

Université de Pau et des Pays de l'Adour

Programmation fonctionnelle

eric.gouarderes@univ-pau.fr

<https://webcampus.univ-pau.fr/courses/PROGFONC/>



Organisation de l'UE

◆ Objectif de l'unité d'enseignement

Initier les étudiants à une approche fonctionnelle et récursive de la programmation à travers le langage Scheme.

◆ Contenu de l'enseignement

- 1. Evaluation d'expressions préfixées
- 2. Syntaxe du langage scheme
- 3. Variables et fonctions
- 4. Récursivité et application au traitement de listes

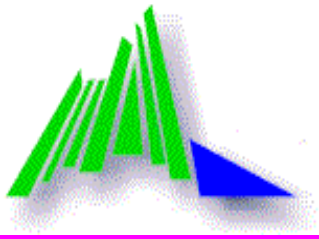
◆ Intervenants

- Eric Gouardères (responsable)
- Samson Pierre



Organisation de l'UE

- ◆ Volume Horaire : 39h00
- ◆ ECTS : 4
- ◆ Organisation des séances
C : 10.5 h – TD : 13.5h – TP: 15 h
- ◆ Evaluation
 - Contrôle continu (au moins un contrôle écrit) : 30 %
 - Examen écrit : 70 %

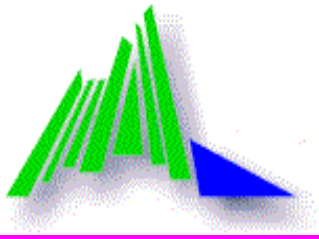


Organisation de l'UE

◆ Planning prévisionnel

Num. Sem.	Date Sem.	Cours (1 gr.)	TD (3 gr.)			TP (4 gr.)			
			Cr1	Gr2	Gr3	GrA	GrB	GrC	
37	11-sept	E. Gouardères - Fonctionnel 1 + TD 1: Présentation de Scheme, évaluation d'expressions							
38	18-sept	E. Gouardères - Fonctionnel 2 : définition de fonction, définition de variables, fonctions de base, citation							
39	25-sept	E. Gouardères - Fonctionnel 3 : affectation, séquence, condition, environnement, Entrée/Sortie	S. Pierre - TD2 : évaluation de variables						
40	02-oct	E. Gouardères - Fonctionnel 4 : Récursivité	S. Pierre - TD3 : fonctions simples						
41	08-oct	E. Gouardères - Fonctionnel 5 : les listes	S. Pierre - TD4 : fonctions conditionnelles			S. Pierre - TP1 : Dr Racket, évaluations, fonctions simples, fonctions E/S, fonctions conditionnelles			E. Gouardères
42	15-oct	E. Gouardères - Fonctionnel 5 : listes et récursivité	S. Pierre - TD5 : récursivité			S. Pierre - TP1 : Dr Racket, évaluations, fonctions simples, fonctions E/S, fonctions conditionnelles			E. Gouardères
43	23-oct	E. Gouardères - Fonctionnel 6 : programmation d'ordre E. Gouardères, S. Pierre - Contrôle (1 grand amphi)	S. Pierre - TD5 : récursivité			S. Pierre - TP2 : récursivité calculette			E. Gouardères
45	06-nov		E. Gouardères - TD6 : listes			S. Pierre - TP2 : récursivité calculette			E. Gouardères
46	13-nov		E. Gouardères - TD6 : listes			S. Pierre - TP2 : récursivité calculette			E. Gouardères
47	20-nov		E. Gouardères - TD6 : listes non linéaires			S. Pierre - TP3 : listes			E. Gouardères
48	27-nov					S. Pierre - TP3 : listes			E. Gouardères
49	04-déc					S. Pierre - TP3 : listes			E. Gouardères
50	11-déc					S. Pierre - TP4 : Backtracking			E. Gouardères

Attention, ce planning peut être modifié. Consulter régulièrement votre emploi du temps.



