	**Structures de données**	**NSI T^{ale}**
	Les Files et les Piles	Cours/TD

Source : https://qkzk.xyz/docs/nsi/cours_terminale/structures_donnees/pile_file/intro/

1. Une classe générique pour les structures en ligne : les sacs

Les piles et les files sont des exemples de structures de données que faute de mieux, nous appellerons des *sacs*. Un sac offre trois opérations élémentaires :

- tester si le sac est vide,
- ajouter un élément dans le sac,
- retirer un élément du sac (tenter de retirer un élément d'un sac vide déclenche une erreur).

Le sac est une structure impérative : un sac se modifie au cours du temps.

En Python, il est logique de représenter un sac (d'éléments E) par un objet (d'une classe Sac) qui possède trois méthodes :

```
class Sac:
    def __init__(self):
        self.contenu = [ ]

    def est_vide(self):
        ...

    def ajouter(self):
        ...

    def retirer(self):
        x = ...
        return x
```

Remarquons le constructeur : `self.contenu = [ ]` : le contenu du sac est référencé dans une liste python. C'est un choix qui sera commode.

Ainsi on ajoute par exemple un élément `e` dans le sac `b` par `b.ajouter(e)`, ce qui modifie l'état interne du sac.

2. Piles et Files : des sacs aux propriétés particulières

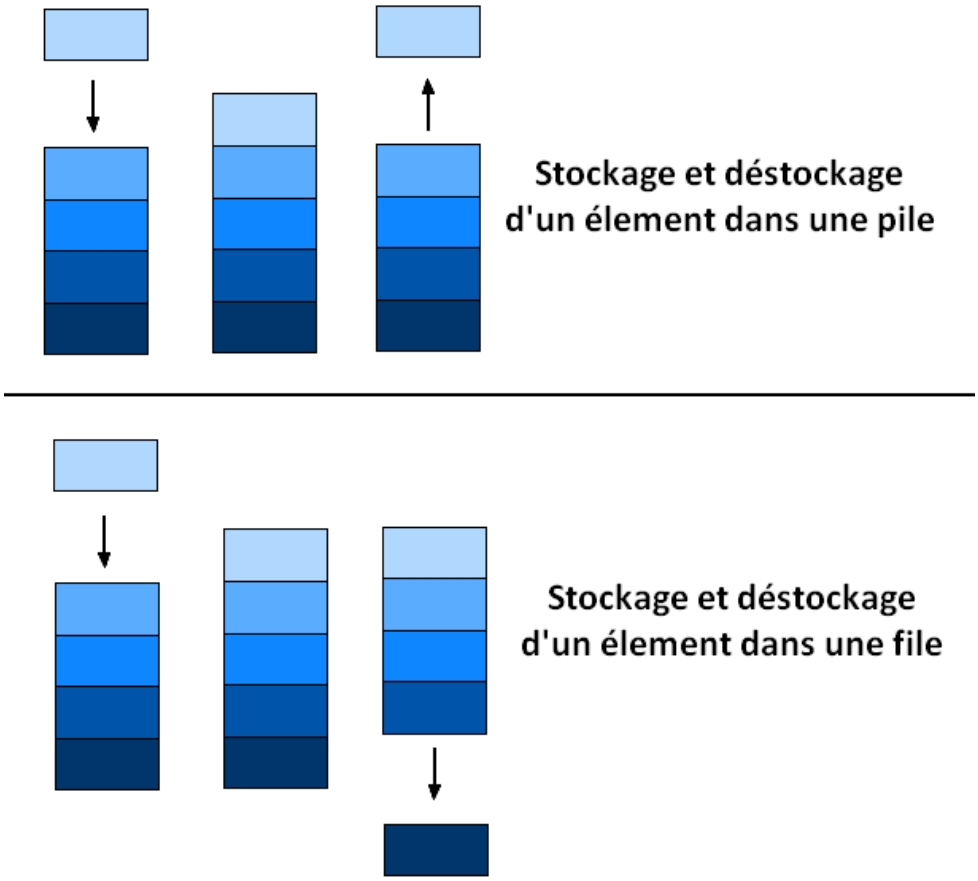
Piles et files se distinguent par la relation entre éléments ajoutés et éléments retirés.

- Dans le cas des **piles**, c'est le **dernier élément** ajouté qui **est retiré**.
- Dans le cas d'une **file** c'est le **premier élément** ajouté qui **est retiré**.

