4. La citation

```
>(quote (1 2 3))
 =(1 2 3)
```

- La fonction quote permet d’éviter que la liste en argument soit évaluée.
- Cette liste est plutôt retournée telle quelle.
- L' utlisation de cette fonction est nécessaire lorsque la premiere expression d' une liste ne s’évalue pas à une fonction. La fonction quote s’écrit plus simplement:

```
'(1 2 3)
```

Une fonction pour construire des listes

```
>(list ‘a ‘b ‘c)
=(a b c)
>(list ‘(a b c))
=((a b c))
```

Définition d'une fonction

- Une définition associe l'expression d'une fonction à un nom:

```
(define (carre x) (* x x))
```

ou, de façon équivalente:

```
(define carre (lambda (x) (* x x)))
(carre 2)
4
```

- L'expression (lambda(var1, var2, …) exp1 exp2 …) retourne une fonction ou les variables sont des paramètres qui seront appliqués aux expressions.
- A noter : > ((lambda (x) (* x x)) 3)
  = 9

Définition d'une fonction

```
(define (F-a-C temperature) ; conversion de oF a oC
        (/ (- temperature 32) 1.8))


>(F-a-C 95)
=35
```

Fonctions Primitives

- Prédicats ?: des fonctions qui retournent **`#t`** ou **#f**.
  - **(symbol? x)**
    #t si **x** est un symbole,
  - **(number? x)**
    #t si **x** est un nombre,
  - **(eq? x y)**
    #t si **x** et **y** sont des symboles égaux
  - **(equal? x y)**
    si **x** et **y** sont des objets identiques (pas nécessairement atomiques)
  - **(null? x)**
    si **x** est **()** – la liste vide
  - **(pair? x)**
    si **x** est soit une liste ou soit une pair
  - **(procedure? x)**
    si **x** est une fonction
  - **(list? x)**
    si **x** est une liste

Tests d'égalité: eq?

- eq? compare si il s'agit du même objet (compare les les adresses)
  - Ne pas utiliser pour comparer des nombres

```
(define chaine "bonjour")
(eq? chaine chaine)
#t
(eq? "bonjour" "bonjour")
#t
```

Tests d'égalité: equal?

- equal? compare les représentations

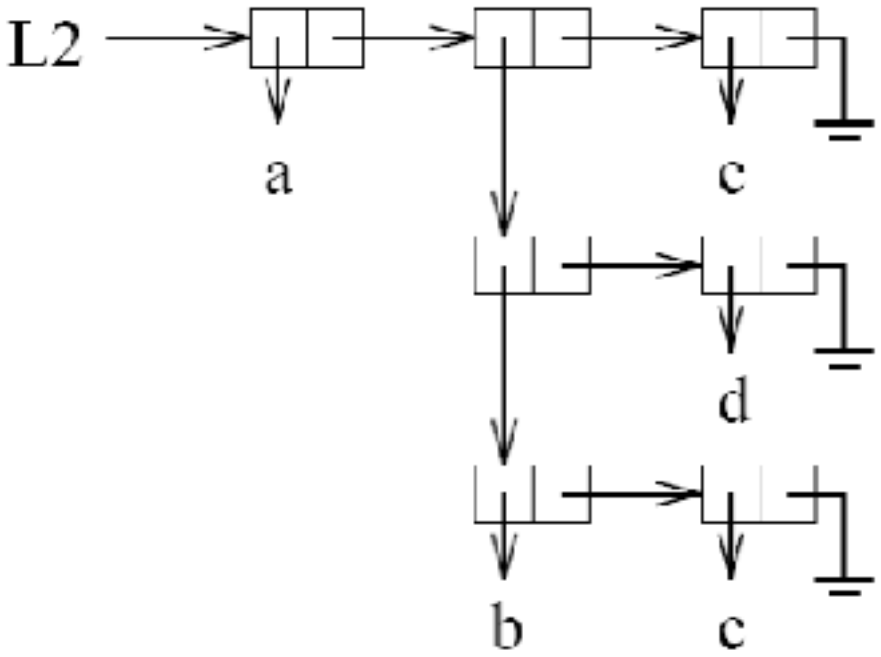
```
>(equal? '(a 1 2) '(a 1 2))
= #t
>(equal? "bonjour" "bonjour")
= #t
>(equal? (list 1 2) '(1 2))
= #t
>(equal? 'a 'a)
= #t
>(equal? 2 2)
= #t
>(equal? '(2)   2)
= #f
```

