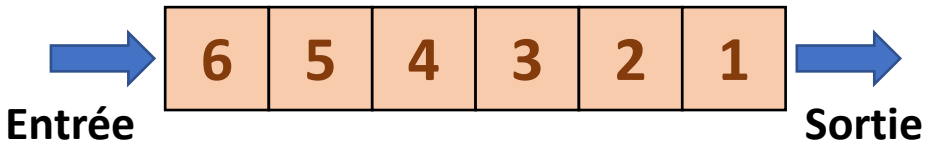
Structures linéaires : files et piles

I./ Les files :

1./ Description d'une file :

Une *file* (*queue* en anglais) est une structure de données abstraite contenant un certain nombre d'éléments ordonnés. Une file ressemble à une liste, mais l'accès à ses éléments est particulier. La file possède en effet deux extrémités, une entrée et une sortie. Lorsqu'on ajoute un nouvel élément à la file (on dit aussi qu'on *enfile* l'élément (*enqueue* en anglais)), il entre dans la file par l'extrémité « entrée ». Lorsqu'on retire un élément de la file (on dit aussi qu'on *défile* l'élément (*dequeue* en anglais)), il sort de la file par l'extrémité « sortie ». Une file est une structure dynamique : sa taille peut varier lors de son utilisation.

Exemple : soit une file. On enfile, dans cet ordre, les nombres de 1 à 6. Si on décide de « défiler » les éléments, on les obtiendra dans le même ordre : 1, 2, 3, 4, 5 puis 6. On parle aussi de *file FIFO* : First In First Out, ou premier arrivé, premier sorti (comme une file d'attente classique).



Pour pouvoir utiliser une file F, il faut un certain nombre de fonctions :

- CREER_FILE() : création d'une file vide
- EST_VIDE(F) : indique si la file F est vide ou non
- LONGUEUR(F) : donne le nombre d'éléments contenus dans la file F (0 si la file est vide)
- ENFILER(F, x) : enfile l'élément x dans la file F
- DEFILER(F) : défile un élément de la file F (si celle-ci n'est pas vide)

2./ Une première implémentation des files en Python :

La solution la plus simple pour implémenter les files en Python est *l'utilisation de listes*. En effet les listes sont des structures de données linéaires ayant deux extrémités.

Tapez le code suivant :

```
def creer_file():
    return []

def file_est_vide(F):
    if F == []:
        return True
    else:
        return False

def longueur_file(F):
    return len(F)

def enfile(F, element):
    F.append(element)
    return None

def defile(F):
    if file_est_vide(F):
        return None
    else:
        return F.pop(0)
```

Une fois le code entré et lancé, tapez dans les lignes suivantes dans la console Python :

```
>>> F = creer_file()
>>> file_est_vide(F)
>>> longueur_file(F)
>>> enfile(F, 1)
>>> enfile(F, 2)
>>> enfile(F, 3)
>>> enfile(F, 4)
>>> enfile(F, 5)
>>> enfile(F, 6)
>>> file_est_vide(F)
>>> longueur_file(F)
>>> defile(F)
>>> defile(F)
>>> defile(F)
>>> defile(F)
>>> defile(F)
>>> defile(F)
>>> defile(F)
```

Dans cette approche, la file F que nous avons créée est en fait une simple liste. Ce fait peut poser des problèmes. En effet, la structure de données « file » ne permet normalement que l'accès à un élément en particulier, celui se trouvant à la sortie de la file. Si on veut obtenir un élément quelconque de la file, on est obligé de défiler tous les éléments qui le précèdent. Or Python permet d'accéder à n'importe quel élément d'une liste. Tapez les lignes suivantes dans la console Python :

```
>>> file = creer_file()
>>> enfile(file, 1)
>>> enfile(file, 2)
>>> enfile(file, 3)
>>> enfile(file, 4)
>>> enfile(file, 5)
>>> enfile(file, 6)
>>> print(file)
>>> file[3]
>>> print(file)
>>> file.pop(3)
>>> print(file)
```

On s'aperçoit qu'en utilisant les méthodes de la classe « **`list`** », on peut accéder et modifier une pile d'une façon qui ne correspond pas à l'utilisation normale de cet objet. Pour empêcher cela, il faudrait pouvoir « cacher » la liste représentant la file de façon à ce qu'on ne puisse pas y accéder directement. Nous pouvons le faire grâce à la programmation orientée objet.

