

Les piles et files

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 de la liste. C'est le principe du « dernier entré, premier sorti »

3 fonctions de base sur les piles :

Une fonction qui permet de savoir si une pile est vide	<pre>def pile_vide(p): """renvoie True si p est vide False sinon""" return p==[]</pre>
Une fonction qui ajoute un élément sur la pile (en anglais « push »)	<pre>def empile(p,x): """ajoute un élément x sur la pile p""" p.append(x)</pre>
Une fonction qui retire l'élément sur la pile	<pre>def desempile(p): """renvoie le sommet de la pile p non vide et le retire de p""" if len(p)>0: return p.pop()</pre>

En pratique une pile n'a pas une capacité illimitée : un débordement et donc une erreur est observée lorsque la capacité maximale est dépassée.

B) Les files

Il s'agit d'une structure pour laquelle on n'autorise l'insertion d'un élément d'un côté et la suppression de l'autres, en suivant le principe du « premier rentré, premier sorti » :

Fonction qui enlève le premier élément de pile	<pre>def defiler(f): """retire le le élément de la file""" return f.pop(0)</pre>
--	--

C) Librairie collection

Dans module *collection*, la structure *deque* permet de générer un objet ayant un comportement similaire à celui d'une liste mais les ajouts et les retraits sont rapides à chaque extrémité. On peut donc l'utiliser pour les files ou pour les piles :

```
from collections import deque
file = deque([1,2,3])
file.append(4)
print(file)#[1,2,3,4]
file.popleft()
print(file)#[2,3,4]
pile=deque([1,2,3])
pile.append(4)
print(pile)#[1,2,3,4]
pile.pop()
print(pile)#[1,2,3]
```

Commenté [AM1]: Ces structures piles se retrouvent (analogues à une pile d'assiettes) :

- Lors des appels récursifs
- Dans les navigateurs ou traitements de texte qui mémorisent les actions effectuées afin de pouvoir revenir en arrière pas à pas

Commenté [AM2]: On peut en citer d'autres :

```
def creer_pile():
    """crée une pile vide"""
    return []
def taille(p):
    """renvoie la taille de p"""
    return len(p)
def sommet(p):
    """renvoie le sommet de p sans le retirer"""
    if len(p)>0:
        return p[-1]
```

Commenté [AM3]: Typiquement une liste de longueur 2000 sous python

Commenté [AM4]: A l'image d'une file d'attente

Commenté [AM5]: Il faut noter que la complexité de :

- p.pop()* pour enlever le dernier élément est en $O(1)$
- p.pop(0)* pour enlever le premier élément est en $O(n)$

La structure *deque* évite ce problème de complexité