Listes - Exercices

Exercice 1

Ouvrir le fichier `LinkedList_stump.py` (*stump* veut dire moignon en anglais) et regarder en détail l'implémentation de la structure de liste chaînée vue en cours.
Pour l'utiliser inclure `from LinkedList_stump import LinkedList` dans votre code. Écrire une fonction `liste_N` qui

- en entrée prend un `int` n ;
- renvoie la liste chaînée composée des éléments 1, 2, ..., n.

Exercice 2 : Terminer l'implémentation objet

Le fichier `LinkedList_stump.py` est incomplet. Tu vas devoir le compléter. Une fois ceci fait tu pourras lui donner le nom `LinkedList.py`.

1. À quoi sert la méthode `__getitem__` ?
2. En s'inspirant de la méthode précédente, écrire la méthode `find`.
3. Coder la méthode `extend` qui
   - en entrée prend une deuxième liste chaînée ;
   - ajoute tous les éléments de cette liste à la première, dans l'ordre

   Par exemple, si `L` vaut `<1, 2, 4>` et si `L2` vaut `<6, 5>` alors `L.extend(L2)` vaut `<1, 2, 4, 6, 5>`.
4. Coder la méthode `__eq__` qui
   - en entrée prend une seconde liste chaînée ;
   - renvoie `True` si les deux listes ont les mêmes éléments aux mêmes places et `False` sinon.

Exercice 3 : Algorithmes récursifs

Écrire un script `recursive_functions.py` qui importe `LinkedList.py` et

1. Écrire la fonction `recursive_length` qui
   - en entrée prend une liste chainée ;

- en sortie renvoie un `int` qui est sa longueur en procédant de manière récursive.

2. Écrire la fonction `recursive_find` qui
   - en entrée prend une liste chainée et un élément ;
   - en sortie renvoie un `int` qui est la position de l'élément s'il figure dans la liste, et -1 sinon.
     Cette fonction procède récursivement.

   Conseil : Pour chacune de ces fonctions, écrire une sous-fonction récursive qui procède sur des données de type `Cell` et qui sera appelée par la fonction principale.