# Débuts en Scheme



Évaluer une expression

Définir une fonction

# Notion de fonction

- Une fonction prend des arguments et retourne un résultat
- Arguments et résultat peuvent être de n'importe quel type :
  - Nombre
  - Booléen
  - Caractère
  - Chaîne de caractères
  - Liste
  - Fonction

# Écriture de l'appel à une fonction (1)

- Syntaxe :
  - Parenthèse ouvrante
  - Nom de la fonction
  - Espace
  - Premier argument
  - Espace
  - Deuxième argument
  - Etc.
  - Parenthèse fermante

(NomFct Arg1 Arg2 … Argn)

# Écriture de l'appel à une fonction (2)

- Sémantique : il faut donner à la fonction le bon nombre d'arguments, et du bon type

- Exemples :
  - (+ 5 13) retourne 18
  - (- 10 b) retourne la différence
    si b a une valeur numérique, une erreur sinon
  - (+ (* 2 5) (- 3 1)) retourne 12
  - (* 5) n'est pas correct
  - (/ 5  "a") non plus

# Évaluation de l'appel à une fonction

- Lorsqu'on lui fournit un appel de fonction, Scheme
  - Évalue chacun des arguments
  - Regarde s'il connaît la fonction,
    sinon affiche un message d'erreur
  - Applique la fonction aux résultats de l'évaluation des arguments
  - Affiche le résultat

- C'est un processus récursif

# Exemples d'erreurs

- (1 + 2) → erreur : 1 n'est pas une fonction
- (toto (1 2 3)) → erreur : 1 n'est pas une fonction

- Dans certains cas particuliers, les arguments ne sont pas évalués avant l'application de la fonction.
  On parle alors de forme spéciale plutôt que de fonction

# Les variables

- Dans le langage Scheme, une variable se nomme symbole
- L'affectation revient à attribuer une valeur à un symbole.
  On utilise pour cela la forme spéciale define
- Exemples :
  - (define a 5)
  - (define b 2)
  - (+ a b) → 7

# La forme spéciale quote

- La forme spéciale quote permet d'empêcher l'évaluation

- Exemples :
  - (define a 5)
  - a → 5

  - (quote a) → a
  - (quote (+ 1 2)) → (+ 1 2)
  - '(+ 1 2) → (+ 1 2)

# La forme spéciale eval

- À l'inverse de quote, eval force l'évaluation
- Exemples :

  (eval '(+ 3 2)) → 5

  (define toto 5)
  (define tata toto)
  tata → 5

  (define titi 'toto)
  titi → toto
  (eval titi) → 5

# Définition d'une fonction

- Syntaxe :

```
(define fonction
        (lambda liste-des-arguments
                instructions))
```

- Exemple :

```
(define plus-1
  (lambda (x)
    (+ x 1)))
```

- Test : (plus-1 3) → 4

# SPÉCIFICATION D'UNE FONCTION

```
; description de ce que fait la fonction
(define fonction ; → type du résultat
    (lambda liste-des-arguments ; type des arguments
        instructions))
```

Exemple :

```
    ; ajoute 1 à un nombre
        (define plus-1 ; → un nombre
            (lambda (x) ; x un nombre
                    (+ x 1)))
```

# L'ALTERNATIVE

- L'alternative est définie en Scheme par la forme spéciale if

- Syntaxe :

  (if condition valeur-si-vrai valeur-si-faux)

- Exemples :
  - (if (> 3 2) 'yes 'no) → yes
  - (if (> 2 3) 'yes 'no) → no
  - (if (> 3 2) (- 3 2) (+ 3 2)) → 1

# Quelques fonctions prédéfinies (1)

- Opérateurs arithmétiques :

  +, -, *, /, sqrt, min, max, abs, ...

  (sqrt 9) → 3
  (min 5 2 1 3) → 1

- Opérateurs booléens :

  not, or, and

  (not #t) → #f
  (and (> 3 2) (> 2 5)) → #f
  (or (> 3 2) (> 2 5)) → #t

# Quelques fonctions prédéfinies (2)

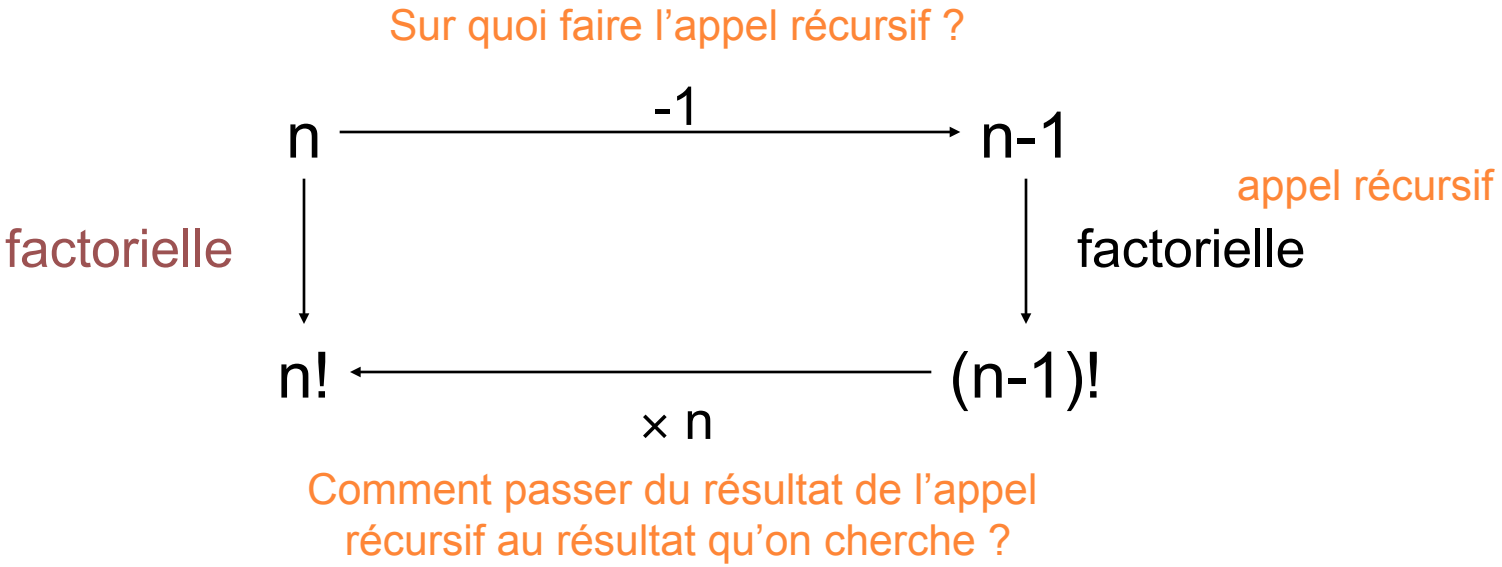
- Opérateurs de comparaison :
  - =, <, >, <=, >= pour les nombres
  - eq? pour tout sauf les listes et les chaînes de caractères
  - equal? pour tout y compris les listes

