Durée de l'épreuve : 45 minutes.

Ces problèmes sont présentés par ordre de difficulté croissante :

1. coder une fonction `aussi` qui prend deux arguments `arg1` et `arg2`, des listes plates, et qui retourne la liste des éléments qui figurent à la fois dans `arg1` et dans `arg2` ; par exemple :
   `(aussi '(p a s t a) '(s p a g h e t t i))` ⇢ `(p a s t)`

2. coder une fonction `unik` qui prend deux arguments `arg1` et `arg2`, des listes plates, et qui retourne la liste des éléments de `arg1` qui ne figurent pas aussi dans `arg2` ; par exemple :
   `(unik '(n o u i l l e) '(r a v i o l i))` ⇢ `(n u e)`

3. soit une liste `L` quelconque, dont on sait que certains éléments sont des atomes numériques[1], par exemple : `(setq L '(a z 7 q 4 w 1 x 3))`
   coder une fonction `augmente` qui opère directement son argument pour en augmenter tous les nombres de 10 ; par exemple, après avoir évalué `(augmente L)`, la valeur de `L` serait modifiée en `(a z 17 q 14 w 11 x 13)`

4. coder la fonction `augmente+`, pareille, à ceci près qu'on veut aussi pouvoir traiter des arbres, c'est-à-dire que les atomes numériques peuvent tout à fait se trouver dans les profondeurs de sous-listes, comme dans le cas de
   `(setq A '(a z (e 5 (r t 8) (6 q s d 7)) f 9))`
   qui devra être chirurgicalement transformé en
   `(a z (e 15 (r t 18) (16 q s d 17)) f 19)` ;

Remarque : peu importe ce que retournent les fonctions `augmente` et `augmente+`, puisqu'on les utilise exclusivement pour leur *effet*, et non pour leur *valeur* !

1. atomes numériques pour lesquels le prédicat `numberp` retournerait `t` bien sûr;