Premières fonctions sur les listes

Exercice 1 : Écrivez une fonction `indice` qui prend en entrée une valeur quelconque `e` et une liste `l`, et fournit en sortie :
- l'indice de l'élément `e` dans la liste `l` si `e` est dans `l` (l'indice est alors compris entre *zéro* et la longueur de la liste *moins un*),
- la longueur de la liste `l` si `e` n'est pas dans `l`.

Exercice 2 : Écrivez une fonction `longueurMoyenne` qui prend en entrée une liste dont les éléments sont des chaînes de catactères (par exemple des séquences d'ARN messagers), et qui retourne en sortie la longueur moyenne des chaînes appartenant à la liste.
- On prendra soin de renvoyer un nombre réel (plus précis qu'un entier).
- Si la liste est vide, renvoyer 0.

Exercice 3 : Certaines expériences consistent à mesurer à intervalles assez réguliers (par exemple une fois par jour) le niveau d'expression de plusieurs gènes (typiquement *via* des mesures de fluorescence sur des puces à ADN). On obtient alors, pour chaque gène, un *profil d'expression* au cours de l'expérience, c'est-à-dire la suite des mesures de niveau d'expression de ce gène. Le plus souvent, les niveaux d'expression sont des entiers compris entre 0 et 255 et un profil d'expression est donc en Python une simple liste d'entiers. La longueur de la liste est alors égale au nombre de mesures effectuées sur la durée de l'expérience.

On peut par exemple mener une expérience sur 2 organismes, l'un soumis à un stress et l'autre pas : les gènes ayant des profils d'expression différents d'un organisme à l'autre participent probablement à la réponse de l'organisme au stress appliqué. Pour les gènes qui ne sont pas impliqués, on ne peut naturellement pas attendre des profils d'expression exactement égaux, cependant on peut attendre qu'il soient « co-exprimés » c'est-à-dire que si l'un augmente d'une mesure à la suivante, alors l'autre aussi, et de même s'il diminue.

Écrivez une fonction `coexprime` qui prend en entrée deux profils d'expression pour un gène donné (`p1` et `p2`) et qui retourne un booléen qui dit s'ils sont covariants. Les deux listes sont supposées de même longueur puisqu'elles représentent des mesures successives faites en même temps sur les deux organismes.

Exercice 4 : Bien comprendre les deux fonctions suivantes :

Compresser un brin d'ADN où il y a beaucoup de répétitions successives de nucléotides. L'idée est de ne pas recopier plusieurs fois un nucléotide répété mais de mémoriser dans la liste son nombre d'occurrences. Par exemple la fonction `comprime` appliquée à la chaîne `"AAAGCTTCCCG"` retourne comme résultat la liste `['A',3,'G','C','T',2,'C',3,'G']`. On suppose sans le verifier que le brin est correct (i.e. ne comprend que des `A`, `T`, `G` ou `C`).

```
>>> def comprime (brin) :
...     r = []
...     compteur = 0
...     precedent = ""
...     for nucl in brin :
...         if nucl == precedent :
...             compteur = compteur + 1
...         else :
...             # mémoriser le nucleotide precedent:
...             if compteur > 1 :
...                 r = r + [ precedent , compteur ]
...             elif compteur == 1 :
...                 r = r + [ precedent ]
...             # préparer la suite:
...             precedent = nucl
...             compteur = 1
...     #traiter le dernier nucleotide en sortant du for:
...     if compteur >= 2 :
...         return( r + [ precedent , compteur ] )
...     else :
...         return( r + [ precedent ] )
```

T.S.V.P. ⟶

et pour décompresser :

```
>>> def deploie (l) :
...     r = ""
...     precedent = ""
...     for elem in l :
...         if type(elem) != type(0) :
...             r = r + elem
...             precedent = elem
...         else :
...             while elem > 1 :
...                 r = r + precedent
...                 elem = elem - 1
...     return(r)
```