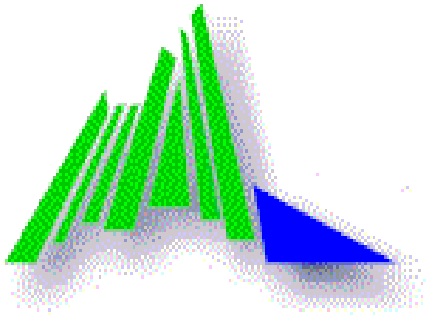


Les fonctions en Scheme

Rappels - Définition de fonction

- **Spécification** d'une fonction : un **domaine de départ**, un **domaine d'arrivée** et **le corps** de la fonction (expression de calcul du résultat).

$$\text{racine} : \mathfrak{R}+ \longrightarrow \mathfrak{R}+ \quad x \longrightarrow \sqrt{x}$$

$$\text{addition} : \mathfrak{R} \times \mathfrak{R} \longrightarrow \mathfrak{R} \quad x, y \longrightarrow x + y$$

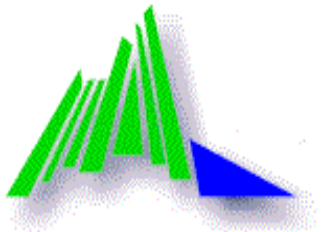
$$f : D1 \times D2 \ldots \times Dn \longrightarrow A \quad x1, x2, \ldots, xn \longrightarrow f(x1, x2, \ldots, xn)$$

- **Arité** d'une fonction : le nombre de **paramètres formels** (éléments de l'ensemble de départ : variables). On parle de fonction **unaire**, **binaire**, **n-aire**.

Application de fonction en Scheme (calcul)

- Pour une fonction n-aire, ceci revient à évaluer une liste (f arg_1 arg_2…arg_n)
- Chaque élément de la liste est séparé des autres par un espace
- *f* est le nom de la fonction
- arg_i représentent les arguments de la fonction appelés également paramètres réels.
- Exemple : (* 2 5)

  application de la fonction ***** (multiplication) aux deux arguments **2** et **5**



Application de fonction en Scheme (calcul)

- Evaluation en trois étapes (rappel arbre pour les expressions)

  1° L'évaluateur vérifie symbole fonction

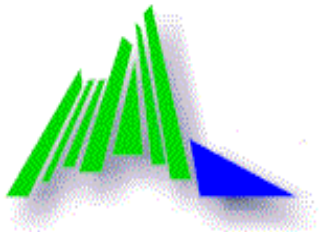
  2° L'évaluateur évalue les arguments

  3° L'évaluateur applique la fonction à la valeur des arguments et retourne le résultat

  (* (+ 1 2) 5) → 15

  ☞ **Il existe des fonctions spéciales qui n'évaluent pas tous leurs arguments**

- Il doit y avoir cohérence entre le nombre de paramètres formels de la spécification et le nombre d'arguments au moment de l'application



La fonction define - définition de variables et de fonctions

- Notion d'environnement
  - Zone mémoire dans laquelle sont stockées toutes les définitions de variables et de fonctions. On peut la voir comme un tableau d'associations entre un symbole (nom de variable ou de fonction) et une valeur (expression scheme).

Symbole	Valeur
a	**10**
pi	**3.1415**
addition	**(lambda (x y) (+ x y))**

- Notation : **{<a, 10>, <pi, 3.1415>, <addition , (lambda (x y) (+ x y))>}**
- L'environnement est utilisé pour évaluer les variables et les fonctions
  - évaluer une variable : lire sa valeur
  - évaluer une fonction : lire sa spécification (arité et corps)

La fonction define - définition de variables et de fonctions

- Définition d'une variable
  - (define *variable expression*) fonction binaire
  - *variable :* un symbole représentant le nom de la variable
  - *expression :* une expression scheme
- L'application de la fonction define
  - n'évalue pas le 1° argument, évalue le second (fonction spéciale)
  - a pour effet d'ajouter l'association à l'environnement
  - donne une valeur **et un type** à la variable

(define x 10)

Symbole	Valeur
x	**10**

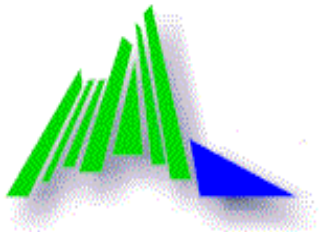
La fonction define - définition de variables et de fonctions

