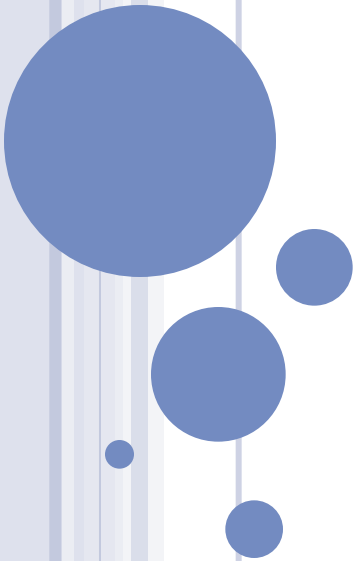


LES LISTES EN SCHEME

car, cdr, cons

cond

list, append

DÉFINITION

- Une liste est une suite d'éléments rangés dans un certain ordre

'(alpha 3 beta "delta" gamma)
'(a b c d e)
'(7 5 9 3 1)

PAIRES EN SCHEME

- Une paire est une structure d'enregistrement avec deux champs appelés car et cdr
  - car représente le premier élément
  - cdr représente le second élément
- Les champs car et cdr sont accédés par les fonctions car et cdr
- Une paire est représentée par la notation
  - ' (element1 . element2)
  - le . permet de faire la différence avec une liste de 2 éléments

LISTES EN SCHEME

- Une liste peut être définie récursivement comme :
  - soit une liste vide : '()
  - soit une paire dont le cdr est une liste
    - '(a . '(1 2 3)), qui se réécrit '(a 1 2 3)

ÉCRITURE D'UNE LISTE

- Les éléments d'une liste sont simplement compris entre parenthèses
  et séparés par des espaces

- La liste vide est écrite '()

- '(a b c d e) et '(a . (b . (c . (d . (e . ())))))
  sont deux notations équivalentes
  pour une liste de symboles

LA FONCTION PAIR?

- (pair? x) retourne #t si x est une paire, #f sinon

- Exemples :
  - (pair? '(a . b)) → #t
  - (pair? '(a b c)) → #t
  - (pair? '()) → #f

LES FONCTIONS CAR ET CDR

- (car x) retourne le champ car de x
  - (car '(a b c)) → a
  - (car '((a) b c d)) → (a)
  - (car '(1 . 2)) → 1
  - (car '()) → error
- (cdr x) retourne le champ cdr de x
  - (cdr '((a) b c d)) → (b c d)
  - (cdr '(1 . 2)) → 2
  - (cdr '(a)) → ()
  - (cdr '()) → error

- Si x est une liste,
  la fonction car retourne le premier élément de la liste,
  et la fonction cdr retourne le reste de la liste (donc une liste !)

LA FONCTION CONS

- (cons x y) retourne une nouvelle paire
  dont le car est x et le cdr est y
  Si y est une liste, la fonction *cons* ajoute x au début de y

- Exemples :
  - (cons 'a '()) → (a)
  - (cons '(a) '(b c d)) → ((a) b c d)
  - (cons "a" '(b c)) → ("a" b c)
  - (cons 'a 3) → (a . 3)
  - (cons '(a b) 'c) → ((a b) . C)

- Pour que le résultat du cons soit une liste,
  il faut donc que le deuxième argument soit une liste

FONCTIONS DE TEST

- (list? x) retourne #t si x est une liste, #f sinon
  - (list? '(a b c)) → #t
  - (list? '()) → #t
  - (list? '(a . b)) → #f

