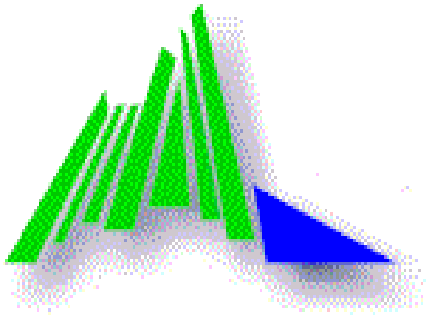
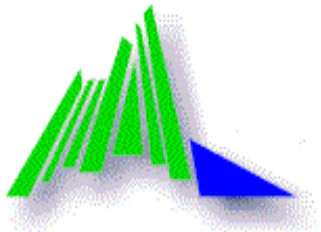


Les listes

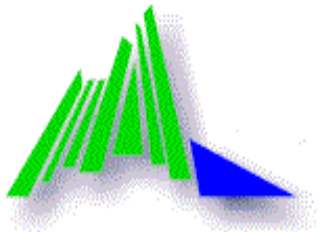


Syntaxe et Sémantique de la liste

- **Syntaxe : (** <suite d'éléments séparés par des espaces> **)**

- **Une liste peut représenter**
  - **l'application d'une fonction à des arguments :** (+ 5 6)
  - **une structure de données :** (a b c d)

- **Rappel évaluation :** évaluer une liste revient à appliquer une fonction
  - (+ 5 6) → 11
  - (a b c) → **ERREUR, *a* n 'est pas un symbole de fonction**



La liste comme structure de données

- Une structure de données est une organisation d'un ensemble de données en mémoire d'un ordinateur

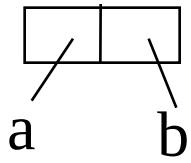
  Ex : Liste, tableau, pile, file…

- Pour considérer une liste comme un ensemble de données, il faut empêcher son évaluation : utilisation de la fonction quote
  - '(a b c) → (a b c)
  - '(+ 5 6) → (+ 5 6)

- Une liste est constituée à partir d'une structure de données élémentaire : la paire pointée

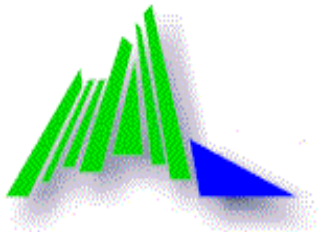
Notion de paire pointée (couple)

- Structure de données à deux éléments
- Notation : **(a . b)** **Attention aux espaces !**
- Représentation :



- **Fonction cons** : constructeur* de paire pointée
- (**cons** *expression1 expression2*)
  - fonction binaire
  - construit une paire pointée :
    - le premier élément est l'évaluation de *expression1*
    - le second élément est l'évaluation de *expression2*

(*) un constructeur est une fonction qui permet de créer une structure de donnée en mémoire

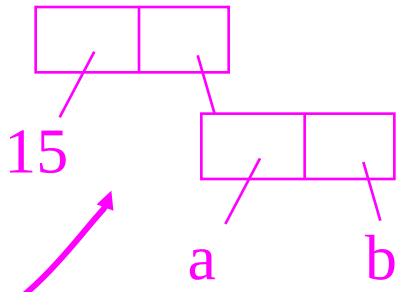


Notion de paire pointée (couple)

- Exemples
  - (cons 'a 'b) → (a . b)
  - (cons 'girafe 'lion) → (girafe . lion)
  - (cons "florence" 25) → ("florence" . 25)
  - (cons (+ 10 5) (boolean? #f)) → (15 . #t)

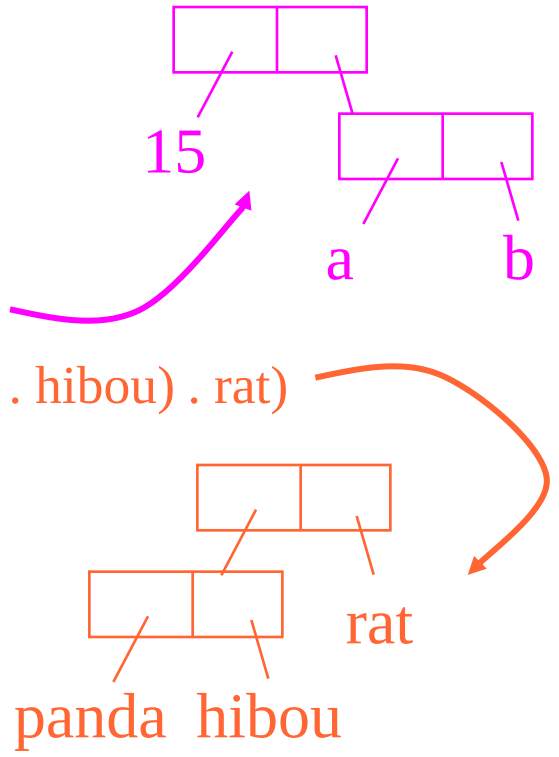
Notion de paire pointée (couple)

