Licence 1ere année UE EDI (2020-2021)

TP N°1

IMPORTANT : Vous devez retourner le fichier ".scm" ou ".rkt". contenant vos réponses aux exercices de la partie 2.
Pour ce faire vous enverrez un mail avec votre fichier attaché, ayant comme **sujet** :

TP1_SCHEME_ NOM1_PRENOM1_NOM2_PRENOM2

L'adresse email est : santucci@univ-corse.fr . Veillez respecter ce format, sinon votre envoi risque de n'être pas pris en compte…

Configuration de DrScheme :
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 disque (utilisez votre compte). 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.
Fonctions spéciales : lecture et écriture
. Display et print pour afficher à l'écran
. Read pour lire au clavier.

Attention : il faut sélectionner un langage (en général Assez Gros Scheme) et n'oubliez pas ensuite de cliquer sur exécuter afin de confirmer la sélection d'un langage.

PARTIE 1 : EVALUATIONS

1. Évaluation d'expressions

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

(car (cdr '(a (b c) d)))
(car '((a (b c) d)))
(cons 'a '(* 3 2))
(cons 'a (* 3 2))
(list? (+ 2 3))
(cons (cadr '(a b c)) (cdar '((d e))))

(+ 1 2)
(1 2 3)
(- (+ 2 5) (* 8 2))
(+ (* 3 2) (+ (* 4 5) 10))
(- 4 5 1)
(/ (+ 2 3) (* 6 2))
(+ (* (+ 2 3) 4) (* (+ 2 3) 5)
(+ (* 1 (- 5 3)) (- (+ (+14 1) 5) 10))
(define a 2)
(+ a 23)
(+ 2 (*5 4))
(/ 12 (+ 1 3))
(* (* (* 1 2) 3) 4)
(* 1 (* 2 (* 3 (* 4 5))))
(+ 1/3 2/3)
(define dix (+ 5 5))
dix
(quotient 15 2)
(max (min 5 8 9) (min 2 7 10))
(+ (abs (- 25 35)) (round 3.6))
(modulo 15 2)
(* (max 3 5 6) (min 2 9 1))
(truncate 6.8)
```

2. Les listes

Evaluez les expressions suivantes et lorsque vous éprouvez des difficultés passez par la représentation graphique.

1. (cons (cons 1 2) (cons 3 4)))
2. (cons 1 (cons 2 (cons 3 4)))
3. (cons 1 (cons 2 (cons 3 '())))
4. (cons '(a b c) '())
5. (list '(a b c) '())
6. (append '(a b c) '())
7. (list 1 2 3 (+ 2 2))
8. (append '((a b)(c d)) '(1 3) '((5)(6)(7)))

```
(define outils (list 'marteau 'tournevis))
outils
(cons 'pince outils)
outils
(define outils (cons 'pince outils))
outils
(append '(scie clef) outils)
outils
(define outils (append '(scie clef) outils))
outils

(define l (list 1 2 3 4 5 6))
(list (car l) (car (cdr (cdr l))))
(list (car l) (cdr l))
```

3. Evaluez les expressions suivantes

1. (length '(quelle taille a cette liste ?))
2. (length '(et (cette autre ?)))
3. (length '((on ne se perd dans toutes ces)))
4. (reverse '(A B C (D)))
5. (reverse '((1 2 3) 4 5 6))
6. (reverse '((1) (2) (3) (4)))

4. Eval et quote

Evaluez les expressions suivantes :

```
(quote (eval (+ 1 2)))

(eval (quote (+ 1 2)))

(eval '(+ (- 3 2) 5))

>(define formule '(* 5 6))
>formule
>(eval formule)
```

```
(eval '(+ '(- 3 2) 5))

(quote (+ (* 2 5) (eval '(+ 2 9))))

(eval (quote ((* 2 5) (eval '(+ 2 9)))))

