Python – Files

Une file avec liste Python

Tout comme les piles, il est possible de faire des files avec des listes Python. En effet, la commande `liste.pop(0)` renvoie la valeur du premier élément de la liste et le supprime.

```
>>> liste = []
>>> liste.append(6)
>>> liste.append(9)
>>> liste.append(15)
>>> liste.pop(0)
6
>>> liste.pop(0)
9
>>> liste.pop(0)
15
```

Par contre, si retirer le dernier élément qui se fait en temps constant, retirer le premier élément se fait en temps linéaire, puisque cela revient à faire avancer toutes les valeurs vers la case précédente et à supprimer le dernier élément.
C'est donc une version qui est simple à réaliser mais qui n'est pas optimale si on manipule de grandes files.

Exercice 1 : Écrire les fontions suivantes en utilisant une liste Python pour représenter la file.

Fonction	Description
`cree_file()`	Renvoie une nouvelle file vide.
`enfile(element, file)`	Rajoute `element` à la fin de la **`file`**.
`defile(file)`	Renvoie l'élément se trouvant au début de **`file`**, qui ne doit pas être vide, et l'enlève.
`est_vide(file)`	Renvoie un booléen indiquant si **`file`** est vide ou non.

```
>>> file = cree_file()
>>> enfile(4, file)
>>> enfile(9, file)
>>> defile(file)
4
>>> enfile(2, file)
>>> defile(file)
9
>>> est_vide(file)
False
>>> defile(file)
2
>>> est_vide(file)
True
```

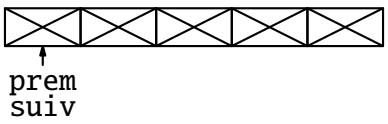
Faire une file bornée avec un tableau

Comme pour les piles, il existe de nombreuses façons de représenter des files. Ces différentes représentations ont des contraites et avantages différents, et également une efficacité variable, en fonction de ce que l'on recherche.
Une des approches les plus simples consiste à utiliser un tableau. L'inconvénient, c'est que le nombre d'objets pouvant se trouver dans la file est alors limité. Cela peut donc s'appliquer dans les cas où la taille de la file est bornée. Pour cela, nous allons créer une nouvelle classe :

```
class FileBornee:
    def __init__(self, n):
        self.tab = [None] * n    # tableau de valeurs
        self.premier = 0         # position du prochain élément à retirer
        self.suivant = 0         # position du prochain élément à ajouter
        self.taille = 0          # nombre d'éléments dans la file
        self.capacite = n        # nombre maximum d'éléments

    def est_vide(self):
        return self.taille == 0

    def est_pleine(self):
        return self.taille == self.capacite

    def enfile(self, v):
        pass

    def defile(self):
        pass
```

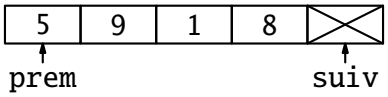
Il est possible de ne pas utiliser l'attribut `taille` qui peut être déduit de `suivant` et `premier`, ainsi que `capacite` qui correspond à `len(self.tab)`, mais cela alourdit un peu le code pour les méthodes suivantes.
La file vide ressemble donc à cela, avec une capacité maximale de 5 éléments :

```
>>> fb = FileBornee(5)
```



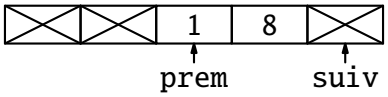
Lors de l'ajout d'éléments, `suivant` va avancer progressivement.

```
>>> fb.enfile(5)
>>> fb.enfile(9)
>>> fb.enfile(1)
>>> fb.enfile(8)
```



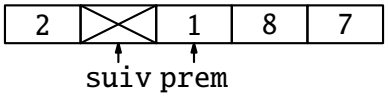
Lors du retrait d'éléments, c'est `premier` qui va avancer. Pour simplifier la compréhension, les éléments retirés sont remplacés par `None` dans le tableau, mais ce n'est pas nécessaire.

```
>>> v = fb.defile()   # v = 5
>>> v = fb.defile()   # v = 9
```



