Observez le code ci-contre. Il implémente en POO la notion de liste chaînée : un ensemble de *cellules* « liées » l'une à la suivante. Le fonctionnement d'une telle liste n'est pas loin de la logique des *piles* (structures abstraites de données où le dernier entré est le premier sorti). D'où le nom de classe .

Notez le recours à **deux** classes : une pour les cellules, une pour les « PListes ». On peut dès lors disposer d'une ***vraie*** « PListe » *vide* : il suffit qu'elle ne contienne *aucune* cellule. On peut donc avoir une PListe *non vide* contenant une cellule dont la *valeur* est `None`.

```
class Cellule:
    def __init__(self, valeur, suivante):
        self.valeur = valeur
        self.suivante = suivante

    def __repr__(self):
        return str(self.valeur)

class PListe:
    def __init__(self):
        """Crée une liste VIDE"""
        self.tete = None

    def insere(self, valeur):
        """Insère une valeur (en tant qu'instance de
        la classe Cellule) EN TÊTE de liste"""
        self.tete = Cellule(valeur, self.tete)

    def est_vide(self):
        return self.tete is None

    def longueur(self):
        if self.est_vide():
            return 0
        else:
            cellule = self.tete
            long = 1
            while cellule.suivante is not None:
                long += 1
                cellule = cellule.suivante
            return long
```

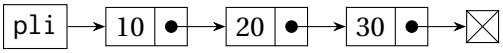
1. D'après le code ci-contre, donner l'instruction permettant de créer une liste vide nommée `pliv` :

   .....

2. Peut-on créer une liste ayant d'emblée une valeur ? Si oui, comment ?

   .....

3. Donner les instructions pour créer la liste chaînée représentée ci-dessous (le symbole ⊠ représente `None`) :



   .....

   .....

   .....

4. En vous inspirant du code fourni, définir une méthode nommée `ieme_element`, qui viendra enrichir la classe `PListe`, qui prendra un paramètre entier `i` et renverra la valeur de la cellule en $i^{ème}$ position à partir de la tête de la liste (les positions seront comptées **à partir de 1**). Vous prendrez la précaution de vérifier à l'aide d'une assertion que cette $i^{ème}$ position existe bien !

   .....

   .....

   .....

   .....

   .....

   .....

   .....

   .....

5. Donner ci-dessous le code le la méthode `depile`, qui viendra elle aussi enrichir la classe **`PListe`**. Elle dépilera l'élément de tête, c'est-à-dire qu'elle supprimera la cellule de tête et renverra sa valeur.

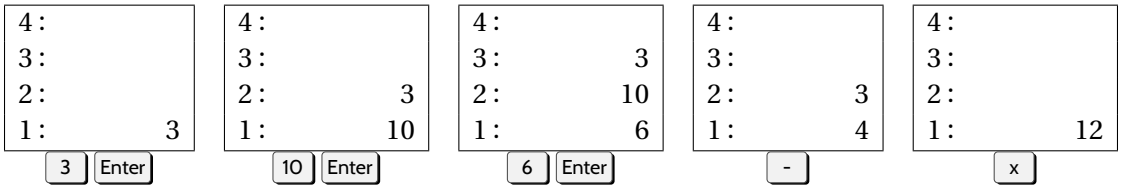
On supposera désormais que, dans la suite, les `PListe` ne contiendront que des *valeurs numériques*.

Les calculatrices HP ont longtemps fait appel à la Notation Polonaise Inverse (RPN en anglais, pour *Reverse Polish Notation*) : l'expression « $3 \times (10 - 6)$ » peut s'écrire « 10 6 − 3 × » en RPN.

Concrètement, chaque nombre rencontré est en fait *empilé* dans une *pile de travail*. À l'écran, elle sera représentée *à l'envers* : le haut de la pile est en bas, les entrées et sorties se font donc *par le bas* !

C'est à la base une pile abstraite, aux fonctionnalités étendues pour permettre la réalisation de calculs, et dotée de fonctionnalités autorisant à modifier son organisation interne (on peut par exemple échanger la position de valeurs dans la pile). Les opérations arithmétiques *agissent sur le sommet de la pile* (opérations *unaires*, comme le changement de signe ou la racine carrée), ou sur les *deux premiers niveaux de la pile* (opérations *binaires* : attention à l'ordre, le niveau le plus élevé intervient en 1re position dans la réalisation du calcul).

Sur une calculatrice RPN, le calcul précédent se réalise ainsi : [3] [Enter] [10] [Enter] [6] [Enter] [-] [×]. L'affichage de la machine évolue comme suit (notez les numéros d'ordre dans la pile, à gauche) :



Notez que le calcul « consomme » le (ou les 2) premier(s) niveau(x) de la pile (ils sont donc *retirés* de la pile), et que le résultat est placé à son *sommet*.

6. Écrire une méthode `affichage` qui viendra enrichir la classe **`PListe`**. Elle n'aura aucun paramètre et son appel provoquera l'affichage de la pile tel qu'encadré ci-dessus.

   Indication : *utilisez les méthodes de chaînes de caractères* `ljust` *(left justify) et* `rjust` *(right justify) :* `'1'.ljust(2) + ':'` *donnera la chaîne* `'1 :'` *(notez l'espace ajouté* si nécessaire *pour avoir* exactement *2 caractères avant le « : ») et si* `valeur = 1.234`, `str(valeur).rjust(10)` *donnera la chaîne* `'␣␣␣␣␣1.234'` *(une chaîne de 10 caractères précisément, avec autant d'espaces en tête que nécessaire).*

7. Implémenter une méthode `neg`, qui change le signe de la valeur située en tête de pile (aucun paramètre, aucune valeur de retour).

8. Implémenter une méthode `plus`, qui ajoute les éléments présents sur les deux premiers niveaux de la pile et place le résultat sur le sommet de la pile (aucun paramètre, aucune valeur de retour).

   Important : *utiliser une assertion pour gérer le cas où la pile n'a pas une taille adaptée.*

9. Implémenter une méthode `moins`, qui soustrait les éléments présents sur les deux premiers niveaux de la pile (**attention :** niveau 2 moins niveau 1) et place le résultat sur le sommet de la pile.

   Important : *utiliser une assertion pour gérer le cas où la pile n'a pas une taille adaptée.*

10. Dessiner l'affichage provoqué par la succession d'instructions suivantes (vous pourrez imiter la présentation de la page 2, ou les affichages sont l'un à côté de l'autre ; inutile de les encadrer !). Vous prendrez soin de préciser clairement les numéros des lignes correspondant aux affichages.

```
1  rpl = PListe()
2  rpl.insere(5)
3  rpl.affichage()
4  rpl.insere(2)
5  rpl.affichage()
6  rpl.insere(7)
7  rpl.affichage()
8  rpl.neg()
9  rpl.affichage()
10 rpl.plus()
11 rpl.affichage()
12 rpl.moins()
13 rpl.affichage()
```