>(define b 123)
>(define a 'b) ;la variable a vaut le symbole b
>a
>(eval a)
```

5. Evaluation de fonctions anonymes

```
((lambda (x y) (* (+ x y) 3)) 2 5)

(* 3 ((lambda (x) (* x x)) 3))

((lambda (x y) (+ (* x y) 2)) (+ 7 11) (+ 5 2))
```

```
((lambda (x y z) (+ (* x y) (* x z))) 2 3 4)

((lambda (x y z u) z) 1023 985 25 79)

((lambda (toto) 33) 0)
```

6**. Evaluations d'expressions booléennes**

(or (> 1 2) (< 1 2))	(and (not #f) (not #t))
(and (> 1 2) (< 1 2))	(and () 2)
(not (not (> 1 2)))	(and 3 4 2

PARTIE 2 : Exercices

1. Transformation d'expressions.
Soit les expressions suivantes:

1. (+ 1 3)	8. (cdr (+ 1 6))	15. (car (cdr (car '((a b) c d))))
2. '(+ 1 3)	9. (car '((a b) c (d e)))	16. (car (quote (b c)))
3. (quote (+ 1 3))	10. (cdr '((a b) c (d e)))	17. (cdr (quote (b c)))
4. (car '(1 2 3))	11. (car (cdr '((a b) c d)))	18. (car '())
5. (car '(2))	12. (cdr (car (cdr '((a b) c d))))	19. (cdr '())
6. (cdr '(2))	13. (car (cdr '(a b c)))	20. (car (cdr '(1)))
7. (cdr '(1 2 3))	14. (car '(cdr '(a b c)))	21. (cdr (cdr '(10)))

Question : Transformer les expressions qui peuvent l'être en utilisant la composition de car et cdr sous la forme cxxxxr.

2. Construction d'expressions sur les listes.

Soit les trois listes suivantes :

- (define l1 '(scheme est un langage))
- (define l2 '(((((comprenez) vous) tout) cela) ?))
- (define l3 '((A B) C (1 4 3 (2)))).

Ecrivez les expression qui après évaluation donneront :

1. langage	6. est	11. scheme
2. (un langage)	7. ()	12. ((comprenez) vous)
3. comprenez	8. ?	13. (tout)
4. (cela)	9. (1 4 3 (2))	14. B
5. C	10. (2)	15. (A B)

3. Construction d'expressions sur les listes (suite)

3.1 Ecriture d'expressions

Soit la déclaration suivante : (define l '(1 2 (3 (4) 5) 6 (7 8) 9))

Donnez les expressions dont l'évaluation renvoie les listes suivantes :

1. (1 3 (4) 5)
2. (6 1)

3.2 Quelle expression faut-il évaluer pour extraire A des expressions suivantes ? Par exemple, pour l'extraire de (B (A D) E), c'est caadr. :
((B A Z))
(Z (B A) (R T))

3.3 D'après le modèle donné, construire les expressions suivantes en utilisant exclusivement la fonction *cons*. Par exemple :
(cons 'a (cons (cons 'b '()) '())) => (a (b))
... => ((a) b g)
... => ((a) (b g))

3.4 Ecrire une commande lisp (une phrase lisp à interpréter), permettant de demander une liste de nombre à lire au clavier et d'en calculer la somme. Indication : utiliser les fonctions eval et read

4. **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.

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

6**. Définir les fonctions suivantes en schème**

f(x) = racine(1 + racine(2 + racine(3 + x)))

g(x) = racine(1 + racine(2 + racine(3 + racine(4 + racine(5 + racine(6 + x))))))

f(x) = 1 + racine(3 + 7/x) + log(1 + racine(3 + 7/2x) - 4)

7 Création de fonctions travaillant sur les listes

Implémenter les fonctions suivantes :

1. *premier* : *liste* → *indiférent.* La fonction premier, prend une liste en paramètre et retourne le premier élément de cette liste.
   Exemple d'appel :
   >(premier '(1 2 3))
   1

2. *second : liste* → *indiférent*. La fonction second, prend une liste en paramètre et retourne le deuxième élément de cette liste.
   Exemple d'appel :
   >(second '(1 2 3))
   2

3. *sauf_premier : liste* → *liste*. La fonction sauf_premier, prend une liste en paramètre et retourne la liste sauf le premier élément.