Langage fonctionnel

- ◆ Ecrire un programme : définir une fonction

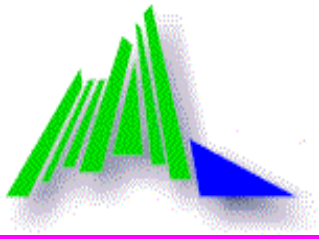
$$f : x, y \longrightarrow x + y$$

- ◆ Exécuter un programme : appliquer une fonction

$$(f \ 2 \ 5)$$

- ◆ Récursivité

$$\begin{array}{l} \text{N} \times \text{N} \\ f : x, n \longrightarrow x^n \\ \left\{ \begin{array}{ll} 1 & n=0 \\ x \cdot x^{n-1} & n>0 \end{array} \right. \end{array}$$



Définition d'une fonction

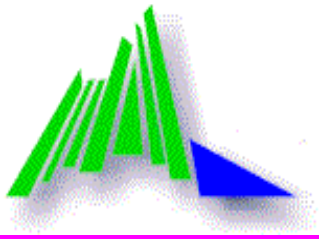
- ◆ **Spécification** d'une fonction : un **domaine de départ**, un **domaine d'arrivée** et le **corps** de la fonction (expression de calcul du résultat).

$$\begin{array}{ccc} \mathbb{R}_+ & \longrightarrow & \mathbb{R}_+ \\ \text{racine : } x & \longrightarrow & \sqrt{x} \end{array}$$

$$\begin{array}{ccc} \mathbb{R} \times \mathbb{R} & \longrightarrow & \mathbb{R} \\ \text{addition : } x, y & \longrightarrow & x + y \end{array}$$

$$\begin{array}{ccc} D_1 \times D_2 \times \dots \times D_n & \longrightarrow & A \\ f : x_1, x_2, \dots, x_n & \longrightarrow & f(x_1, x_2, \dots, x_n) \end{array}$$

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



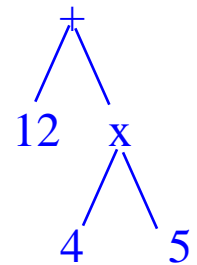
Notion d'expression

◆ Moyen de dialogue

- mots + règles de syntaxe
- sens de lecture (de la gauche vers la droite)

◆ Exemple : expression arithmétique

- notation commune (linéaire) : $(12 + (4 \times 5))$
- représentation sous forme d'arborescence
 - expression (parenthèses) : arbre
 - opérateur : racine
 - opérande : sous-arbre (éventuellement réduit à une feuille)





Exercice : représenter sous forme d'arbres

$$(3 + 4) * 5$$

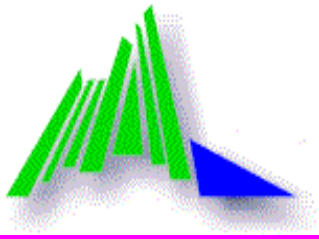
$$3 + (4 * 5)$$

$$4 + 5 + (6 * 7)$$

$$125 * (12 - \frac{204}{6})$$

$$\frac{1 + \sqrt{2}}{1 - \sqrt{2}}$$

$$e(-\frac{1}{3})$$



Différentes notations

- ◆ Infixée : opérateur situé entre les opérandes (opérateur binaire)

$1 + 2$

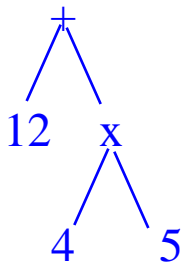
- ◆ Préfixée : opérateur situé avant opérandes

$+ 1 2, \sin x$

- ◆ Postfixée : opérateur situé après opérandes

$1 2 +, x \sin$

- ◆ Ordre de lecture d'un arbre



— infixée : sous-arbre-gauche racine sous-arbre-droit $\rightarrow (12 + (4 \times 5))$

— préfixée : racine sous-arbre-gauche sous-arbre-droit $\rightarrow (+ 12 (x 4 5))$

— postfixée : sous-arbre-gauche sous-arbre-droit racine $\rightarrow (12 (4 5 x) +)$

— structure de l'arbre \rightarrow parenthèses

— si arité opérateurs connue, parenthèses facultatives en préfixé et postfixé (en infixé, il faut aussi connaître les règles de priorité)

