Chapitre 6 - Les files

Objectifs :

- ▷ Connaître et comprendre le type abstrait file.
- ▷ Modéliser puis simuler le comportement d'une file.
- ▷ Distinguer les modes FIFO et LIFO des piles et des files.

1 Introduction

Dans ce chapitre nous allons décrire des **structures de données linéaires** appelées **files**. On rappelle qu'une structure de données est une organisation logique des données permettant de simplifier ou d'accélérer leur traitement.

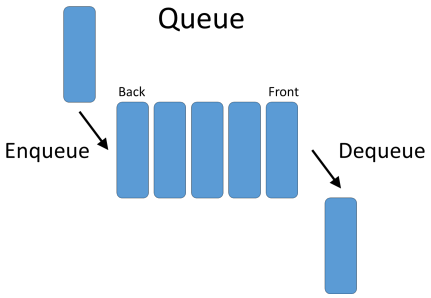
Il faut bien comprendre que lorsqu'on parle de structure de données, on parle d'une représentation **abstraite** qui n'est pas en lien direct avec son implémentation qui peut être réalisée de diverses manières suivant le langage de programmation, voire au sein d'un même langage de programmation.

2 Définition

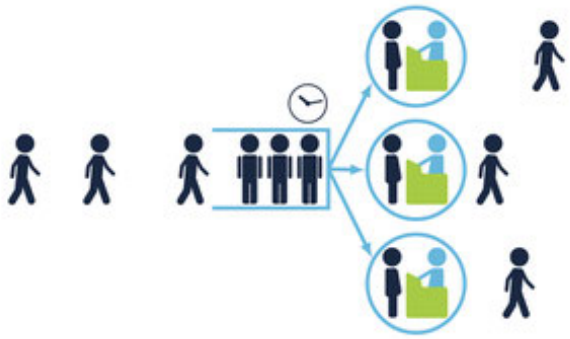
A retenir !

En informatique, une **file** (appelée *queue* en anglais) est une structure de données basée sur le principe « **Premier entré, premier sorti** », en anglais **FIFO** (*First In, First Out*).

Cela signifie que les premiers éléments ajoutés à la file seront les premiers à être récupérés.

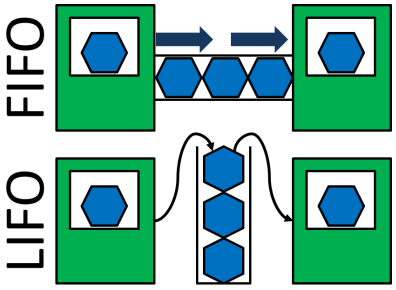


Le fonctionnement ressemble à la **queue** d'une caisse de supermarché par exemple. Ainsi, les premières personnes à arriver sont les premières personnes à sortir de la file.



Warning !

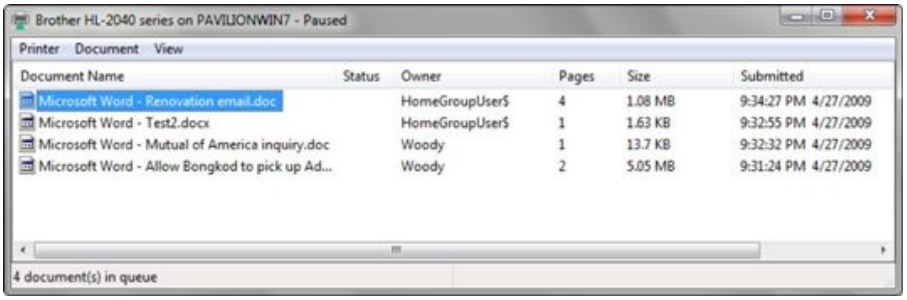
Attention à ne pas confondre la **pile** qui utilise le principe du **LIFO** *Last In, First Out* avec la **file** qui utilise le principe du **FIFO** *First In, First Out*.



3 Exemples d'usage

En général, on utilise des files pour mémoriser temporairement des transactions qui doivent attendre pour être traitées, comme par exemple :

- les **serveurs d'impression**, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).