- Définition d'une variable
  - (define *variable expression*) fonction binaire
  - *variable :* un symbole représentant le nom de la variable
  - *expression :* une expression scheme
- L'application de la fonction define
  - n'évalue pas le 1° argument, évalue le second (fonction spéciale)
  - a pour effet d'ajouter l'association à l'environnement
  - donne une valeur **et un type** à la variable

(define x 10)

(define Pi 3.1415)

Symbole	Valeur
x	**10**
pi	**3.1415**



La fonction define - définition de variables et de fonctions

- Définition d'une variable
  - (define *variable expression*) fonction binaire
  - *variable :* un symbole représentant le nom de la variable
  - *expression :* une expression Scheme

- L'application de la fonction define
  - n'évalue pas le 1° argument, évalue le second (fonction spéciale)
  - a pour effet d'ajouter l'association à l'environnement
  - donne une valeur **et un type** à la variable

(define x 10)

(define Pi 3.1415)

(define y (+ x 5))

Symbole	Valeur
x	**10**
pi	**3.1415**
y	**15**

La fonction define - définition de variables et de fonctions

- Définition d'une variable
  - (define *variable expression*) fonction binaire
  - *variable :* un symbole représentant le nom de la variable
  - *expression :* une expression scheme
- L'application de la fonction define

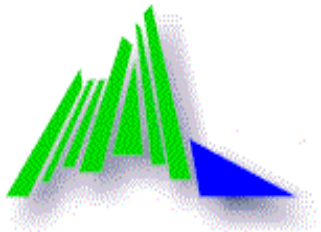
Ne retourne pas de résultat !

(define x 10) → Indéfini

(define Pi 3.1415) → Indéfini

(define y (+ x 5)) → Indéfini

Symbole	Valeur
x	**10**
pi	**3.1415**
y	**15**



La fonction define - définition de variables et de fonctions

- Définition d'une fonction

  (define *nom_fonction* (lambda (p_1 p_2...p_n) *corps_fonction*))

  (define (*nom_fonction* p_1 p_2...p_n) *corps_fonction*)

  - p_i : paramètres formels
  - *corps_fonction* : une expression Scheme

- Exemple

  (define addition (lambda (x y) (+ x y)))

  (define (addition x y) (+ x y))

- Ecriture pour améliorer la lisibilité

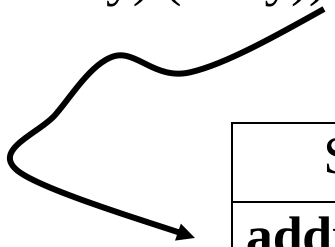
  (define (*nom_fonction* p_1 p_2...p_n)
  corps_fonction
)

  (define (addition x y)
  (+ x y)
)

La fonction define - définition de variables et de fonctions

- Mêmes principes d'application que pour les variables
  - n'évalue pas le 1° argument, évalue le second (fonction spéciale)
  - a pour effet d'ajouter l'association à l'environnement
  - ne retourne pas de résultat
  - définit l'**arité** et le **corps** de la fonction

(define (addition x y) (+ x y)) → Indéfini



Symbole	Valeur
addition	**(lambda (x y) (+ x y))**

Quelques fonctions arithmétiques

n-aire

- (+ 2 3 5 3) → 13
- (* 2 3 5) → 30
- (- 2 3 4) → -5
- (/ 2 3 3) → 0.222222222222222

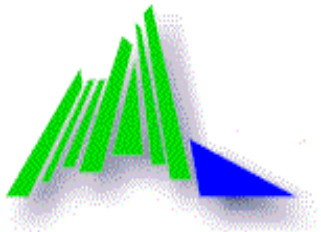
binaire

- (**quotient** 5 2) → 2
- (**remainder** 5 2) → 1
- (**modulo** 5 2) → 1
- (**expt** 2 3) → 8

(**remainder** -5 2)	→ -1
(**modulo** -5 2)	→ 1
(**remainder** 5 -2)	→ 1
(**modulo** 5 -2)	→ -1

unaire

- (**abs** -3) → 3
- (**sqrt** 9) → 3

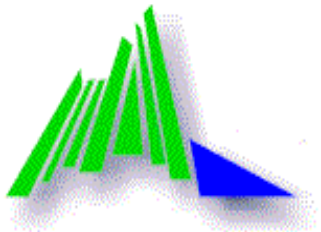


Quelques fonctions arithmétiques
Exercices d'évaluation

- On souhaite évaluer les deux expressions suivantes dans l'environnement {<x, 3>, <y, 2>, <z, 4>}:
  - (+ x (expt z y)) →
  - (- (sqrt z) (* x 2 z))→

a. Donner le résultat de l'évaluation

b. Représenter ces expressions sous forme d'arbre et montrer comment elles sont évaluées