- (null? x) retourne #t si x est la liste vide, #f sinon

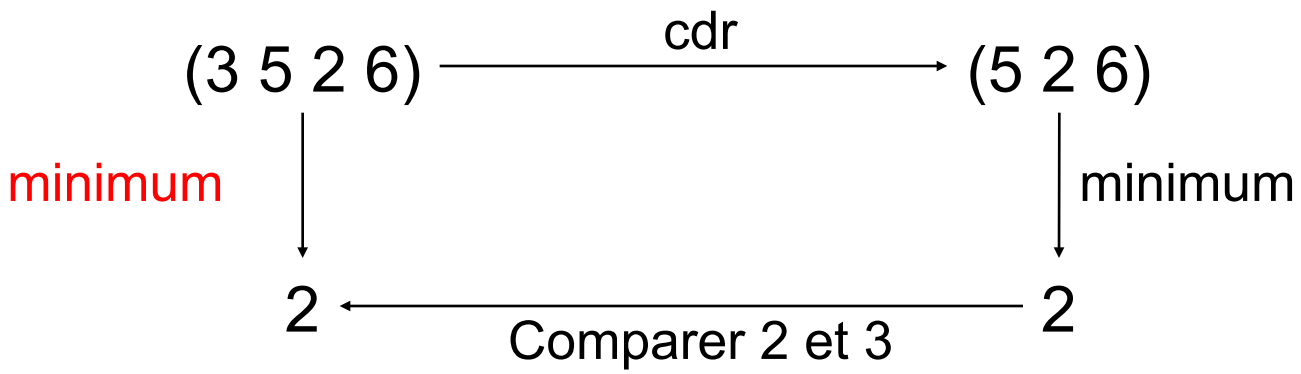
IMPLÉMENTATION DE L'ALGORITHME RÉCURSIF DU MINIMUM

Définition de la fonction minimum(L)

```
Si vide?(reste(L)) Alors
    retourne premier(L)
Sinon
    Si premier(L) < minimum(reste(L))
    Alors
            retourne premier(L)
    Sinon
            retourne minimum(reste(L))
    FinSi
FinSi
```

```
(define minimum ; → nombre
    (lambda (l) ; l liste de
                nombres non vide
            (if (null? (cdr l))
                (car l)
                (if  (< (car l)
                        (minimum (cdr l)))
                    (car l)
                    (minimum (cdr l))))))
```

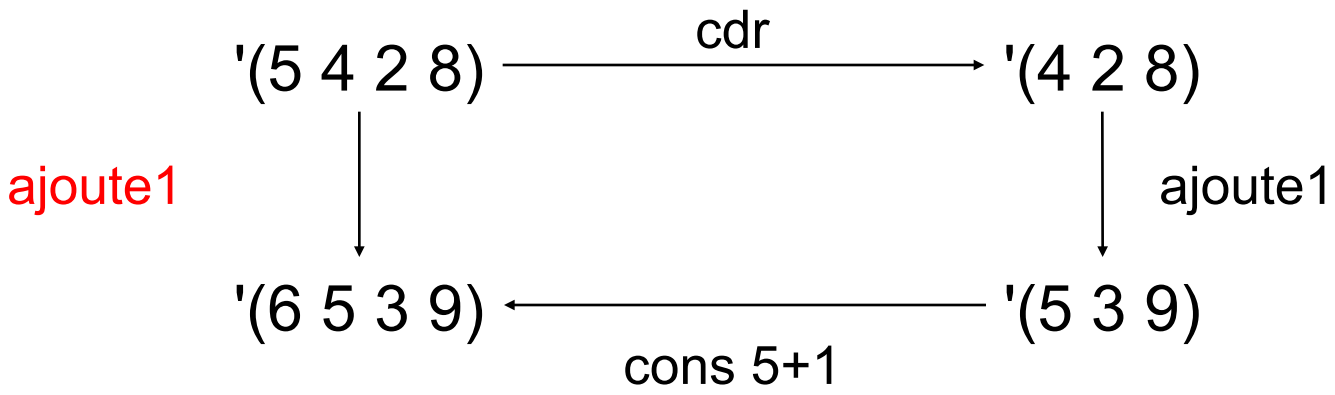
ILLUSTRATION DE LA MÉTHODE



UNE FONCTION QUI RETOURNE UNE LISTE

- Une fonction qui ajoute 1 à tous les éléments d'une liste de nombres

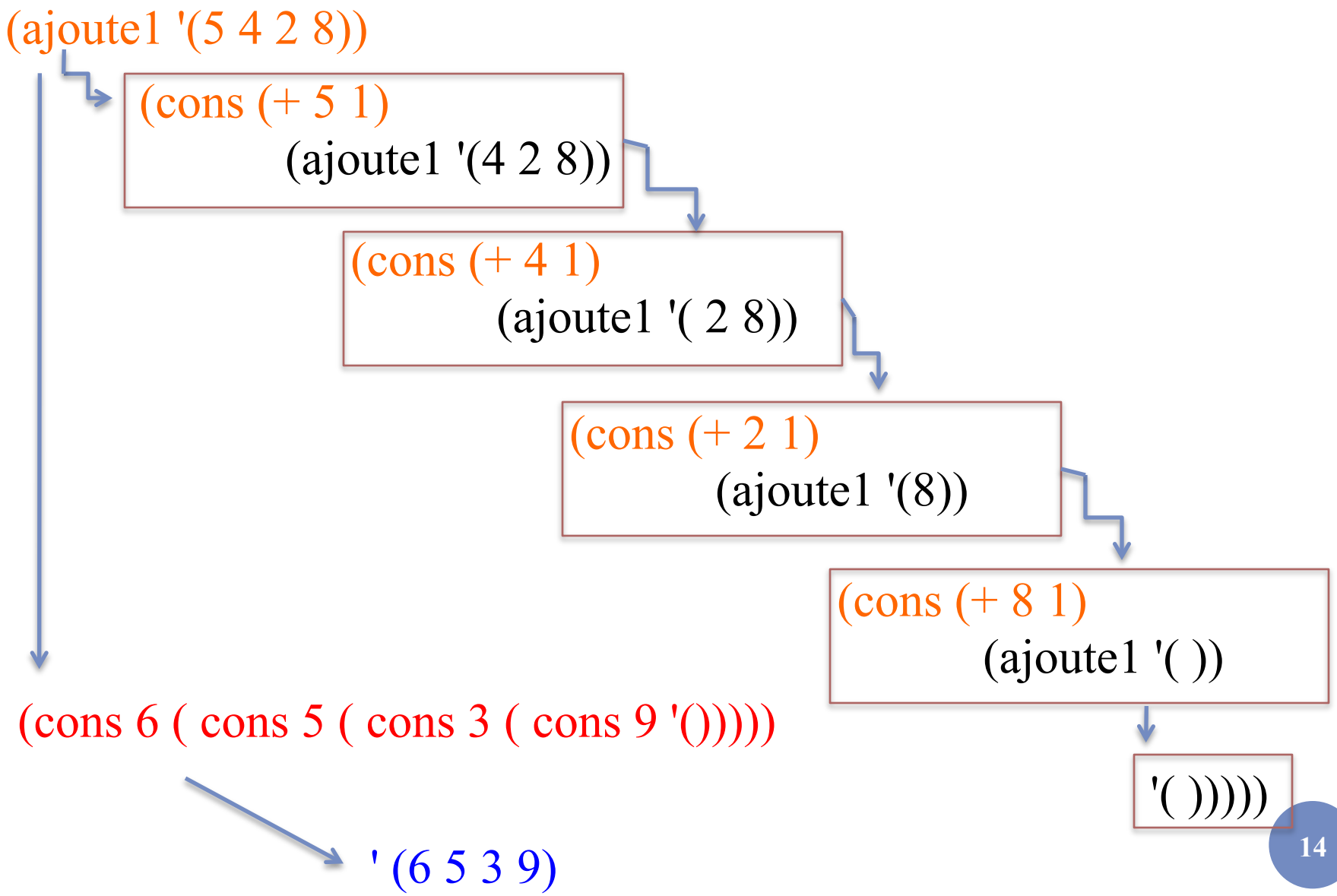
(ajoute1 '(5 4 2 8)) → (6 5 3 9)



