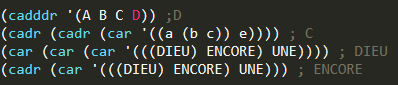
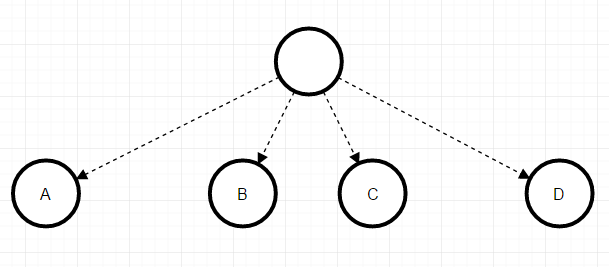
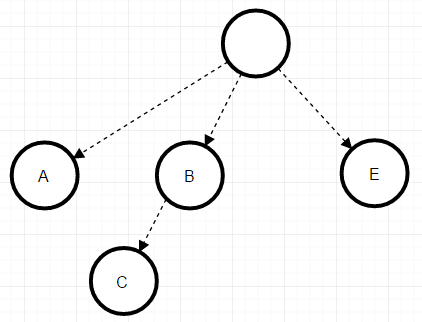
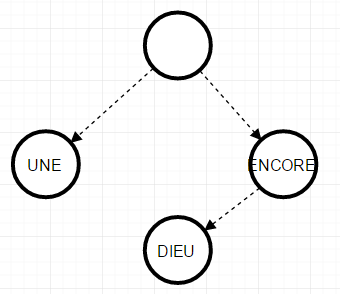
IA01 – Rapport TP1  
Montée en compétences Lisp

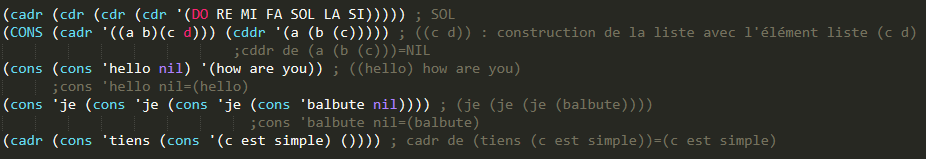
# Exercice 1 : Mise en condition

1.   


2.

(A B C D) :  
  
(A (B C) E) :  
  
(((DIEU) ENCORE) UNE) :

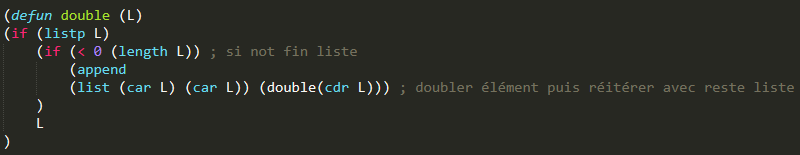
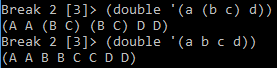


3.  


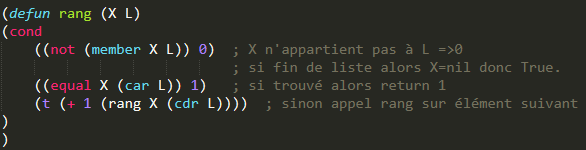
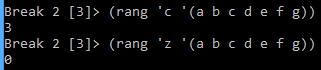
4.

## double (L) :

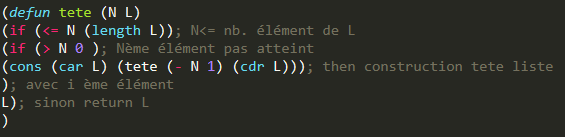
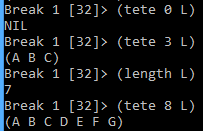
renvoie une liste avec chaque élément en double



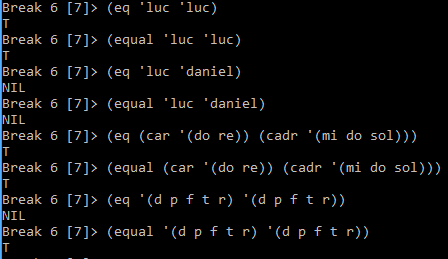
rang (X L) : Retourne l’index de l’élément X dans L.  
Renvoie 0 si X n’appartient pas à L.



tete (N L) : renvoie les N premiers éléments.  
Si N est plus grand que le nombre d’éléments de L alors la liste L est renvoyée.  
Si N=0 alors nil est renvoyée.



