PARTIE A

On donne l'interface d'une pile définie ci-dessous :
Pour exprimer l'interface des piles, nous notons **`Pile[T]`** le type des piles contenants des éléments de type **`T`**.

Fonction	Description
`creer_pile() → Pile[T]`	Crée une pile vide
`est_vide(p : Pile[T]) → bool`	Renvoie **`True`** si **`p`** est vide et **`False`** sinon
`empiler(p : Pile[T], e : T) → None`	Ajoute **`e`** au sommet de **`p`**
`depiler(p : Pile[T]) → T`	Retire et renvoie l'élément au sommet de **`p`**

Exercice 1
On dit qu'une chaîne de caractères comprenant, en autres choses, des parenthèses " (" et ") " est bien parenthésée lorsque chaque parenthèse ouvrante est associée à une unique fermante, et réciproquement.

1) Implémenter une fonction **`controle(E)`** prenant en paramètre une expression **`E`** (chaine de caractères) et qui renvoie **`True`** si l'expression **`E`** est bien paramétrée et **`False`** sinon.
   a) A l'aide d'une fonction en Python. (fonction **`controle-py(E)`**)
   b) En utilisant la structure **`Pile(T)`** énoncée ci-dessus.

2) En utilisant la structure **`Pile(T)`** énoncée ci-dessus, écrire une fonction **`controlePile(E, f)`** prenant en paramètre une expression **`E`** **bien parenthésée** et l'indice **`f`** d'une parenthèse fermante, et qui renvoie l'indice de la parenthèse ouvrante associée.

Exercice 2
On considère une expression **`E`** (chaine de caractères) incluant à la fois des parenthèses rondes " (" et ") " et des parenthèses carrées (crochets) " [" et "] ".
L'expression est bien parenthésée si chaque ouvrante est associée à une unique fermante *de même forme*, et réciproquement.
Par exemple, l'expression "[3 + (5 – 7] × 3) " n'est pas correctement parenthésée car la parenthèse fermante ne peut pas venir après le crochet fermant.

En utilisant la structure **`Pile(T)`** énoncée ci-dessus, écrire une fonction **`bonne_par(E)`** prenant en paramètre une expression **`E`** (chaine de caractères) et qui renvoie **`True`** si l'expression **`E`** est bien paramétrée et **`False`** sinon.

PARTIE B

On donne, ci-dessous, la réalisation d'une pile avec une liste chaînée :

```
class Cellule:
    """une cellule d'une liste chaînée"""
    def __init__(self, v, s):
        self.valeur = v
        self.suivante = s
```

```
class Pile:
    """structure de pile"""
    def __init__(self):
        self.contenu = None

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

    def empiler(self, v):
        self.contenu = Cellule(v, self.contenu)

    def depiler(self):
        if self.est_vide():
            raise IndexError("depiler sur une pile vide")
        v = self.contenu.valeur
        self.contenu = self.contenu.suivante
        return v

def creer_pile():
return Pile()
```

Exercice 1

Compléter la classe Pile définie ci-dessus avec trois méthodes additionnelles :

1) **`consulter`**
2) **`vider`**
3) a) **`taille`**

   b) Quelle est l'ordre de grandeur du nombre d'opérations effectuées par la fonction **`taille`** ?
4) Pour éviter le problème du calcul de taille à la question 3)b), on se propose de revisiter la classe **`Pile`** en lui ajoutant un attribut **_taille** indiquant à tout moment la taille de la pile.
   Quelles ***méthodes*** doivent être modifiées, et comment ?

Exercice 2
En utilisant la **`classe Pile,`** réécrire les fonctions **`controle(E)`** et **`controlePile(E, f)`** de *l'exercice 1 Partie A*.

Exercice 3
Une **pile bornée** est une pile dotée à sa création d'une **capacité maximale**.

On propose l'interface suivante :

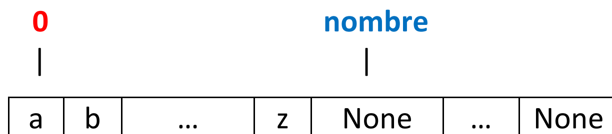
Fonction	Description
`creer_pile(c)`	Crée et renvoie une pile bornée de capacité c
`est_vide(p)` → `bool`	Renvoie **True** si **p** est vide et **`False`** sinon
`est_pleine(p)` → `bool`	Renvoie **True** si **p** est pleine et **`False`** sinon
`empiler(p, e)`	Ajoute **e** au sommet de **p** si **p** n'est pas pleine, et lève une exception **`IndexError`** sinon
`depiler(p)`	Retire et renvoie l'élément au sommet de **p** si **p** n'est pas pleine, et lève une exception **`IndexError`** sinon

On se propose de réaliser une telle **pile bornée** à l'aide d'un tableau dont la taille est fixée à sa création et correspond à la capacité.

Les éléments de la pile sont stockés consécutivement à partir de **l'indice 0** (qui contient l'élément du fond de la pile).

On se donne également un entier enregistrant le **nombre d'éléments** dans la pile, qui permet donc également de désigner l'indice de la prochaine case libre.

Ainsi dans le schéma ci-dessous, les éléments sont ajoutés et retirés du côté droit de la pile :



Travail
Réaliser une telle structure à l'aide d'une classe `PileBornee`, ayant pour attributs le **tableau fixe** et le **nombre d'éléments** dans la pile bornée.
Vous pouvez vous inspirer de la définition de la classe `Pile` ci-dessus.