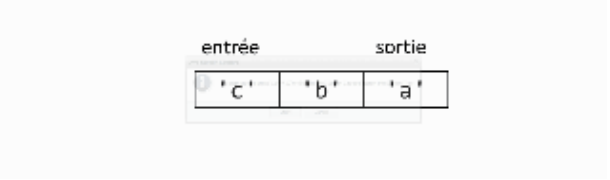
Structures linéaires : Files

Cours

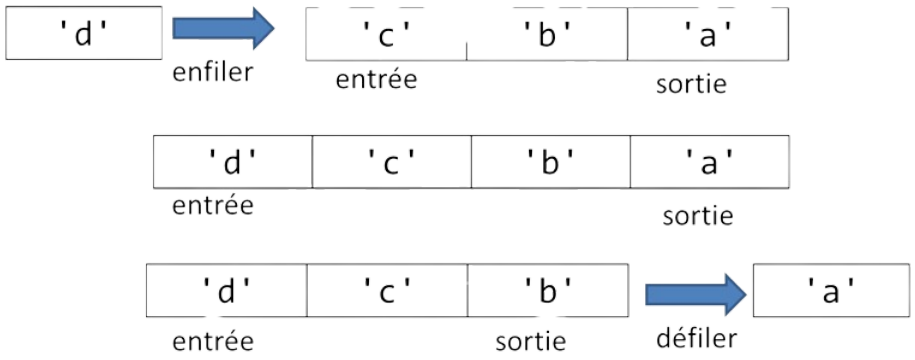
En informatique, une **file** (en anglais ***queue***) est un type abstrait de données sur le principe « premier arrivé, premier sorti » (ou **FIFO** pour ***First In, First Out***).



Le premier élément ajouté à la file, ou **enfilé**, est le premier qui sera sorti, ou **défilé**.

Le premier élément enfilé est en **sortie** de la file, le dernier est en **entrée**.

Le fonctionnement est celui d'une **file d'attente** : les premières personnes qui arrivent dans la file sont ensuite les premières qui en sortiront.



Ici aussi, les files trouvent de nombreuses applications en informatique, en général pour mémoriser temporairement des transactions qui doivent attendre pour être traitées, par exemple par les logiciels d'imprimantes qui traitent les demandes dans l'ordre dans lequel elles arrivent en les plaçant dans une file d'attente ; ou par l'ordonnanceur d'un système d'exploitation qui accorde du temps machine à chaque processus dans l'ordre où il arrive, sans en privilégier aucun.

Interface

Les principales primitives constituant l'interface d'une file sont :

- `creer() → file` : construire d'une file vide.
- `est_vide() → bool` : vérifier si une file est vide ou non.
- `enfiler(element)` : ajouter un élément dans la file (*enqueue* en anglais).
- `défiler() → element` : enlèver un élément de la file et le renvoyer (*dequeue* en anglais).
- `taille() → int` : renvoyer le nombre d'éléments dans la file.

Exemples :

Soit une file `F` composée des éléments suivants : 12, 14, 8, 7, 19 et 22 (le premier élément rentré dans la file est 22 ; le dernier élément rentré dans la file est 12). Regardons chaque exemple ci-dessous en repartant de la file d'origine :

- `enfiler(F,42)` la file `F` est maintenant composée des éléments suivants : 42, 12, 14, 8, 7, 19 et 22 (le premier élément rentré dans la file est 22 ; le dernier élément rentré dans la file est 42)
- `défiler(F)` la file `F` est maintenant composée des éléments suivants : 12, 14, 8, 7, et 19 (le premier élément rentré dans la file est 19 ; le dernier élément rentré dans la file est 12)
- après `défiler(F)` 6 fois de suite, `est_vide(F)` renvoie vrai.
- après avoir appliqué `défiler(F)` une fois, `taille(F)` renvoie 5.

Implémentation

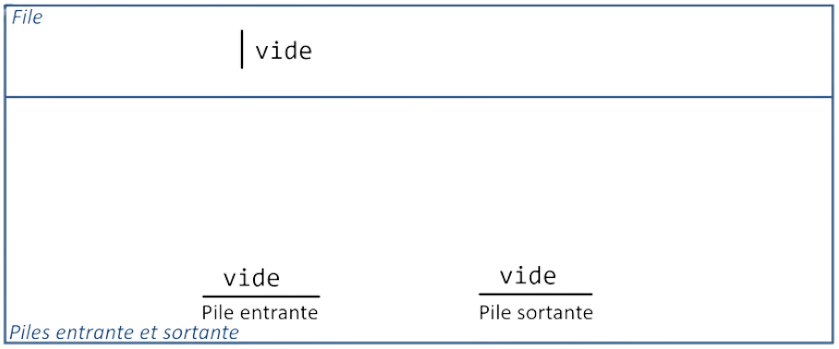
Avec le type `list` de Python

La liste est un type abstrait, son implémentation peut se faire sous différentes formes, par exemple en reprenant l'implémentation d'une Pile avec une variable de type `list` il suffit de modifier `pop()` en `pop(0)` pour écrire la méthode `defiler()`.

