TP n°1 - Récursivité

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.
```

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

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

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)`.

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 $n!$.