**Cours LISP (1h30)**

**Fonction FUNCALL**

? (funcall '+ 1 2)

= 3

Ecrire la fonction *findAll* (fct l) qui renvoie la liste des élements de la liste ***l*** qui satisfont la fonction ***fct***.

?(findAll 'numberp '(2 k b 4)) = (2 4)

?(findAll 'atom '(a (h) p)) = (a p)

Vérifier que la fonction prédéfinie **remove-if-not** est identique à la fonction **findAll.**

(remove-if-not 'numberp '(2 k b 4)) = (2 4)

(remove-if-not 'atom '(a (h) p)) = (a p)

La fonction ***remove-if***  existe également.

? (remove-if 'numberp '(1 m 2)) = (m)

==========================

**Fonction LAMBDA**

Définit une fonction anonyme

?((lambda (x) (+ x 2)) 3) = 5

syntaxe : (lambda (<arg1> ..<argn>) <corps de la fonction>)

pour l’appliquer

((lambda (<arg1> ..<argn>) <corps de la fonction>) <val1> …<valn>)

**EX1**

Ecrire à l’aide de *remove-if-not* et d’une *lambda* fonction , la fonction ***linfan(n l***) qui prend en argument un nombre n et une liste de nombres l et renvoie la liste des nombres inférieurs à n.

? (linfan '6 ' (7 4 1 9)) = (4 1)

**EX2**

Ecrire à l’aide d’une *lambda* fonction , la fonction ***infan(n)*** qui renvoie la fonction qui teste si un nombre est inférieur à n.

? (funcall (infan 2) 6) = T

Que renvoie (infan 2)  ?

Réécrire à l’aide de *remove-if-not* et de la fonction *infan* la fonction *linfan*(*n l*).

==========================

**Fonction APPLY**

? (apply '+ ' (1 2)) = 3

A la différence de la fonction *funcall*, la fonction *apply* prend en arguments une fonction et la liste des arguments de cette fonction.

**EX3**

Ecrire la fonction ***multnum*** (l) qui calcule le produit de tous les éléments d’une liste plate de nombres.

? (multnum ' ( 8 4 2))= 64

==========================

**EX4**

Ecrire à l’aide des fonctions ***apply*** et ***remove-if-not*** la fonction ***mult*** (l) qui calcule le produit de tous les nombres d’une liste.

? (mult ' ( 8 (4) 2 a))= 16

==========================

**Fonction MAPCAR**

(mapcar <fct> <l1> …<ln>)

applique la fonction <fct> avec tous les CAR de toutes les listes comme argument, puis tous les CADR, jusqu’à épuisement des listes et renvoie la liste des résultats.

? (mapcar 'list ' (je tu (il elle)) ' (nous vous (ils elles)) )

= ((je nous) (tu vous) ((il elle)(ils elles)))

? (mapcar '1+ ' (1 2 3)) = (2 3 4)

Evaluer

(mapcar '+ ' (1 2 3) ' (8 7 9))

(mapcar '+ ' (1 2 3 4) ' (8 7 9))

(mapcar 'car ' ((5) (1 2 3 4))

==========================

**EX5**

Ecrire la fonction **ajoutliste** (l1 l2) qui ajoute les éléments de 2 listes 2 à 2

? (ajoutliste ' (1 2) ' (3 4))= (4 6)

============================

**EX6**

Ecrire la fonction **premierliste (l)** qui prend en argument une de listes et renvoie le premier élément de chaque liste

?(premierliste '((b n)(2 3 1)(+ 1 2)) ) = (b 2 +)

==========================

**EX7**

Ecrire la fonction **carreListe** qui renvoie la liste des carrés d’une liste plate de nombres

? (carreListe ' ( 5 8) ) = (25 64)

==========================

**EX8**

Ecrire la fonction produit scalaire **ps** (l1 l2) à l’aide de ***apply*** et ***mapcar***

? (ps ' (1 2 3) ' (3 4 5))  = 26 ; 1\*3+2\*4+3\*5

==========================

**EX9**

Ecrire la fonction ***sommeF (fct l)*** qui renvoie la somme des images par la fonction *fct* des éléments de la liste *l*.

? (sommeF 'abs '(1 -1 -2 2)) = 6

============================

**EX10**

Ecrire à l’aide de la fonction ***ajoutn*** précédente, la fonction ***ajout*** (n l) qui ajoute un nombre ***n*** à tous les nombres d’une liste de nombres ***l***

?(ajout '3 ' ( 5 8)) = (8 11)

==========================

**EX11**

Ecrire la fonction ***ajout2*** qui ajoute un nombre n à tous les nombres de premier niveau d’une liste

?(ajout2 '3 ' (a 5 (7) 6) = (a 8 (7) 9)

==========================

**EX12**

Ecrire la fonction ajout2liste(n l1 l2) qui fait la somme des éléments des 2 listes l1 et l2 membre à membre et y ajoute n.

? (ajout2liste '3 ' ( 5 8) ' ( 1 2)) = (9 13)

==========================

**EX13**

Ecrire la fonction ***remplace*** (x y l) à l’aide de la fonction **mapcar** et d’une lambda fonction.

? (remplace 'c 's ' (g c n k c))= (g s n k s)

==========================

**EX14**

Ecrire la fonction **devant** (x l) qui prend en argument un atome et une liste de listes plates et met l’atome à la tête de chaque liste de la liste.?(devant 'd ' ((c v) (i o)( k l m)) )= ((d c v) (d i o)( d k l m)) )

(Une liste plate ne possède que des éléments de premier niveau. Exemple (y i p). Contre-exemple ((s) d (j(k)) ).

**mapcan** is like **mapcar**, except that the results of applying *function* are combined into a *list* by the use of **nconc** rather than **list**.

***nconc*** est la fonction de concaténation équivalente à ***append***.

**EX** 15

Réécrire la fonction ***findAll*** à l’aide de la fonction **mapcan**

==========================

Fonction ***every***

La fonction prédéfinie ***every*** (fct l) teste si tous les éléments de la liste ***l*** satisfont la fonction ***fct***.

? (every ' numberp ' (1 2 3)) = t

**EX16**

Ecrire à l’aide de la fonction ***every*** la fonction ***inclusDans*** (l1 l2) qui teste si tous les éléments de la liste ***l1*** sont dans ***l2***.

==========================

Fonction ***some***

La fonction prédéfinie ***some*** (fct l) teste si au moins un élément de la liste ***l*** satisfait la fonction ***fct***.

? (some ' numberp ' (h 2 3)) = t

**EX17**

Ecrire à l’aide de la fonction ***some*** la fonction ***nonDisjoints*** (l1 l2) qui teste si les ensembles ***l1*** et ***l2*** ne sont pas disjoints.