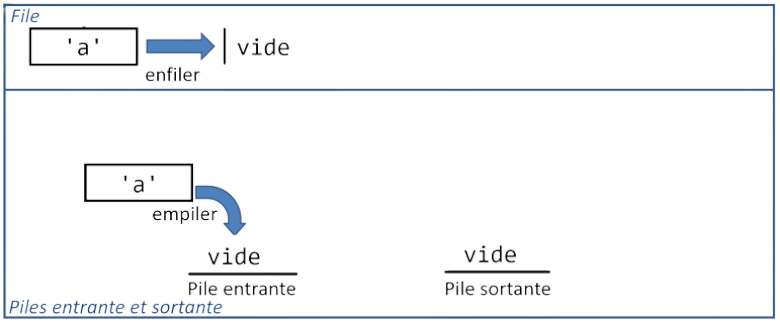
```
    def defiler(self):
        if self.est_vide(): raise IndexError("la file est vide")
        return self.pile.pop(0)
```

Avec deux piles

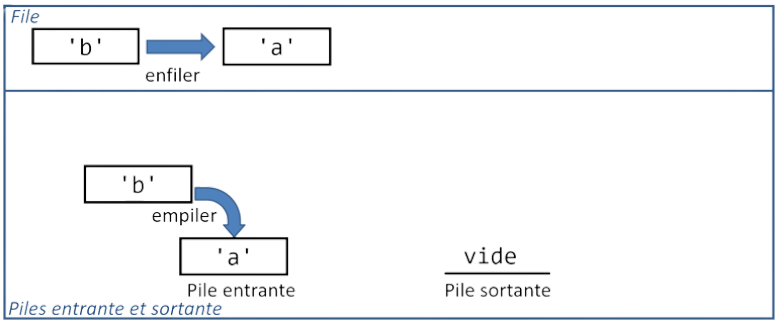
Voici une autre approche d'implémentation d'une file, en utilisant deux piles : une pile « entrante » et une pile « sortante ».



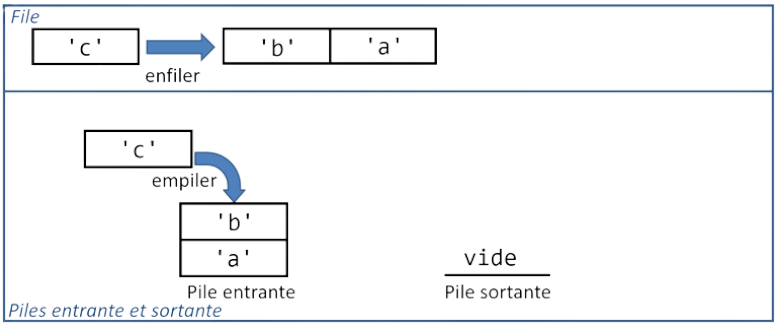
1. **Enfiler** : Chaque fois qu'un nouvel élément est enfilé dans la file, il est empilé dans la pile entrante. Pour commencer enfilons trois nouveaux éléments `'a'`, `'b'` et `'c'`.
    - `'a'` est enfilé dans la file et empilé dans la pile entrante :



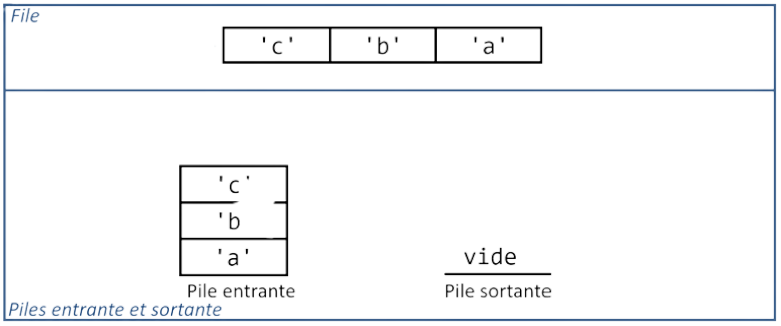
- `'b'` est enfilé dans la file et empilé dans la pile entrante :



- `'c'` est enfilé dans la file et empilé dans la pile entrante :

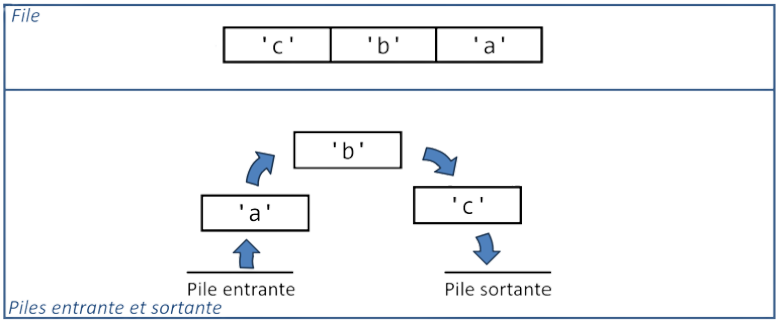


- Les trois éléments `'a'`, `'b'` et `'c'` sont présents dans la file et dans la pile entrante. Noter comment l'élément `'a'` qui est en tête de file se trouve tout en bas de la pile.

