TDTP 2 : listes

À préparer avant la séance

• Pour chaque expression Scheme écrite ci-dessous, réfléchissez au résultat qu'elle devrait donner. Vérifiez avec Racket le résultat, et nous discuterons en TDTP des cas qui posent question.

```
(car (a b c d))
(car '(a b c d))
(cdr '(a b c d))
(cadr '(a b c d))
(cadr '(abc d))
(cdddr '(a b c d))
(cdadr '(a b c d))
(cdadr '(a (b c) d))
```

```
(cdr '(a (b c d)))
(cdr '((a b c d)))
(car '((a b c d)))
(cdadr '(a (b c)(d e)))

(cons a '(b c d))
(cons (a) '(b c d))
(cons '(a) '(b c d))
```

```
(cons '(a b) '(c d))
(cons 'a (cons'b'(c d)))
(cons '(a b) 'c)
(cons 'a (* 3 2))
(cons 'a '(* 3 2))

(list? (+ 2 3))
(list? '(+ 2 3))
```

• Définir en Scheme :

- une fonction qui retourne le deuxième élément d'une liste d'au moins deux éléments
  `(deuxieme '(a b c d))` → `b`
- une fonction qui calcule la longueur d'une liste (i.e. son nombre d'éléments)
  `(longueur '(a b c d))` → `4`

À faire pendant la séance

• Construction de listes

(TP) Construisez les listes suivantes en utilisant exclusivement la fonction *cons*.

(cons 'paon (cons (cons 'poule '()) '())) →
(paon (poule))

... → (() () poule)
... → (oie poule (poule) paon)
... → (poule oie poule ((poulet)))
... → (((poule) oie) paon)

• Accès aux éléments d'une liste

(TP) Quelle fonction faut-il utiliser pour extraire *oie* des listes suivantes ?

Par exemple, pour l'extraire de :
((poule oie pie) paon aigle), c'est cadar.

'(aigle (paon (pie oie)))
'(((oie) poule))
'(poule paon (oie) aigle)
'(paon (poule (poulet (oie))))

• Définition de fonctions

(TD) Définir en Scheme :

- une fonction qui retourne vrai si et seulement si une liste n'a qu'un seul élément
  `(singleton? '(a b c d))` → `#false`
  `(singleton? '(a))` → `#true`
- une fonction qui teste l'appartenance d'un élément à une liste
  `(estdans? 'b '(a b c d))` → `#true`
  `(estdans? 'e '(a b c d))` → `#false`

- une fonction qui retourne le $n^{ième}$ élément d'une liste
  `(nieme 3 '(a b c d))` → `c`
- une fonction qui insère un élément dans une liste après le $i^{ème}$ élément
  `(insere 'r 3 '(a b c d e f))` → `(a b c r d e f)`
- une fonction qui renverse une liste
  `(renverse '(a b c d))` → `(d c b a)`

TD Donner la spécification de la fonction mystère ci-dessous :

```
(define mystere
  (lambda (x l)
     (cond ((null? l) 0)
           ((equal? x (car l)) (+ 1 (mystere x (cdr l))))
           (else (mystere x (cdr l))))))
```

TP Définir en Scheme :

- une fonction qui retourne le dernier élément d'une liste non vide
  `(dernier '(a b c d))` → `d`
- une fonction qui échange les 2 premiers éléments d'une liste d'au moins 2 éléments
  `(echange '(a b c d))` -> `(b a c d)`
- une fonction qui répète chaque élément d'une liste
  `(repete '(a b c d))` → `(a a b b c c d d)`
- une fonction qui retourne la liste formée de tous les symboles d'une liste
  `(symboles '(a 2 b (6 z) "toto" 7 f))` → `(a b f)`
- une fonction qui remplace toutes les occurrences d'un élément e_1 dans une liste L par un autre élément e_2
  `(remplace 'o 'i '(b o n j o u r))` → `(b i n j i u r)`

Pour s'entrainer (exercices supplémentaires facultatifs)

• Écrire une fonction qui enlève le dernier élément d'une liste non vide
`(tsd '(a b c d))` → `(a b c)`

• Écrire une fonction qui supprime tous les éléments d'une liste qui sont égaux à un élément passé en paramètre.
`(supprime '(a b a d e f a) 'a)` → `(b d e f)`

• Écrire une fonction qui retourne la liste formée des n premiers éléments d'une liste.
`(premiers 3 '(a b c d e))` → `(a b c)`

• Écrire une fonction qui, étant donnée une liste de longueur paire, regroupe ses éléments deux par deux dans une liste.
`(regroupe '(a b c d e f))` → `((a b) (c d) (e f))`