Conseils et aide mémoire pour les TP de LIF3

Conseils et recommandations

Sauvegarde

- Sauvegardez votre fichier sur votre compte dès le début du TP.
- Faites un fichier par TP.
- Bannissez les espaces, accents et autres caractères spéciaux dans les noms de fichiers, de fonctions, de variables, etc.
- Sauvegardez régulièrement au cours du TP (personne n'est à l'abri d'une panne de courant ou d'un plantage système).

Code

- Indentez votre code suivant les suggestions de l'éditeur.
- Commentez votre code : au minimum ce que fait la fonction (en français), ce que retourne la fonction et les variables manipulées (type).
- Pensez aussi à préciser s'il y a des conditions d'utilisation particulières pour cette fonction (fonction qui ne manipule que des listes de nombres par exemple).

Trucs et astuces

La fenêtre d'interactions (celle du bas) se "vide" à chaque fois que l'on clique sur "exécuter". Si vous devez retaper plusieurs fois la même commande pour tester une de vos fonctions, cela peut vite devenir pénible.

Trois solutions :

- faire du copier-coller, mais on oublie souvent de copier !
- rappeler la dernière commande avec ***Echap+P***
- écrire les appels dans la fenêtre de définitions (celle du haut) et les commenter lorsque l'on n'en a plus besoin.

Exemple :

```
; retourne une liste dont les deux premiers éléments ont été intervertis
(define echange ;-> une liste
  (lambda (l)     ;l est une liste
    (cons (cadr l) (cons (car l) (cddr l)))))

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

Il suffit ensuite d'exécuter pour obtenir le résultat dans la fenêtre d'interactions.

Remarque :

Quand vous utilisez souvent une liste pour des tests, vous pouvez lui donner un nom pour l'utiliser en lieu et place de la liste.

Exemple :

```
(define l '(1 2 3 4))
(car l) équivaut alors à  (car '(1 2 3 4))
```

Raccourcis clavier

Ctrl+T : exécute (fait la même chose que l'appui sur le bouton exécuter en haut à droite)

Ctrl+S : sauvegarde le fichier (le bouton sauvegarder doit disparaître)

Liens utiles

Le site de l'UE LIF3 : https://liris.cnrs.fr/~fzara/LIF3/

Le site de Racket : http://racket-lang.org/

Aide mémoire

Type d'opération	Syntaxe	Exemple d'application

Définition d'une fonction	`(define nomDeLaFonction`
	`  (lambda  (un  ou  plusieurs arguments espacés)`
	`  instructions))`
		`(define carre ; -> un nombre`
		`  (lambda (a) ; a: entier`
		`  (* a a)))`

Condition : if	`(if test`
	`  instructionSiTestVrai`
	`  instructionSiTestFaux)`
		`(if (null? l)`
		`  0`
		`  (car l))`

Condition : cond	`(cond`
	`  (test1 instruction1)`
	`  (test2 instruction2)`
	`  ...`
	`  (else instructionN))`
		`(cond`
		`  ((< n 0) 'negatif)`
		`  ((> n 0) 'positif)`
		`  (else 'nul))`

Mémorisation : let	`(let ((identificateur1 valeur1)`
	`  (identificateur2 valeur2)`
	`  ...`
	`  (identificateurN valeurN)`
	`  )`
	`  instructions avec`
	`  utilisation`
	`  des identificateurs`
	`)`
		`(let ((a (sqr x)`
		`  (b (sqr y))`
		`  (c (sqr z))`
		`  )`
		`  (if (< a b)`
		`  0`
		`  (+ a b c)))`
		`)`

Fonctions prédéfinies	**Syntaxe**	**Exemple d'application**
Opérateurs arithmétiques	`+, -, *, /`	`(+ 3 6 1)` → `10` `(- 6)` → `-6`

Opérateurs booléens	`or, and, not`	`(not #t)` → `#f`
		`(not #f)` → `#t`
		`(and #t #t #f)` → `#f`

Opérateurs de comparaisons sur les nombres	`=, <, >, <=, >=`	`(= 2 4)` → `#f`
		`(< 3 9)` → `#t`
		`(>= 5 5)` → `#t`

Fonctions de comparaison	`eq?` (compare deux éléments sauf listes et chaines de caractères)
	`equal?` (compare deux éléments)
		`(eq? 'a 'a)` → `#t`
		`(eq? 4 5)` → `#f`
		`(equal? "toto" "toto")` → `#t`
		`(equal? '(5 r ) '(u 4 df 5))` → `#f`

Fonctions mathématiques	`sqr` (= carré)
	`sqrt` (= racine carrée)
	`abs` (= valeur absolue)
	`max, min`
	`modulo` (= reste division entière)
	`quotient` (= division entière)
		`(sqr -5)` → `25`
		`(sqrt 9)` → `3`
		`(abs -6)` → `6`
		`(max 2 6 7 5)` → `7`
		`(modulo 17 3)` → `2`
		`(quotient 341 10)` →`34`

Fonctions de test	`symbol?, number?, integer?, string?, list?, boolean?, even?` (pair), `odd?` (impair)	`(symbol? 'a)` → `#t` `(symbol? 5)` → `#f`

Fonctions sur les listes	Accès : `car, cdr`
	Construction : `cons, list, append`
	Test : `null?`
	Longueur : `length`
	Appartenance : `member?`
		`(car '(a b c)` → `a`
		`(cdr '(a b c)` → `(b c)`
		`(cons 'a '(b c)` → `(a b c)`
		`(list 'a 'b 'c)` → `(a b c)`
		`(append '(a b) '(c))` → `(a b c)`
		`(length '(a b c))` → `3`
		`(member? 'a '(a b c)` → `#t`

Fonctions diverses	`map, apply` `eval` (force l'évaluation) `random` (renvoie un entier aléatoire dans [0 X[) `begin` (permet d'exécuter un bloc d'instructions) `display` (permet l'affichage), `newline` (renvoie à la ligne)	`(map even? '(3 7 2))` → `(#f #f #t)` `(apply + '(3 7 2))` → `12`