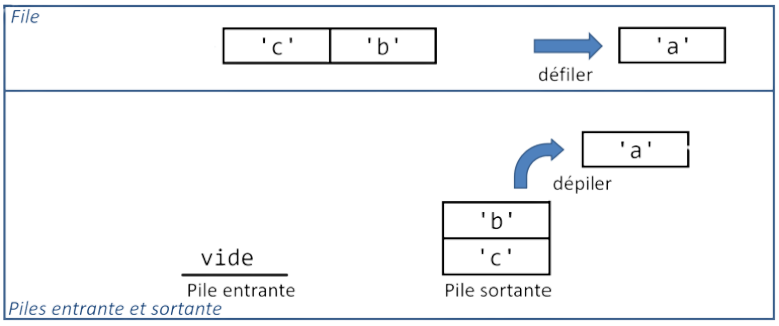


2. **Défiler** *quand la pile sortante est vide* : Il n'y a pas aucun élément prêt à être dépilé de la pile sortante, il faut d'abord la « remplir » avec **tous** les éléments présents dans la pile entrante, puis dépiler le sommet de la pile sortante. Défilons l'élément `'a'` qui se trouve en tête de la file :

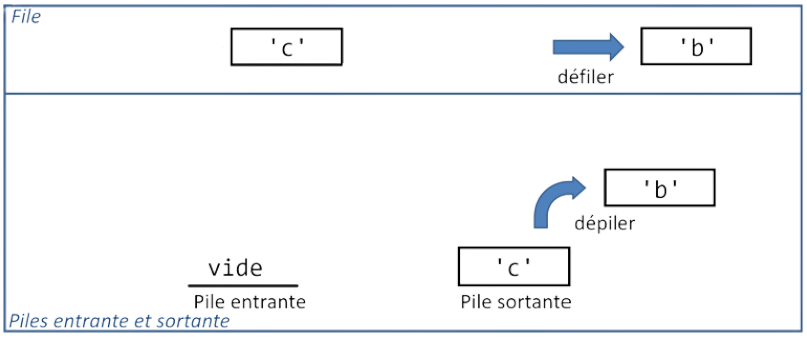
   - **Tous** les éléments présents dans la pile entrante, `'a'`, `'b'` et `'c'`, sont dépilés et empilés à la suite dans la pile sortante.



- L'élément `'a'` en tête de file est défilé. Il se trouve au sommet de la pile sortante, il est dépilé.



3. **Défiler** *quand la pile sortante n'est pas vide* : Il suffit de dépiler le sommet de la pile sortante. Défilons l'élément `'b'` qui se trouve maintenant en tête de la file :



Transcrivons cela en Python en créant un nouveau fichier « file.py » dans le même répertoire que « pile.py » et importons ce module `pile`.

```
import pile
```

Il est maintenant possible d'instancier des objets de la classe `Cellule` ou `Pile` et d'utiliser les méthodes associées :

```
c = pile.Cellule('a', None)
p = pile.Pile()
```

Créons nos deux piles :

```
import pile

class File:
    ''' File sous forme de deux Piles
    '''
    def __init__(self):
```

```
        self.entrante = pile.Pile()
        self.sortante = pile.Pile()
```

La méthode `est_vide` est immédiate, il suffit que les deux piles soient vides :

```
    def est_vide(self):
        return self.entrante.est_vide() and self.sortante.est_vide()


>>> f = File()
>>> f.est_vide()
True
```

Pour enfiler un élément dans la file, il suffit de l'empiler dans la pile entrante :

```
    def enfiler(self, v):
        self.entrante.empiler(v)


>>> f = File()
>>> f.est_vide()
True
>>> f.enfiler('a')
>>> f.est_vide()
False
```

Mais pour défiler, il faut gérer les deux cas vus précédemment :

```
    def defiler(self):
        if self.est_vide():
            raise IndexError ("La file est vide")
        if self.sortante.est_vide():
            #  TOUTE la pile entrante est dépilée dans la sortante
            while not self.entrante.est_vide():
                self.sortante.empiler(self.entrante.depiler())
        return self.sortante.depiler()

>>> f = File()
>>> f.enfiler('a')
>>> f.enfiler('b')
>>> f.enfiler('c')
>>> f.defiler()
'a'
```