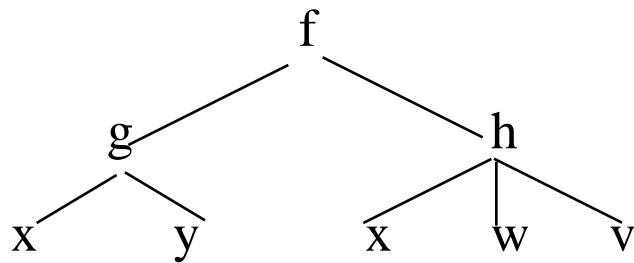
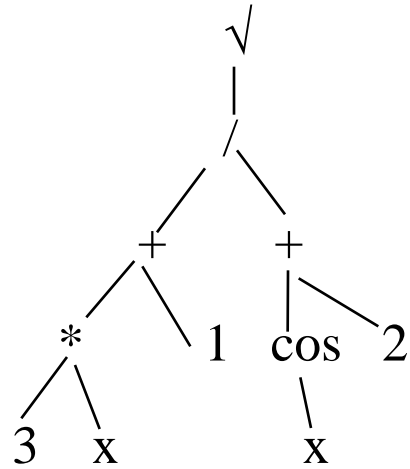
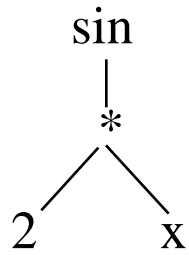
infixée $\rightarrow 12 + 4 \times 5$

préfixée $\rightarrow + 12 \times 4 5$

postfixée $\rightarrow 12 4 5 \times +$



Examples




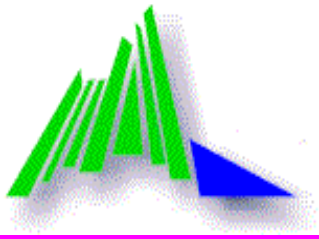


Langage Scheme

- ◆ Langage fonctionnel de la famille LISP (Steele et Sussman, MIT 1975)
 - traitement de listes
 - expressions symboliques
- ◆ Langage interprété
 - interpréteur : programme de traduction d'un programme source en langage machine.
 - généralement une ligne de commande (saisie des instructions) et un évaluateur (calcul)
 - le programme source est traduit au moment de l'exécution
 - ☞ interpréteur ≠ compilateur
 - évaluateur : cycle R.E.P.L.
comparable au fonctionnement
d'une calculatrice

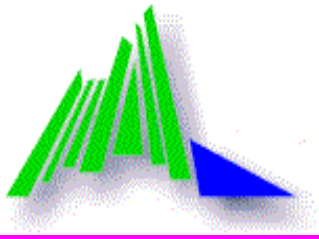
Read
Eval
Print
Listen





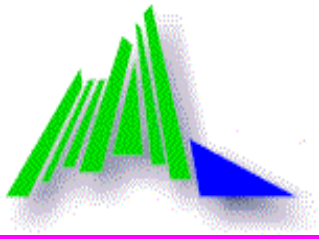
Les expressions Scheme

- Notation préfixée totalement parenthésée
- simple
 - nombres : 5, 6.5, 1/3
 - les valeurs booléennes : #t, #f
 - chaînes : « bonjour », « au revoir »
- symbole :
 - nom d'une expression (variable, fonction)
 - valeur symbolique
 - Ex : +, -, toto, and, or...
- liste :
 - application d'une fonction : $5 + 6 \rightarrow (+\ 5\ 6)$
 - ensemble de données : $(a, b, c) \rightarrow (a\ b\ c)$
- lambda expression ou fonction anonyme
Ex : $(\text{lambda}\ (x\ y)\ (+\ x\ y))$



Quelques références

- **La programmation, une approche fonctionnelle et récursive avec Scheme** - Laurent Ardit, Stéphane Ducasse - Eyrolles, 1996
- **Débuter la programmation avec Scheme** - Jean-christophe Routier et Eric Wegrzynowski -International Thomson Publishing, 1997
- **Programmer avec Scheme, de la pratique à la théorie** - Jacques Chazarain - International Thomson Publishing, 1996
- **L'environnement de programmation DrRacket** : <https://racket-lang.org/>