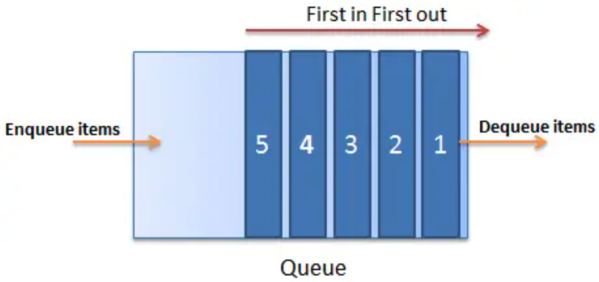


- les **moteurs multitâches**, dans un système d'exploitation, qui doivent accorder du temps-machine à chaque tâche, sans en privilégier aucune.

4 Implémentation

Pour implémenter une structure de file, on a besoin d'implémenter seulement un nombre réduit d'opérations de bases qui sont :

- **emfiler** : ajoute un élément dans la file. Terme anglais correspondant : *enqueue*
- **défiler** : enlève un élément de la pile et le renvoie. En anglais : *dequeue*
- **vide** : renvoie vrai si la pile est vide, faux sinon
- **remplissage** : renvoie le nombre d'éléments dans la pile



La structure de file est un **concept abstrait**. Comment faire pour réaliser une pile dans la pratique ?

L'idée principale étant que les fonctions de bases pourront être utilisées indépendamment de l'implémentation choisie.

4.1 Implémentation n°1

Dans cette implémentation, nous utiliserons une simple liste Python pour représenter la file. On utilise donc encore `append(x)` et `pop(0)` pour réaliser les méthodes `enfiler` et `defiler`.

Voici les fonctions de base :

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

def vide(f):
    '''Renvoie True si la file est vide et False sinon'''
    return f == []

def enfiler(p,x):
    '''Ajoute l'element x a la file f'''
    return f.append(x)

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

A faire 1 : Tester 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))
```

Expliquer les affichages obtenus :

A faire 2 : Réaliser les **fonctions** `taille(f)` et `sommet(f)` qui retournent respectivement la taille de la liste et le premier élément de la liste (mais sans le supprimer !).

4.2 Implémentation n°2

Nous allons utiliser la POO afin de créer une classe `File` pour implémenter cette structure. En vous inspirant de ce que l'on a vu pour la classe `Pile` dans le chapitre précédent, compléter puis tester l'implémentation ci-dessous :

```
class File:
    '''classe File, creation d'une instance File avec une liste'''

    def __init__(self):
        '''Initialisation d'une file vide'''
        ...

    def vide(self):
        '''Teste si la pile est vide'''
        return ...

    def defiler(self):
        '''Defile'''
        ...
        ...

    def enfiler(self,x):
        '''Enfile l'element x dans la file'''
        ...
        ...
```

Écrire les instructions permettant de :

- ▷ créer une file
- ▷ la remplir avec les entiers 0,2,4,6,8
- ▷ la faire afficher
- ▷ "défiler" la file en faisant afficher l'élément récupéré.

A faire 3 : Réaliser les **méthodes** `taille(self)` et `sommet(self)` qui retournent respectivement la taille de la liste et le premier élément de la liste (mais sans le supprimer !).

5 Exercices

Exercice 1 : Soit une file `F` initialement vide.

On considère l'enchaînement d'opérations ci-dessous.

```
enfiler(F,6)
enfiler(F,3)
a = defiler(F)
enfiler(F,9)
b = taille(F)
enfiler(F,17)
c = defiler(F)
enfiler(F,2)
d = taille(F)
```

Donner le contenu de la file `F`, la valeur de `a`, la valeur de `b`, la valeur de `c` et la valeur de `d`.

Exercice 2 : Soit le programme Python suivant :