3./ Création d'une classe File :

Dans cette approche, nous allons créer une *classe* File. Les objets correspondants possèderont un attribut unique, qui sera la liste contenant les éléments de la file. Pour empêcher tout accès non autorisé aux éléments de la file, cet attribut sera privé (voir le cours sur la POO).

La classe File contiendra aussi toutes les méthodes permettant de créer une file vide, tester si une file est vide, connaître la longueur d'une file, enfiler un élément et défiler un élément. Ces méthodes seront publiques, de façon à ce qu'on puisse les utiliser : elles constitueront l'interface de la classe.

Tapez le code suivant (pensez aux *docstrings* qui permettront l'utilisation de la fonction **`help`**) :

```
class File:
    """
    Classe permettant d'utiliser des files FIFO
    """

    def __init__(self):
        """
        abc = File()  :  Crée une file vide
        """
        self.__body = []
        return None

    def __str__(self):
        """
        print(abc)  :   Affiche le contenu de la file passée en argument
                        Affiche "File vide" si la file ne contient aucun élément
        """
        if self.__body == []:
            return "File vide"
        else:
            return "File : " + str(self.__body)

    def est_vide(self):
        """
        Renvoie un booléen True si la file est vide, False sinon
        """
        if self.__body == []:
            return True
        else:
            return False

    def longueur(self):
        """
        Donne le nombre d'éléments contenus dans la file
        """
        return len(self.__body)

    def enfile(self, element):
        """
        Ajoute un élément dans la file
        """
        self.__body.append(element)
        return None

    def defile(self):
        """
        Retire un élément de la file
        """
        if self.__body == []:
            return None
        else:
            return self.__body.pop(0)
```

Une fois ce code tapé et lancé, tapez les lignes suivantes dans la console Python :

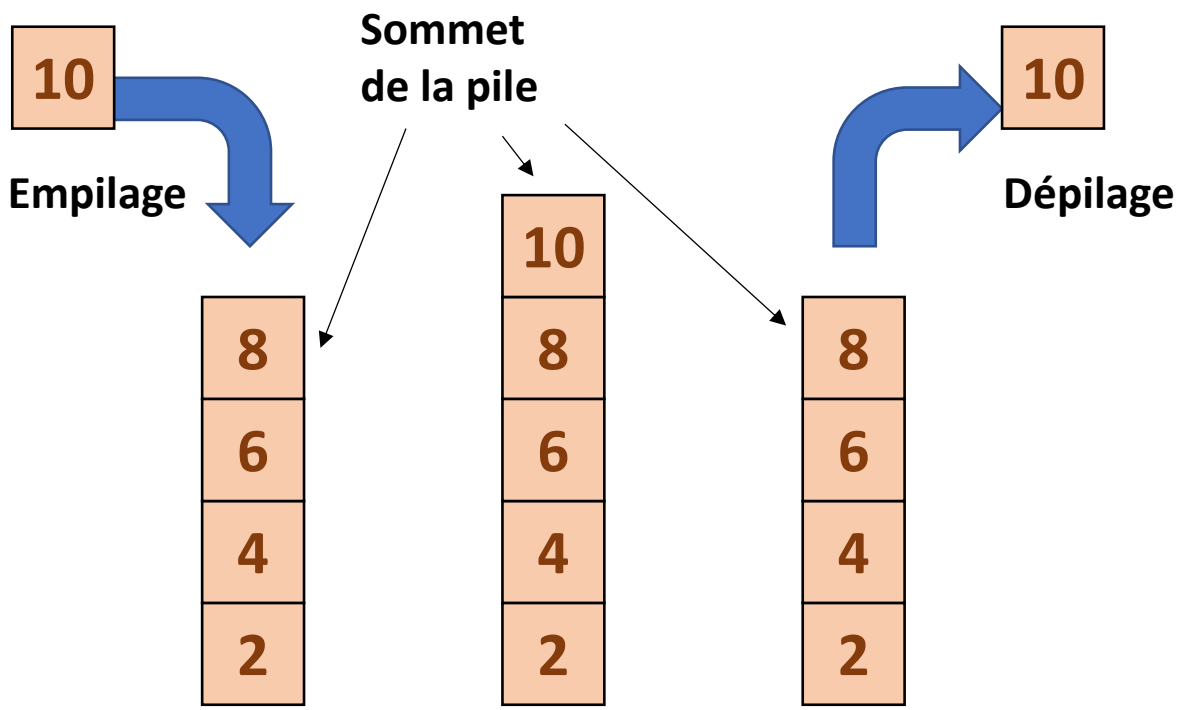
```
>>> F = File()
>>> F.est_vide()
>>> F.enfile(2)
>>> F.enfile(4)
>>> F.enfile(6)
>>> F.enfile(8)
>>> F.enfile(10)
>>> F.longueur()
>>> F.defile()
>>> F.defile()
>>> F.defile()
>>> F.defile()
>>> F.defile()
>>> F.defile()
```

