TD Informatique TSI2

Recursion and sorting recursion

EXERCICE 01

1. Écrire une fonction `somme(n)` qui calcule, au moyen d'une boucle, la somme $\sum_{k=0}^{n} k$.

   C'est la méthode impérative.

   Indication : *on utilise une variable* `s` *initialisée à* 0*, à laquelle on ajoute successivement* 0*, puis* 1*, … et enfin* n*.*

2. Écrire une fonction `sommeRec(n)` qui calcule la somme $\sum_{k=0}^{n} k$ de façon récursive.

   Indication : *il suffit de remarquer que pour l'initialisation,* `sommeRec(0)=0` *et l'hérédité donne* `sommeRec(n)= n + sommeRec(n-1)`

EXERCICE 02

1. Écrire une fonction `fact(n)` qui renvoie au moyen d'une boucle $n!$ C'est la méthode impérative.

   Indication : *Pour la version impérative, il suffit de calculer le produit de tous les entiers jusqu'à* n*. On utilise donc une variable* p *initialisée à* 1 *que l'on multiplie successivement par* 1, 2, *…*
2. Écrire une fonction `factRec(n)` qui calcule la factorielle $n!$ de façon récursive.

   Indication : *Pour la version récursive, on sait que* $0! = 1$ *et que* $n! = n.(n-1)!$ *pour tout* $n \geqslant 1$.

EXERCICE 03

1. Écrire une fonction `puissance(x,n)` qui calcule, au moyen d'une boucle x^n avec $n \geqslant 0$.

   Indication : *Il s'agit de faire un produit* $x \times x \times ... \times x$ *(avec* n *facteurs).*
   On initialise à `p=1` *puis on boucle avec* `p *= x`
2. Écrire une fonction `puissanceRec(x,n)` qui calcule x^n de façon récursive (sur le n).

   Indication : *Pour la version récursive, on utilise le fait que* $x^0 = 1$ *et* $x^n = x \times x^{n-1}$ *pour* $n \geqslant 1$.

EXERCICE 04

On suppose que l'on ne peut faire que des additions et des soustractions d'entiers (pas de multiplication, ni de division).

1. Écrire une version impérative `quotient2(n)` et une version récursive `quotient2Rec(n)` de la fonction qui calcule le quotient d'un entier (positif ou nul) dans la division par 2.

   Indication : *Pour la version impérative, partant d'un entier* n*, on utilise une variable qui va compter combien de paquets de* 2 *on peut retrancher à* n *avant de tomber sur un entier inférieur ou égal à* 1*. Pour cela, chaque fois que l'on retire* 2 *à l'entier, on ajoute* 1 *au compteur.*
   Par exemple : $n = 11$, $q = 0$, $r = 11$.
   Puis $11 > 1$ *donc* $q = 1$ *et* $r = 11 - 2 = 9$.
   Puis $9 > 1$ *et on continue,* $q = 2$ *et* $r = 9 - 2 = 7$.
   Puis $7 > 1$ *et on continue,* $q = 3$ *et* $r = 7 - 2 = 5$.
   Puis $5 > 1$ *et* $q = 4$ *et* $r = 5 - 2 = 3$.
   Puis $3 > 1$ *et* $q = 5$ *et* $r = 3 - 2 = 1$*. On s'arrête. On retourne* 5*. Donc dans la procédure, on initialise* `q=0` *et* `r=n` *et on boucle avec* `while r >1`

   Pour la version récursive, les cas de base sont $n = 0$ *et* $n = 1$*, pour lesquels le quotient est nul. Dans les autres cas, le quotient de* n *est égal au quotient de* $n - 2$ *augmenté d'une unité.*
2. Même question pour le reste avec une version impérative nommée `reste2(n)` et une version récursive nommée `reste2Rec(n)`

   Indication : *il suffit de renvoyer* `r` *au lieu de* `q` *en sortie de boucle pour la version impérative.*

EXERCICE 05

1. Écrire une fonction `nbChiffresRec(n)` qui calcule, de façon récursive, le nombre de chiffres de l'écriture décimal d'un entier n (on convient que 0 s'écrit avec un seul chiffre).

   Indication : *Si $n \leqslant 9$, le nombre n s'écrit avec un seul chiffre.*
   Sinon, il suffit de savoir avec combien de chiffres s'écrit le nombre n privé de son dernier chiffre (i.e le quotient dans la division par 10) et d'ajouter 1. La commande `nbChiffresRec(n // 10)` donne le nombre de chiffres de n privé de son dernier chiffre.
2. Écrire une fonction `nbChiffres(n)` qui calcule, de façon impérative, le nombre de chiffres de l'écriture décimal d'un entier n (on convient que 0 s'écrit avec un seul chiffre).