- Exemples
  - (cons 'a 'b) → (a . b)
  - (cons 'girafe 'lion) → (girafe . lion)
  - (cons "florence" 25) → ("florence" . 25)
  - (cons (+ 10 5) (boolean? #f)) → (15 . #t)
  - (cons (+ 10 5) (cons 'a 'b)) → (15 . (a . b))



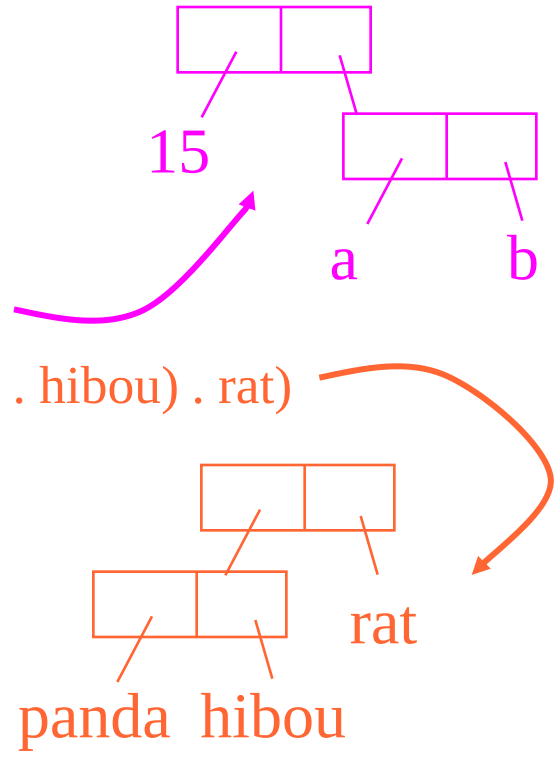
Notion de paire pointée (couple)

- Exemples
  - (cons 'a 'b) → (a . b)
  - (cons 'girafe 'lion) → (girafe . lion)
  - (cons "florence" 25) → ("florence" . 25)
  - (cons (+ 10 5) (boolean? #f)) → (15 . #t)
  - (cons (+ 10 5) (cons 'a 'b)) → (15 . (a . b))
  - (cons (cons 'panda 'hibou) 'rat) → ((panda . hibou) . rat)



Notion de paire pointée (couple)

- Exemples
  - (cons 'a 'b) → (a . b)
  - (cons 'girafe 'lion) → (girafe . lion)
  - (cons "florence" 25) → ("florence" . 25)
  - (cons (+ 10 5) (boolean? #f)) → (15 . #t)
  - (cons (+ 10 5) (cons 'a 'b)) → (15 . (a . b))
  - (cons (cons 'panda 'hibou) 'rat) → ((panda . hibou) . rat)

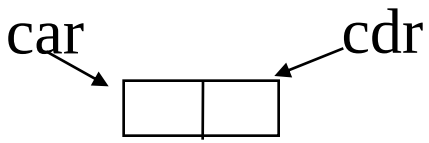


- Prédicat de type : **pair?**

  (pair? <*paire*>)

  (pair? ‘ (a . b)) → #t

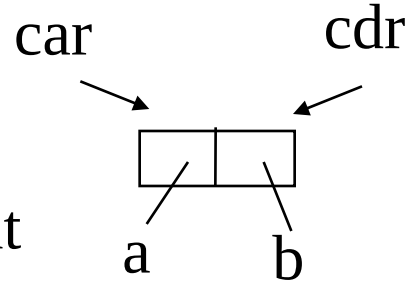
Accès aux éléments d'une paire pointée



- La fonction **car** : accès au premier élément

  (car *<paire>*)

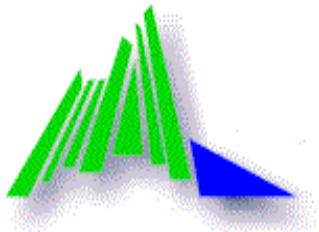
  (car '(a . b)) → a



- La fonction **cdr** : accès au second élément

  (cdr *<paire>*)

