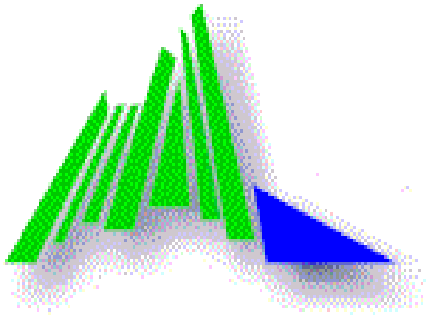
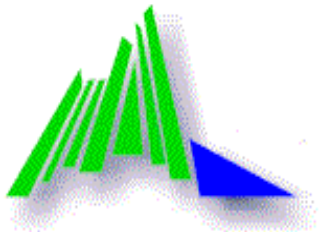




# Les fonctions en Scheme (suite)



# Séquence et affectation

- (**begin** <*suite d'expressions*>)
  - évaluation en séquence des expressions
  - retourne le résultat de la dernière évaluation
  - Exemple : (begin (+ 5 1) (* 6 5)) → 30

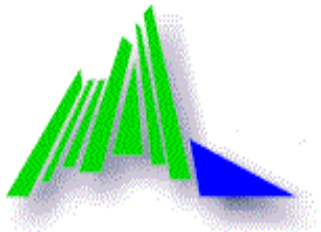
- (**set!** *symbole expression*)
  - evalue *expression* et affecte le résultat à symbole
  - ne retourne rien
  - attention *symbole* doit être défini avant

    (define x 10) →indéfini

    (set! x (+ x 1)) →indéfini

    x → 11

☞ Ne pas abuser de l'affectation



# Fonction conditionnelle If

- (**if** *expr_test expr_alors expr_sinon*)
  - fonction spéciale :

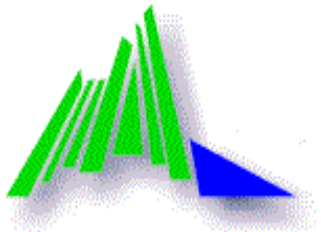
    évalue soit *expr_alors* soit *expr_sinon* en fonction de *expr_test*

  - Exemple :

    (if (< x  0)
    'négatif
    'positif_ou_nul)

  - Forme simplifiée : (if *expr_test expr_alors*)

    ☞ Attention : *expr_alors* et *expr_sinon* représentent une expression unique. Pour évaluer plusieurs expressions utiliser begin



# Fonction conditionnelle Cond

- **(cond**

  (*expr_test_1 <suite d 'expressions_1>*)

  (*expr_test_2 <suite d 'expressions_2>*)

  …

  (*expr_test_n <suite d 'expressions_n>)*

  (**else** *<suite d 'expressions>*)

  )

  - Fonction n-aire, généralisation de la fonction IF
  - fonction spéciale :

    1) évalue successivement chaque test jusqu'à trouver un test vrai ou trouver le mot clé **else**

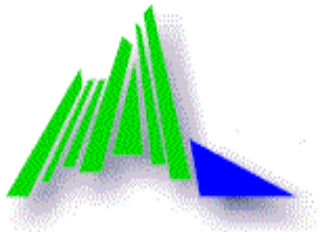
    2) évalue la suite d'expressions correspondantes et retourne le résultat de la dernière expression évaluée.

# Fonction conditionnelle Cond

```
(cond
    ((> 3 5) (+ 100 3))
    ((> 10 1) (+ 100 4))
    ((< 0 2) (+ 100 5))
    (else 88)) → 104
```

```
(define (signe x)
    (cond
        ((< x 0) « le nombre est négatif »)
        ((= x 0) « le nombre est nul »)
        (else « le nombre est positif »))
)
```

```
(define (couleur animal)
   (cond
    ((equal? animal 'elephant)  '(couleur : gris))
    ((equal? animal 'zebre) '(couleur  : blanc_raye_noir))
    ((equal? animal 'chien) '(couleur : je ne sais pas))
    (else '( je ne connais pas cet animal)))
)
```



# Notion d'environnement temporaire (environnement local)

- (**let** ((*variable*$_1$ *expression*$_1$)
  (*variable*$_2$ *expression*$_2$)
  ...
  (*variable*$_n$ *expression*$_n$)
  )
  *expression*
  )
- Fonction binaire
- Définition et utilisation de variables temporaires (locales)
- Le premier argument est une **liste de couples** (en bleu). Un couple associe un symbole (variable) à une expression.
- Le second argument est le **corps** de la fonction let. C'est une expression (en rose) .

# Notion d'environnement temporaire (environnement local)

- (**let** ((*variable*$_1$ *expression*$_1$)
  (*variable*$_2$ *expression*$_2$)
  ...
  (*variable*$_n$ *expression*$_n$)
  )
  *expression*
  )

| **Environnement *initial (global)*** |
|---|

| **Environnement *Et*** | |
|---|---|
| Symbole | Valeur |
| **variable$_1$** | **eval_expression$_1$** |
| **variable$_2$** | **eval_expression$_2$** |
| … | … |
| **Variable$_n$** | **eval_expression$_n$** |

- Application d'une fonction let :
  1) création d'un environnement temporaire *Et*, lié (lien hiérarchique) à l'environnement *initial*, appelé également environnement *global*,
  2) pour chaque couple (*variable*$_i$ *expression*$_i$) évaluation de *expression*$_i$ et création de la liaison dans l'environnement *Et*,
  3) évaluation du corps *expression* dans l'environnement *Et*,
  4) suppression de l'environnement *Et*.

