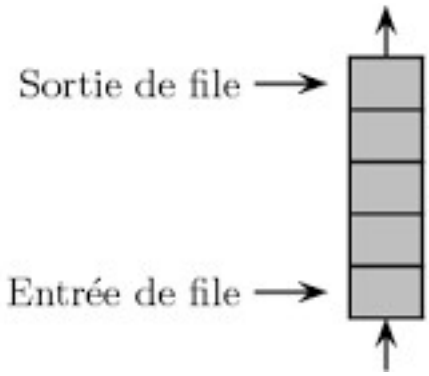
Les files

Comme les piles, les files ont des points communs avec les listes.

Différences majeures:

Dans une file on ajoute des éléments à une extrémité de la file et on supprime des éléments à l'autre extrémité.

On prend souvent l'analogie de la file d'attente devant un magasin pour décrire une file de données.



Les files (queue) sont basées sur le principe FIFO (First In First Out : le premier qui est rentré sera le premier à sortir).

Une file est une collection d'objets sur lesquels il est possible d'utiliser diverses opérations.

(Ici aussi, on retrouve souvent ce principe FIFO en informatique).

Voici les opérations que l'on peut réaliser sur une file:

•Savoir si une file est vide

- (file_vide?, is_empty)

•Ajouter un nouvel élément à la file

- (ajout, enfiler, enqueue)

•Récupérer l'élément situé en bout de file tout en le supprimant

- (retire, defiler, dequeue)

•Accéder à l'élément situé en bout de file sans le supprimer de la file

- (premier)

•Connaitre le nombre d'éléments présents dans la file

- (taille)

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

Pour chaque exemple ci-dessous on repart de la file d'origine:

•ajout(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

•retire(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

•premier(F) renvoie 22, la file F n'est pas modifiée

•si on applique retire(F) 6 fois de suite, file_vide?(F) renvoie vrai

•Après avoir appliqué retire(F) une fois, taille(F) renvoie 5.

Une 1° implémentation de la structure:

```
def file():
        """
        Retourne une file vide
        """
        return []

def vide(f):
        """
        Vérifie si la file est vide
        Renvoie True si elle est vide, False sinon
        """
        return f == []

def enfiler(f, x):
        """
        Ajoute l'élément x à la file
        """
        return f.append(x)

def defiler(f):
        """
        Enlève et renvoie le premier élément de la file
        """
        assert not vide(f), 'file vide'
        return f.pop(0)
```

– Testez les instructions suivantes:

```
f = file()
for i in range(5):
        enfiler(f, 2*i)
print(f)
a = defiler(f)
print(a)
print(f)
print(vide(f))
```

– Réaliser la fonction taille(f), qui renvoie la taille de la file

– Réalisez la fonction sommet(f) qui renvoie le premier élément à sortir, sans le supprimer

Une 2° implémentation de la structure:

```
class File:
        """
        classe File
        Créé une instance de la classe File avec une liste
        """

        def __init__(self):
                """
                Initialisation de la classe File
                """
                ..........................................

        def vide(self):
                """
                Vérifie si la file est vide
                """
                ..........................................

        def defiler(self):
                """
                Enlèce le premier élément de la file
                """
                ..........................................
                ..........................................

        def enfiler(self, x):
                """
                Ajoute l'élément x à la fin de la file
                """
                ..........................................
```

- En vous inspirant de la class Pile, complétez la class File
- Ecrire les instructions pour:
  - créer une file
  - remplir la file avec 0, 2, 4, 6, 8
  - afficher la file
  - "défiler" la file en affichant l'élément récupéré
- Ecrire la méthode taille, qui retourne la taille de la file
- Ecrire la méthode sommet, qui retourne le premier élément à sortir de la file, sans le supprimer

Croisement routier:

Pour simuler un croisement routier, à sens unique, on utilise 3 files (f1, f2 et f3) représentant respectivement les voitures arrivant des routes r1 et r2, et allant sur la route r3.

La route r2 à un stop, les voitures venant de r2 ne peuvent avancer que si la route r1 est vide.

On souhaite écrire un algorithme qui simule le départ des voitures sur la route r3.

- Dans la file f1, on va représenter la présence de voiture par le nombre 1, et l'absence de voiture par le nombre 0
- Dans la file f2, on va représenter la présence de voiture par le nombre 2, et l'absence de voiture par le nombre 0
- On n'utilisera que les méthodes 'enfiler', 'defiler', 'sommet' et 'vide'
- On testera l'algorithme sur f1 : tête ->[0, 1, 1, 0, 1] <- queue
- On testera l'algorithme sur f2 : tête ->[0, 2, 2, 2, 0, 2, 0] <- queue
- Le résultat attendu : f3: tête ->[0, 1, 1, 2, 1, 2, 2, 0, 2, 0] <- queue

Sources:

- https://pixees.fr/informatiquelycee/n_site/nsi_term_structDo_liste.html
- Prépabac Tle générale spécialité NSI, édition Hatier
- https://isn-icn-ljm.pagesperso-orange.fr/NSI-TLE/co/section_chapitre2.html