Vérifiez sur d'autres exemples qu'on a bien le même comportement que l'implémentation précédente. Essayez ensuite d'accéder directement aux éléments du milieu d'une file pleine : y parviendrez-vous ?

II./ Les piles :

1./ Description d'une pile :

Une *pile* (*stack* en anglais) est une structure de données abstraite contenant un certain nombre d'éléments ordonnés. Une pile ressemble à une file, mais l'accès à ses éléments est différent. Lorsqu'on ajoute un nouvel élément à la pile (on dit aussi qu'on *empile* l'élément (*push* en anglais)), il se met au *sommet* de la pile. Lorsqu'on retire un élément de la pile (on dit aussi qu'on *dépile* l'élément (*pop* en anglais)), on enlève l'élément qui se trouve au sommet. Un exemple concret pour illustrer ce fonctionnement est une pile d'assiettes. Une file est une structure dynamique : sa taille peut varier lors de son utilisation.



Pour pouvoir utiliser une pile P, il faut un certain nombre de fonctions :

- CREER_PILE() : création d'une pile vide
- EST_VIDE(P) : indique si la pile P est vide ou non
- LONGUEUR(P) : donne le nombre d'éléments contenus dans la pile P (0 si la pile est vide)
- EMPILER(P, x) : empile l'élément x dans la pile P
- DEPILER(P) : dépile un élément de la pile P (si celle-ci n'est pas vide)

Si on empile les éléments dans un certain ordre (exemple : 1, 2, 3, 4, 5 puis 6), on se rend compte qu'on va les dépiler dans le sens inverse (ici 6, 5, 4, 3, 2 puis 1). On parle donc de *pile LIFO* : Last In First Out, ou dernier arrivé, premier sorti.

2./ Implémentation des piles en Python :

En vous inspirant des implémentations précédentes portant sur les files, implémentez un système de piles. Vous pourrez notamment créer une classe Pile.

Une fois votre classe Pile créée, vous pourrez l'enregistrer dans un fichier `.py` pour pouvoir l'utiliser comme module plus tard. Vous pouvez faire de même avec la classe File. De cette façon, vous pourrez vous constituer une boîte à outils de divers modules vous permettant d'implémenter et d'utiliser différents objets dans vos propres programmes.

III./ Utilisation des files et des piles :

L'utilisation d'une des ces structures de données va dépendre de l'application souhaitée :

- Pour les files, on peut citer le système des files d'impression. Chaque document envoyé à une imprimante réseau va être stocké dans la file d'impression, et l'imprimante les traitera dans leur ordre d'arrivée.
- Pour les piles, nous avons vu déjà vu plusieurs exemples d'utilisation : le langage assembleur se sert d'une pile pour stocker des données. Python utilise une pile, appelée pile d'exécution pour pouvoir traiter les fonctions récursives.