Lorsqu'un des deux indices arrive au bout du tableau, il faut repartir au début. Ainsi `suivant` peut se trouver avant `premier`.

```
>>> fb.enfile(7)
>>> fb.enfile(2)
```



Exercice 2 (sur papier) : Représenter le contenu du tableau et la position de `premier` et de `suivant` après l'exécution des instructions suivantes. Vous supposerez que les valeurs retirées sont remplacés par **`None`**.

```
>>> fb = FileBornee(5)
>>> fb.enfile(3)
>>> fb.enfile(5)
>>> v = fb.defile()
>>> fb.enfile(1)
```

```
>>> fb.enfile(4)
>>> v = fb.defile()
>>> v = fb.defile()
>>> fb.enfile(7)
>>> v = fb.defile()
```

Exercice 3 : Écrire la méthode `enfile(self, v)` qui rajoute `v` à la file. Si la file est pleine, il faut lever l'exception `IndexError('file pleine')`.

Exercice 4 : Écrire la méthode `defile(self)` qui renvoie la valeur en première position dans la file. Si la file est vide, il faut lever l'exception `IndexError('file vide')`. Vous pouvez remplacer la valeur dans la file par **`None`**, mais ce n'est pas obligatoire.

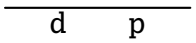
Exercice 5 : Rajouter la méthode `__str__(self)` qui renvoie un texte correspondant à la file. Si la file est vide, le message est `'file vide'` et sinon `'premier <- ... <- dernier'`.

```
>>> fb = FileBornee(5)
>>> str(fb)
'file vide'
>>> fb.enfile(3)
>>> fb.enfile(5)
>>> str(fb)
'3 <- 5'
```

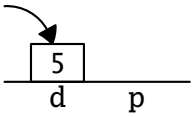
Une file avec deux piles

Une autre méthode classique pour faire des files, c'est d'utiliser 2 piles, qui fonctionnent comme la pioche et la défausse dans un jeu de carte. Chaque élément ajouté se place au sommet de la défausse et pour en retirer un, on le prend au sommet de la pioche. Si la pioche est vide, on retourne la défausse et on la place sur la pioche. Voici un exemple d'utilisation :

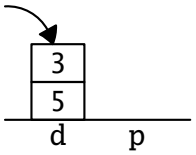
`f = File2()`



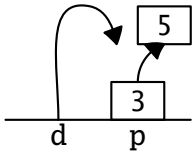
`f.enfile(5)`



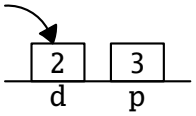
`f.enfile(3)`



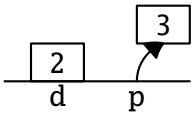
`v = f.defile()`



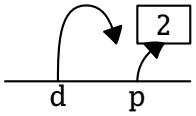
`f.enfile(2)`



`v = f.defile()`



`v = f.defile()`



Exercice 6 (sur papier) : Représenter l'état des deux piles après l'exécution des commandes suivantes :

```
>>> f = File2()
>>> f.enfile(0)
>>> f.enfile(2)
>>> f.enfile(5)
```

```
>>> v = f.defile()
>>> f.enfile(1)
>>> v = f.defile()
>>> f.enfile(7)
```

```
>>> v = f.defile()
>>> v = f.defile()
>>> f.enfile(4)
>>> v = f.defile()
```

Exercice 7 : Créer la classe `File2` et son constructeur `__init__(self)`. Les objets de cette classe n'ont que deux attributs : `defausse` et `pioche` qui sont deux piles vides. Vous pouvez reprendre la classe `Pile` déjà définie dans la feuille précédente.

Exercice 8 : Créer la méthode `est_vide(self)` qui renvoie un booléen indiquant si la file est vide. Pour qu'elle soit vite, il faut que les deux piles soient vides.

