Exercices liste chaînée
Correction

Christophe Viroulaud

Terminale - NSI

Archi 04

Sommaire

Exercice 1

```
1  def longueur(lst: tuple) -> int:
2      if len(lst) == 0:
3          return 0
4      else:
5          return 1 + longueur(lst[1])
```

```
1 def afficher(lst: tuple) -> str:
2     if len(lst) == 0:
3         return "fin"
4     else:
5         return lst[0] + " - " + afficher(lst[1])
```

Sommaire

Exercice 2

```
1  def inserer_rec(self, val: int, n: int, m: object) -> None:
2      """
3      méthode interne pour placer val au rang n
4      si n est trop grand, place l'élément en fin de liste
5      """
6      if n == 0: # ajout en première place
7          nouveau = Maillon(val, m) # m est self.tete
8          self.tete = nouveau
9      elif n == 1: # position trouvée
10         nouveau = Maillon(val, m.suivant)
11         m.suivant = nouveau
12     elif m.suivant is None: # n trop grand
13         nouveau = Maillon(val, m.suivant)
14         m.suivant = nouveau
15     else:
16         self.inserer_rec(val, n-1, m.suivant)
```

```
1  def inserer(self, val: int, n: int) -> None:
2      """
3      appel principal de l'insertion pour placer
       val en n
4      """
5      self.inserer_rec(val, n, self.tete)
```

Sommaire

Exercice 3

```python
def dernier_imp(self) -> int:
    # gestion liste vide
    if self.tete is None:
        return -1

    en_cours = self.tete
    while en_cours.suivant is not None:
        en_cours = en_cours.suivant
    return en_cours.valeur
```

```python
def dernier_aux(self, en_cours: Maillon) -> int:
    if en_cours.suivant is None:
        return en_cours.valeur
    else:
        return self.dernier_aux(en_cours.suivant)

def dernier_rec(self) -> int:
    # gestion liste vide
    if self.tete is None:
        return -1

    return self.dernier_aux(self.tete)
```

Sommaire

Exercice 4

- Initialiser le **`Maillon`** en cours avec la tête.
- Tant que le **`Maillon`** en cours n'est pas vide :
  - Créer un **`Maillon`** et l'attacher au précédent.
  - Prendre le **`Maillon`** suivant.
- Réinitialiser la tête avec le dernier **`Maillon`** visité.

```python
def renverser(self) -> None:
    res = None
    en_cours = self.tete
    while en_cours is not None:
        # crée maillon et l'attache au précédent
        res = Maillon(en_cours.valeur, res)
        # va voir le maillon suivant
        en_cours = en_cours.suivant
    self.tete = res
```

Sommaire

Exercice 5

- Initialiser le **`Maillon`** en cours avec la tête.
- Tant que le **`Maillon`** en cours n'est pas vide :
  - Créer un **`Maillon`** doublon du **`Maillon`** en cours.
  - Faire pointer le **`Maillon`** en cours sur le doublon.
  - Initialiser le **`Maillon`** en cours avec le suivant du doublon.

```
1 def dupliquer(self):
2     en_cours = self.tete
3     while en_cours is not None:
4         doublon = Maillon(en_cours.valeur, en_cours.suivant)
5         en_cours.suivant = doublon
6         en_cours = doublon.suivant
```

Sommaire

Exercice 6

```
1 from liste import Liste, Maillon
```

```
1  l1 = Liste()
2  l1.ajoute(8)
3  l1.ajoute(5)
4  l1.ajoute(3)
5  l1.ajoute(9)
6  l1.ajoute(10)
7
8  l2 = Liste()
9  l2.ajoute(2)
10 l2.ajoute(4)
11 l2.ajoute(7)
```

```
1  def concatener(l1: Liste, l2: Liste) -> Liste:
2      def concatener_rec(tete1: Maillon, tete2: Maillon) -> Maillon:
3          """
4          fonction interne pour additionner 2 listes
5          """
6          if tete1 is None:
7              return tete2
8          else:
9              return Maillon(tete1.valeur,
10                            concatener_rec(tete1.suivant, tete2))
11
12     res = Liste()
13     res.tete = concatener_rec(l1.tete, l2.tete)
14     return res
```

Remarque

- La liste 2 n'est pas recopiée dans la nouvelle liste. Une modification du contenu de l'une d'elle modifie l'autre : c'est un **effet de bord** (pas nécessairement désiré).
- La complexité dépend de la taille de la première liste, qui est entièrement copiée.