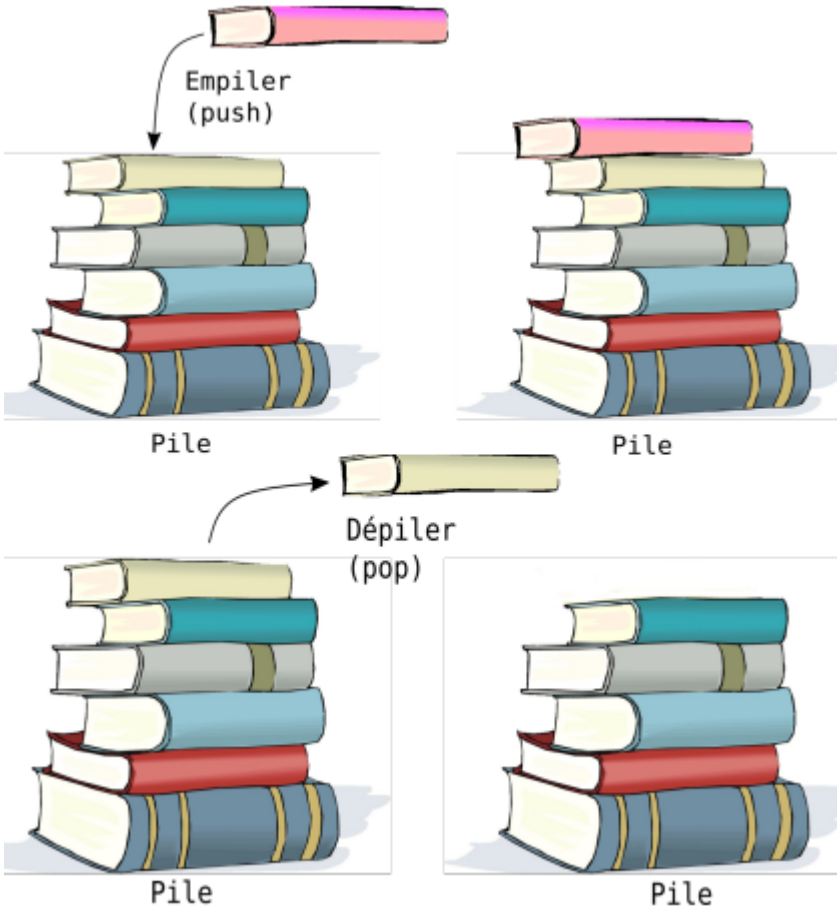
Chapitre 14 : Les piles, les files et les files de priorité

A) Les piles

Il s'agit d'une structure pour laquelle l'insertion et la suppression d'un élément s'effectue uniquement à la fin: C'est le principe du dernier entré, premier sorti.



3 fonctions de base sur les piles (que nous allons simuler par des listes)

In [1]:
```python
#une fonction qui permet de savoir si la pile est vide
def pile_vide(p):
    """fonction qui renvoie True si la pile est vide, False dans le
    return p==[]
# fonction qui ajoute un élément sur la pile
def empile(p,x):
    "ajoute l'élément x à la fin"
    p.append(x)
# fonction qui retire l'élément en haut de la pile
def desempile(p):
    "renvoie le sommet de la pile p et retir ce sommet"
    if len(p)>0:
        return p.pop()
```