  (cdr ' (a . b)) → b



Accès aux éléments d'une paire pointée

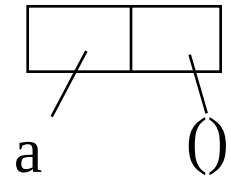
- (car (cons 'elephant 'toucan)) → elephant
- (car (cons (cons 'girafe 'zebre) 'zebu)) → (girafe . zebre)
- (car (car '(((12 . 13) . 25) . (5 . 8)))) → (12 . 13)

- (cdr (cons 'elephant 'toucan)) → toucan
- (cdr (cons (cons 'girafe 'zebre) 'zebu)) → zebu
- (cdr (car '(((12 . 13) . 25) . (5 . 8)))) → 25

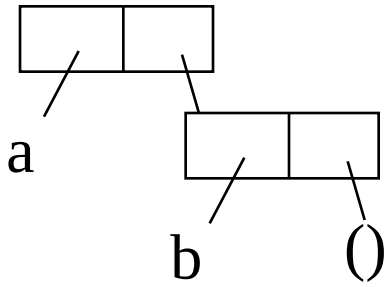
Définition d'une liste

- Une liste peut être :
  - la liste vide notée **()**
  - une paire pointée dont le second élément est **une liste**

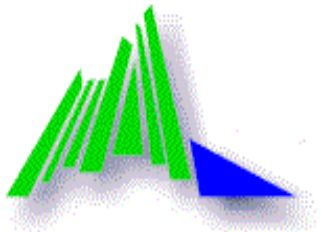
- Exemples
  - (a . ()) liste à un seul élément



  - (a . (b . ())) liste à deux éléments

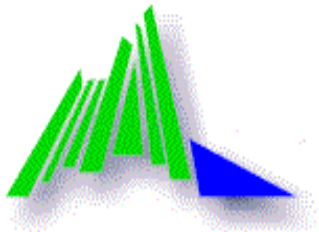


☞pour chaque élément : une paire pointée



Définition d'une liste

- Notation simplifiée
  - **(** <suite d 'éléments séparés par des espaces> **)**
  - (a . ()) → (a)
  - (a . (b . ())) → (a b)
  - (a . (b . (c . ()))) → (a b c)

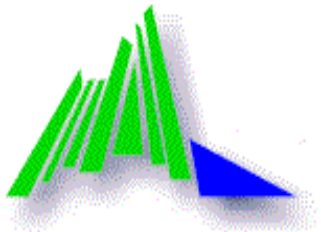


Les Prédicats sur les listes

- Prédicat de type : **list?**

  (list? *<liste>*)

  - (list? '()) → #t
  - (list? '(a)) → #t
  - (list? (cons 'herisson '())) → #t
  - (list? '(koala . (singe . ()))) → #t
  - (list? '((souris . rat) . ())) → #t
  - (list? '(souris . rat)) → #f
  - (list? '((chameau . ()) . dromadaire)) → #f
  - (list? (cons 'tigre 'girafe)) → #f



Les Prédicats sur les listes

- Prédicat de liste vide : **null?**

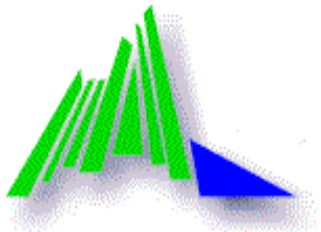
  (null? *<liste>*)

  (null? '()) → #t

- Prédicat **pair?**

  (pair? *<liste_non_vide>*)

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

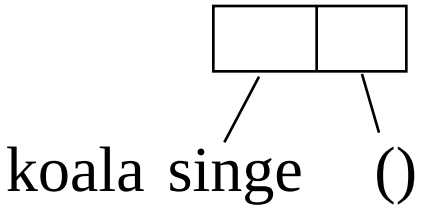


Constructeurs de liste

- Fonction **cons**
- (**cons** *expression1 expression2)*
  - fonction binaire
  - construit une paire pointée représentant une liste :
    - le premier élément est l'évaluation de *expression1*
    - le second élément est l'évaluation de *expression2* et doit obligatoirement être une liste
  - (cons 'a '()) → (a)
  - (cons 'koala '(singe)) → (koala singe)

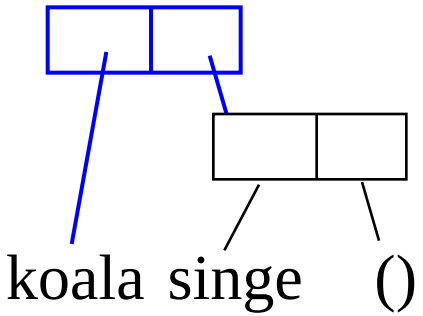
Constructeurs de liste

- Fonction **cons**
- (**cons** *expression1 expression2)*
  - fonction binaire
  - construit une paire pontée représentant une liste :
    - le premier élément est l ’évaluation de *expression1*
    - le second élément est l ’évaluation de *expression2* et doit obligatoirement être une liste
  - (cons 'a '()) → (a)
  - (cons 'koala '(singe)) → (koala singe)



