> Version pdf

Type abstrait Pile

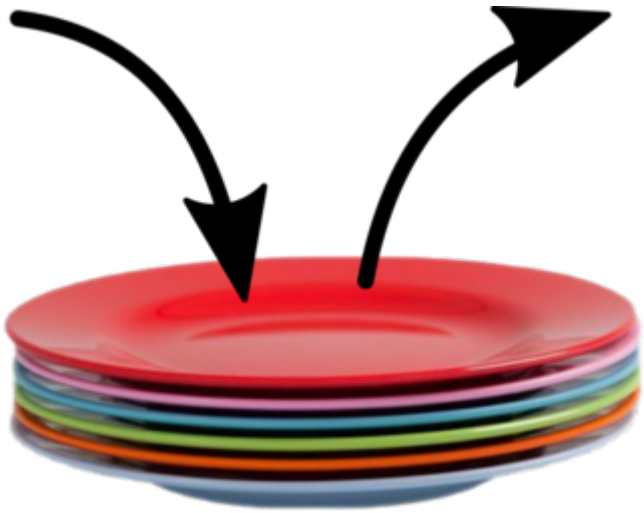
Interface

"Point de cours 1"

Une **pile** est un ensemble ordonné d'éléments qui se manipule comme une *pile d'assiettes* :

- on peut ajouter une assiette au *sommet* de la **pile**, c'est l'opération **empiler**
- on peut retirer l'assiette du *sommet* de de la **pile**, c'est l'opération **dépiler**

Comme dans une **liste** un seul élément est accessible directement, le *sommet* de la **pile**.



