TP numéro 1

Connectez-vous à l'aide de votre numéro d'étudiant. Vous pouvez lancer DrRacket via un raccourci sur le bureau (ou via le menu *Démarrer>Programmes>PLT Scheme*).

Configuration de DrRacket :
Choisissez *Interagir avec DrRacket en Français*, puis *Accepter et Quitter* et relancez DrRacket. Ne prenez pas la peine de chercher une version plus récente (*Suivant*). Vous devez alors configurer le niveau de langage : choisissez *Niveau débutant avec abréviations pour les listes*, puis dans *Montrer les détails*, choisissez *write* pour *syntaxe de sortie* (si vous ne voyez pas le bouton Fin, cachez les détails).

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 devez enregistrer (bouton *Sauvegarder*) sur disque (utilisez votre compte sur W). L'extension ajoutée est `.scm`. 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 *Step* 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.

(+ 3 5)

(+ 2 3 4 5 6)

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

(if (< 1 2) 1 2)

(define alpha 10)

(symbol? alpha)

(symbol? 'alpha)

(symbol? "alpha")

Imaginez d'autres expressions simples et testez-les!

2 Premières fonctions

• Écrire une fonction qui calcule le carré d'un nombre.

• Écrire une fonction qui retourne la racine carrée d'un nombre s'il est positif et retourne 0 sinon.

• Écrire une fonction réalisant le ou-exclusif de deux booléens.

X	Y	X OU-ex Y
V	V	F
V	F	V
F	V	V
F	F	F

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

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))

(cons 1 '(2 3 4))

(cons (1) '(2 3 4))

(cons '(1) '(2 3 4))

(cons '1 '(2 3 4))

(cons alpha '(20 30))

(cons 'alpha '(20 30))

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

• Écrivez 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)
```

• Écrivez une fonction qui renvoie le dernier élément d'une liste non vide.

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

• Écrivez une fonction qui renvoie le calcul de la factorielle d'un nombre positif.

• Écrivez une fonction qui prend une liste de nombres en argument et qui renvoie son maximum.

• Écrivez une fonction qui teste si un nombre est impair.

• Écrivez une fonction qui prend une liste en argument et qui renvoie cette liste privée de son dernier élément.

• Écrivez une fonction qui prend une liste en argument et rend la liste privée d'un élément sur deux.

```
(elimine '(a b c d e f)) -> (a c e)
```

• Écrivez une fonction qui prend une liste en argument et qui répète chacun de ses éléments.

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

TP numéro 2

1 Fonctions sur les listes

• Écrivez une fonction qui ajoute 10 à tous les éléments d'une liste de nombres.

```
(ajoute10 '(1 3 2 4)) -> (11 13 12 14)
```

• Écrivez une fonction qui renvoie la sous-liste formée de tous les symboles d'une liste.

```
(symboles '(a 2 b (6 z) "toto" 7 f)) -> (a b f)
```

• Écrivez une fonction qui renvoie la sous-liste formée des n premiers éléments d'une liste.

```
(premiers 3 '(a b c d e)) -> (a b c)
```

• Écrivez 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)
```

2 Mémorisation

• Écrire une fonction qui calcule $\frac{n!+100}{n!+4}$ avec un seul appel à (factorielle n).

• Écrire une fonction qui rend la liste des n+1 premiers nombres de la suite de Fibonacci (de u_0 à u_n) sans faire plusieurs fois les mêmes calculs.

```
(fibo-liste 5) → (8 5 3 2 1 1)
```

• Utiliser la fonction `fibo-liste` pour écrire une nouvelle version de la fonction écrite en TD qui calcule le $n^{ième}$ terme de la suite de Fibonacci. Comparez le nombre de calculs effectués pour n=4. Testez les deux fonctions pour n=30. Etes-vous maintenant convaincu(e) de l'intérêt de calculer la complexité d'un algorithme ?

• Écrire une fonction qui étant donnée une liste, construit une liste de deux sous-listes : celle contenant les symboles et celle contenant les nombres.

```
(trie '(tor 1 tue la 2 3 pin 4)) → ((tor tue la pin) (1 2 3 4))
```

• Écrire une fonction qui étant donnée une liste, construit une liste où les éléments de la liste initial ont été permutés deux à deux.

```
(permute2a2 '(a b c d e f) → (b a d c f e))
```

• Écrire la fonction `som-prod` qui, étant donné une liste de nombres, renvoie une liste de deux nombres correspondant à la somme et au produit des éléments de la liste.

```
(som-prod '(1 2 3 4)) → (10 24)
```

• Écrire la fonction `tous-egaux` qui, étant donné une liste, teste si tous les éléments de la liste sont égaux.

• Écrire la fonction `parite` qui, étant donné une liste de nombres, renvoie une liste de deux sous-listes contenant les éléments pairs et impairs de la liste initiale.

```
(parite '(1 6 7 10 5)) → ((1 7 5) (6 10))
```

TP numéro 3

1 Tri par sélection du minimum

Le tri du minimum consiste à sélectionner le minimum de la liste à trier, à le mettre en début de liste, puis à recommencer sur la liste à laquelle on a enlevé ce minimum.