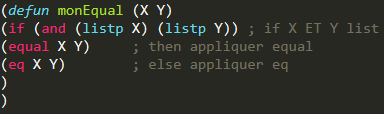
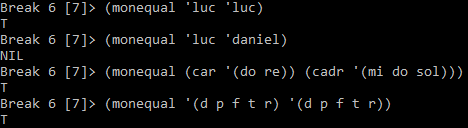
## monEqual (X Y) :



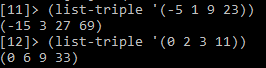
La différence entre *eq* et *equal* se remarque pour le test de l’égalité de liste.

*equal* renvoie t si liste identique alors que *eq* renvoie nil.

Nous pouvons donc construire la fonction monEqual de cette manière :

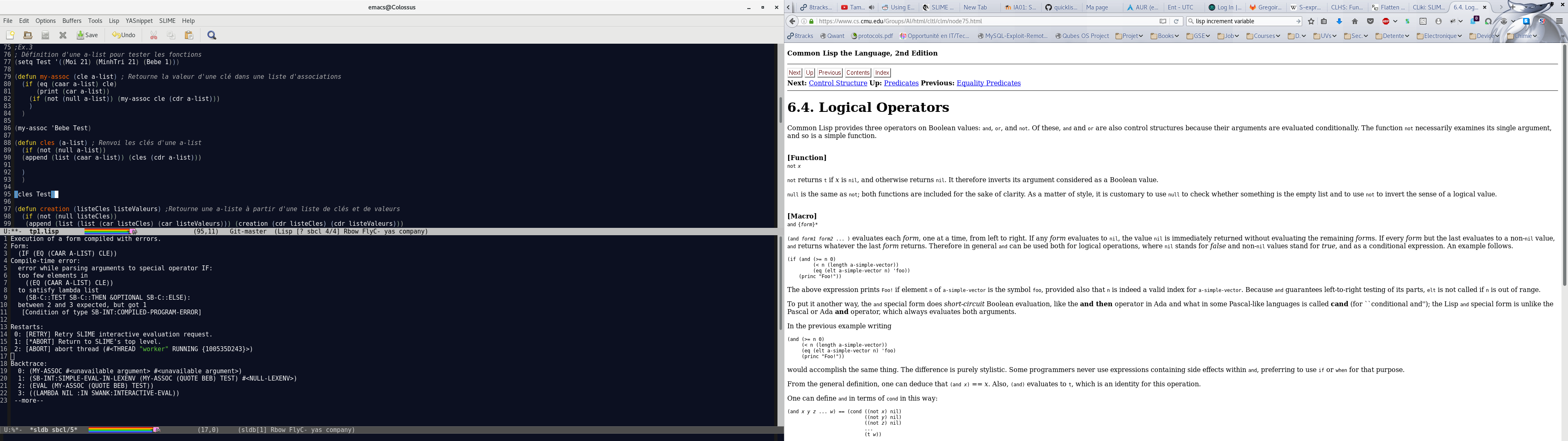
  
monEqual combine les retours des fonction *equal* et *eq*. Equal est uniquement appelé si X et Y sont des listes.