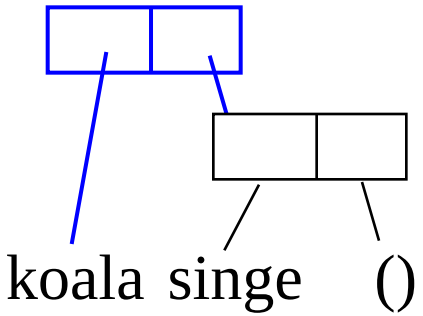
Constructeurs de liste

- Fonction **cons**
- (**cons** *expression1 expression2)*
  - fonction binaire
  - construit une paire pontée représentant une liste :
    - le premier élément est l ’évaluation de *expression1*
    - le second élément est l ’évaluation de *expression2* et doit obligatoirement être une liste
  - (cons 'a '()) → (a)
  - (cons 'koala '(singe)) → (koala singe)



Constructeurs de liste

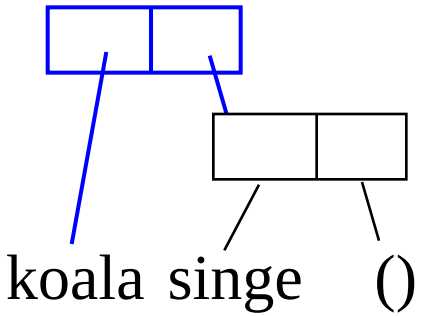
- Fonction **cons**
- (**cons** *expression1 expression2)*
  - fonction binaire
  - construit une paire pointée représentant une liste :
    - le premier élément est l ’évaluation de *expression1*
    - le second élément est l ’évaluation de *expression2* et doit obligatoirement être une liste



  - (cons 'a '()) → (a)
  - (cons 'koala '(singe)) → (koala singe)
  - (cons 'pie (cons 'chouette '(aigle))) → (pie chouette aigle)
  - (cons ’koala ’singe) → (koala . singe)

Constructeurs de liste

- Fonction **cons**
- (**cons** *expression1 expression2)*
  - fonction binaire
  - construit une paire pointée représentant une liste :
    - le premier élément est l ’évaluation de *expression1*
    - le second élément est l ’évaluation de *expression2* et doit obligatoirement être une liste
  - (cons 'a '()) → (a)
  - (cons 'koala '(singe)) → (koala singe)

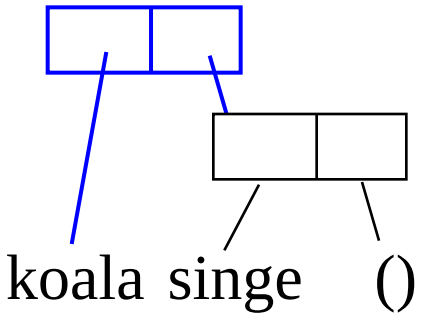


  - (cons 'pie (cons 'chouette '(aigle))) → (pie chouette aigle)
  - (cons ’koala ’singe) → (koala . singe)

koala singe

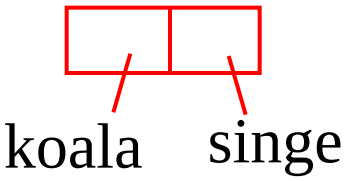
Constructeurs de liste

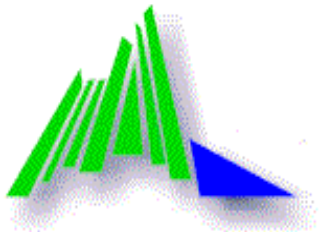
- Fonction **cons**
- (**cons** *expression1 expression2)*
  - fonction binaire
  - construit une paire pontée représentant une liste :
    - le premier élément est l ’évaluation de *expression1*
    - le second élément est l ’évaluation de *expression2* et doit obligatoirement être une liste



  - (cons 'a '()) → (a)
  - (cons 'koala '(singe)) → (koala singe)
  - (cons 'pie (cons 'chouette '(aigle))) → (pie chouette aigle)
  - (cons ’koala ’singe) → (koala . singe)

    ☞ ce n ’est pas une liste





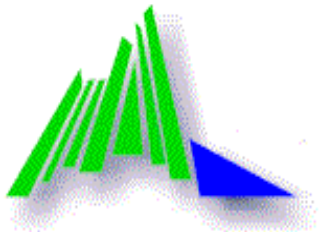
Constructeurs de liste

- **Fonction list**

(**list** *expression*$_1$ *expression*$_2$… *expression*$_n$)