• Écrire une fonction qui renvoie le minimum d'une liste de nombres.

• Écrire une fonction qui enlève un élément à une liste. Cet élément ne sera enlevé qu'une seule fois.

```
(retire 7 '(6 4 7 9 7 9)) → (6 4 9 7 9)
```

• Écrire la fonction `tri-minimum` qui utilise les deux fonctions précédentes.

2 Listes ordonnées

• Écrire une fonction qui enlève tous les nombres entre a et b dans une liste triée.

```
(enleve 7 10 '(3 4 8 9 11)) → (3 4 11)
```

Attention à bien utiliser le fait que la liste est triée.

3 Listes de listes : la suite de Conway

La suite de Conway : 0, 10, 1110, 3110, 132110, 1113122110, 311311222110, ...

• On veut écrire une fonction qui calcule un terme de la suite de Conway à partir du précédent.

Exemple : (conway '(1 1 1 0)) -> (3 1 1 0)

Nous allons voir deux manières de programmer cette fonction.

1. Définissez une version récursive de cette fonction de la manière habituelle, en utilisant le résultat de l'appel récursif sur le reste de la liste.

2. Envisageons maintenant une version qui, bien qu'étant récursive, s'inspire de la programmation itérative, en utilisant un compteur. On définit pour cela une fonction `conway2` qui renvoie le même résultat, mais prend un argument supplémentaire : un entier qui sert à compter le nombre de répétitions du premier élément de la liste.

La fonction `conway` s'écrira alors :

```
(define conway
   (lambda (l)
      (if  (null? l)
           l
           (conway2 1 l))))
```

• Écrire une fonction qui calcule les n premiers termes de la suite de Conway en fonction du premier terme.

Exemple : (conwayn 4 '(0)) -> ((0) (1 0) (1 1 1 0) (3 1 1 0))

4 Listes de listes

Définir une fonction `(sp n x y)` qui, étant donnés deux nombres x et y, calcule les n premiers termes de la suite $x_n = x_{n-1} + y_{n-1}$ et $y_n = x_{n-1} * y_{n-1}$.

```
(sp 4 5 2) → ((5 2) (7 10) (17 70) (87 1190))
```

TP numéro 4

1 Récursivité profonde

• Écrire une fonction qui calcule en profondeur le nombre d'occurrences d'un élément dans une liste.

• Écrire une fonction qui calcule la profondeur d'une liste.

```
(profondeur '(a (b c) d)) -> 2
(profondeur '(a (b) (c ((d e) f)))) -> 4
```

2 Arbres binaires

Commencez par implémenter les fonctions primitives sur les arbres vues en cours et en TD :
vide, vide?, valeur, fils-g, fils-d, cons-binaire, arbre=?, feuille ?

Pensez bien à tester vos fonctions sur un arbre où au moins un nœud a un seul fils.

• Écrire une fonction qui calcule le minimum d'un arbre binaire contenant des nombres.

• Écrire une fonction qui renvoie la liste des valeurs des feuilles d'un arbre.

• Écrire une fonction qui prend un arbre binaire et renvoie son miroir : pour tous les nœuds de l'arbre, le fils gauche devient le fils droit et le fils droit devient le fils gauche.

3 Listes de listes : une autre version du triangle de Pascal (cf. TD5)

• Écrire une fonction `som2` qui, étant donnée une liste de nombres, retourne la liste des sommes des éléments consécutifs de cette liste.

```
(som2 '(1 2 3 4)) -> (3 5 7)
```

• Utiliser la fonction `som2` pour écrire une autre version de la fonction qui construit une liste de listes correspondant aux n premières lignes du triangle de Pascal.

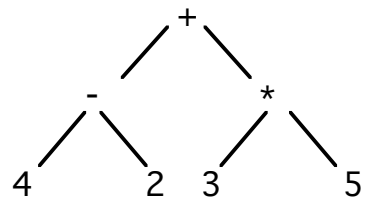
• Que pensez-vous de la complexité de cette version de la fonction par rapport à celle écrite en TD ?

TP numéro 5

1 Arbres binaires pour représenter des expressions arithmétiques

On considère des arbres qui représentent des expressions arithmétiques. Les valeurs des nœuds peuvent donc être soit des nombres, soit les symboles +, -, * et /.

L'expression arithmétique (+ (- 4 2) (* 3 5)) sera représentée par l'arbre binaire suivant :



• Écrire une fonction qui prend un tel arbre et qui renvoie la valeur numérique de l'expression correspondante. On supposera que chaque nœud possède soit 0 fils soit 2 fils.

2 Modification d'arbres binaires

• Définir une fonction qui prend en paramètre un arbre binaire de nombres et qui retourne une liste composée de cet arbre dont toutes les valeurs sont désormais positives et du nombre de changements qui ont été effectués pour obtenir ce nouvel arbre.

```
(arbre-positif '(12(-4(2()())())(-14(-6(8()())(12()()))(18()()))))
               → ((12(4(2()())()))(14(6(8()())(12()()))(18()()))) 3)
```

3 Arbres binaires de recherche

• Écrire une fonction qui calcule le minimum d'un arbre de nombres ordonné. Attention à bien tenir compte du fait qu'il s'agit d'un arbre binaire de recherche.