   Indication : *Pour la version impérative, on manipule une variable `a` (initialisée à n) et un compteur `c` initialement égal à 1. Si $a \leqslant 9$, le nombre s'écrit avec un chiffre, qui est la valeur du compteur. Puis dans une boucle `while`, on remplace a par son quotient dans la division par 10 et on ajoute 1 au compteur. Et ceci jusqu'à tomber sur le cas $a \leqslant 9$: la valeur du compteur indique alors combien l'entier comporte de chiffres. On utilisera une boucle `while` avec la condition `a>9`*

EXERCICE 06

On définit la fonction récursive suivante :

```
In [1]: def f(a,b):
   ...:     if b == 0:
   ...:         return 0
   ...:     else :
   ...:         return a+f(a,b-1)
```

1. Calculer à la main la valeur `f(3,5)` en détaillant les appels récursifs.
2. Pour quelles valeurs des arguments la fonction se termine-t-elle ?
3. Que calcule la fonction ? Le démontrer (par récurrence sur b).

■ Fonctions définies sur des listes

EXERCICE 07

On veut écrire une fonction récursive qui calcule le plus grand élément d'une liste non vide.

1. Écrire une fonction récursive `plusGrandEntre` qui calcule le plus grand élément entre deux positions `deb` et `fin`

   Indication : *Si `fin = deb` la plage ne comporte qu'un élément, qui est le plus grand ; sinon, on calcule récursivement le plus grand élément de la plage $l[deb..fin-1]$, que l'on compare à `l[fin]`*
   On a le droit d'utiliser la fonction prédefinie `max`
2. En déduire le plus grand élément d'une liste non vide en usant de `plusGrandEntre`

EXERCICE 08

1. Écrire une fonction récursive `chercheEntre` qui cherche un élément entre les indices `deb` et `fin` dans une liste **triée**.

   Indication : *Si la plage est vide, l'élément est absent.*
   On le traduira par : `if fin < deb` alors `return False`
   Sinon, on note `m` l'indice median `m = (deb + fin) // 2`
   On compare `x` et `l[m]` Si `x` vaut `l[m]` alors on retourne `True` et sinon, si `x < l[m]` alors il suffit de chercher dans la plage $l[deb..m-1]$ et si par contre `x>l[m]` alors il suffit de le chercher dans la plage $l[m+1..fin]$.

2. Quelle est la complexité de cette fonction dans le pire des cas ? On s'intéressera au nombre de comparaisons effectuées.

   Indication : *Le pire des cas est celui où l'élément est absent. Si la longueur initiale de la plage est* n*, elle est au plus égale à* $n/2^p$ *au bout de* p *appels récursifs. L'algorithme se termine quand cette longueur est strictement inférieure à* 1*. Majorer alors* p *avec une fonction de* n*.*

EXERCICE 09

1. Écrire une fonction récursive `chiffresFortVersFaible(n)` qui renvoie la liste des chiffres de l'écriture en base 10 d'un nombre `n`, chiffre de poids fort en tête (par exemple pour le nombre 547, la fonction doit renvoyer [5,4,7]). On conviendra de représenter 0 par la liste vide.

   Indication : *On considérera un seul cas terminal, celui où* $n = 0$ *pour lequel on retourne* `[ ]` *la liste vide. Dans les autres cas, si* q *est le nombre (éventuellement nul) obtenu en supprimant le dernier chiffre de* n *alors* q *est le quotient dans la division de* n *par* 10*, c'est-à-dire en Python* `n // 10` *Pour connaître la liste des chiffres de* n*, il suffit de rentrer récursivement dans la liste* `L` *les chiffres de* q *et d'ajouter le dernier chiffre de* n *(i.e. son reste dans la division par* 10*) en queue. On écrit alors* `L.append(n % 10)`

2. On considère le programme suivant, où `L` est une liste de chiffres et `dernier` un indice entier.

```
In [1]: def f(L,dernier):
   ...:     if dernier < 0 :
   ...:         return 0
   ...:     else :
   ...:         q = f(L,dernier-1)
   ...:         return 10 * q + L[dernier]
```

   Calculer à la main `f([4,5,8],0)` puis `f([4,5,8],2)`

   Interpréter alors `f` par rapport à `chiffresFortVersFaible`

■ Algorithmes de tri récursifs

EXERCICE 10

On reprend la procédure `partition(l,deb,fin)` du cours en remplaçant `return m` par `return (m,l)`

1. Déterminer à la main le résultat de `partition([3,-1,2,3,5,4,8,0,4,2,5,-5],8,11)` puis retrouver ce résultat avec Python.
2. Faire de même avec `partition([3,-1,2,3,5,4,8,0,4,2,5,-5],3,8)`
3. Faire de même avec `partition([3,-1,2,3,5,4,8,0,4,2,5,-5],0,11)`

EXERCICE 11

On reprend la procédure `fusion(l,deb,mil,fin)` du cours en y ajoutant `return l`
Faire à la main `fusion([-2,2,3,7,9,10,1,8,1,4],1,5,7)` puis retrouver le résultat avec Python.