TP numéro 1 : débuts en Scheme, listes

Connectez-vous à l'aide de votre numéro d'étudiant. Vous pouvez lancer DrRacket *via* le menu *Démarrer>Tous les programmes>Racket*.

Configuration de DrRacket : dans le menu Langage, Sélectionnez le langage *Niveau débutant avec abréviations pour les listes*, puis dans *Montrer les détails*, choisissez *write* pour *Style d'impression des résultats* (si vous ne voyez pas le bouton OK, cachez les détails). Utilisez le bouton *Exécuter* pour valider votre choix de langage.

La partie inférieure de la fenêtre vous permet d'évaluer des expressions. Vous pouvez l'utiliser pour effectuer la première partie du TP.

La partie supérieure de la fenêtre vous permet de définir des fonctions que vous pouvez enregistrer (bouton *Sauvegarder*) sur votre compte (l'extension ajoutée est `.rkt`). Prenez l'habitude de créer un fichier par TP.

Le bouton *Vérifier* vous permet de vérifier la syntaxe de votre fonction. Le bouton *Exécuter* compile la fonction, pour que vous puissiez ensuite la tester dans la partie inférieure de la fenêtre. Le bouton *Stopper* vous permet d'interrompre l'exécution d'une fonction. Le bouton *Pas* vous permet de dérouler pas-à-pas l'exécution d'une fonction.

1. Évaluation d'expressions

Avant de faire évaluer ces expressions par Scheme, essayez de prédire la réponse, et de l'expliquer.

```
(/ 12 3)
(- (+ 2 5 3) (* 8 4))
(> (+ 2 3) 4)
(or #t (< 12 3))
(define toto 5)
(number? toto)
(number? 'toto)
(integer? 3.5)
(even? (/ 6 2))
```

2. Premières fonctions

• Écrire une fonction qui calcule l'aire d'un cercle de rayon r.

• Écrire une fonction qui calcule la valeur absolue d'un nombre.

• Écrire une fonction qui calcule récursivement la somme des entiers de 1 à n. Dérouler (somme 3) pas à pas.

• Écrire une fonction qui teste si un entier strictement positif est une puissance de 2. Dérouler pas à pas la fonction pour les valeurs 3 et 4.

3. Évaluation d'expressions sur les listes

```
(car (1 2 3 4))
(car '(1 2 3 4))
(cdr '(1 2 3 4))
(cadr '(1 2 3 4))
(cdddr '(1 2 3 4))
(cdadr '(1 2 3 4))
(cdadr '(1 (2 3) 4))
(car '((1 (2 3) 4)))
(cons 1 '(2 3 4))
(cons (1) '(2 3 4))
(cons '(1) '(2 3 4))
(cons '1 '(2 3 4))
(cons 'a '(* 3 2))
(cons 'a (* 3 2))
(list? (+ 2 3))
(cons (cadr '(a b c)) (cdar '((d e))))
```

4. Exercices sur les listes

• D'après le modèle donné, construisez les expressions suivantes en utilisant exclusivement la fonction *cons*.

Par exemple :

(cons 'paon (cons (cons 'poule '()) '()))
=> (paon (poule))

... => (() () poule)

... => (oie poule (poule) paon)

... => (poule oie poule ((poulet)))

... => (((poule) oie) paon)

• Quelle expression faut-il évaluer pour extraire *oie* des expressions 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))))

5. Fonctions sur les listes

• Écrire une fonction qui prend une liste d'au moins deux éléments en argument et rend la liste équivalente où les deux premiers éléments ont été échangés.

```
(echange '(a b c d)) -> (b a c d)
```

• Écrire une fonction qui, étant donnée une liste non vide, rend la liste privée de son dernier élément.

```
(tsd '(a b c d)) -> (a b c)
```

• Écrire une fonction qui répète chaque élément d'une liste.

```
(repete '(a b c d)) -> (a a b b c c d d)
```

Pour s'entrainer (exercices supplémentaires facultatifs)

• Soit la suite de Syracuse, définie comme suit :

$u_{n+1} = 1 + 3u_n$ si u_n impair ;
$u_{n+1} = u_n / 2$ si u_n pair.

Quel que soit le nombre entier u_0 strictement positif, cette suite finit par engendrer le nombre 1.

• Écrire une fonction `syracuse?` testant si la suite partant d'un nombre u_0 donné en argument finit par passer par 1.

• Dérouler `(syracuse? 5)` pas-à-pas.

• Écrire une fonction qui concatène deux listes en utilisant uniquement la fonction *cons* (c'est-à-dire sans utiliser la fonction *append*).

```
(concatene '(a b c) '(d e)) → (a b c d e)
```