Exercice 1 : Listes chaînées - nouvelles fonctionnalités
Reprendre la classe *Liste* vue en cours et ajouter les méthodes ci-après :

1. ___*str*___ s'appelle directement : **print(nom_objet)**.

```
    def afficher_rec(self, maillon: object)->str:
        """
        crée la chaîne d'affichage pour __str__
        """
        if maillon is None:
            return "\n" # renvoi àla ligne
        else:
            return str(maillon.valeur)+" "+self.afficher_rec(
                maillon.suivant)

    def __str__(self):
        return self.afficher_rec(self.tete)
```

2. Version impérative.

```
    def renverser(self)->None:
        res = None
        maillon_en_cours = self.tete
        while maillon_en_cours is not None:
            res = Maillon(maillon_en_cours.valeur, res)
            maillon_en_cours = maillon_en_cours.suivant
        # la liste est retournée on récupère la tête
        self.tete = res
```

3. S'appuyer sur le schéma

```
    def concatener_rec(self, tete1: object, tete2: object)->
        object:
        """
        méthode interne pour additionner 2 listes
        """
        if tete1 is None:
            return tete2
        else:
            return Maillon(tete1.valeur, self.concatener_rec(
                tete1.suivant, tete2))

    def concatener(self, l: object)->object:
        """
        appel principal de la méthode récursive pour
            additionner 2 listes
        """
        res = Liste()
        res.tete = self.concatener_rec(self.tete, l.tete)
        return res
```

4. La complexité dépend de la taille de la première liste mais pas du tout de celle de la seconde.

Christophe Viroulaud

Exercice 2 :

```
1     def inserer_rec(self, val: int, lst: object)->object:
2         """
3         méthode interne pour recréer la liste avec l'élément insér
              é
4         """
5         if lst is None or val <= lst.valeur:
6             return Maillon(val, lst)
7         else:
8             return Maillon(lst.valeur, self.inserer_rec(val, lst.
                  suivant))
9
10    def inserer(self, val: int)->None:
11        """
12        appel principal pour l'insertion dans une liste triée
13        """
14        self.tete = self.inserer_rec(val, self.tete)
```


Christophe Viroulaud