Chapitre 15 : 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 (que nous allons simuler par des listes) :

Une fonction qui permet de savoir si une pile est vide	`Def pile_vide(p) :`
	`  """renvoie True si p est vide False sinon"""`
	`  return p==[]`

Une fonction qui ajoute un élément sur la pile (en anglais « push »)	`def empile(p,x):`
	`  """ajoute un élément x sur la pile p"""`
	`  p.append(x)`

Une fonction qui retire l'élément sur la pile	`def desempile(p):`
	`  """renvoie le sommet de la pile p non vide`
	`  et le retire de p"""`
	`  if len(p)>0:`
	`  return p.pop()`

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 autorise l'insertion d'un élément d'un côté et la suppression de l'autre, en suivant le principe du « premier rentré, premier sorti » :

Fonction qui enlève le premier élément de file	`def defiler(f):`
	`  """retire le 1e élément de la file"""`
	`  return f.pop(0)`

C) 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 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]`

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.

```
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))#Ajout manuel
file_priorite.put((val2,nom2))
file_priorite.put((val3,nom3))
print(file_priorite.empty())#True si vide
val,nom=file_priorite.get() #couple(val3,nom3) est le 1e couple sorti
```

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