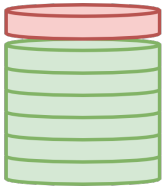
Chap. 6 — Piles et files

6.1 — Introduction

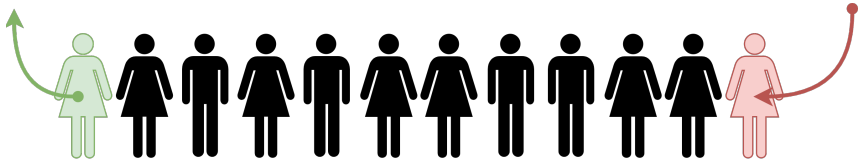
Tout comme les *tableaux*, **Pile** et **File** sont des structures de données qui permettent de (1) **stocker** des ensembles d'objets et (2) **ajouter**/**retirer** des objets un à un.

Dans une **pile** (en anglais *stack*), chaque opération `depile` retire l'élément arrivé le plus récemment. Pour imaginer cette structure, il suffit de penser à une *pile d'assiettes* : on ajoute une assiette sur le sommet et quant on retire une assiette, c'est forcément celle du sommet.



Dernier entré, premier sorti (en anglais **LIFO** pour *last in, first out*)

Dans une **file** (en anglais *queue*), chaque opération `defile` retire l'élément *qui avait été ajouté* le premier. Pour imaginer cette structure, on pense à une file d'attente dans laquelle (1) les personnes arrivent à tour de rôle, (2) patientent et (3) sont servies dans leur ordre d'arrivé.



Premier arrivé, premier sorti (en anglais **FIFO** pour *first in, first out*)

6.2 — Interface commune aux piles et aux files

Classiquement, chacune de ces deux structures a une **interface** proposant au minimum les quatre opérations suivantes :

pile	file	opérations
`Pile()`	`File()`	créer une structure initialement vide
`est_vide()`	`est_vide()`	tester si une structure est vide
`empile()`	`enfile()`	ajouter un élément à une structure
`depile()`	`defile()`	retirer et obtenir un élément d'une structure

Comme pour les tableaux et les listes chaînées, on préconisé pour les piles et les files une **structures homogènes**. C'est-à-dire que tous les éléments stockés aient le même type.

Dans ce cours, nos structures de pile et de file seront considérées **mutables** : chaque opération d'ajout ou de retrait d'un élément **modifie la pile ou la file** à laquelle elle s'applique.

Mais il aurait été tout à fait possible d'en décider autrement.

6.3 — Interface et utilisation d'une pile

Détaillons l'interface des piles.

interface	explications et commentaires
`Pile[T]`	le type des piles contenant des éléments de type `T`. Par exemple `T` peut être `int` pour les nombres entiers ou encore `str` pour les chaînes de caractères

interface	explications et commentaires
`est_vide(p: Pile[T]) -> bool`	prend en paramètre une pile `p` et renvoie un booléen indiquant si la pile est vide ou pas
`creer_pile() -> Pile[T]`	aucun paramètre et renvoie une pile vide capable de contenir n'importe quel type d'élément `T`
`empiler(p: Pile[T], e: T) -> None`	ajout de l'élément `e` au sommet de la pile `p` (*push* en anglais), fonction qui prend en paramètre une pile `p` et l'élément `e` de type `T` homogène avec celui des éléments de la pile
`depiler(p: Pile[T]) -> T`	retrait de l'élément au sommet de la pile `p` (*pop* en anglais) qui prend en paramètre la pile `p` et renvoie l'élément qui en a été retiré. On suppose que la pile est non vide et une exception est levée le cas échéant

Exemple d'utilisation des piles : Considérons un navigateur web dans lequel on s'intéresse à deux opérations : **aller** à une nouvelle page et **revenir** à la page précédente. On veut que le bouton de retour en arrière permette de remonter une à une les pages précédentes, et ce jusqu'au début de la navigation.

En plus de l'adresse courante, qui peut être stockée dans une variable à part, il nous faut donc conserver l'ensemble des pages précédentes auxquelles il est possible de revenir. *Puisque le retour en arrière se fait vers la dernière page qui a été quittée*, la discipline ***dernier entré, premier sorti*** des piles est exactement ce dont nous avons besoin pour cet ensemble.

6.4 — Interface et utilisation d'une file

Comme pour les piles, on note `File[T]` le type des files contenant des éléments de type `T`.

interface	explications et commentaires
`File[T]`	le type des files contenant des éléments de type `T`
`creer_file() -> File[T]`	créer une file vide
`est_vide(f: File[T]) -> bool`	renvoie `True` si `f` est vide et `False` sinon
`enfiler(f: File[T], e) -> None`	ajoute l'élément `e` à la fin de la file `f`
`defiler(f: File[T]) -> T`	retirer et renvoyer l'élément situé au début de la file `f`

Exemple d'utilisation des files : Considérons le jeu de cartes de la bataille. Chaque joueur possède un paquet de cartes et pose à chaque manche la carte prise **sur le dessus du paquet**. Le vainqueur de la manche récupère alors les cartes posées, pour les placer **au-dessous de son paquet**.

En plus des cartes posées au centre de la table nous avons besoin de conserver en mémoire le paquet de cartes de chaque joueur. *Puisque les cartes sont remises dans un paquet à une extrémité et prélevées à l'autre*, la discipline ***premier entré, premier sorti*** des files est exactement ce dont nous avons besoin pour chacun de ces ensembles.

6.5 — Réalisation d'une pile avec une liste chaînée

La structure de **liste chaînée** donne une manière élémentaire de réaliser une pile. Empiler un nouvel élément revient à ajouter un nouveau maillon en tête de liste, tandis que dépiler un élément revient à supprimer le maillon de tête.

On peut ainsi construire une classe `Pile` définie par un unique attribut contenu associé à l'ensemble des éléments de la pile, stockés sous la forme d'une liste chaînée.

Implémenter le constructeur de la classe `Pile` qui construit une pile vide en définissant son attribut `contenu` comme la liste vide `None`.

Exemple et test :

```
>>> p = Pile()
>>> print(p.contenu)
None
```

Étendre la classe `Pile` en implémentant la méthode `est_vide()`.

Exemples et tests :

```
>>> p = Pile()
>>> print(p.est_vide())
True
>>> p.contenu = Maillon(1, None)
>>> print(p.est_vide())
False
```

6.5 — Réalisation d'une file avec une liste (mutable)

6.6 — Réalisation d'une file avec deux piles