 & \text{si } b \leq a \\ a & \text{si } b = a+1 \\ \pi(a,c) \times \pi(c,b) & \text{pour } a < c < b \text{ sinon.} \end{cases}$$

**Exercice 3.**

En utilisant les trois relations précédentes dont la dernière relation avec $c = E(\frac{a+b}{2})$, écrire une fonction récursive `prodRange(a,b)` qui calcule $\pi(a,b)$.

En déduire une fonction `facto(n)` qui retourne la factorielle de son argument.
Chronométrer le temps de calcul de `facto(900)` et de `facto(1000)`.

Correction.

Correction calcul de $\pi(a,b)$

```
def prodRange(a,b):
    """Calcul du produit des entiers entre a et b-1 où a,b sont des entiers
        naturels"""
    if b<=a:
        return 1
    if b==a+1:
        return a
    c=(a+b)//2
    return prodRange(a,c)*prodRange(c,b)
```

Correction factorielle récursive maline

```
def facto(n):
    """Calcul de factorielle n où n est un entier naturel. Utilise la fonction
        récursive prodRange"""
    return prodRange(1,n+1)
```

### d. Dernier Test pour la route

**Exercice 4.**

Chronométrer `facto(150000)` et `factoIt(150000)`. Qu'en pensez-vous ?

On expliquera précisément cette différence quand on reverra la notion de complexité.

### e. Petit exercice "amusant" !

Voici deux petits exercices pour les curieux : d'abord un exercice de Maths, puis l'application en Info (mais là il n'y a pas grand chose à faire !).

**Exercice 5.**

Pour $n \in \mathbb{N}$, déterminer une formule donnant le nombre exact de 0 à la fin du nombre $n!$ (en fonction de $n$ bien sûr).

Écrire une fonction `nombreZeroFacto(n)` qui renvoie le nombre de 0 à la fin de $n!$.

Correction.

On peut conclure directement en utilisant la formule de Legendre mais on ne la connaît pas ! Alors, on se débrouille ! Ce qui va suivre n'est pas une démonstration rigoureuse, mais une explication heuristique qui permet, sans s'encombrer de formalisme, de trouver le résultat.

On remarque que le nombre de 0 à la fin de $n!$ correspond à la plus grande puissance de 10 qui divise $n!$.

Si $10^i$ est plus petit que $n$, alors $10^i$ apparaît dans la formule de $n! = n \times (n-1) \times ... \times 1$. Si c'est le cas, cela a pour effet d'ajouter $i$ zéros à la fin de $n!$.

Si on va un peu plus loin, on remarque que pour chaque multiple de 5 (disons $5k$) plus petit que $n$, il existe un multiple de 2 inférieur à ce multiple de 5 (facile : $2k$) et comme $n!$ "contient" le produit de ces deux multiples et que $2 \times 5 = 10$, ce qui a pour effet de rajouter un zéro à la fin de $n!$

Allons plus loin : pour chaque multiple de $5^2$ inférieur à $n$, il existe un multiple de $2^2$ inférieur à ce multiple, et comme $5^2 \times 2^2 = 10^2$ ; donc 2 zéros de plus à la fin de $n!$. Ah mais non ! car $5^2$ faisait déjà partie de nos multiples de 5, donc on avait déjà compté un des deux 0 juste avant. Donc en fait chaque multiple de $5^2$ rajoute un zéro à la fin de $n!$.

Etc... Etc... jusqu'à la plus grande puissance de 5 plus petite que $n$ : pour chaque multiple de $5^i$ inférieur à $n$, il existe un multiple de $2^i$ inférieur à ce multiple, et comme $5^i \times 2^i = 10^i$ ; un zéros de plus à la fin de $n!$ (et pas $i$ car on a on en a déjà rajouté $i-1$ au tours précédents).

Notre problème revient donc à déterminer, pour chaque $k = 1, 2, ...$, le nombre de multiple de $5^k$ inférieurs à $n$ et de sommer sur $k$ tous ces résultats !
Première chose, si $5^k > n$ alors il y en a 0 !
Maintenant, si $5^k \leq n$, et bien il faut savoir "combien de fois il y a $5^k$ dans $n$" : c'est-à-dire le quotient de la division euclidienne de $n$ par $5^k$ i.e. $E(\frac{n}{5^k})$ où $E$ est la fonction partie entière (en python : `n//5**i` ).

Au final, on obtient donc, en remarquant que $E(\frac{n}{5^k}) = 0$ si $5^k > n$ :

nombre de zéros à la fin de $n!$ égal à

$$\sum_{i=1}^{+\infty} E(\frac{n}{5^k}).$$

D'où en python :

```python
def nombreZeroFacto(n):
    """Nombre de zeros dans factorielle n où n est un entier positif"""
    S,i=0,1
    while 5**i <= n:
        S+=n//5**i
        i+=1
    return S
```