☞ Les *expression*$_i$ ne peuvent pas utiliser les variables du let

# Notion d'environnement temporaire (environnement local)

- (**let** ((*variable*$_1$ *expression*$_1$)
  (*variable*$_2$ *expression*$_2$)
  ...
  (*variable*$_n$ *expression*$_n$)
  )
  *expression*
  )

| **Environnement *initial (global)*** |
|---|

| **Environnement *Et*** | |
|---|---|
| Symbole | Valeur |
| **variable$_1$** | **eval_expression$_1$** |
| **variable$_2$** | **eval_expression$_2$** |
| **…** | **…** |
| **Variable$_n$** | **eval_expression$_n$** |

- Si l'*expression* contient un ou plusieurs symboles, deux cas peuvent se produire lors de l'évaluation :
  - le symbole est une des variables du let : l'évaluation retourne la valeur associée dans l'environnement *Et*,
  - le symbole n'est pas une variable du let : l'évaluation retourne la valeur de l'environnement que l'on aurait trouvée sans le let (environnement initial).

# Notion d'environnement temporaire
# Exemples

- (**let** ((*x 5*)
  (*y 3*)
  )
  (* *x y*)
  ) → 15

| **Environnement *initial*** |
|---|

| Environnement *Et* | |
|---|---|
| **symbole** | **valeur** |
| x | 5 |
| y | 3 |

- (define x 100)
- (**let** ((*x 2*)
  (*y (+ x 3)*)
  )
  (* *x y*)
  ) → 206

| Environnement *initial* | |
|---|---|
| **symbole** | **valeur** |
| x | 100 |

| Environnement *Et* | |
|---|---|
| **symbole** | **valeur** |
| x | 2 |
| y | 103 |

Liens hiérarchiques

# Notion d'environnement temporaire
# Exemple de hiérarchie

- ( let ((x 100)
  )
  (**let** ((*y 2*)
  (*z (+ x 3)*)
  )
  *(* y z)*
  *)*
  ) → 206

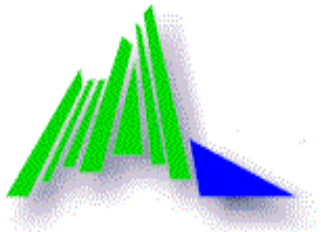
| **Environnement *initial*** |
|---|

| Environnement *Et1* | |
|---|---|
| **symbole** | **valeur** |
| x | 100 |

Liens hiérarchiques

| Environnement *Et2* | |
|---|---|
| **symbole** | **valeur** |
| y | 2 |
| z | 103 |

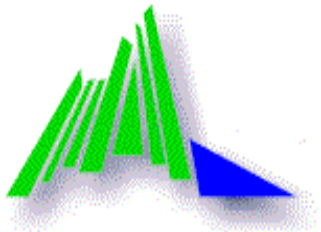
Liens hiérarchiques



# Imbrication de let - simplification d'écriture let*

- ( let ((x 100)
  )
  (**let** ((*y 2*)
  (*z (+ x 3)*)
  )
  *(* y z)*
  *)*
  ) → 206

( let* ((x 100)
(*y 2*)
(*z (+ x 3)*)
)
*(* y z)*
) → 206



# Entrée/Sortie

- (**Read** )
  - lecture d'une expression, retourne l'expression sans l'évaluer
  - Exemples

| **Lecture sans mémorisation du résultat** | **Lecture avec mémorisation du résultat (variable globale)** | **Lecture avec mémorisation du résultat (variable locale)** |
|---|---|---|
| ➢(read) ; début lecture<br>➢(+ 2 3) ; expression saisie<br>➢(+ 2 3) ; expression résultat | ➢(define x (read)) ; début lecture<br>➢(+ 2 3) ; expression saisie<br>➢ x → (+ 2 3) ; valeur variable | ➢(let (( x (read)) ; début lecture<br>)<br>x)<br>➢(+ 2 3) ; expression saisie<br>➢(+ 2 3) ; valeur variable |

- (**display** expression)
  - affiche le résultat de l'évaluation de *expression*, résultat indéfini
  - fonction unaire
- (**newline**)
  - passage à la ligne, résultat indéfini

# Entrée/Sortie

```
(define (f x)
    (cond ((number? x) (display x) (display  « est un nombre » ))
          ((symbol? x) (display x) (display « est un symbole »))
          (else (display x) (display « : je ne sais pas ce que c'est »)))
```

(f 5) → indéfini

5 est un nombre

(f « bonjour ») → indéfini

bonjour : je ne sais pas ce que c'est