# Exercice 2 : Objets fonctionnels

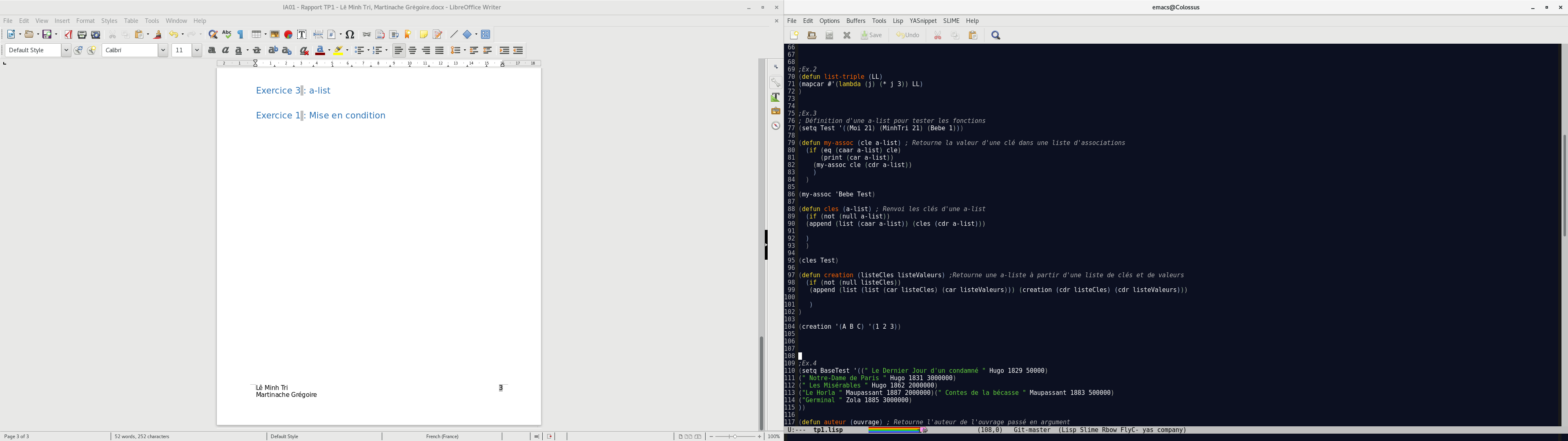


# Exercice 3 : a-list

D’abord, nous avons commencé par définir une liste Test qui permettra de tester les différentes fonctions.

