Parcours récursif d'une liste

- cdr vers le bas, cons vers le haut

```
(define (traite-liste L)
      (if (null? L)
      ()
      (cons (traite (car L))
            (traite-liste (cdr L)))))
```

Parcours avec fonction comme paramètre

```
(define (applique fct L)
   (if (null? L)
   `()
   (cons (fct (car L))
         (applique fct (cdr L)))))

(applique (lambda(x) (+ x 4)) `(1 2 3 4))
'(5 6 7 8)
```

Ajouter un prefixe

```
(define (prefixe pre L)
   (applique
      (lambda(el) (cons pre el))
      L)
)
```

```
(prefixe 'a '((b c) (d e))
(prefixe 'a '(b c))
(prefixe 'a ())
(prefixe 'a '(()))
```

Génération de combinaisons

```
(define (combine dim set)
   (cond
      ((= dim 0) '(()))
      ((null? set) '())
      (else
         (append (prefixe (car set)
                          (combine (- dim 1) (cdr set)))
                 (combine dim (cdr set))
         )
       )
 ))
```

Réducteur

```
(define (reduce F F0 L)
  (if (null? L)
      F0
      (F (car L)
         (reduce F F0 (cdr L)))
))

(reduce * 1 '(1 2 3 4))
```

Boucles

- Boucle à N répétitions

```
(define (boucle P N)
  (cond ((zero? N) ())
  (#T (manipule P) (boucle P (- N 1)))))
```

- Boucle de inf a sup

```
(define (boucle2 P inf sup)
  (cond ((> inf sup) ())
  (#T (manipule P) (boucle2 P (+ inf 1) sup))))
```

NOTE: Ces fonctions contiennent une recursivite terminale (*tail recursion*), plus facile a optimiser par un compilateur

La boucle do

(do ((*var init update*) ...) (*test resultIfTrue* ...) *exprIfTestFalse* ...)

```
(define fibonacci
  (lambda (n)
    (if (= n 0)
        0
        (do ((i n (- i 1)) (a1 1 (+ a1 a2)) (a2 0 a1))
            ((= i 1) a1)))))

(fibonacci 6)  8
```

Parcours avec récursivité terminale

- Toute fonction récursive peut être mise sous forme de récursivité terminale en utilisant des variables accumulant les résultats intermédiaires

```
(define (traite-liste2 L Lacc)
    (if (null? L)
        Lacc
       (traite-liste2 (cdr L)
          (append Lacc (list (traite (car L)))))))

(define (traite-liste L)
    (traite-liste2 L ()))
```

Exemple avec factoriel

```
(define (factorial n)
     (if (<= n 0)
         1
        (* n (factorial (- n 1))) ) )
```

Pour être en récursivité terminale, la fonction doit retourner le résultat de l'appel récursif sans modifications

```
(define (factorial n) (factorialb n 1))

(define (factorialb n answer)
       (if (<= n 0)
           answer
          (factorialb (- n 1) (* n answer)) ) )
```