On dit que

- la **pile** est une structure **LIFO** (*last-in first-out*),
- la **file** est une structure **FIFO** (*first-in first-out*).

Si on représente pile et file par des tas de cases, on voit que la pile possède un sommet, où sont ajoutés et d'où sont retirés les éléments, tandis que la file possède une entrée et une sortie.



Tout ceci est parfaitement conforme à l'intuition d'une pile d'assiettes, ou de la queue devant un guichet, et suggère fortement des techniques d'implémentations pour les piles et les files.

Question

Soit la fonction `build` suivante qui ajoute tous les éléments d'une liste dans un sac puis construit une liste en vidant le sac.

```
def build(tableau):
      sachet = Sac()

      for k in tableau:
            sachet.ajouter(k)

      tableau_sortie = []
      while not sachet.est_vide():
            tableau_sortie.append(sachet.retirer() )

      return tableau_sortie
```

Quelle est la liste renvoyée dans le cas où le sac est une pile, puis une file ?

Traitons un exemple pour mieux comprendre :

1) Si le sac est une pile.

```
tableau = [1, 2, 3]

On exécute l'instruction :

for k in tableau:
  sachet.ajouter(k)

sachet contient alors les éléments [1, 2, 3]

On exécute alors l'instruction :

while not sachet.est_vide():
  tableau_sortie.append(sachet.retirer())

Comme la liste est une pile, on retire successivement 3, 2, 1.

Qu'on ajoute à la fin de tableau_sortie :

Donc tableau sortie contient [3, 2, 1]
```

2) Si le sac est une file.

```
tableau = [1, 2, 3]

On exécute l'instruction :

for k in tableau:
  sachet.ajouter(k)

sachet contient alors les éléments [1, 2, 3]

On exécute alors l'instruction :

while not sachet.est_vide():
  tableau_sortie.append(sachet.retirer())

Comme le sac est une file, on retire successivement 1, 2, 3

Qu'on ajoute à la fin de tableau_sortie :

Donc tableau sortie contient [1, 2, 3]
```

Solution. Si le sac est une pile, alors `build` renvoie une copie de la liste `p`. Si le sac est une file, alors `build` renvoie une copie en ordre inverse de la liste `p`.

Il y a un peu de vocabulaire spécifique aux piles et aux files. Traditionnellement, ajouter un élément dans une **pile** se dit **empiler** (`push`), et l'enlever **dépiler** (`pop`), tandis qu'ajouter un élément dans une **file** se dit **enfiler** (`enqueue`), et l'enlever **défiler** (`dequeue`).

3. Python et les piles et files

Le moyen le plus simple de modéliser une **pile** ou une **file** est d'utiliser un objet `list` et les méthodes `append` et `pop`

Modéliser une pile avec un objet list

```
>>> pile = [ ]
>>> pile.append(1)
>>> pile.append(2)
>>> pile.append(3)
>>> pile.pop() # sort le dernier élément
3
>>> pile.pop() # sort le dernier élément
2
>>> pile.pop() # sort le dernier élément
1
```

Modéliser une file avec un objet list

```
>>> file = [ ]
>>> file.append(1)
>>> file.append(2)
>>> file.append(3)
>>> file.pop(0) # sort le premier élément
1
>>> file.pop(0) # sort le premier élément
2
>>> file.pop(0) # sort le premier élément
3
```

Bien sûr il est parfaitement possible de créer une classe et d'implémenter les notations traditionnelles (`push`, `pop`, `enqueue`, `dequeue`).

4. Utilisation des piles et files en informatique

Les gestionnaires de messages, utilisent un "**tampon**" (*buffer*) pour accumuler les messages à traiter. Si le traitement est long et qu'ils peuvent recevoir plusieurs messages simultanément, les messages sont rangés dans une **file**.

Un serveur (de fichier, web ou autre) est généralement accompagné d'un gestionnaire de message qui enregistre les requêtes avant de les faire exécuter par le serveur lui même.

Ainsi une machine peu puissante traite les requêtes à son rythme et cela évite que les requêtes soient perdues.

Les **appels successifs d'une fonction récursive** s'accumulent dans la **pile de récursion**. Le dernier appel retourne une valeur au précédent, qui retourne au précédent etc.

5. TD piles et files

Créer les classes `Pile` et `File` et implémenter les méthodes traditionnelles (`push`, `pop`, `enqueue`, `dequeue`).