TD n°7
Manipulations de listes

I Les basiques de la manipulation de listes…

Ecrire les fonctions suivantes :

`Premier` qui récupère le premier élément d'une liste
`Deuxieme` qui récupère le deuxième élément d'une liste
`Troisieme` qui récupère le troisième élément d'une liste
`Saufpremier` qui récupère une liste privée du premier élément
`Saufundeux` qui récupère une liste sans les deux premiers éléments

II Les bons comptes font les bons amis…

1) Ecrire les fonctions :

`Compte` qui compte le nombre d'éléments d'une liste **plate**
`Comptegene` qui compte le nombre d'éléments d'une liste quelconque (donc avec des sous listes)

Idée : Réfléchir à la différence entre ces deux fonctions quant a) à la condition d'arrêt b) aux appels récursifs

2) ***Le serpent qui se mord la queue....***

A l'exécution, faire évaluer `comptegene` en prenant comme liste argument la fonction `comptegene` elle même : vous saurez ainsi combien vous avez d'atomes dans la définition de votre fonction. Qui en aura la moins ? Le match est ouvert.

3) ***De la dérécursivation***

Ecrire `compte_term` une version récursive terminale de `compte`

III Et la liste continue…

Ecrire les fonctions suivantes :

`Dernier` qui retourne le dernier élément d'une liste (cas particulier : retourne ( ) si la liste est vide
`Saufdernier` qui retourne la liste privée de son dernier élément
`Concatene` qui fait une seule liste avec deux listes
[ex : (`concatene` '(1 2 3) '(4 5 6)) => (1 2 3 4 5 6)]
`Aplatit` qui retourne une liste avec sous-listes en une liste plate
[ex : (`aplatit` '(1 2 (3 4) 5)) => (1 2 3 4 5)]
`Retourne` qui retourne la liste argument plate renversée
[ex : `Retourne` '(1 2 3) => (3 2 1))]quelle que soit la longueur de la liste
`Retournegene` qui retourne la liste et toutes les sous listes inversées
[ex : (`retournegene` '(1 2 (3 4) 5)) => (5 (4 3) 2 1)]

IV De l'écriture préfixée à l'ecriture infixée et réciproquement

Ecrire les fonctions :

1) `pre->in` qui transforme une expression arithmétique de l'écriture préfixée vers infixée
[ex : (`pre->in` '(+ 5 3)) => (5 + 3)]

`in->pre` qui transforme une expression arithmétique de l'écriture infixée vers préfixée
[ex : (`in->pre` '(5 + 3)) => (+ 5 3)]

Rem : On admet dans cette première partie que les opérateurs ne comportent que deux arguments et les expressions sont donc écrites :
(opérateur operande1 opérande2) en préfixée ou (operande1 opérateur opérande2) en infixée

2) `pre->in2` et `in->pre2` .. améliorant les versions de la question 1)
en tentant compte que les expressions peuvent être également :

a) unaire ex : avec les fonctions sin, cos, tan, log, exp, etc.
b) n_aire ex : l'addition ou la multiplication où l'on peut par ex écrire (+ 1 2 3 4 5 6…)