```
;;;;
;;;; Corrigé.stk        -- Corrigés de la feuille 4
;;;;
;;;;           Author: Erick Gallesio [eg@unice.fr]
;;;;    Creation date: 28-Feb-2000 09:06 (eg)
;;;; Last file update: 28-May-2003 11:47 (eg)
;;;;

;=============================================================================
;                                      EXO 1
;=============================================================================

(define (remove-if predicate? l)
  (cond
     ((null? l)              '())
     ((predicate? (car l))  (remove-if predicate? (cdr l)))
     (else                  (cons (car l) (remove-if predicate? (cdr l))))))

;;;   STk> (remove-if number? '(a 1 (1) 2 (x y) ()) )
;;;   (a (1) (x y) ())
;;;   STk> (remove-if (lambda (x)(< x 5)) '(6 2 9 3 7 1) )
;;;   (6 9 7)
;;;   STk> (remove-if null? '(a () 1 () (())))
;;;   (a 1 (()))

(define (count-if predicate? l)
  (cond
     ((null? l)              0)
     ((predicate? (car l))  (+ 1 (count-if predicate? (cdr l))))
     (else                  (count-if predicate? (cdr l)))))

;;;   STk> (count-if symbol? '(a (a) () c (x y) b) )
;;;   3
;;;   STk> (count-if (lambda (x)(> x 0)) '(-1 2 0 3 -5 -10 7) )
;;;   3


(define (find-if predicate? l)
  (cond
     ((null? l)              #f)
     ((predicate? (car l))  (car l))
     (else                  (find-if predicate? (cdr l)))))

;;;    STk> (find-if list? '(a 1 (x y) () (a b)) )
;;;    (x y)
;;;    STk> (find-if (lambda (x)(> x 10)) '(2 5 12 1 15) )
;;;    12

(define (remove-if-not predicate? l)
  (remove-if (lambda (x) (not (predicate? x))) l))

;;;   STk> (remove-if-not (lambda (x)(< x 5)) '(6 2 9 3 7 1) )
;;;   (2 3 1)
;;;   STk> (remove-if-not null? '(a () 1 () (())) )
;;;   (() ())

;=============================================================================
;                                      EXO 2
;=============================================================================

(define (depth l)
  (if (pair? l)
```



```
      (+ 1 (apply max (map depth l)))
      0))

;; Autre écriture (bof)
(define (depth l)
  (apply max (map (lambda (x)
                    (if (pair? x) (+ 1 (depth x)) 1))
                  l)))

(define (flat l)
  (cond
     ((null? l) l)
     ((pair? l) (apply append (map flat l)))
     (else      (list l))))

;=============================================================================
;                                      EXO 3
;=============================================================================

;;
;; Un map avec des fonctions unaires
;;
(define (simple-map fn l)
  (if (null? l)
      '()
      (cons (fn (car l)) (simple-map fn (cdr l)))))

;;
;; Un map simplifié (toutes les listes sont de longueur égale)
;;
(define (amap fn . l)

  (define (my-map fn l)
    (if (null? (car l))
        ()
        (cons (apply fn  (simple-map car l))
              (my-map fn (simple-map cdr l)))))
  (my-map fn l))


;;; (amap cons '(1 2 3) '(4 5 6))
;;; ((1 . 4) (2 . 5) (3 . 6))
;;; (amap cons '(1 2 3) '(4 5))
;;; ==> plante


;; Le vrai map (les listes peuvent être de longueur différente. On
;; s'arrête dès que l'on a atteint la fin d'une liste). Note: cette
;; version n'était pas demeandée

(define (Xmap fn . l)

  (define (null-in l)
    (and (not (null? l))
         (or (null? (car l))
             (null-in (cdr l)))))

  (define (my-map fn l)
    (if (null-in l)
        ()
        (cons (apply fn  (simple-map car l))
              (my-map fn (simple-map cdr l)))))
```

```
  (my-map fn l))

;;; (Xmap cons '(1 2 3) '(4 5 6))
;;; ((1 . 4) (2 . 5) (3 . 6))
;;; (Xmap cons '(1 2 3) '(4 5))
;;; ((1 . 4) (2 . 5))

;===============================================================================
;                                    EXO 4
;===============================================================================

(define (sum term a next b)
  (if (> a b)
      0
      (+ (term a)
         (sum term (next a) next b)))))

;; Somme des 100 premiers entiers
(sum (lambda (x) x) 1 (lambda (x) (+ x 1)) 100)

;; Somme des carrés des 20 premiers entiers
(sum (lambda (x) (* x x)) 1 (lambda (x) (+ x 1)) 20)

;; Intégrale de f(x)=x^2-1 entre 3 et 5
(let ((pas 0.001))
  (sum (lambda (x) (* (- (* x x) 1) pas)) 3 (lambda (x) (+ x pas)) 5))

;===============================================================================
;                                    EXO 5
;===============================================================================

;; En fait, les ensembles sont représentés par des prédicats. Cela
;; veut dire que x appartient à E ssi (E x) renvoie #t et, par
;; conséquent, si le prédicat renvoie #f, on considère que l'élément
;; n'est pas dans l'ensemble.
;;
;; Qu'est ce qu'on gagne avec cette représentation? Avec cette
;; représentation, on n'a pas besoin d'avoir uns structure de données
;; qui contienne en mémoire effectivement les éléments de
;; l'ensemble. Ce qu'on a en mémoire c'est la propriété que partagent
;; les élements de l'ensemble. On peut donc représenter des ensemble
;; infini sans problème.

(define (vide)
  (lambda (x) #f))

(define (entiers)
  (lambda (x) #t))

(define (multiple k)
  (lambda (x) (= (modulo x k) 0)))

(define (singleton k)
  (lambda (x) (= x k)))

(define (intervalle m n)
  (lambda (x) (<= m x n)))


(define (appartient? x E)
  (E x))
```



```
(define (union E1 E2)
  (lambda (x)
    (or (appartient? x E1)
        (appartient? x E2))))

(define (intersection E1 E2)
  (lambda (x)
    (and (appartient? x E1)
         (appartient? x E2))))

(define (complementaire E)
  (lambda (x)
    (not (appartient? x E))))

(define (difference E1 E2)
  (lambda (x)
    (and (appartient? x (union E1 E2))
         (not (appartient? x (intersection E1 E2))))))

;===============================================================================
;                                    EXO 6
;===============================================================================

;; Là c'est pas compliqué, il faut juste ce concentrer un peu et ça
;; devient évident.

(define (kons a b)
  (lambda (f)
    (f a b)))

(define (kar obj)
  (obj (lambda (a b) a)))

(define (kdr obj)
  (obj (lambda (a b) b)))
```