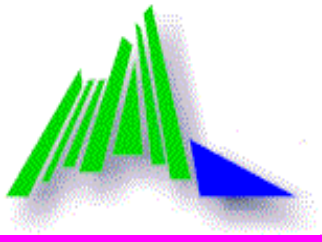


Université de Pau et des Pays de l'Adour

# Programmation avancée en Scheme (programmation d'ordre supérieur)



# Fonction d'évaluation

---

## ◆ Evaluer une expression

**(eval *expression environnement*)**

- $(+ 5 10) \rightarrow 15$
- $'(+ 5 10) \rightarrow (+ 5 10)$
- $(\text{eval } '(+ 5 10) \text{ (interaction-environment)}) \rightarrow 15$
  
- $(\text{define } x \text{ '+})$
- $x \rightarrow +$
- $(+ 5 10) \rightarrow 15$
- $(x 5 10) \rightarrow \text{erreur}$
- $((\text{eval } x \text{ (interaction-environment)}) 5 10) \rightarrow 15$



## Fonction *apply*

---

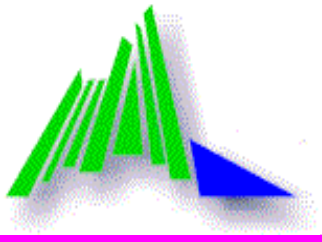
- ◆ Applique une fonction à une liste d'arguments

(**apply** *fonction* <*liste*>)

- ◆ Si  $liste = (a_1 a_2 \dots a_n)$  et  $fonction = f$

Equivaut à évaluer : (**f**  $a_1 a_2 \dots a_n$ )

- (apply + '(5 4 6))  $\rightarrow$  15
- (apply cons '(a b))  $\rightarrow$  (a . b)



## Fonction *map*

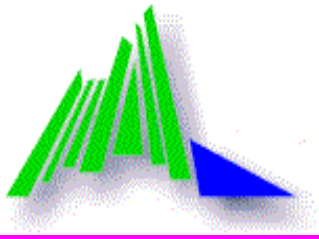
- ◆ Applique une fonction à chaque élément d'une liste  
(**map** *fonction* <liste>)

Si  $liste = (a_1 a_2 \dots a_n)$  et  $fonction = f$

Equivaut à évaluer :  $(f a_1)$  et  $(f a_2)$  et ...  $(f a_n)$

- ◆ Retourne la liste des évaluations :  $((f a_1) (f a_2) \dots (f a_n))$   
(map number? '(5 « toto » x 5))  $\rightarrow$  (#t #f #f #t)

- ◆ Généralisable à des fonctions n-aires  
(**map** *fonction* <liste1> <liste2>...<listen>)  
(map cons '(a b c) '(1 2 3))  $\rightarrow$  ((a . 1) (b . 2) (c . 3))

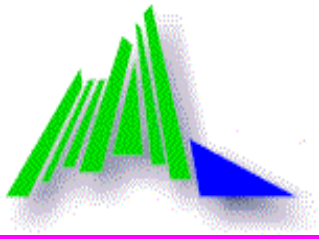


## Exemple 1

---

- ◆ Ecrire une fonction **applique** qui prend en paramètres une liste non vide  $lf$  de fonctions unaires et un entier  $n$  positif ou nul et qui retourne la liste des résultats successifs de l'application de chaque fonction de  $lf$  au nombre entier  $n$ .


Remarque : On considère que les valeurs des paramètres sont valides.



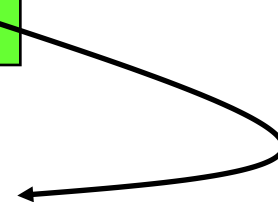
# Retour sur l'évaluation d'une lambda expression

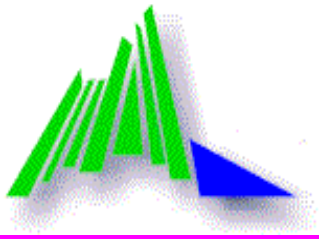
- ◆ Le résultat de l'évaluation d'une lambda est **une fermeture** (closure). Structure précisant les paramètres formels, le corps et l'environnement d'évaluation.

