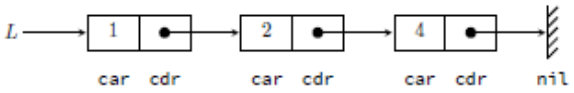
Structures de données : les listes, les piles et les files
Bilan sur les types abstraits listes, piles et files

1 - Les listes

a) Caractéristiques

Une liste est une structure de données permettant de regrouper des données.
Une liste L est composée de 2 parties :
- sa *tête* (souvent noté car), qui correspond au *dernier élément ajouté à la liste*,
- et sa *queue* (souvent noté cdr) qui correspond au *reste de la liste*.



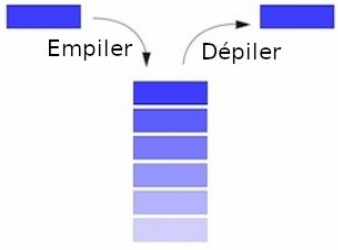
b) Opérations possibles

Voici les opérations qui peuvent être effectuées sur une liste :
- créer une liste vide,
- tester si une liste est vide,
- ajouter un élément en tête de liste,
- supprimer la tête x d'une liste L et renvoyer cette tête x,
- compter le nombre d'éléments présents dans une liste.

2 - Les piles

a) Caractéristiques

On retrouve dans les piles une partie des propriétés vues sur les listes.
Dans les piles, il est uniquement possible de manipuler le dernier élément introduit dans la pile.
Les piles sont basées sur le principe LIFO : dernier entré, premier sorti.



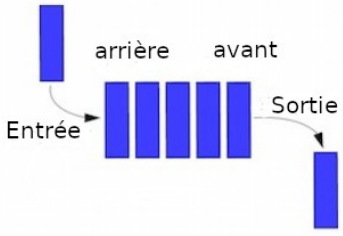
b) Opérations possibles

Voici les opérations que l'on peut réaliser sur une pile :
- créer une pile vide,
- savoir si une pile est vide,
- ajouter un nouvel élément sur la pile, on dit qu'on empile,
- récupérer l'élément au sommet de la pile tout en le supprimant, on dit que l'on dépile,
- accéder à l'élément situé au sommet de la pile sans le supprimer de la pile,
- connaître le nombre d'éléments présents dans la pile.

3 - Les files

a) Caractéristiques

Comme les piles, les files ont des points communs avec les listes.
Différences majeures : dans une file on ajoute des éléments à une extrémité de la file et on supprime des éléments à l'autre extrémité.
Les files sont basées sur le principe FIFO : premier entré, premier sorti.



b) Opérations possibles

Voici les opérations que l'on peut réaliser sur une file :
- créer une file vide,
- savoir si une file est vide,
- ajouter un nouvel élément à la file, on dit qu'on enfile,
- récupérer l'élément situé en bout de file tout en le supprimant, on dit qu'on défile,
- accéder à l'élément situé en bout de file sans le supprimer de la file,
- connaître le nombre d'éléments présents dans la file.

4 - Ce qu'il faut savoir faire
Implémenter les structures abstraites liste, pile et file en Python

a) Utilisation des listes comme des piles
Les méthodes des listes rendent très facile leur utilisation comme des piles, où le dernier élément ajouté est le premier récupéré (« dernier entré, premier sorti » ou LIFO .
Pour ajouter un élément sur la pile, utilisez la méthode append().
Pour récupérer l'objet au sommet de la pile, utilisez la méthode pop() sans indicateur de position.
Par exemple :

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

b) Utilisation des listes comme des files
Il est également possible d'utiliser une liste comme une file, où le premier élément ajouté est le premier récupéré (« premier entré, premier sorti » ou FIFO.
Toutefois, les listes ne sont pas très efficaces pour réaliser ce type de traitement.
Alors que les ajouts et suppressions en fin de liste sont rapides, les opérations d'insertions ou de retraits en début de liste sont lentes car tous les autres éléments doivent être décalés d'une position.

```
>>> queue = [8,1,24,9,0,16]
>>> queue.append(4)
>>> queue
[8, 1, 24, 9, 0, 16, 4]
>>> queue.append(5)
>>> queue
[8, 1, 24, 9, 0, 16, 4, 5]
>>> queue.pop(0)
8
>>> queue
[1, 24, 9, 0, 16, 4, 5]
>>> queue.pop(0)
1
>>> queue
[24, 9, 0, 16, 4, 5]
```

Pour implémenter une file, on peut utiliser la classe collections.deque qui a été conçue pour réaliser rapidement les opérations d'ajouts et de retraits aux deux extrémités.
Par exemple :

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")                       # Terry arrive
>>> queue.append("Graham")                      # Graham arrive
>>> queue.popleft()                             # le premier arrivé est enlevé
'Eric'
>>> queue.popleft()                             # le deuxième arrivé est enlevé
'John'
>>> queue                                       # liste dans l'ordre d'arrivée
deque(['Mickael','Terry', 'Graham'])
```