- fonction n-aire
- construit une liste :
  - le premier élément est l'évaluation de *expression*$_1$
  - le second élément est l'évaluation de *expression*$_2$
  - ...
  - le nième élément est l'évaluation de *expression*$_n$

- (list 'a 'b 'c 'd) → (a b c d)
- (list 'a '(b c d)) → (a (b c d))
- (list (+ 2 11) 5 (* 3 4)) → (13 5 12)
- (list 'ouistiti '() 4 'kiwi) → (ouistiti () 4 kiwi)
- (list (list 'girafe)) → ((girafe))



Constructeurs de liste

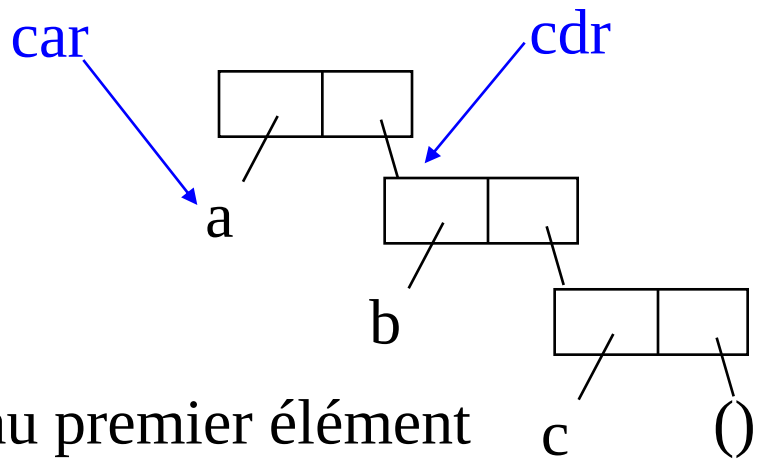
- **Fonction append**

(**append** *expression*$_1$ *expression*$_2$… *expression*$_n$)

- fonction n-aire
- l'évaluation de chaque *expression*$_i$ doit être une liste
- réalise la fusion de plusieurs listes

- (append '(a b) '(c d) '(e f)) → (a b c d e f)
- (append '(tatou) '(tapir)) → (tatou tapir)
- (append '(tatou)) → (tatou)
- (append '()) → ()
- (append '() '(guepard autruche)) → (guepard autruche)
- (append '(+ 2 3) '(* 5 6)) → (+ 2 3 * 5 6)
- (append (+ 2 3) '(* 5 6)) → ERREUR

Accès aux éléments d'une liste



- La fonction **car** : accès au premier élément

  (car <liste>)

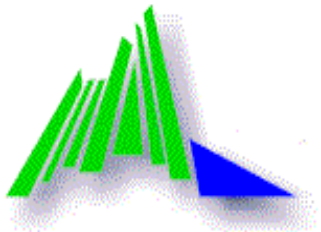
  (car '(a b c)) → a

- La fonction **cdr** : accès au reste de la liste (second élément de la paire pointée)

  (cdr <liste>)

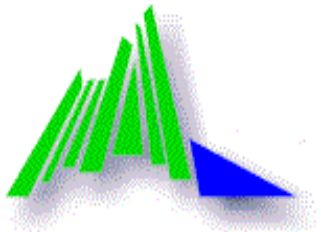
  (cdr '(a b c)) → (b c)

  ☞ le cdr d'une liste est la liste privée de son premier élément



Accès aux éléments d'une liste

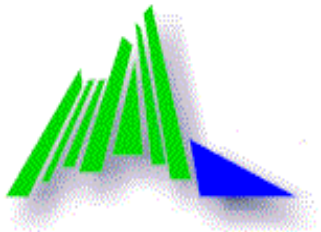
- Exemples
  - (car '(koala kangourou kiwi)) → koala
  - (cdr '(koala kangourou kiwi)) → (kangourou kiwi)
  - (car '()) → ERREUR
  - (cdr '()) → ERREUR
- Notation condensée : composition de fonctions
  - (car (car l)) ⇔ (caar l)
  - (car (cdr l)) ⇔ (cadr l)
  - (car (cdr (cdr (car l)))) ⇔ (caddar l)



Accès aux éléments d'une liste

- Exemples
  - (car '(koala kangourou kiwi)) → koala
  - (cdr '(koala kangourou kiwi)) → (kangourou kiwi)
  - (car '()) → ERREUR
  - (cdr '()) → ERREUR

