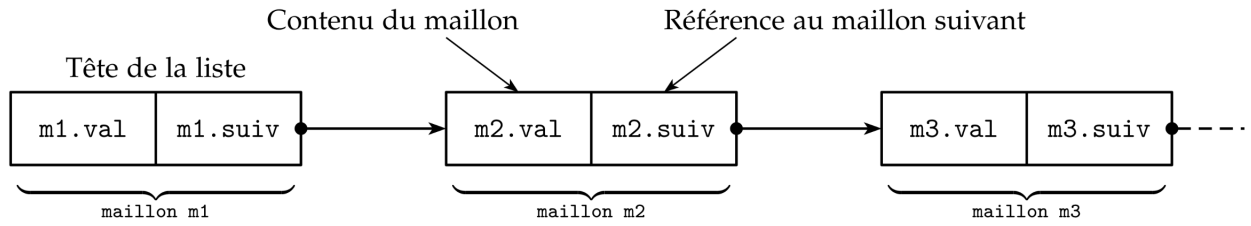
# Les listes (chaînées)

## Capacités attendues

- ✓ Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.
- ✓ Distinguer des structures par les méthodes qui les caractérisent.
- ✓ Choisir une structure de données adaptée à la situation à modéliser.

### Définition

En informatique, une liste (simplement chaînée) est une structure de données composée d'une séquence d'éléments reliés entre eux : chaque élément, ou maillon, fait référence à l'élément suivant.
La tête de la liste est le premier maillon de celle-ci.




Dans un tableau Python (de type `List`) tous les éléments sont stockés les uns à la suite des autres dans un espace mémoire donné. Il est facile d'ajouter un élément à la fin d'un tel tableau (méthode `append`) mais l'opération d'insertion d'un élément est coûteuse car elle nécessite de décaler en mémoire tous les éléments suivants l'élément inséré.

Les listes chaînées présentent l'avantage de faciliter l'insertion car les maillons ont des emplacements mémoire distincts. Il suffit de faire *pointer* un maillon vers un autre emplacement mémoire pour insérer un nouvel élément, les suivants n'ayant pas à être déplacés.

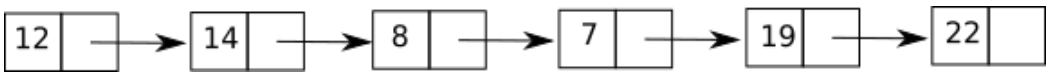
**Exercice 1**

En quoi une liste chaînée est-elle une structure récursive ?

Car son élément suivant est lui-même une liste.

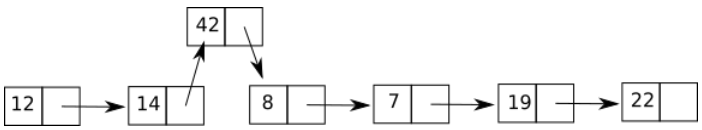
**Exercice 2**

On considère la liste chaînée suivante :




Modifier ce schéma pour représenter l'ajout de la valeur $42$ entre les maillons dont les valeurs sont $14$ et $8$.

Décrire avec une phrase la façon de faire.



Il faut modifier l'élément suivant de l'élément précédent le nouvel élément, et faire pointer l'élément suivant du nouvel élément vers l'élément qui était à sa place.

**Exercice 3**

On considère l'implémentation suivante d'un maillon.

```
class Maillon :

    def __init__(self, valeur):
        """ Maillon(obj)
            renvoie un maillon contenant l'élément valeur """
        self.valeur = valeur
        self.suivant = None
```

1. Quelles sont les instructions qui permettent de définir la liste `0 -> 1 -> 2` ?

```
maillon = Maillon(0)
maillon.suivant = Maillon(1)
maillon.suivant.suivant = Maillon(2)
```

2. Étant donné le maillon de tête d'une liste, quelles instructions permettent d'afficher le nombre d'éléments qu'elle contient ?

```
compteur = 1
while maillon.suivant:
| compteur = compteur + 1
| maillon = maillon.suivant

print(f'Cette liste comporte {compteur} maillons.')
```

**Exercice 4**

1. Télécharger sur notre site le fichier `maillon-e.py`. La classe `Maillon` vous est donnée, il n'y a pas à la changer.

   Une batterie de tests vous est aussi donnée. Le but est qu'il n'y ait pas d'erreur dans les tests en exécutant le fichier. Pour ceci, vous devez compléter les **fonctions** du fichier. Seule la fonction `longeur` est déjà écrite. Aidez-vous des *docstrings*.

2. Télécharger sur notre site le fichier `listes-e.py`. Recopier des morceaux du fichier précédent comme indiqué en commentaire. (Recopier la classe `Maillon` à l'emplacement donné, puis les fonctions à l'autre emplacement donné.)

   Enfin, comme précédemment, une batterie de tests est donnée et doit fonctionner. Pour ceci, compléter les méthodes de la classe `Liste`. Aidez-vous des *docstrings*.

**Exercice 5**

1. Quel est le rapport entre une liste chaînée et un arbre ? C'est un arbre filiforme
2. Quel est le coût en temps (la complexité) de la recherche d'un élément dans une liste chaînée ? linéaire
3. Que pourrait-on proposer comme solution pour qu'on puisse parcourir une liste chaînée dans les deux sens ? liste doublement chaînée avec pointeur vers maillon précédent