Représentation des listes

- A chacune des expressions formant une liste est associée une cellule mémoire constituée de deux pointeurs. Le premier de ces pointeurs donne l'adresse de l' atome ou de la liste correspondant, alors que le second pointeur donne l' adresse de la prochaine cellule.

Exemple

Si L2 est lié à (a ((b c) d) e)

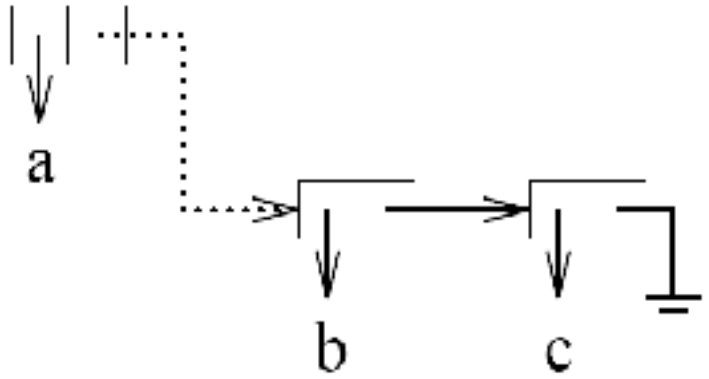


La fonction de construction

- Le premier paramètre de la liste est un atome à être placé en tête de la liste spécifiée comme second paramètre.
- Pour ce faire, une nouvelle cellule mémoire est créée
  - le premier de ses pointeurs pointe sur la première expression passée en paramètre
  - le second pointeur pointe sur la seconde expression

CONS

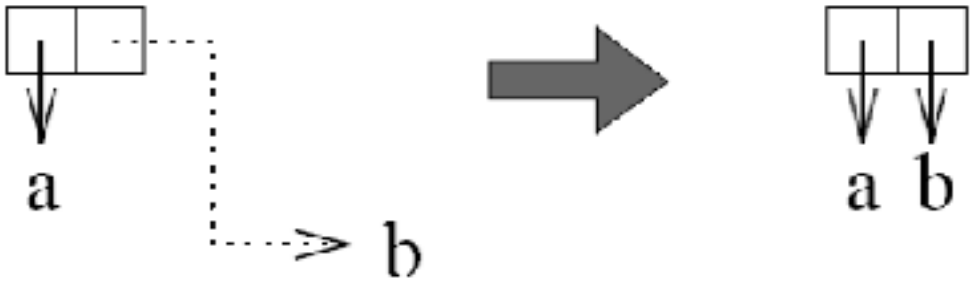
>(cons ‘a ‘(b c))
= (a b c)



>(cons ‘(a b) ‘(b c))
= ((a b) b c)

Une paire pointée : A EVITER!

>(cons ‘a ‘b)



L’usage des paires pointée en Scheme est toutefois déconseillée (les paires pointées ne sont pas des listes!)

CAR

- Content of the Address Register

```
>(car '(a b c))
=a
>(car '((a b) b c))
=(a b)
```

CDR

- Content of the Decrement Register

```
>(cdr '(a b c))
=(b c)
>(cdr '((a b) b c))
=(b c)
>(cdr '(a (b c)))
=((b c))
```

Utilisation cascadée

>(cdr (car (cdr '(a (b c d) e))))
peut s'écrire:
>(cdadr '(a (b c d) e))
=(c d)

>(cons (car '(a b c)) (cdr '(a b c)))
=(a b c)

TP N°1