- Notation condensée : composition de fonctions
  - (car (car l)) ⇔ (caar l)
  - (car (cdr l)) ⇔ (cadr l)
  - (car (cdr (cdr (car l)))) ⇔ (caddar l)

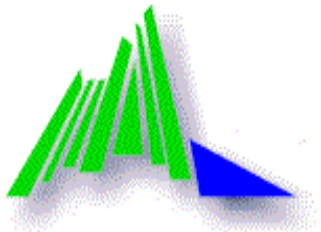


Accès aux éléments d'une liste

- Exemples
  - (car '(koala kangourou kiwi)) → koala
  - (cdr '(koala kangourou kiwi)) → (kangourou kiwi)
  - (car '()) → ERREUR
  - (cdr '()) → ERREUR

- Notation condensée : composition de fonctions
  - (car (car l)) ⇔ (caar l)
  - (car (cdr l)) ⇔ (cadr l)
  - (car (cdr (cdr (car l)))) ⇔ (caddar l)
  - (car (car (cdr (cdr (car l))))) ⇔ (car (caddar l))

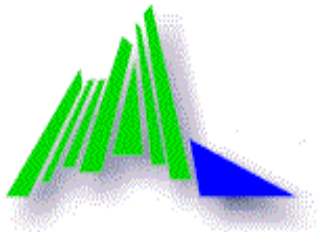
    ☞ composition de 4 fonctions maximum



Exemple

- Ecrire la fonction **min** qui prend une liste de 3 nombres en argument et retourne le minimum.

  Ex. : (min '(8 1 4)) ➔ 1



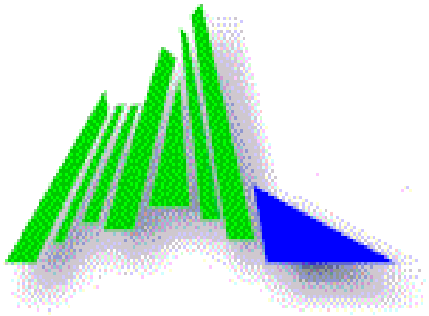
Exemple

- Ecrire la fonction **min** qui prend une liste de 3 nombres en argument et retourne le minimum.

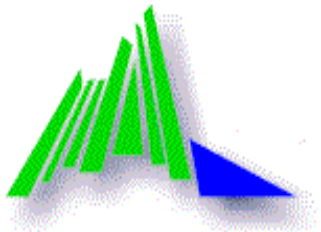
  Ex. : (min '(8 1 4)) ➔ 1

```
(define (min l)
  (if (not (= (length l) 3))
      'erreur_nombre_elements
  (let ((min2premier (if (< (car l) (cadr l))
                         (car l)
                         (cadr l))))
    (if (< (caddr l) min2premier)
        (caddr l)
        min2premier))))
```

```
(define (min1 l)
  (if (not (= (length l) 3))
      'erreur_nombre_elements
  (let ((min2dernier (if (< (cadr l) (caddr l))
                         (cadr l)
                         (caddr l))))
    (if (< (car l) min2dernier)
        (car l)
        min2dernier))))
```

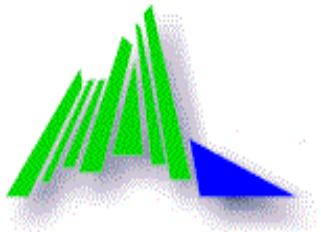


Récursivité sur les listes



Principe

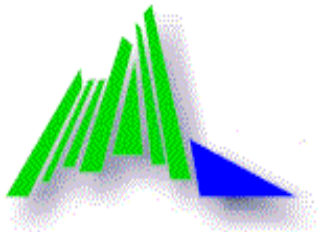
Une liste est une donnée **structurellement récursive** : elle est soit vide, soit composée d'un élément, **le car**, mis en tête d'une autre liste, **le cdr**. Pour cette raison, il est intéressant d'utiliser des fonctions récursives pour résoudre de nombreux problèmes sur les listes.



Exemple

La fonction **min** qui prend une liste de 3 nombres en argument et retourne le minimum.

```
(define (min1 l)
  (if (not (= (length l) 3))
      'erreur_nombre_elements
  (let ((min2dernier (if (< (cadr l) (caddr l))
                         (cadr l)
                         (caddr l))))
    (if (< (car l) min2dernier)
        (car l)
        min2dernier))))
```



