**Exercice 1**

Ce qu’on veut :

|  |
| --- |
| >(setq ll '( A 1 BB 2 CCC 3 DDDD 4))  (A 1 BB 2 CCC 3 DDDD 4)  >(f ll)  A  1  BB  2  CCC  3  DDDD  4  NIL |

Pour obtenir ce résultat on se propose d’utiliser plusieurs fonctions différentes. Les voici :

|  |  |
| --- | --- |
| CG-USER(13):(defun f1 (ll)  (dolist (x ll)  (print x)  )  )  CG-USER(17):(defun f2 (ll)  (mapcar print ll)  )  CG-USER(3):( (defun f3 (ll)  (dotimes ( i (length ll))  (print (nth i ll))  )  )  CG-USER(21): (defun f4 (ll)  (cond  ((null ll) nil)  (T (print (car ll))  (f4 (cdr ll))  )  )  )  CG-USER(6): (defun f5 (ll)  (loop for i in ll  do (print i))  )  CG-USER(7): (defun f6 (ll)  (loop  (unless ll (return from nil "done"))  (print (pop ll))  )  ) | Parcourt la liste en incrémentant x.  Afficher le x-ième élément de la liste.  Utilisation de mapcar. Problème avec la syntaxe à l’utilisation.  Dotimes : de i=0 à (length ll) -1.  Fonction récursive. On affiche le car, puis le cadr, puis le caddr ... jusqu’à ce que la liste en argument soit vide.  Afficher le i-ième élément de ll. |

A l’utilisation, toutes les fonctions sauf f2 et f6 renvoient :

|  |
| --- |
| A  1  BB  2  CCC  3  DDDD  4  NIL |

f2 renvoie :

|  |
| --- |
| >(f2 ll)  Error: Attempt to take the value of the unbound variable `PRINT'.  [condition type: UNBOUND-VARIABLE] |

Défaut de compréhension du mapcar.

Quant à f6 :

|  |
| --- |
| >(f6 ll)  A  1  BB  2  CCC  3  DDDD  4  Error: Attempt to take the value of the unbound variable `FROM'.  [condition type: UNBOUND-VARIABLE] |

Visiblement un problème avec la syntaxe au niveau du return.

**Exercice 2**

Ce qu’on veut :

|  |
| --- |
| >(setq l '(html  (header  (title "ma page" )  )  (body  (h1 "un titre")  (p "soror et aemula romae")  )  )  )  >(make-html l)  <HTML><HEADER><TITLE>ma page</TITLE></HEADER><BODY><H1>un titre</H1><P>soror et aemula romae</P></BODY></HTML> |

Voici la fonction make-html :

|  |  |
| --- | --- |
| CG-USER(14): (defun make-html (l)  (if (listp l)  (progn  (princ (concatenate 'string "<" (string (car l)) ">"))  (dolist (e (cdr l))  (make-html e)  )  (princ (concatenate 'string "</" (string (car l)) ">"))  )  (princ l)  )  ) | Si l est une liste on effectue le bloc dans progn.  Sinon on affiche l. |

Il est également possible d’utiliser cond. Rappelons que (if x then else) ≡ (cond (x then) (T else)). Dans le cas du cond, le « then » (ou le « else ») peut être plusieurs instructions, mais pour le if seule une instruction est acceptée (d’où l’utilisation du bloc progn).

**Exercice 3**

Il s’agit maintenant de créer un fichier html. Il faut donc créer une fonction qui créera un fichier html puis écrire dedans ce qu’affiche make-html. Dans la mesure où on ne peut pas utiliser le retour de make-html, mieux vaut modifier la fonction la fonction de telle sorte que les princ soient inscrits dans le fichier. Cela devrait fonctionner en modifiant de la sorte la fonction :

|  |
| --- |
| (defun make-html (l file)  (if (listp l)  (progn (princ (concatenate 'string "<" (string (car l)) ">") file)  (dolist (e (cdr l))  (make-html e)  )    (princ (concatenate 'string "</" (string (car l)) ">") file)  )  (princ l file)  )  ) |

Et en utilisant :

|  |
| --- |
| (with-open-file (file "test"  :if-does-not-exist :create  :if-exists :overwrite  :direction :output)  (make-html l file)) |

Néanmoins l’interpréteur LISP retourne une erreur : make-html ne recevrait qu’un seul argument dans une telle configuration ...