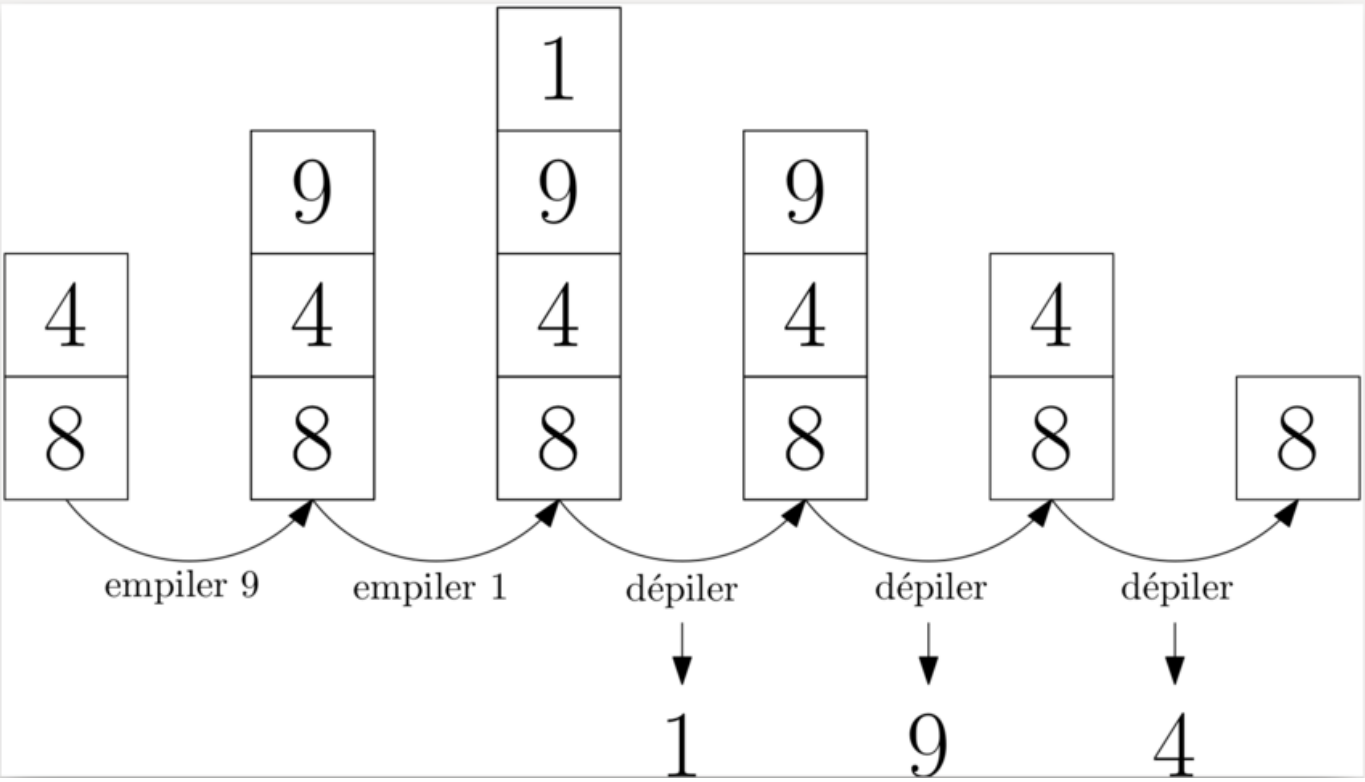
> *Source : Gilles Lassus*

L'interface du type abstrait **Pile** peut se réduire à quatre opérations :

Opération	Signature	Description
creer_pile	creer_pile()	Renvoie une pile
pile_vide	pile_vide(pile)	Renvoi un booléen indiquant si la pile est vide
empiler	empiler(pile, elt)	Ajoute `elt` au sommet de la pile
depiler	depiler(pile)	Retire l'élément au sommet de la pile et le renvoie

Quelques propriétés à retenir :

- Le premier élément qu'on peut retirer d'une pile est forcément le dernier à y être entré, on dit que c'est une structure **Last In First Out (LIFO)**.
- En particulier si on dépile tous les élément, l'ordre dans lequel on les retire de la pile est l'inverse de leur ordre d'insertion.
- L'opération `empiler` peut renvoyer une nouvelle pile si on implémente une pile *immuable* ou modifier la pile par *effet de bord*, si on implémente une pile *mutable* (voir cours liste mutable).
- Les implémentations correctes du type Pile garantissent une complexité constante en $O(1)$ pour les opérations `depiler` et `empiler` .



> *Source : Pierre Duclosson*

"Exemple 1 : applications des piles"

On retrouve la structure Pile dans de nombreuses situations :

- *L'historique* d'un navigateur Web conserve les pages parcourues dans une pile : la flèche gauche permet de revenir en arrière (dépiler) et la flèche droite d'avancer (empiler)
- Lors de l'évaluation d'une *fonction récursive* les contextes des appels récursifs imbriqués sont stockés dans une pile dont la taille est d'ailleurs limité pour éviter les appels infinis.

Lorsque la *pile d'appels* déborde, c'est le fameux *stack overflow* !
Pour éviter le dépassement de capacité de la *pile d'appels*, on peut exprimer une fonction récursive sous forme de boucle en utilisant une pile. Nous verrons une application dans l'algorithme d'exploration de graphe en profondeur.

- On peut évaluer une expression arithmétique en notation postfixé en utilisant une pile.
- à la fin d'un devoir un enseignant récupére une pile de copies et souvent il les corrige en commençant par le sommet de la pile.
- les couches géologiques forment évidemment une pile

Néanmoins si la pile est une structure naturelle pour stocker des objets car les opérations `depiler` et `empiler` sont peu coûteuses, ce n'est pas toujours une solution stisfaisante :

- dans un rayon de supermarché il n'est pas raisonnable de stocker des yaourts sous forme de pile, sinon les plus anciens risquent de n'être jamais achetés
- les personnes qui attendent depuis longtemps devant une salle de concerts avec placement libre ne souhaitent pas que les derniers arrivés soient les premiers entrés !

"Exemple 2"

On donne ci-dessous un exemple d'utilisation de l'interface pour calculer la somme des valeurs des éléments d'une pile d'entiers. Notez qu'on utilise une pile auxiliaire pour reconstituer la pile initiale après avoir dépilé tous les éléments pour effectuer le calcul

```
def somme(pile):
    autre = creer_pile()
    s = 0
    while not pile_vide(pile):
        sommet = depiler(pile)
        empiler(autre, depiler(pile))
        s = sommet + s
    while not pile_vide(autre):
        empiler(pile, depiler(autre))
    return s
```

Implémentations

On présente plusieurs implémentations possibles du type abstrait Pile.