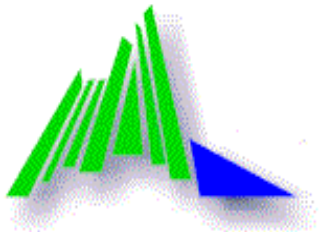
Exemple

La fonction **min** qui prend une liste de 3 nombres en argument et retourne le minimum.

```
(define (min1 l)
  (if (not (= (length l) 3))
      'erreur_nombre_elements
  (let ((min2dernier (if (< (cadr l) (caddr l))
                         (cadr l)
                         (caddr l))))
    (if (< (car l) min2dernier)
        (car l)
        min2dernier))))
```

La même fonction pour une liste de longueur quelconque

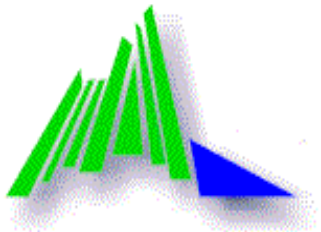
```
(define (min2 l)
  (cond ((null? l) 'Erreur_liste_vide)
        ((null? (cdr l)) (car l))
        (else (let ((minn-1derniers (min2 (cdr l))))
                (if (< (car l) minn-1derniers)
                    (car l)
                    minn-1derniers))))))
```



Méthode

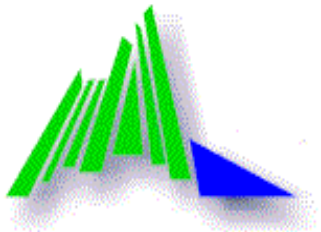
- La méthode d'écriture de ces fonctions est très souvent la même.
  - La variable de récursivité est la liste,
  - sa valeur minimale est **(),**
  - la fonction de variation est **cdr** : partant d'une liste non vide quelconque, puis en faisant un certain nombre de fois **cdr**, on arrive toujours à **().**

- Quand on écrit une fonction prenant en paramètre une liste, il faut avoir le réflexe programmation récursive : **est-ce que je peux ramener le calcul de cette fonction au calcul de la même fonction sur le cdr de la liste** ?



Exemple : recherche d'un élément

- Ecrire le prédicat *appartient?* qui étant donné un élément *e* et une liste *l*, retourne :
  - *#t* , si *e* appartient à *l*
  - *#f* , si *e* n'est pas dans *l*



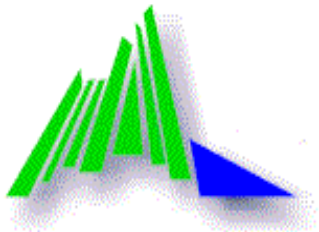
Liste plate, liste quelconque

- Une liste **plate** (ou linéaire) est une liste qui ne contient pas de sous-listes.

  Ex : (a b c)

- Une liste **quelconque** (ou non linéaire) contient des sous-listes.

  Ex : (a (b c) (d (e f g) h) i)



Liste plate, liste quelconque

- Une liste **plate** (ou linéaire) est une liste qui ne contient pas de sous-listes.

  Ex : (a b c)

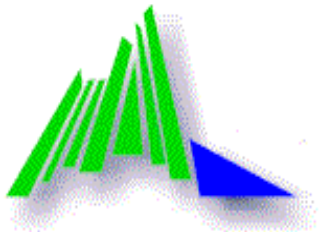
  (appartient ? ‘b ‘(a b c)) → #t

- Une liste **quelconque** (ou non linéaire) contient des sous-listes.

  Ex : (a (b c) (d (e f g) h) i)

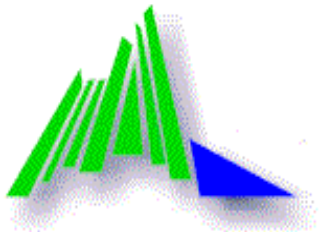
  (appartient ? ‘(b c) ‘(a (b c) (d (e f g) h) i)) → #t

  (appartient ? ‘b ‘(a (b c) (d (e f g) h) i)) → #f



Récursivité dans les listes quelconques

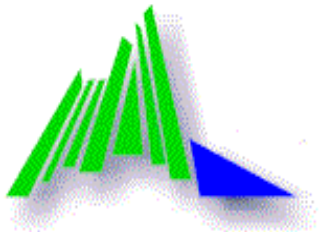
- Il faut aussi explorer les sous-listes
- Par rapport à la méthode d'écriture précédente, il faut traiter le cas ou le premier élément de la liste est une sous-liste :
  - tester le premier élément de la liste (**car**) :
    - Si c'est une liste, rappeler récursivement la fonction sur cet élément et combiner le cas échéant avec l'appel récursif sur le reste de la liste.
    - Sinon traiter l'élément selon la fonction, comme pour une liste plate.



