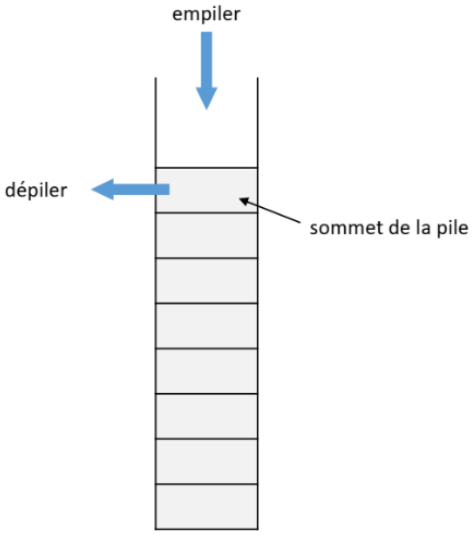
La pile est une structure de données qui donne accès uniquement à la dernière donnée insérée : le sommet de pile. On décrit souvent ce comportement par « dernier entré, premier sorti » LIFO en anglais (Last In First Out)

Représentation :



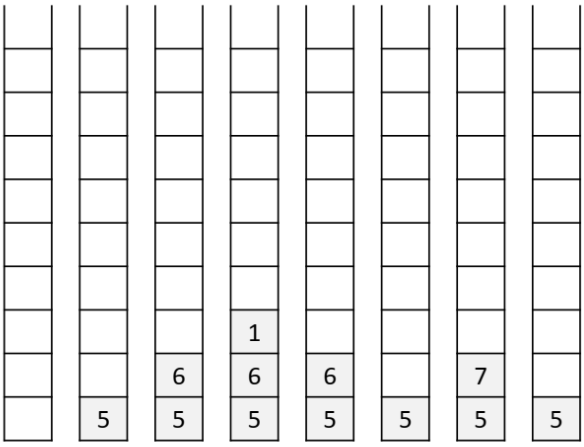
1. TAD

Opérations :

- Constructeur : **Pile()**
  Postconditions : La pile est une pile vide
- Procédure **empiler (e)**
  Postcondition : e est ajouté en sommet de la pile
- Procédure **dépiler ()**
  Précondition : la pile n'est pas vide
  Postcondition : le sommet de la pile est dépilé
  Résultat : retourne le sommet de la pile
- Procédure **vider ()**
  Postcondition : la pile ne contient plus aucun élément, la pile est une pile vide
- Fonction **estVide ()** : renvoie un booléen
  Résultat : retourne vrai si la pile est vide, faux sinon
- Fonction **sommet ()** : renvoie tout type
  Précondition : la pile n'est pas vide
  Résultat : retourne le sommet de la pile
- Fonction **hauteur ()** : renvoie un entier
  Précondition : la pile n'est pas vide
  Résultat : retourne le nombre d'éléments dans la pile

Exemple d'utilisation :

```
Variables locales :
    p : Pile, s : entier, v : booléen
Début
    v ← p.estVide()
    p.empiler(5)
    p.empiler(6)
    p.empiler(1)
    p.dépiler()
    s ← p.sommet()
    p.dépiler()
    p.empiler(7)
    s ← p.sommet()
    p.dépiler()
    v ← p.estVide()
Fin
```



2. Implémentation

Deux implémentations possibles, par exemple :

- Par une liste simplement chaînée : Sommet de pile = tête de liste (pour éviter l'accès en fin de liste en O(n))
  - empiler = ajouter en tête
  - dépiler = supprimer la tête

➢ coût constant O(1)

- Par un tableau dynamique (list Python) : sommet de pile = dernière case (pour éviter les recopies)
  - empiler = ajouter en dernière position
  - dépiler = supprimer l'élément en dernière position

➢ coût amorti constant 0(1)

Coût amorti signifie que l'on ne prend pas en compte les cas rare et coûteux en opérations et donc en temps comme par exemple quand la liste python est pleine et qu'il faut la recopiée en mémoire dans une liste de taille le double.

Implémentation python en POO avec une list :

```
# ********************************************************************************
# CLASSE FILE - CREATION D'UNE STRUCTURE DE TYPE PILE en POO avec une List Python
# ********************************************************************************

class Pile:
    '''Classe définissant une structure de données de type Pile (LIFO: Last In First Out) avec méthodes'''

    # Méthode: Constructeur de la pile
    def __init__(self,L=None):
        '''Méthode qui construit la Pile à partir d'une liste existante qui peut être vide ou non'''
        if L==None:
            L=[]
        self.liste=L

    # SURCHARGE de la Méthode de destruction de la Pile
    def __del__(self):
        self.vider()
        print ("Pile vidée et détruite")

    # SURCHARGE de la Méthode d'AFFICHAGE
    def __repr__(self):
        '''Méthode qui redéfinie l'affichage de la Pile de façon personnalisée, et renvoie False si la Pile est vide'''
        if not self.estVide():
            chaine="Pile:  "
            nbElem=self.hauteur()
            for i in range (0,nbElem-1):
                chaine += eval("f'{self.liste[i]}   '")
            chaine += eval("f'{self.liste[nbElem-1]}'")
            return chaine
        else:
            return "Pile vide !"
```

Travail 1 - Créer un fichier Pile.py, copier le code ci-dessus, est implémenter les méthodes manquantes : empiler, dépiler, vider, estVide, sommet, hauteur

3. Exemple d'utilisation

3.1 Evaluation d'une expression

Il est possible à l'aide d'un Pile et d'un tableau d'évaluer une expression mathématique comme par exemples :

- a = (12 x 5)/6 + 18
- 5 + 56x9 – 54
- ...

Voir fichier pdf