"Implémentation par tableau dynamique Python"

Si on représente une pile par un tableau dynamique de Python (type `list`), en considérant le dernier élément du tableau comme le sommet de la pile,
certaines méthodes implémentent les opérations `empiler` et `depiler` du type abstrait Pile :

Méthode de tableau dynamique	Opération du type abstrait Pile
append	empiler
pop	depiler

Une implémentation fonctionnelle du type abstrait Pile avec un tableau dynamique Python est donc immédiate. Il s'agit d'une **pile mutable**.

```
# Interface du type abstrait Pile
def creer_pile():
    return []

def pile_vide(pile):
    return pile == []

def depiler(pile):
    sommet = pile.pop()
    return sommet

def empiler(pile, elt):
    pile.append(elt)
```

"Implémentation par liste chaînée"

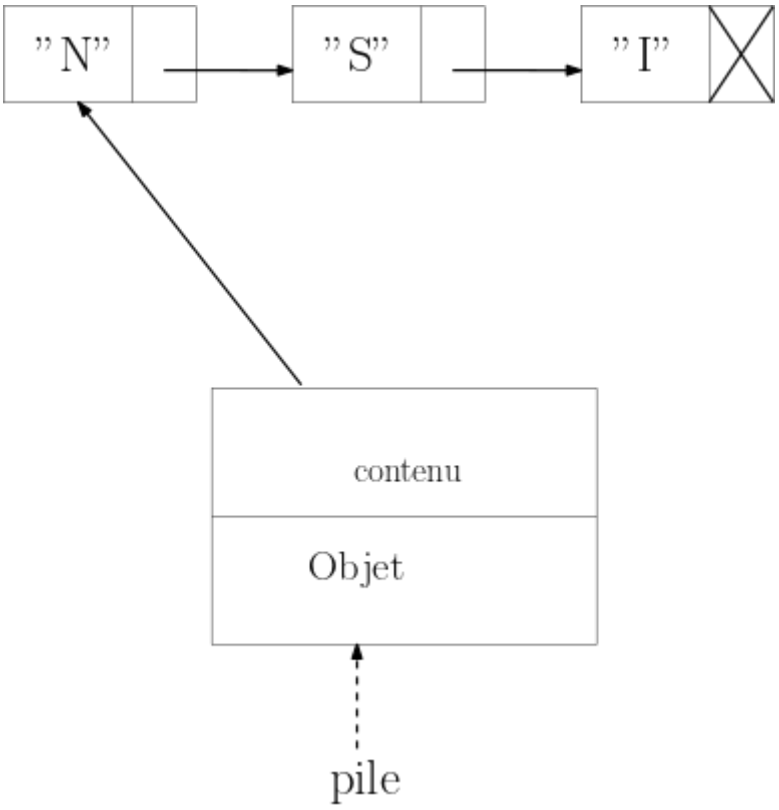
On peut remarquer que si on implémente le *type abstrait Pile* par une liste chaînée dont la *tête* correspondrait au *sommet* de la pile alors l'interface du type Pile est incluse dans celle du type Liste (seule l'opération `queue` ne figure pas dans l'interface d'une Pile) :

Opération du type abstrait Liste	Opération du type abstrait Pile
creer_liste	creer_pile
liste_vide	pile_vide
inserer	empiler

Opération du type abstrait Liste	Opération du type abstrait Pile
tete	depiler

> 💡 Il y a une petite difference entre `tete` qui renvoie la valeur de l'élément en tête de liste et `empiler` qui retire l'élément au sommet de la pile en plus de renvoyer sa valeur.
> 💡 On peut choisir une implémentation de **pile immuable** avec l'implémentation d'une liste chaînée par des *tuples* ou de **pile mutable** avec l'implémentation par une classe que nous présentons ici.

Il est donc naturel d'implémenter le type abstrait Pile par une liste chaînée. Avec le paradigme objet (POO), on écrit une classe `Pile` avec un seul attribut `contenu` qui contient un lien vers la première cellule de la liste chaînée contenant les éléments.



```python
class Cellule:

    def __init__(self, elt, suivant=None):
        self.element = elt
        self.suivant = suivant

class Pile_chaine:

    def __init__(self):
        self.contenu = None

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

    def depiler(self):
        assert not self.pile_vide(), "Pile Vide"
        sommet = self.contenu.element
        self.contenu = self.contenu.suivant
        return sommet

    def empiler(self, elt):
        if self.pile_vide():
            self.contenu = Cellule(elt)
        else:
            self.contenu = Cellule(elt, self.contenu)
```