INFO0054 - Programmation fonctionnelle

Répétition 1: l'interpréteur Scheme

Jean-Michel Begon

11 février 2020

Évaluations simples

Exercice 1.
Donner le résultat de l'évaluation des expressions suivantes :

```
3
#t
(+ 1 2)
(/ 2 3)
(+ (* 3 4) 10)
(* 3 (- 12 5))
(+ (+ 2 3) (+ 4 5))
(- (+ 5 8) (+ 2 4))
(define a 4)
a
'a
(define b a)
b
(define a 6)
a
b
(define d #t)
(define Robert 'Bob)
Robert
(null? 'a)
(number? 1)
(number? '1)
(number? #t)
(number? 'a)
(number? a)
(boolean? 3)
(boolean? #t)
(boolean? #f)
(boolean? '#t)
(boolean? 'd)
(boolean? d)
(symbol? 'b)
(symbol? #f)
(cons 'a '())
(cons 'a '(b))
(car '(a b))
(cdr '(a b))
(car (cdr '(a b)))
(cdr (cdr '(a b)))
(cadr '(a b c d))
(cadar '((a b) (c d) (e f)))
(null? (car '(())))
(equal? '(car '((b) c)) (cdr '(a b)))
```

Premières formes et fonctions

Exercice 2.
Calculer en une seule forme Scheme le nombre de secondes dans une année (non bissextile).

Exercice 3.
Écrire une forme qui rend `un` si `x` est égal à 1, ..., `cinq` si `x` est égal à 5 et `inconnu` sinon.

De là, définir une fonction `sayit` qui rend `un` si l'argument est 1, ..., `cinq` si l'argument est égal à 5 et `inconnu` sinon.

Exercice 4.
Donner le résultat de l'évaluation des expressions suivantes :

```
(lambda (y x) (cons x y))                    (id 1)
((lambda (y x) (cons x y)) '() 'a)           (id '(1 2 3))
(define id (lambda (x) x))                   (((id id) (id id)) 3)
```

Remarque : La valeur renvoyée par `(define ...)` n'est pas spécifiée.

Exercice 5.
Écrire une fonction `sum-int` prenant en argument un naturel n et calculant la somme des naturels inférieurs ou égaux à n.

Exercice 6.
Définir la fonction `sum-list` qui calcule la somme des éléments d'une liste de nombres.

Exercice 7.
Définir la fonction `mod` qui renvoie le modulo de deux nombres.

Remarque : la fonction *modulo* est prédéfinie, nous la redéfinissons sous un autre nom à titre d'exercice.

Exercice 8.
Écrire une fonction `big` qui prend comme arguments un nombre entier `n` et une liste `l` de nombres entiers, telle que `(big n l)` est la liste des éléments de `l` plus grands que `n`.

`(big 5 '(7 -3 6 1 -2 5 6))` $\Rightarrow$ `(7 6 6)`

Exercice 9.
Définir la fonction `prod-list` qui calcule le produit des éléments d'une liste de nombres.

Exercice 10.
Écrire une fonction `prefix-n` à deux arguments, une liste `ls` et un naturel `n`, et qui renvoie la listes des `n` premiers éléments de `ls`.