Exemple : recherche d'un élément

Ecrire le prédicat *appartientq?* qui étant donné un élément *e* et une liste *l* quelconque, retourne :

- *#t* , si *e* appartient à *l*
- *#f* , si *e* n'est pas dans *l*



Exercice d'application – Listes quelconques

On représente une population par une liste d'individus. Chaque individu est lui-même représenté par une liste contenant le numéro INSEE, le nom, le prénom et une liste de caractéristiques. Chaque caractéristique est représentée par une paire *(symbole . valeur)*. Dans la mesure où l'on ne possède pas toutes les informations sur les individus, la liste de caractéristiques est de longueur variable selon les individus et ceux-ci n'ont pas tous forcément les mêmes caractéristiques.

Exemple de population de trois individus:

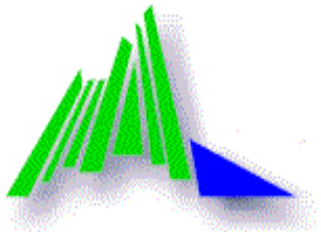
((1650964545270 Dupont Jean ((taille . 1.70) (poids . 80) (yeux . noir)))

(1650964545270 Durand Pierre ((cheveux . noir) (yeux . bleu)))

(1650964545270 Dubois Paul ((taille . 1.70) (poids . 90) (yeux . marron) (cheveux . blond))))

1) Définir une fonction **valeur** qui étant donné un symbole de caractéristique et un individu, retourne la valeur de cette caractéristique pour cet individu. Si l'individu ne possède pas cette caractéristique, la fonction retourne le symbole *indefini*.

Ex : (valeur 'yeux '(1650964545270 Durand Pierre ((cheveux . noir) (yeux . bleu))))→ bleu



Exercice d'application – Listes quelconques

On représente une population par une liste d'individus. Chaque individu est lui-même représenté par une liste contenant le numéro INSEE, le nom, le prénom et une liste de caractéristiques. Chaque caractéristique est représentée par une paire *(symbole . valeur)*. Dans la mesure où l'on ne possède pas toutes les informations sur les individus, la liste de caractéristiques est de longueur variable selon les individus et ceux-ci n'ont pas tous forcément les mêmes caractéristiques.

Exemple de population de trois individus:

((1650964545270 Dupont Jean ((taille . 1.70) (poids . 80) (yeux . noir)))

(1650964545270 Durand Pierre ((cheveux . noir) (yeux . bleu)))

(1650964545270 Dubois Paul ((taille . 1.70) (poids . 90) (yeux . marron) (cheveux . blond))))

2) Ecrire une fonction **meme-valeur** qui étant donné une caractéristique et une population retourne la liste, éventuellement vide, des individus ayant la même caractéristique.

Ex : (meme-valeur '(taille . 1.70) pop) → ((1650964545270 Dupont Jean ((taille 1.70) (poids . 80) (yeux . noir))) (1650964545270 Dubois Paul ((taille . 1.70) (poids . 90) (yeux . marron) (cheveux . blond))))

Exercice d'application – Listes quelconques

On représente une population par une liste d'individus. Chaque individu est lui-même représenté par une liste contenant le numéro INSEE, le nom, le prénom et une liste de caractéristiques. Chaque caractéristique est représentée par une paire *(symbole . valeur)*. Dans la mesure où l'on ne possède pas toutes les informations sur les individus, la liste de caractéristiques est de longueur variable selon les individus et ceux-ci n'ont pas tous forcément les mêmes caractéristiques.

Exemple de population de trois individus:

((1650964545270 Dupont Jean ((taille . 1.70) (poids . 80) (yeux . noir)))

(1650964545270 Durand Pierre ((cheveux . noir) (yeux . bleu)))

(1650964545270 Dubois Paul ((taille . 1.70) (poids . 90) (yeux . marron) (cheveux . blond))))

3) On suppose qu'il existe des prédicats de tests spécifiques aux caractéristiques, c'est-à-dire des prédicats qui, appliqués à un individu, retourne *#t* (le test est positif) si l'individu possède cette caractéristique et *#f* (le test est négatif) sinon. Exemples de prédicats de tests : *taille-170?, poids_80?, yeux_marron?…* Définir la fonction **partition-test** qui étant donné un prédicat de test et une population retourne une liste contenant deux sous-listes éventuellement vides. La première contient les individus pour lesquels le test est positif, la seconde contient les individus pour lesquels le test est négatif.

Ex : (partition-test taille_170? pop) →

(((1650964545270 dupont jean ((taille . 1.7) (poids . 80) (yeux . noir))) (1650964545270 dubois paul ((taille . 1.7) (poids . 90) (yeux . marron) (cheveux . blond))))((1650964545270 durand pierre ((cheveux . noir) (yeux . bleu)))))