Quelques méthodes de mutation de listes

La classe `list` propose plusieurs dizaines de méthodes sur les listes, dont certaines provoquent des mutations de la liste, comme `.append`, `.insert`, `.pop` ou `.sort`.

```
>>> L = [8, 3, -2, 3, 7, 5]
>>> id (L)                      # l'adresse de L en mémoire (cf. page 87)
4522466832
>>> L.append(4)                 # ajout de 4 en queue de L (aucun résultat)
>>> L                           # L a muté !
[8, 3, -2, 3, 7, 5, 4]
>>> id(L)
4522466832                      # L n'a pas bougé !
```

Il ne faut pas confondre cette mutation avec une simple affectation qui construirait une autre liste ailleurs en mémoire, avec un coût $\mathcal{O}(n)$.

```
>>> L = [8, 3, -2, 3, 7, 5]
>>> id (L)                      # l'adresse de L en mémoire
4522466912
>>> L = L + [4]                 # concaténation puis affectation
>>> L                           # L a été affecté à une nouvelle liste !
[8, 3, -2, 3, 7, 5, 4]
>>> id(L)
4522488368                      # donc L a bougé en mémoire !
```

La méthode `.append` sur les listes est très efficace (coût $\mathcal{O}(1)$ en moyenne) et ne dépend pas de la longueur de la liste. La méthode `pop` permet de supprimer un élément de liste (`L.pop(i)` supprime l'élément numéro i avec un coût en $\mathcal{O}(n)$). La méthode `.insert` permet d'insérer un nouvel élément à un numéro donné dans la liste (coût $\mathcal{O}(n)$). En particulier, ajouter un nouvel élément en tête d'une liste coûte $\mathcal{O}(n)$, que ce soit par recopie ou par mutation. Enfin, la méthode `.sort` – très rapide – permet de trier une liste *sur place*, sans construire de copie de la liste (contrairement à la fonction `sorted` qui ne modifie pas l'original).

```
>>> L = [8, 3, -2, 3, 7, 5]
>>> L.insert(4,10)              # mutation sur place, aucun résultat
>>> L
[8, 3, -2, 3, 10, 7, 5]         # insertion de 10 en numéro 4 de L
>>> L.sort()                    # tri croissant de L sur place
>>> L
[-2, 3, 3, 5, 7, 8, 10]

```

```
>>> L.pop(3)                    # on arrache l'élément numéro 3
5
>>> L
[-2, 3, 3, 7, 8, 10]            # L.pop() arrache le dernier, coût 𝒪(1)
>>> sorted([8, 3, -2, 3, 10, 7, 5])     # construction d'une copie triée
[-2, 3, 3, 5, 7, 8, 10]
```