évaluation de (lambda (x y) (+ x y))

Args :	Corps :	Env. :
( x y)	(+ x y)	

**Environnement  
d'évaluation**  
[liaisons initiales]





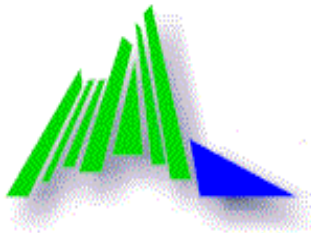
## Exemple 2

- ◆ On considère un environnement  $e$  dans lequel sont définis :
  - une variable  $v$  à valeur entière positive ou nulle,
  - une variable  $l$  contenant une liste de fonctions unaires,
  - une fonction  $f$  à deux paramètres  $p1$  et  $p2$ .
- ◆ Le tableau ci-dessous décrit le comportement de  $f$  en fonction de la valeur de ses deux paramètres.

$P1$ (symbole)	$P2$	Action effectuée par $f$
<b>get-v</b>	()	Retourne la valeur courante de la variable $v$
<b>set-v !</b>	Expression entière	Affecte la valeur de $p2$ à la variable $v$
<b>add-l !</b>	Un symbole de fonction unaire	Modifie la liste $l$ en lui ajoutant la valeur de $p2$ (en tête de liste)
<b>execute</b>	()	Retourne la liste des résultats successifs de l'application de chaque fonction de $l$ à la variable $v$

1. Ecrire la fonction  $f$ .
2. Ecrire la fonction **gen-env** sans paramètre qui crée un environnement local selon le modèle de  $e$  et retourne comme résultat l'évaluation de  $f$ . Initialement  $v$  est nulle et  $l$  est vide.

Remarque : On considère que les valeurs des paramètres sont valides



## Exemple 2

<i>P1</i> (symbole)	<i>P2</i>	Action effectuée par <i>f</i>
<b>get-v</b>	()	Retourne la valeur courante de la variable <i>v</i>
<b>set-v !</b>	Expression entière	Affecte la valeur de <i>p2</i> à la variable <i>v</i>
<b>add-l !</b>	Un symbole de fonction unaire	Modifie la liste <i>l</i> en lui ajoutant la valeur <i>dep2</i> (en tête de liste)
<b>execute</b>	()	Retourne la liste des résultats successifs de l'application de chaque fonction de <i>l</i> à la variable <i>v</i>

Soit les définitions suivantes :

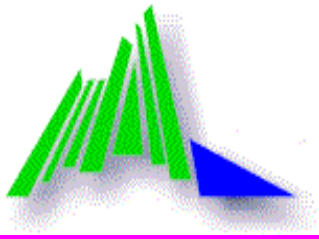
*(define f1 (gen-env))*

*(define f2 (gen-env))*

1. Donner l'expression à évaluer pour affecter la valeur 4 à la variable *v* de *f1*.
2. Donner l'expression à évaluer pour obtenir la valeur de la variable *v* de *f2*. Donner également cette valeur.
3. Ecrire une fonction **add-exe** qui prend en paramètre deux symboles de fonctions unaires *fu1* et *fu2*, et dont l'exécution ajoute *fu1* et *fu2* à la liste *l* de *f1* puis retourne la liste des résultats successifs de l'application de chaque fonction de *l* de *f1* à la valeur de la variable *v* de *f1*.

Remarque : On considère que les valeurs des paramètres sont valides.





# Fonctions à arité variable

- ◆ Avec paramètres obligatoires
  - (lambda (x1 x2 ... xn . Lparam)  
expression)
- ◆ Sans paramètres obligatoires
  - (lambda Lparam  
Expression )

Exemple :

```
(define mcons (lambda (a . l) ; cons itéré 1 paramètre au moins
  (cons a
    (if (null? l)
        '()
        (apply mcons l))))
```

(mcons 1) → (1)

(mcons 1 2) → (1 2)

(mcons 1 2 3 '(a b)) → (1 2 3 (a b))



# Macro caractères

---

## ◆ Quasi-citation et virgule

- ``s` = (quasiquote s)
- Se comporte presque comme quote : ``(a b) → (a b)`
- Différence concerne les listes avec possibilité d'évaluation sélective grâce au caractère ``,``
  - `(define x 1)`
  - `(define y '(u v))`
  - `'(a b x y) → (a b x y)`
  - ``(a b ,x ,y) → (a b 1 (u v))`



# Macro caractères

---

- ◆ Quasi-citation et couple virgule arobasque
  - (define x 1)
  - (define y '(u v))
  - `(a b ,x ,@y) → (a b 1 u v)
- L'expression qui suit ,@ est évaluée. Sa valeur doit être une liste et c'est le contenu de la liste qui est mis à sa place