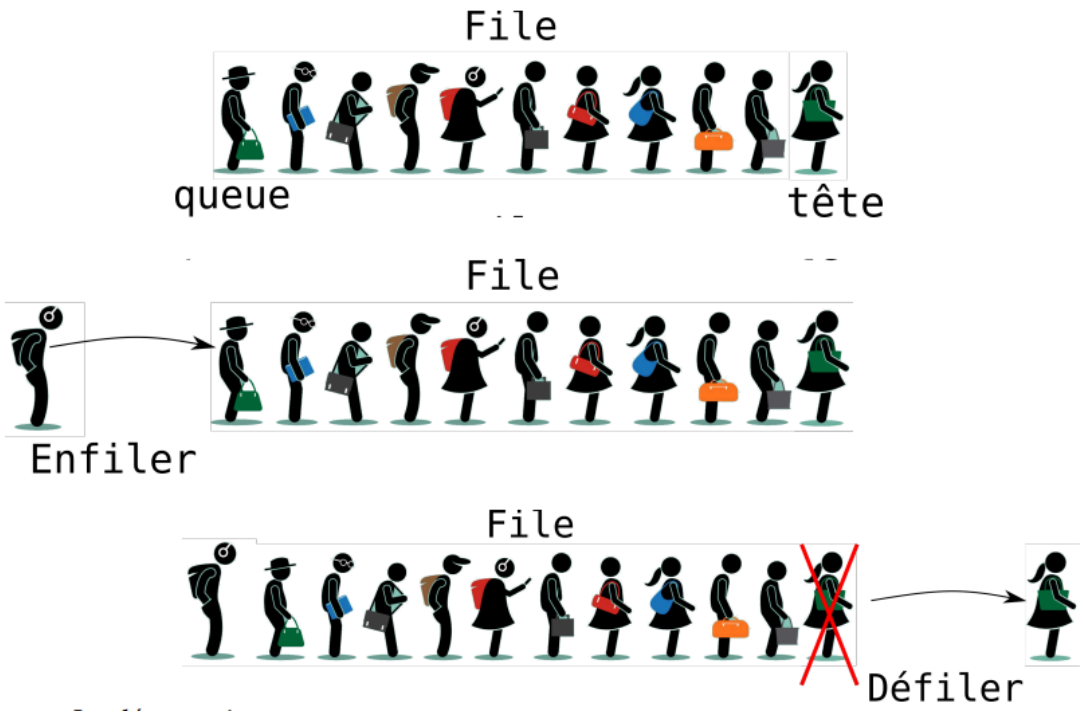
En partique, une pile a une capacité limitée : un débordement et donc une erreur est observée lorsque la capacité maximale est dépassée.

In [1]:
```python
def recursif(i):
    if i==0:
        return print("pas de dépassement")
    else :
        recursif(i-1)
recursif(1000)
```

```
pas de dépassement
```

B) les files

Il s'agit d'une structure pour laquelle on autorise l'insertion d'un élément d'un côté et la supression de l'autre, en suivant le principe du "premier rentré, premier sorti"



In [1]:
```
# fonction qui enlève le premier élément de la file
def defiler(f):
    """retire le 1e élément de la file"""
    if len(file)>0:
        return f.pop(0)#le pb de cette fonction est qu'elle de comp
```

C) La librairie collection

Dans la librairie *collections*, la fonction *deque* permet de générer un objet ayant un comportement similaire à celui d'une liste mais les ajouts et les retraits à chaque extrémité ont été optimisés. On peut donc l'utiliser pour les files et aussi les piles

In [4]:
```
from collections import deque
#pour les files
file = deque([1,2,3])
file.append(4)
print(file)#deque([1,2,3,4])
file.popleft()
print(file)#deque([2,3,4])

#pour les piles
pile = deque([1,2,3])
pile.append(4)
print(pile)#deque([1,2,3,4])
pile.pop()
print(pile)#deque([1,2,3])

```

```
deque([1, 2, 3, 4])
deque([2, 3, 4])
deque([1, 2, 3, 4])
deque([1, 2, 3])
```

D) File de priorité

Il est possible de générer des files pour lesquelles l'extraction d'un élément est faite suivant un critère de priorité

In [7]:
```
from queue import PriorityQueue

file_priorite=PriorityQueue()#initialisation
val1,nom1=1,"B"
val2,nom2=2,"C"
val3,nom3=0,"A"
file_priorite.put((val1,nom1))
file_priorite.put((val2,nom2))
file_priorite.put((val3,nom3))
print(file_priorite.empty())#renvoie True si vide
val,nom=file_priorite.get()#par défaut renvoie le couple associé à
print(val,nom)
```

```
False
0 A
```

In [ ]:
```
1
```