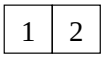
Programmation applicative – L2
TD 5 : Introduction aux paires et aux listes

G.Artignan {artignan@lirmm.fr} M.Lafourcade {lafourcade@lirmm.fr}
S.Daudé {sylvain.daude@univ-montp2.fr} A.Chateau {chateau@lirmm.fr}
B.Paiva Lima da Silva {bplsilva@gmail.com} C.Dony {dony@lirmm.fr}

1 La structure de données `pair`

Une paire, ou doublet, est constituée de deux valeurs : `(cons 1 2)` ⇝ 

Le constructeur de paire est l'opérateur `cons`, l'opérateur `car` permet d'accéder au premier élément de la paire et `cdr` permet d'accéder au second élément de la paire.

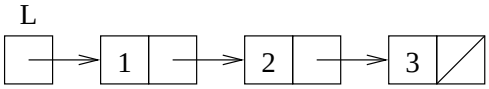
2 Structure de données liste

Rappel : `(car l)` donne la tête de la liste et `(cdr l)` donne la queue de la liste (c'est-à-dire la liste privée de la tête), `(cons t r)` réunit une tête et une queue de liste pour construire une liste. Écrire `'e` retourne l'expression `e` elle-même, et non le résultat de son évaluation. Par exemple, `'(+ 5 6)` retourne la liste `(+ 5 6)` alors que l'expression `(+ 5 6)` renvoie `11`.

Une liste se définit à partir de la structure de paire et de la valeur **liste vide**. La liste vide est considérée comme une expression atomique et se note `null` ou `()`.

Une expression `L` est une liste soit :
- Si `L` est la liste vide
- Si `(cdr L)` est une liste

Une liste est donc une chaîne de doublets qui se termine par la valeur liste vide.

`(define L (cons 1 (cons 2 (cons 3 ()))))` 

La structure de liste émerge de la *propriété de fermeture* du `cons`, c'est-à-dire le fait de créer des paires dont les éléments sont des paires.

Exercice 1 *Répondez aux questions suivantes :*

1. *L'expression suivante est-elle une liste ?* `(cons (cons 1 2) (cons 3 4))`
2. *Est-ce que la liste* `(1 2 3)` *est une paire ?*
3. *Est-ce que toutes les listes sont des paires ?*

Exercice 2 *Indiquer si les listes suivantes sont bien parenthésées.*

```
(5 (((6 7 9 'r))) 8 12)
(9 (((8 4 5 (4) 25) 3) 7)
(() ())
(56 'p ('K 6) 8 (() (7 6 2)))
```

Exercice 3 *Donner le* `car` *et le* `cdr` *des listes suivantes :*

```
(2)
()
((1 2) 3 4)
(() ())
(1 (2 (3 4 5) 6) 7)
(((8 4) 6 (5 4))(7 8 9))
```

Exercice 4 *Indiquer ce que retournent les expressions suivantes :*

```
1.    (+ ( 2 6) 10)
2.    '(+ ( 2 6) 10)
3.    (car (car  '((5 9) 7 5)))
4.    (car (cdr '((5 9) 7 5)))
5.    (cdr (car '((5 9) 7 5)))
6.    (cdr (cdr '((5 9) 7 5)))
7.    (cons '(a b) '(c d))
8.    (cons (+ 2 5) '(e f))
9.    (cons 'a '(b c d))
10.   (cons 'a '())
11.   (cons (car '(a b c)) (cdr '(a b c)))
12.   (let ((l '(a (b c e))))
           (if (equal? (cdr l) ()) 'yes 'nope))
```

Exercice 5 *Soit l'expression suivante :*

```
(define a (cons '+ (cons (cons ' * (cons (+ 2 6) (cons 3 ()))) (cons (+ 1 3) ()))))
```

1. *Donner la valeur de* `a`
2. *Donner la valeur de* `(cdr a)`
3. *Donner la valeur de* `(cdadr a)`

3 Fonctions de manipulations de listes

Exercice 6 *Écrire une fonction* `premdeux?` *qui indique si un élément se trouve soit en première place soit en seconde place d'une liste. Par exemple :*

```
> (premdeux? 'a '(a b c d))
>  #t
> (premdeux? 'b '(a b c d))
>  #t
> (premdeux? 'c '(a b c d))
>  #f
```

La fonction `list` prend un nombre quelconque d'éléments en paramètre, et construit la liste formée de ces éléments. Par exemple :

```
(list 1 2 3 4 5)       ⇒  (1 2 3 4 5)
(list '1 '(+) '(2 3))  ⇒  (1 (+) (2 3))
```

La fonction `append` prend un nombre quelconque de listes en paramètre, et construit une nouvelle liste qui est la concaténation du contenu de ces listes. Par exemple :

```
(append '(a b) '(c d) '() '((e)))  ⇒  (a b c d (e))
(append '(+) '(2 3))               ⇒  (+ 2 3)
```

Exercice 7 *Pour être sur de ne pas confondre, quelle est la valeur des expressions suivantes ?*

```
(cons '(a b) '(c d))
(list '(a b) '(c d))
(append '(a b) '(c d))
```

Exercice 8 *Donnez la valeur des expressions suivantes (utilisant* `append` *et* `list`*) :*

```
1.  (append (cons 'a (cons 'b ()))  (list (+ 2 3) 'a))
2.  (+ 2 (cadar (list (cons 3 (list 7 (+ 2 4))) (+ 4 5))))
3.  (append (list (cons 'a 'b)) (cons 'c (list 'd)))
4.  (list (cons 'a ()) 'b)
5.  (list (cons 'a (cons 'b ()))  (list (+ 2 3) 'a))
6.  (* 2 (cadar (list (append (cons 3 (list 7 (+ 2 4))) (list (+ 4 5))))))
```

4 Appliquer un traitement à tous les éléments d'une liste

Pour appliquer un opérateur sur tous les éléments d'une liste :

1. On applique l'opérateur sur le `car` de la liste,
2. Tant que la liste n'est pas vide, on appelle récursivement l'opérateur sur le `cdr` de la liste.

Exercice 9 *Écrire une fonction* `length` *qui calcule la longueur d'une liste, c'est-à-dire qui compte son nombre d'éléments.*

Exercice 10 *La fonction* `map` *est la fonction générique pour appliquer une fonction à chaque élément d'une liste. Elle prend en paramètre la fonction à appliquer, et la liste sur laquelle l'appliquer. Exemple :*

```
> (define (plusun n)
     (+ n 1))
> (map plusun '(1 2 3 4 5 6))
> (2 3 4 5 6 7)
```

Écrire une fonction `double` *qui double chaque élément d'une liste de nombres :*

1. *Sans utiliser* `map`
2. *En utilisant* `map`