– avec des champs nommés (dictionnaires)

REMARQUE

Variantes des listes chaînées

Il existe de nombreuses variantes des listes chaînées :

– listes cycliques (le dernier élément est lié au premier)
– listes doublement chaînées où chaque élément est lié à l'élément suivant **et** à l'élément précédent dans la liste
– listes cycliques doublement chaînées
– etc.

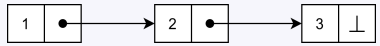
2 — Structure de liste chaînée

Une **liste chaînée** permet avant tout de réprésenter une liste dont les éléments sont chaînés entres eux, permettant ainsi le passage d'un élément à l'élément suivant.

Ainsi, chaque élément est stocké dans un bloc mémoire que l'on pourra appeler **maillon**. Ce maillon possède deux informations : la `valeur` stockée et l'adresse mémoire du maillon `suivant`.

Exemple

Par exemple on a illustré une liste contenant trois éléments, respectivement 1, 2 et 3.



Chaque élément de la liste est matérialisé par un emplacement en mémoire contenant d'une part sa valeur (dans la case de gauche) et d'autre part l'adresse mémoire de la valeur suivante (dans la case de droite).

Dans le cas du dernier élément, qui ne possède pas de valeur suivante, on utilise une valeur spéciale désignée ici par le symbole ⊥ et marquant la fin de la liste

Une façon traditionnelle de représenter une liste chaînée en Python consiste à utiliser une classe décrivant les maillons de la liste, de sorte que **chaque élément de la liste est matérialisé par un objet de cette classe**.

Implémenter la classe `Maillon` possédant deux attributs : `valeur` pour la valeur de l'élément (l'entier, dans notre exemple) ; et `suivant` pour le maillon suivant de la liste.

Exemple

Exemple 1 : pour affecter à `m2` le maillon contenant la valeur `3` et n'ayant aucun maillon suivant on écrira l'instruction `m2 = Maillon(3, None)`.

Exemple

Exemple 2 : Pour construire une liste, on applique le constructeur autant de fois qu'il y a d'éléments dans la liste. Ainsi pour construire la liste 1, 2, 3 du schéma que l'on stocke dans la variable `lst` on écrira : `lst = Maillon(1, Maillon(2, Maillon(3, None)))`.

[11]:
```
from doctest import testmod

class Maillon:
    """
    Une classe pour représenter le maillon d'une liste

    Attributs
    ---------
    valeur : type
        valeur contenue dans le maillon
    suivant : maillon
        maillon suivant ou None si pas de maillon
    """

    def __init__(self, valeur, suivant):
        """Constructeur de classe

        Args:
            valeur (type): valeur stockée dans le maillon
            suivant (maillon): maillon suivant ou None

        Exemples et tests:
        >>> l3 = Maillon(3, None)
        >>> assert (l3.valeur == 3)
        >>> assert (l3.suivant == None)

        >>> l2 = Maillon(2, l3)
        >>> assert (l2.valeur == 2)
        >>> assert (l2.suivant.valeur == 3)

        >>> l1 = Maillon(1, l2)
        >>> assert (l1.valeur == 1)
        >>> assert (l1.suivant.valeur == 2)
        >>> assert (l1.suivant.suivant.valeur == 3)
        >>> assert (l1.suivant.suivant.suivant == None)

        >>> l1.suivant.suivant.suivant.valeur
        Traceback (most recent call last):
```

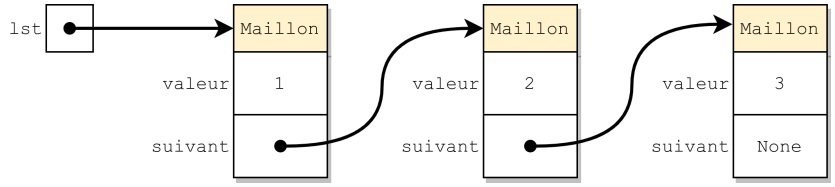


```
        AttributeError: 'NoneType' object has no attribute 'valeur'
        """
        self.valeur = valeur
        self.suivant = suivant


lst = Maillon(1, Maillon(2, Maillon(3, None)))

testmod()
```

1]: `TestResults(failed=0, attempted=12)`



La valeur contenue dans la variable lst est l'adresse mémoire de l'objet contenant la valeur 1,

Définition récursive d'une liste chaînée

Comme on le voit, une liste est soit la valeur `None`, soit un objet de la classe `Maillon` dont l'attribut `suivant` contient une liste. C'est là une **définition récursive** de la notion de liste.

REMARQUE

Représentation alternatives

On peut représenter une liste chaînée par autre chose qu'un classe :

- avec des couples : `lst = (1, (2, (3, None)))`
- avec des tableaux à deux éléments : `lst = [1, [2, [3, None]]]`