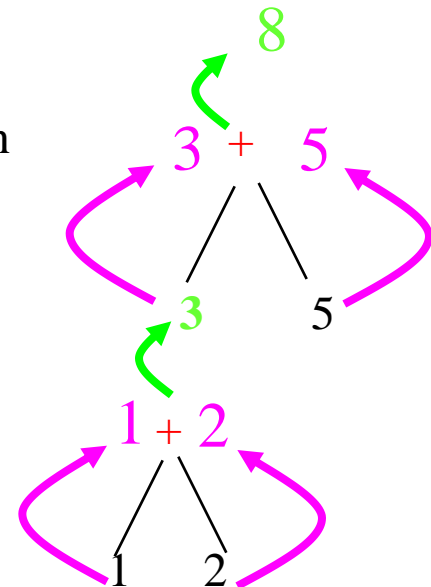


Evaluation des expressions

- ◆ Les expressions simples s'auto-évaluent
- ◆ Les symboles :
 - notion d'environnement : ensemble de liaisons symbole/valeur
 - ☞ contient tous les symboles (prédéfinis par la plateforme et définis par le programmeur)
 - évaluer un symbole c'est retourner sa valeur

- ◆ Les listes :
 - évaluer une liste revient à appliquer une fonction
 - 1) évaluer le 1^{er} élément (symbole de fonction)
 - 2) évaluer les autres éléments (arguments)
(sauf pour les fonctions spéciales)
 - 3) appliquer la fonction

Ex : évaluer $(+ (+ 1 2) 5)$





Quelques fonctions simples

- ◆ Arithmétique : +, -, /, *, sqrt, remainder, quotient
 - (sqrt 9) \rightarrow 3

- ◆ Booléenne : and , or , not
 - (and #t #f) \rightarrow #f

- ◆ Relationnelle : =, <, >, <=, >=
 - (= X 0)

- ◆ Prédicats : equal?, zero?, number?...
 - (equal? X « oui »)



Ecrire en Scheme les expressions suivantes

$$125 * (12 - \frac{204}{6}) \quad , \quad \frac{1 + \sqrt{2}}{1 - \sqrt{2}} \quad , \quad 2^{-\frac{1}{3}}$$

$$x \geq 0 \text{ et } x \leq 5 \quad , \quad x > 0 \text{ et } y > 0 \text{ et } x + y < 4$$

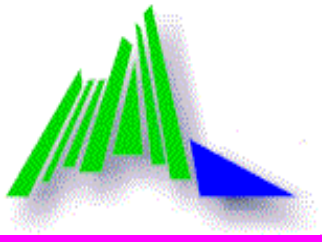
$$(x > 5 \text{ et } y > x - 2) \text{ ou } (7 < x < 10 \text{ et } y > 2 * x)$$

Sachant que Scheme offre les fonctions suivantes :

sqrt : Réel \rightarrow Réel expt : Réel, Réel \rightarrow Réel

$$x \mapsto \sqrt{x}$$

$$a, x \mapsto a^x$$



Exemples d'évaluation d'expression : résultats

Expression

Résultat d'évaluation

- 486

→ 486

- #t

→ #t

- « bonjour »

→ « bonjour »

- (+ 2 3)

→ 5

- (+ (- 3 1) (+ 2 3) 4 (* 3 7))

→ 32

- (and (> 12 45) (= 3 (/ 12 4)))

→ #f

- (or #t (< 12 3))

→ #t



Exemples d'évaluation d'expression : méthode

Expression

- 486
- #t
- « bonjour »
- (+ 2 3)
- (+ (- 3 1) (+ 2 3) 4 (* 3 7))
- (and (> 12 45) (= 3 (/ 12 4)))
- (or #t (< 12 3))

**Représenter les expressions de
manière arborescente et montrer
comment elles sont évaluées**