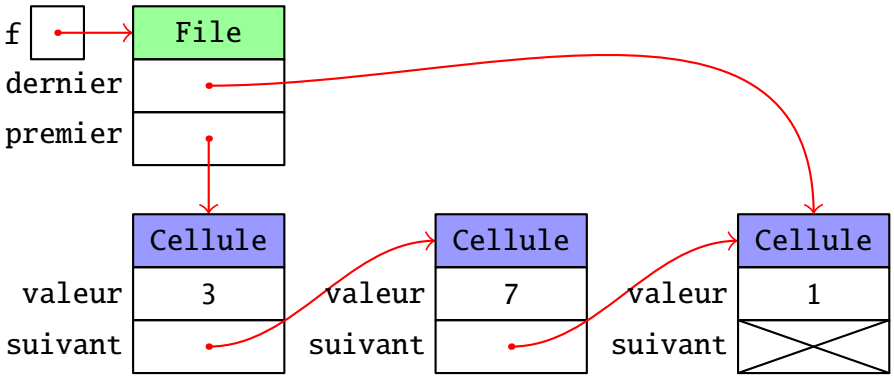
Exercice 9 : Créer la méthode `enfile(self, v)` qui ajoute `v` à `defausse`.

Exercice 10 : Créer la méthode `defile(self)` qui renvoie la valeur au sommet de `pioche`, tout en l'enlevant. Si `pioche` est vide, il faut retourner `defausse` et la mettre sur `pioche`, avant de retirer la valeur au sommet. Si les deux piles sont vides, il faut lever l'exception `IndexError` et le message `'file vide'`

Exercice 11 (optionnel) : Créer la méthode `__str__(self)` qui renvoie un texte correspondant à la file. Si la file est vide, le message est `'file vide'` et sinon `'premier <- ... <- dernier'`. Il faudra peut-être rajouter des méthodes `affichage(self))` et `affichage_inverse(self))` dans la classe `Pile`.

Une file avec une liste chaînée

La version avec un tableau pose le problème de la limite du nombre de valeurs et celle avec deux piles rend linéaire la complexité du retrait, alors que cela devrait être constant. Nous allons donc utiliser une troisième approche qui permet de lever ces deux limitations. Nous allons reprendre le principe de la liste chaînée mutable, sauf que nous ajouterons un attribut pointant sur le dernier élément ajouté à la file.



Lorsqu'un élément est rajouté à la file, il est mis après le dernier maillon. Pour retirer un élément, il faut enlever celui en première position. Dans le cas d'une liste vide, les valeurs de `premier` et `dernier` sont **`None`**.

Exercice 12 (sur papier) : Représenter l'état d'une file qui ne contient qu'un seul élément.

Exercice 13 (sur papier) : Représenter l'état de la file de l'exemple ci-dessus après l'appel `f.enfile(2)`.

Exercice 14 (sur papier) : On reprend la file comme dans l'exemple. Représenter l'état de la file après l'appel `f.defile()`.

Exercice 15 : Créer la classe `File` et son constructeur **`__init__`**`(`**`self`**`)`. Les objets de cette classe n'ont que deux attributs : `premier` et `dernier` qui sont initialisés avec **`None`**. Vous aurez aussi besoin de la classe `Cellule` déjà définie précédemment.

Exercice 16 : Créer la méthode `est_vide(`**`self`**`)` qui renvoie un booléen indiquant si la file est vide.

Exercice 17 : Créer la méthode `enfile(`**`self`**`, v)` qui ajoute une cellule contenant `v` après `dernier`.

Exercice 18 : Créer la méthode `defile(`**`self`**`)` qui renvoie la valeur de la cellule `premier` et l'enlève de la liste. Si la file est vide, il faut lever l'exception `IndexError` et le message `'file vide'`

Exercice 19 (optionnel) : Créer la méthode **`__str__`**`(`**`self`**`)` qui renvoie un texte correspondant à la file. Si la file est vide, le message est `'file vide'` et sinon `'premier <- ... <- dernier'`.