   Exemple d'appel :
   >(sauf_premier '(1 2 3))
   (2 3)

4. *construit : indifférent * liste* → *liste*. La fonction construit, prend un élément quelconque e (atome ou liste) et une liste l en paramètre et retourne la liste dont le premier élément est e et dont les suivant sont ceux de l.
   Exemple d'appel :
   >(construit 'A '(1 2 3))
   (A 1 2 3)

5. *singleton?*. Donnez la spécification et la définition en scheme du prédicat singleton? Qui rend vrai si une liste n'a qu'un seul élément.

6. *decalage_gauche : liste* → *liste*. La fonction décalage_gauche, prend en paramètre une liste et retourne cette même liste après permutation à gauche. Toutes les positions sont décalées vers la gauche : la tête de liste se retrouve à la queue, le deuxième élément se retrouvre en tête, le troisième passe en deuxième...
   Exemple d'appel :
   >(decalage_gauche '(1 2 3 4 5))
   (2 3 4 5 1)

   De même implémentez la fonction decalage_droite

8. Implémentez les fonctions suivantes :

1. *le_plus_grand : nombre * nombre → nombre.* Retourne le plus grand de deux nombres passés en paramètre de la fonction

2. *signe : nombre →{positif, négatif, nul}*. La fonction signe retourne le signe du nombre qui lui est passé en paramètre.

3. Implémentez le ou exclusif se définissant comme suit : la proposition "p ou_ex q" est vrai si et seulement si une et une seule des deux propositions est vraie.

4. Concevoir et coder en scheme une fonction calculant l'aire d'une couronne définie par deux cercles concentriques de rayon R1 et R2

5. Un marchand de vin expédie une quantité q de vin de prix unitaire p. Si le total de la commande est d'au moins 500€; le port est gratuit. Sinon, il est facturé 10% de la commande. Ecrire une fonction, qui étant donné q et p retourne la somme totale à payer.

6. Même chose que précédemment mais dans le cas ou le port n'est pas gratuit, il est de 10% de la commande avec un minimum de 10€.

9 Définir et tester les fonctions suivantes

a) *La fonction constante* $k2 : x \mapsto 2$.

b) *La fonction identité* $id : x \mapsto x$.

c) *La fonction* $proj2 : (x, y) \mapsto y$.

d) *La fonction distance prenant 4 paramètres* x_1, y_1, x_2 *et* y_2 *qui retourne la distance euclidienne*[2] *du point* $M_1(x_1, y_1)$ *au point* $M_2(x_2, y_2)$.

[2] La distance euclidienne est donnée par : $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

10 Ecrire les fonctions suivantes :

a-Faire une fonction (max3 a b c) donnant le maximum des trois nombres a, b, c.

b-Faire une fonction (max4 a b c d) donnant le maximum des quatre nombres a, b, c, d.

Indication : Vérification des exercices
En utilisant la fonction trace on peut vérifier l'execution des fonctions.
Cette fonction fait partie d'une librairie qu'il faut importer pour pouvoir l'utiliser.
Cette opération est effectuée par l'expression (require (lib « trace.ss »)).

La trace est appliquée sur une fonction et elle imprime l'expansion de cette fonction chaque fois que celle-ci est appelée.
Exemple :

```
>(define (square x) (* x  x))
>(define a 3)
>(trace square)
>(trace *)
>(square  a)
```

Donne :

```
|(square 3)
  |( * 3  3)
  |9
|9
```

Remarque : pour ne plus avoir l'impression de la trace il faut untracer la fonction : (untrace square)