

# Exercices listes chaînées

## Correction

Christophe Viroulaud

Terminale - NSI

**Archi 04**

1. Exercice 1

2. Exercice 2

3. Exercice 3

Exercice 1

Exercice 2

Exercice 3

# Exercice 1 - 1

Exercice 1

Exercice 2

Exercice 3

```
1 class Maillon:
2     """
3     Crée un maillon de la liste chaînée
4     """
5
6     def __init__(self, val: int, suiv: object) -> None:
7         self.valeur = val
8         self.suivant = suiv
```

# Exercise 1 - 2

Exercice 1

Exercice 2

Exercice 3

```
1 class Liste:
2     """
3     Crée une liste chaînée
4     """
5
6     def __init__(self):
7         self.tete: Maillon = None
```

# Exercice 1 - 3

Exercice 1

Exercice 2

Exercice 3

```
1 def ajoute(self, val: int) -> None:
2     self.tete = Maillon(val, self.tete)
```

# Exercise 1 - 4

Exercice 1

Exercice 2

Exercice 3

```
1 lst = Liste()  
2 lst.ajoute(8)  
3 lst.ajoute(5)  
4 lst.ajoute(3)  
5 lst.ajoute(9)  
6 lst.ajoute(10)
```

# Exercice 1 - 5

Exercice 1

Exercice 2

Exercice 3

```
1 def dernier(self) -> int:
2     # gestion d'erreur
3     if self.tete is None:
4         return -1
5
6     en_cours = self.tete
7     while en_cours.suivant is not None:
8         en_cours = en_cours.suivant
9     return en_cours.valeur
```

# Exercise 1 - 6

Exercice 1

Exercice 2

Exercice 3

```
1 def dernier_aux(self, m: Maillon) -> int:
2     if m.suivant is None:
3         return m.valeur
4     else:
5         return self.dernier_aux(m.suivant)
6
7 def dernier_rec(self) -> int:
8     # gestion d'erreur
9     if self.tete is None:
10         return -1
11
12     return self.dernier_aux(self.tete)
```



# Exercice 1 - 7

Exercice 1

Exercice 2

Exercice 3

fin

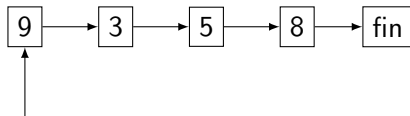


FIGURE 1 – Tant que le Maillon en cours n'est pas None

# Exercice 1 - 7

Exercice 1

Exercice 2

Exercice 3

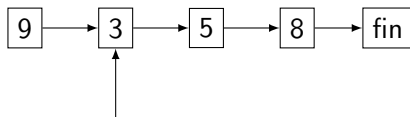
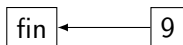


FIGURE 2 – Créer un nouveau Maillon et l'attacher au précédent.

Exercice 1

Exercice 2

Exercice 3

```
1 def renverser(self) -> None:
2     res = None
3     en_cours = self.tete
4     while en_cours is not None:
5         # crée maillon et l'attache au précé
        dant
6         res = Maillon(en_cours.valeur, res)
7         # va voir le maillon suivant
8         en_cours = en_cours.suivant
9     self.tete = res
```

Exercice 1

Exercice 2

Exercice 3

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

1. Exercice 1

2. Exercice 2

3. Exercice 3

Exercice 1

Exercice 2

Exercice 3

## Exercice 2 - 1

Exercice 1

Exercice 2

Exercice 3

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

## Exercice 2 - 2

Exercice 1

Exercice 2

Exercice 3

```
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(4)
10 l2.ajoute(0)
11 l2.ajoute(2)
```

## Exercice 2 - 3

Exercice 1

Exercice 2

Exercice 3

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



Exercice 1

Exercice 2

Exercice 3

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**.

Exercice 1

Exercice 2

Exercice 3

Le complexité dépend de la taille de la première liste, qui est entièrement copiée.

1. Exercice 1

2. Exercice 2

3. Exercice 3

Exercice 1

Exercice 2

Exercice 3

## Exercice 3 - 1

Exercice 1

Exercice 2

Exercice 3

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

## Exercise 3 - 2

Exercice 1

Exercice 2

Exercice 3

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