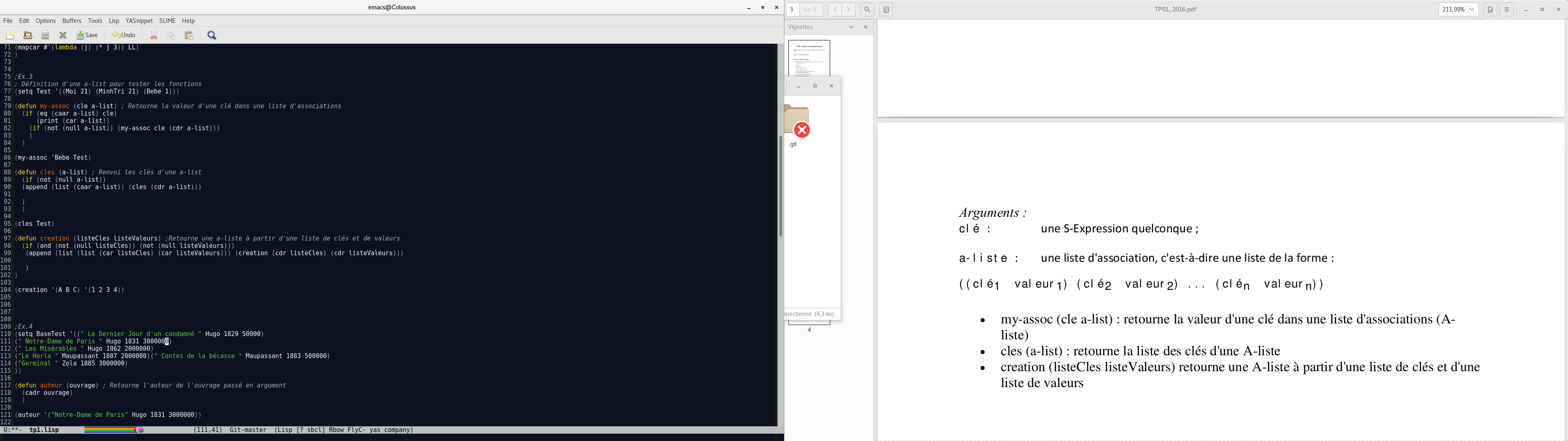


Le fonctionnement de my-assoc est assez simple : On regarde si le premier élément de la a-list correspond à la clé recherché. Si c’est le cas, alors on renvoi le couple, sinon, on continue dans la liste de façon récursive. Si on arrive à la fin de la liste, la condition (not (null a-list)) permet de renvoyer nil.



On définit maintenant la fonction cles permettant de récupérer tous les clés d’une a-liste.

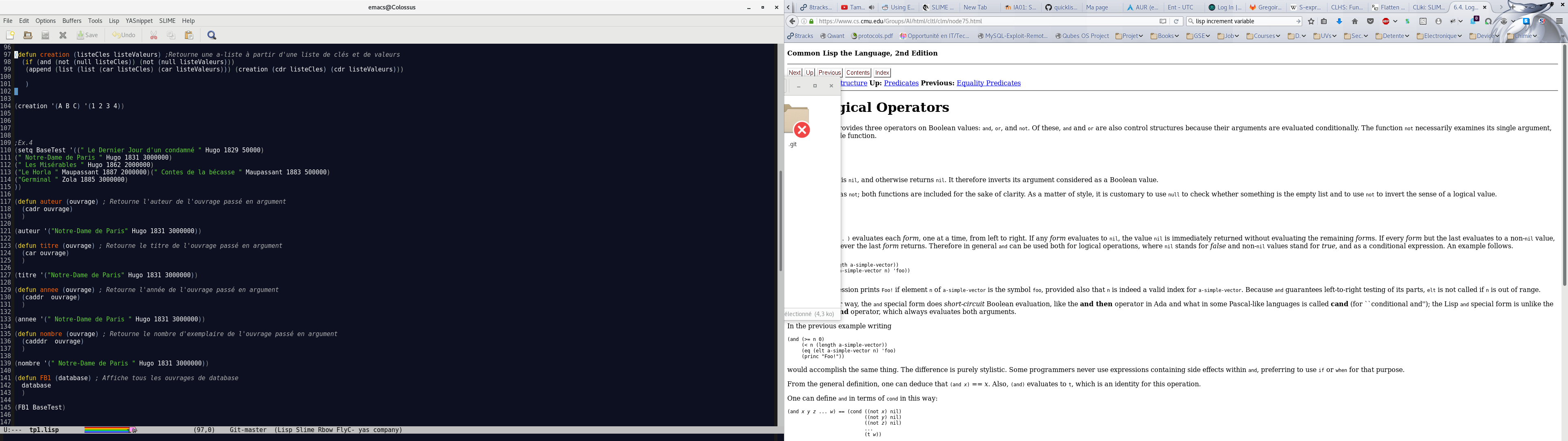
Pour cela, on parcours la liste récursivement (avec comme condition d’arrêt que la liste ne soit pas vide) en ajoutant à la liste des clés la clé actuelle suivie des prochaines.



Enfin, concernant la fonction permettant la création d’une a-liste : On prend le premier élément de la liste des clés, le premier élément de la liste des valeurs et on forme un couple avec ces deux éléments. Ensuite, on relance la fonction sur la queue de chacune de ses listes. On s’arrête dès que l’une des deux listes est vide. En procédant ainsi, on gère le cas où les deux listes ne font pas la même taille.