```
pile = []
tab = [5,8,6,1,3,7]
pile.append(5)
pile.append(10)
pile.append(8)
pile.append(15)
for i in tab:
    if i > 5:
        pile.pop()
```

Donner l'état de la pile `pile` après l'exécution de ce programme.

Exercice 3 : On dispose d'une **file**. Ecrire une fonction qui renvoie la file inversée (l'élément de la tête sera situé à la queue et ainsi de suite). On utilisera une pile et seulement les méthodes associées aux piles et aux files.

Exercice 4 : On dispose d'une **pile**. Ecrire une fonction qui renvoie la **pile inversée** (l'élément en haut de la pile sera situé en bas de la pile et ainsi de suite). On utilisera une pile et seulement les méthodes associées aux piles et aux files.

Exercice 5 : On dispose d'une **file** contenant des entiers, écrire une fonction qui renvoie une **file** où on aura séparé les nombres pairs des impairs.

Exercice 6 : On dispose d'une **pile** contenant des entiers, écrire une fonction qui renvoie une **pile** où l'on aura séparé les nombres pairs des impairs.

Exercice 7 : On dispose d'une **pile** contenant des entiers, écrire une fonction ou une méthode qui renvoie une **pile** où l'on aura supprimé le premier élément entré.

Exercice 8 : On dispose d'une **file** contenant des entiers, écrire une fonction ou une méthode qui renvoie une **file** où l'on aura supprimé le dernier élément entré.

Exercice 9 : *Sujet Bac 2022 (Amérique du Nord)*

Les interfaces des structures de données abstraites `Pile` et `File` sont proposées ci-dessous :

Structure de données abstraite : Pile
Opérations :
`creer_pile_vide()` renvoie une pile vide
`est_vide(pile)` renvoie `True` si `pile` est vide, `False` sinon
`empiler(pile,element)` ajoute `element` à la pile
`depiler(pile)` renvoie l'élément au sommet de la pile en le retirant de la pile

Structure de données abstraite : File
Opérations :
`creer_file_vide()` renvoie une file vide
`est_vide(file)` renvoie `True` si `file` est vide, `False` sinon
`enfiler(file,element)` ajoute `element` à la file
`defiler(pile)` renvoie le premier élément de la file en le retirant de la file

On considère la file `F` suivante :

enfilement → `"rouge", "vert", "jaune", "rouge", "jaune"` → défilement

On utilisera uniquement les fonctions ci-dessus.

1. Quel sera le contenu de la pile `P` et de la file `F` après l'exécution du programme Python suivant ?

```
P = creer_pile_vide()
while not(est_vide(F)):
    empiler(P, defiler(F))
```

2. Créer une fonction `taille_file` qui prend en paramètre une file `F` et qui renvoie le nombre d'éléments qu'elle contient. Après appel de cette fonction la file `F` doit avoir retrouvé son état d'origine.

```
def taille_file(F):
    """File -> Int"""
    ...
```

3. Écrire une fonction `former_pile` qui prend en paramètre une file `F` et qui renvoie une pile `P` contenant les mêmes éléments que la file.

   Le premier élément sorti de la file devra se trouver au sommet de la pile ; le deuxième élément sorti de la file devra se trouver juste en-dessous du sommet,..., etc.

4. Écrire une fonction `nb_elements` qui prend en paramètres une file `F` et un élément `elt` et qui renvoie le nombre de fois où elt est présent dans la file `F`. Après appel de cette fonction la file `F` doit avoir retrouvé son état d'origine.

5. Écrire une fonction `verifier_contenu` qui prend en paramètres une file `F` et trois entiers : `nb_rouge`, `nb_vert` et `nb_jaune`. Cette fonction renvoie le booléen `True` si `"rouge"` apparaît au plus `nb_rouge` fois dans la file `F`, `"vert"` apparaît au plus `nb_vert` fois dans la file `F` et `"jaune"` apparaît au plus `nb_jaune fois` dans la file `F`. Elle renvoie `False` sinon. On pourra utiliser les fonctions précédentes.

Exercice 10 : *Le problème de Josèphe*

Flavius Josèphe était un historien juif du premier siècle. Il participa aux révoltes contre les Romains et fut assiégé avec quarante de ses compatriotes dans la forteresse de *Jotapata* en 67 après J.C. Les 41 soldats (dont *Flavius Josèphe*), cernés par des soldats romains, décident de se suicider. Ils se mettent en cercle, et un premier soldat est choisi au hasard pour être exécuté ; puis le troisième à partir de sa gauche est exécuté. Tant qu'il y a des soldats, la sélection continue de la même façon.

Le but est de trouver à quelle place doit se tenir un soldat pour être le dernier. Josèphe, peu enthousiaste à l'idée de mourir, parvint à trouver cette place. Quelle est-elle ?

En utilisant une file trouver la position à choisir parmi les 41 soldats pour être le dernier éliminé.