# Programmation fonctionnelle

# Répétition 1

15 février 2007

## 1 Avant propos

Informations pratiques :

- François Van Lishout, `vanlishout@montefiore.ulg.ac.be`, 04/3662619, bureau I82a.
- http ://www.montefiore.ulg.ac.be/∼vanlishout
- Ce cours ne nécessite pas une énorme quantité de travail, *mais un travail régulier est indispensable*.
- Vu le nombre d'étudiants, il est impensable de travailler sur les machines au cours des répétitions. Ceci dit, il est *indispensable* de travailler sur les machines en dehors des répétitions. Une bonne préparation implique un travail sur papier, suivit du travail sur la machine.
- Un cdrom est disponible à l'AEES. Ce cdrom contient plusieurs versions installables du MIT-Scheme (pour windows et pour linux) ainsi qu'un document très intéressant : *Revised5 Report on the Algorithmic Language Scheme*.

Quelques points délicats :

- Attention à ne pas comfondre `cons`, `list`, et `append`.
- Savoir spécifier est aussi important que savoir programmer. Les énoncés des exercices sont de bons exemples de spécifications.
- Les répétitions contiennent de nombreux exercices d'examen (marqués d'une étoile $\star$) et d'interrogation (marqués d'un trèfle ♣).
- Il faut privilégier les solutions simples et courtes.
- Les solutions doivent avoir une complexité acceptable.

# 2 Exercices

## Evaluations simples

**1.** Donner le résultat de l'évaluation des expressions suivantes

```
3
#t
(+ 1 2)
(/ 2 3)
(+ (* 3 4) 10)
(+ 3 4)
(* 3 (- 12 5))
(+ (+ 2 3) (+ 4 5))
(- (+ 5 8) (+ 2 4))
(define a 4)
a
(quote a)
'a
(define b a)
b
(define a 6)
a
b
(define c (quote a))
c
(define d #t)
(define Robert 'Bob)
Robert
(car '(a b))
(car (quote (a b)))
(cdr '(a b))
(car (cdr '(a b)))
(cdr (cdr '(a b)))
(cons 'a '())
(cons 'a '(b))
(cons '() '())
(cons '(a) '(b))
(list '(a) '(b))
(append '(a) '(b))
(cons 'a (cons 'b (cons 'c '())))
(car (cons 'a '()))
(cdr (cons 'a '()))
(cons a (a b))
```

```
(cons (car '(a b c)) (cons
                       (car (cdr '(a b c)))
                       (cons (car (cdr (cdr '(a b c)))) '())))
(cadr '(a b c d))
(cadar '((a b) (c d) (e f)))
```

## Deux exercices plus complexes à faire chez soi

**2.** La variable `phrase` est définie comme suit

```
(define phrase
  '(((e) x (e r (c (i c e) c (o m (p l (i q))) u e)))))
```

Extraire le `m` au moyen de `car`, `cdr`, `cadr`, ...

**3.** Construire la liste `(s c h e m e)` au seul moyen de la liste `ls` définie comme suit

```
(define ls '(C E H M S))
```

et des fonctions `cons`, `car`, `cdr`, `cadr`, ...

## Premières formes

**4.** Calculer en une seule forme SCHEME le nombre de secondes dans une année (non bissextile).

**5.** Ecrire une forme qui rend `un` si `x` est égal à 1,...`cinq` si `x` est égal à 5 et `inconnu` sinon.

## Récursion sur les nombres

**6.** Ecrire une fonction $sumOfIntegers$ prenant en argument un naturel $n$ et calculant la somme des naturels inférieurs ou égaux à $n$.

**7.** Définir la fonction `mod` qui renvoie le modulo de deux nombres.

PS : la fonction *modulo* est prédéfinie, nous la redéfinissons sous un autre nom à titre d'exercice.

**8.** Définir une fonction `pgcd` qui renvoie le plus grand commun diviseur de deux nombres.

**9.** Ecrire une fonction prenant en argument deux nombres $x$ et $n$ et renvoyant $x^n$.

PS : la fonction *expt* est prédéfinie, nous la redéfinissons sous un autre nom à titre d'exercice.

## Récursion sur les listes

**10.** Définir la fonction *sumList* qui calcule la somme des éléments d'une liste de nombres.

**11.♣** (24/02/1997, Q3) Ecrire une fonction `removeFirst` à deux arguments `l` et `n` dont les valeurs sont respectivement une liste et un nombre entier, qui retourne la liste privée de la première occurence de `n`. Ainsi on a
`(removeFirst '(2 7 1 7 3 1) 1)` ⇒ `(2 7 7 3 1)`

## Spécification

**12.♣** Spécifier la fonction suivante

```
(define xxx
  (lambda (u)
    (if (null? u)
        0
        (if (> (car u) 0)
            (+ (car u) (xxx (cdr u)))
            (xxx (cdr u))))))
```

## Exercices proposés

**13.** Définir la fonction `min` qui renvoie le minimum de la liste non vide de nombres donnée en argument.

**14.♣** (3/3/1999, Q4) Ecrire une fonction `big` qui prend comme arguments un nombre entier `n` et une liste `l` de nombres entiers, telle que `(big n l)` est la liste des éléments de `l` plus grands que `n`.
`(big 5 '(7 -3 6 1 -2 5 6))` ⇒ `(7 6 6)`