Fonctions booléennes

- not : négation, fonction unaire
  - (not #t) → #f
  - (not #f) → #t
- and : et logique, fonction n-aire
  - (and ...#f ...) → #f
  - (and #t #t #t) → #t
- or : ou logique, fonction n-aire
  - (or … #t ...) → #t
  - (or #f #f #f) → #f

☞(or (= 2 2) (= 3 (/ 5 0))) → #t, sans erreur

☞(and (= 2 3) (= 3 (/ 5 0))) → #f, sans erreur

Fonctions relationnelles

- Fonctions n-aires
- Résultat booléen (#t ou #f)
- =, <, >, <=, >=

(= 2 3 4) → #f

(<= x 5)
- → #t si x est inférieur ou égal à 5
- →#f si x est supérieur à 5

Fonctions prédicats

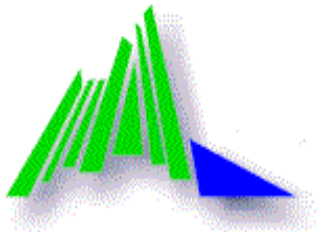
- Résultat Booléen (#t, #f)
- Le nom se termine par **?**

- Equal?
  - fonction binaire
  - teste l'égalité de deux expressions (pas seulement des nombres)

(equal? x « a »)

→ #t si x vaut « a »

→ #f si x différent de « a »



Fonctions prédicats

- zero?
  - unaire
  - teste si une expression est nulle

(zero? x)

→ #t si x vaut 0

→ #f si x différent de 0

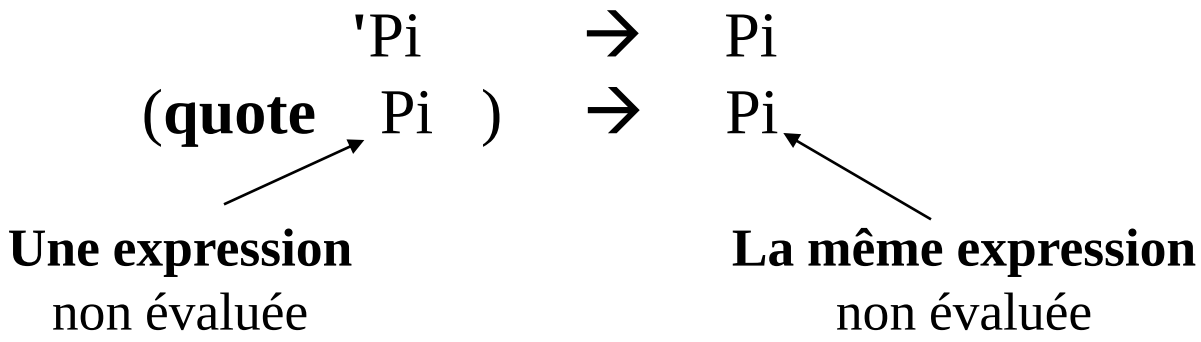
- boolean?, number?, symbol?, string?, integer?, real?...
  - unaire
  - teste le type d'une expression

(integer? x)

→ #t si x est un nombre entier

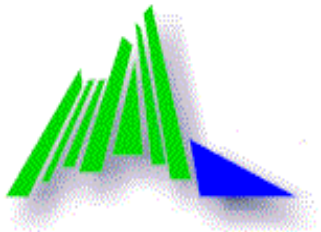
→ #f si x n'est pas un nombre entier

Le mécanisme de citation



- L'application de la fonction quote retourne l'expression non évaluée
- permet de manipuler des valeurs symboliques

```
>(define a 10)
> a
10
> ' a
a
```



Le mécanisme de citation

- (define robot R2D2)
  - ☞problème car R2D2 doit être évalué
  - ☞symbole non défini
- (define robot 'R2D2)
  - ☞crée la liaison: robot ⟶ R2D2
  - ☞ *robot* variable symbolique de valeur R2D2
- (define l (+ 5 6))
  - ☞crée la liaison: l ⟶ 11
- (define l '(+ 5 6))
  - ☞crée la liaison : l ⟶ (+ 5 6)