TP n° 4 - Fonctions élémentaires sur les listes

Exercice 1. *Récursion !*

Dans cet exercice les listes seront représentées par des listes chaînées comme il a été vu en cours :

```
class Noeud:
    pass
def noeud(x):
    n = Noeud()
    n.valeur = x
    n.suivant = None
    return n
class Liste:
    pass
def liste(n = None):
    l = Liste()
    l.premier = n
    return l
```

1. Réécrivez les fonctions suivantes :
- `vide(l)` qui renvoie `True` si la liste `l` est vide et `False` sinon ;
- `cons(x, l)` qui renvoie la liste commençant par l'élément `x` suivi de la queue `l` ;
- `tete(l)` qui renvoie le premier élément de la liste `l` ;
- `queue(l)` qui renvoie la queue de la liste `l`.

(ces fonctions sont toutes très simples à écrire, il ne faut pas plus de 3 lignes pour chacune)

2. Construisez la représentation de la liste [1, 2, 3, 4, 5, 6] en utilisant les fonctions de la question 1 (indication : il faut utiliser `liste` et `cons`).

3. Écrivez de manière récursive les fonctions suivantes en ne manipulant les listes qu'à l'aide des fonctions de la question 1 (donc indépendamment de l'implémentation utilisée) :
- `longueur(l)` qui donne la longueur de la liste ;
- `element(i, l)` qui renvoie le `i`-ème élément de la liste ;
- `supprimer(i, l)` qui renvoie la liste `l` à laquelle on a enlevé le `i`-ème élément ;
- `ajouter(x, i, l)` qui renvoie la liste `l` à laquelle on a ajouté l'élément `x` en position `i`.

4. Écrivez les fonctions un peu plus avancées suivantes (toujours de manière récursive, en n'utilisant que les primitives de la question 1) :
- `map(f, l)` qui applique à chaque élément de `l` la fonction `f` ;
- `count(x, l)` qui compte le nombre d'occurrences de `x` dans la liste `l` ;
- `filter(f, l)` qui renvoie la liste des éléments de `l` pour lesquels la fonction `f` renvoie `True` (on suppose que la fonction `f` ne renvoie que `True` ou `False`)
- `reduce(f, l, x)` : on suppose que la fonction `f` prend 2 arguments. Elle commence par calculer `x1 = f(x, y0)` où `y0` est le premier élément de la liste `l`, puis elle calcule `x2 = f(x1, y1)` où `y1` est le second élément de la liste, et ainsi de suite jusqu'à épuiser tous les éléments de la liste. On renvoie alors la dernière valeur obtenue.

Exercice 2. *Itération ! (?)*

L'autre encodage des listes vu en cours (version itérative) utilise des tableaux :
- Une liste est représentée par un tableau de taille `MAX` et un entier qui représente la taille de la liste ;
- La liste vide est donc un tableau de taille `MAX` (comme toutes les autres) dont les valeurs n'ont pas d'importance et dont la taille est 0 ;

– La liste `[10, 20, 30, 40]` est représentée par un tableau de taille `MAX` (au risque de se répéter...) dont les 4 premières valeurs sont respectivement 10, 20, 30 et 40 et l'entier 4 qui indique la taille de la liste.

Remarque : les listes sont donc ici encore représentées par un couple mais cette fois-ci le un couple de la forme `(tab, n)` contenant un tableau et un entier.

Autre remarque : On prendra dans le TP la valeur `MAX = 100`.

1. Écrivez les fonctions suivantes pour qu'elles manipulent des listes représentées par des tableaux :
- `liste();`
- `longueur(l);`
- `element(i, l);`
- `ajouter(x, i, l);`
- `supprimer(i, l).`

2. En n'utilisant que les fonctions de la question 1 (mais de l'exercice 2 cette fois-ci !) pour manipuler les listes, écrivez de manière itérative (avec des boucles quand nécessaire) les fonctions `vide(l)`, `tete(l)`,`queue(l)` et `cons(x, l)`.

3. Écrivez de manière itérative les fonctions de la question 4 de l'exercice 1.