# Exercice 4 : Gestion d’une base de connaissances en Lisp

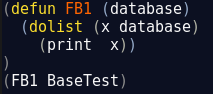
A. Fonctions de service :

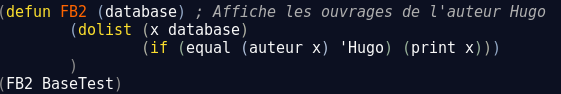


Les fonctions de service se passent de commentaires détaillés, il s’agit la simplement de manipulation de liste à l’aide des fonctions car et cdr ainsi que de leur combinaison.

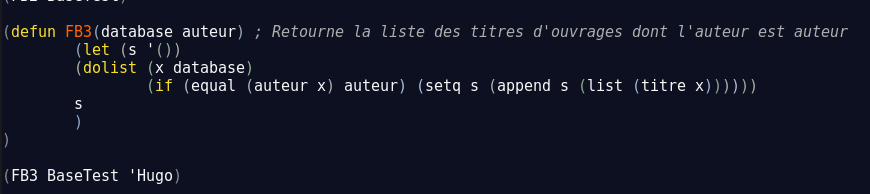
B. Autres fonctions :

La fonction FB1 doit simplement afficher tous les ouvrages de la base de données. Il suffit pour cela de parcourir la base de données avec un dolist et d’en afficher chacun de ses éléments.



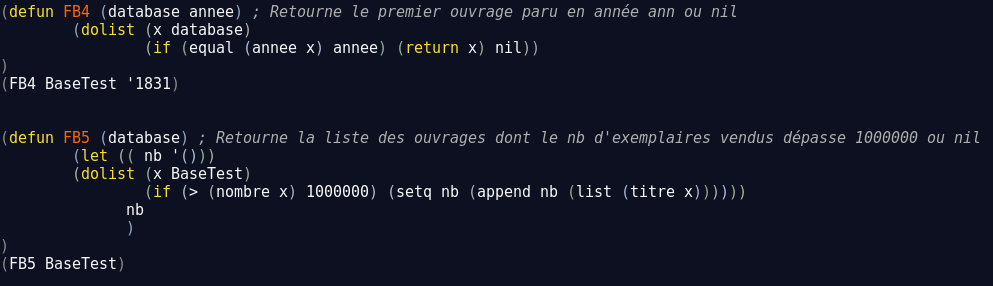
La fonction FB2 fonctionne sur le même principe que FB1 sauf qu’à chaque itération, il y’a une condition supplémentaire pour n’afficher que les auteurs correspondant à « Hugo ».

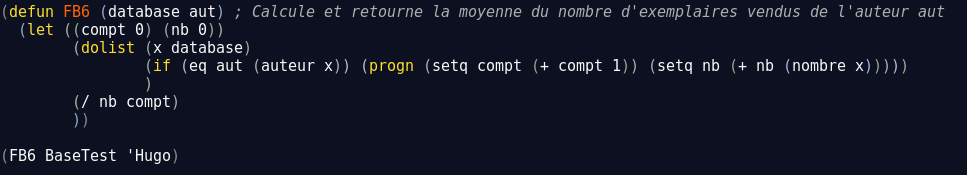
Pour retourner la liste des titres d’ouvrages d’un certain auteur (FB3), on utilise une variable locale s. À chaque itération, si l’auteur correspond, on rajoute le titre de son œuvre à la liste s. À la fin, on renvoi simplement s.



Pour FB4, on parcourt la base de donnée en cherchant l’année x. Si celle-ci est trouvé, on utilise la fonction return pour ne pas chercher d’autres ouvrages puisqu’on souhaite seulement le premier ouvrage paru cette année.

Pour FB5, on utilise toujours la même méthode : on parcourt la liste des œuvres, si on trouve un ouvrage respectant la condition, on l’ajoute à la liste.



Enfin, pour FB6, on utilise deux variables locales. Le compteur compte le nombre de livres pris en compte et la variable nb calcule la somme du nombres de livres vendus. Pour obtenir la moyenne, on divise simplement compt par nb.