- Pour tester le type d'un objet :
  boolean?, number?, symbol?, string?

- modulo : reste de la division entière
  - (modulo 12  5) → 2
  - (modulo 5 12) → 5

# Ma première fonction récursive : choix de la méthode

- On veut écrire une fonction qui étant donné un entier n calcule n!



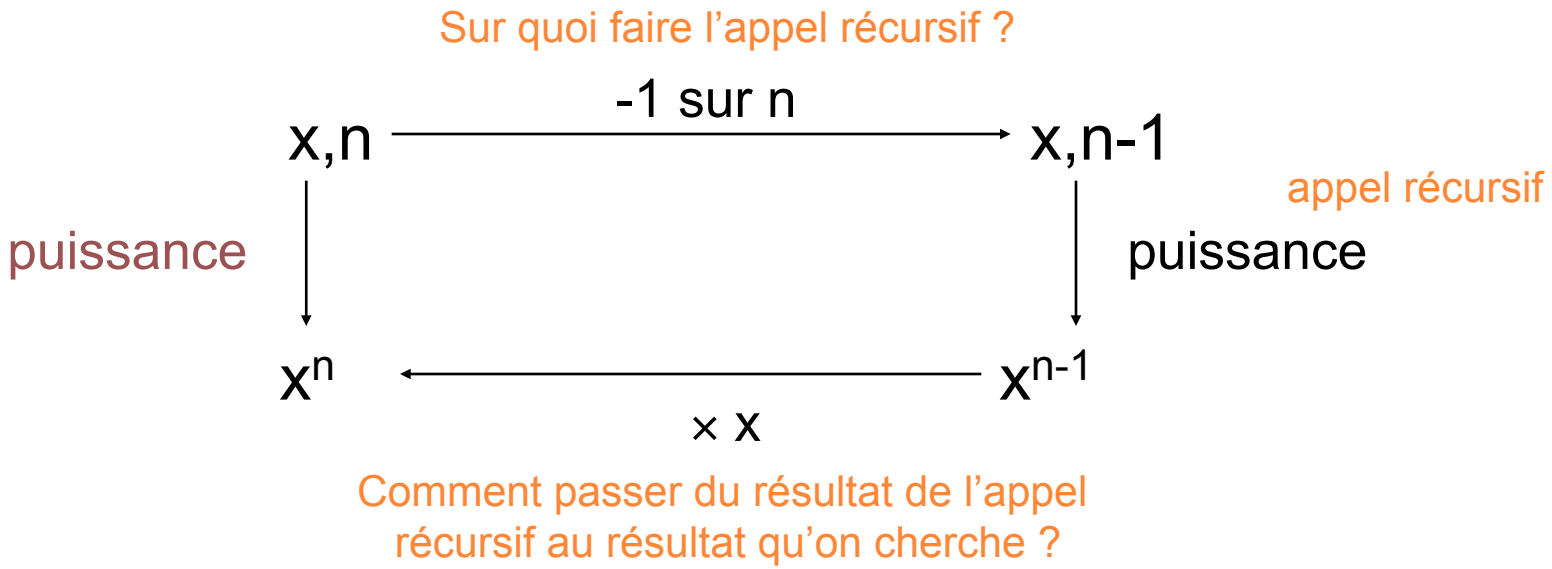
# Ma première fonction récursive : écriture

Cas d'arrêt : 0! = 1
Récursivité : n! = 1 x 2 x 3 x … x n = n x (n-1)!

```
(define factorielle ; → entier positif
  (lambda (n) ; n entier positif
        (if     (= n 0)
                1
                (* n (factorielle (- n 1))))))
```

# Une autre fonction récursive : choix de la méthode

- On veut écrire une fonction qui étant donné un nombre x et un entier positif n calcule $x^n$



# UNE AUTRE FONCTION RÉCURSIVE : ÉCRITURE

Cas d'arrêt : $X^0 = 1$

Récursivité : $X^n = X \times X \times \ldots \times X = X \times X^{(n-1)}$

```
(define puissance ; → nombre
  (lambda (x n) ; x nombre, n entier positif
      (if     (= n 0)
              1
              (* x (puissance x (- n 1))))))
```