• Écrire une fonction qui construit la liste ordonnée des valeurs d'un ABR comprises entre deux valeurs données. Attention à ne pas parcourir inutilement certaines branches de l'arbre.

4 Parcours selon un chemin

On définit un chemin comme une liste de symboles g ou d (pour gauche et droite). Il s'agit d'une liste des directions à suivre à partir de la racine de l'arbre.

• Écrire une fonction booléenne qui, à partir d'un arbre et d'un chemin, dit si le chemin spécifié sort de l'arbre ou non.

• Écrire une fonction qui calcule la somme des valeurs des nœuds qui se trouvent le long d'un chemin donné. On supposera que le chemin ne sort pas de l'arbre.

TP numéro 6

Fonctions en argument

• Écrire une fonction qui applique une fonction unaire à tous les nombres d'une liste quelconque.

```
(applique-nombres sqr '(a 3 "ert" (t 5) 4 b)) → (a 9 "ert" (t 5) 16 b)
```

Abstraction sur les arbres

• En utilisant la fonction abstraite sur les arbres vue en cours, écrire une nouvelle version des fonctions hauteur et miroir.

Map et apply

• Redéfinir la fonction map pour les fonctions binaires (comme applique-à-tous est la fonction map pour les fonctions unaires).

```
(map_binaire + '(1 2 3) '(4 5 6)) → (5 7 9)
```

Définir les fonctions suivantes en utilisant map et/ou apply

- Une fonction qui calcule les racines carrées des nombres d'une liste de nombres.

  ```
  (racines '(4 16 9)) → (2 4 3)
  ```

- Une fonction qui calcule la somme des racines carrées des nombres d'une liste de nombres.

  ```
  (somme-racines '(4 16 9)) → 9
  ```

- Une fonction qui calcule les racines carrées des nombres d'une liste de listes de nombres.

  ```
  (racines-listes '((4 16) (9 1 4))) → ((2 4) (3 1 2))
  ```

- Une fonction qui calcule la moyenne des nombres d'une liste de nombres.

  ```
  (moyenne '(4 6 13 1)) → 6
  ```

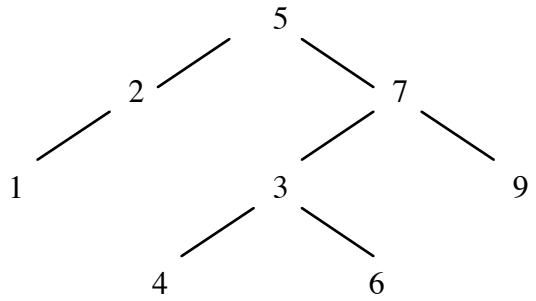
- La fonction applique-nombres définie ci-dessus.

- Une fonction qui calcule la somme des nombres d'une liste de listes de nombres.

  ```
  (somme '((1 2 3) (4 5) (6 7))) → 28
  ```

Révisions

On définit l'arbre a suivant :



• Définir une fonction qui, étant donné x un nombre et l une liste de couples (nombre, liste), retourne le deuxième élément d'un couple de l dont le premier élément est x.

```
(cherche 6 '((3 (a d r)) (6 (a z e)) (2 (e r)))) → (a z e)
```

• Définir une fonction qui, étant donné un nombre x et une liste de nombres l, construit deux listes : celle des nombres de l inférieurs ou égaux à x, et celle des nombres de l supérieurs à x.

```
(separe 3 '(5 1 2 3 6 4)) → ((1 2 3) (5 6 4))
```

• Définir une version de la fonction précédente qui traite la liste en profondeur.

```
(separe-prof 3 '(5 (6 (1 (2)) 4) 7)) → ((1 2) (5 6 4 7))
```

• Définir une fonction qui, étant donné un nombre x et un arbre de nombres a, construit la liste des valeurs de a supérieures à x.

```
(valeurs 3 a) → (5 7 4 6 9)
```

• Définir une fonction qui, étant donné une fonction f et un arbre a, applique la fonction f à chaque valeur de l'arbre a.

```
(applique-arbre sqr a)→(25(4(1()())())(49(9(16()())(36()()))(81()()))))
```

• Utiliser la fonction précédente pour définir une fonction qui enlève 1 à toutes les valeurs impaires d'un arbre d'entiers.

```
(rend-pair a) → (4(2(0()())())(6(2(4()())(6()()))(8()()))))
```

• Définir une fonction qui calcule le minimum d'une liste de listes de nombres.

```
(min-liste '((2 2 5) (1 6 3) (5 7))) → 7
```

• Définir une fonction qui fait la somme des seconds éléments des couples d'une liste de couples.

```
(ajoute2nd '((a 5) (b 7) (c 3))) → 15
```