ÉCRITURE DE LA FONCTION

```
(define ajoute1; → liste de nombres
  (lambda (l) ; l liste de nombres
        (if  (null? l)
                '()
                (cons (+ (car l) 1)
                        (ajoute1 (cdr l))))))
```

ILLUSTRATION DE LA FONCTION



RELÂCHER DES PRÉCONDITIONS

- Une fonction qui ajoute 1 à tous les nombres d'une liste quelconque

```
(define ajoute1; → liste
  (lambda (l) ; l liste
        (if  (null? l)
             '()
             (if (number? (car l))
                    (cons  (+ (car l) 1)
                           (ajoute1 (cdr l)))
                    (cons (car l)
                           (ajoute1 (cdr l)))))))
```

- (ajoute1 '(2 "a" 1 (toto) 5)) → (3 "a" 2 (toto) 6)

LA FORME SPÉCIALE COND

- La forme spéciale cond permet d'écrire plus simplement des if imbriqués

```
(if test1 valeur1
    (if test2 valeur2
            ...
            valeurN) …))
```

```
(cond  (test1 valeur1)
       (test2 valeur2)
       …
       (else valeurN))
```

APPLICATION À AJOUTE1

```
(define ajoute1; → liste
  (lambda (l) ; l liste
    (cond
      ((null? l) '())
      ((number? (car l)) (cons (+ (car l) 1)
                               (ajoute1 (cdr l))))
      (else (cons (car l) (ajoute1 (cdr l))))
    )))
```

METTRE EN FACTEUR DANS UN IF

```
(if (number? (car l))
    (cons (+ (car l) 1)
          (ajoute1 (cdr l)))
    (cons (car l)
          (ajoute1 (cdr l)))
)
```

```
(cons
    (if (number? (car l))
        (+ (car l) 1)
        (car l)
    )
    (ajoute1 (cdr l)))
```

Ces deux expressions sont équivalentes

LA FONCTION LIST

- La fonction list est une fonction qui permet de construire une liste, comme la fonction cons

- Elle prend un nombre quelconque d'arguments et les met tous dans une liste

  (list 'a (+ 3 2) "toto" 'b) → (a 5 "toto" b)

- Ne pas confondre list et list?

LA FONCTION APPEND

- La fonction append permet de concaténer des listes
- Elle prend un nombre quelconque d'arguments

(append '(a (b c) d) '(e f)) → (a (b c) d e f)

CONS, LIST ET APPEND : LES DIFFÉRENCES

Ces trois fonctions permettent toutes de construire des listes,
mais ont chacune un comportement différent

- cons prend deux arguments, le deuxième étant obligatoirement une liste
- list prend un nombre quelconque d'arguments de types quelconques
- append prend un nombre quelconque d'arguments qui doivent tous être des listes

CONS, LIST ET APPEND : EXEMPLES

- (cons '(a b) '(c d)) → ((a b) c d)
- (list '(a b) '(c d)) → ((a b) (c d))
- (append '(a b) '(c d)) → (a b c d)

RACCOURCIS POUR COMPOSER LES FONCTIONS CAR ET CDR

- Au lieu d'écrire (car (cdr L))
  on peut écrire (cadr L)

- Au lieu d'écrire (cdr (cdr (cdr L)))
  on peut écrire (cdddr L)

- Et ainsi de suite …
  (au maximum 4 caractères entre le c et le r)