Les listes chaînées, piles et files

https://info.blaisepascal.fr/nsi-listes-chainees

Wikipedia

Type abstrait / structure de données

LISTE

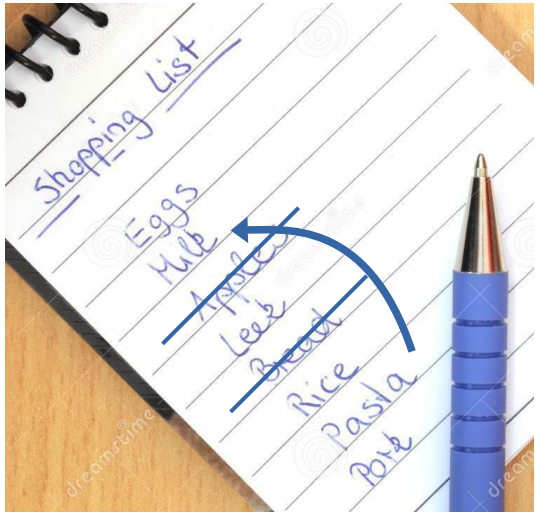
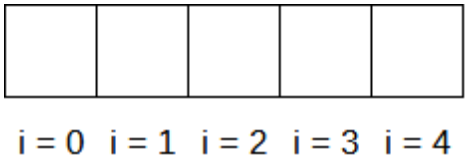
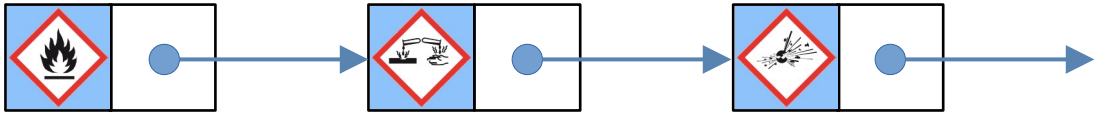


Tableau statique



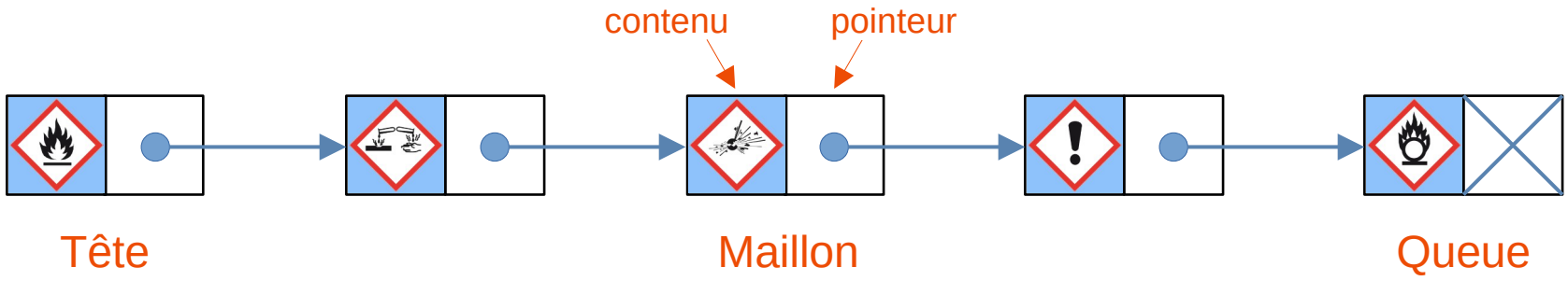
Liste chaînée



Structure d’une liste (simplement) chaînée

Constituée d’un ensemble de maillons ou éléments

Liés entre eux via leurs adresses mémoire



Une liste chaînée est entièrement défini par l’adresse son maillon de tête

Implémentations d’une liste chaînée

Avec un tableau : un maillon est un tableau de deux éléments [valeur, adresse]

Liste de taille 1 : [5, None]

Liste de taille 2 : [5, [3, None]]

Liste de taille 4 : [5, [3, [9, [3, None]]]]

Avec un objet : un maillon est un objet Maillon(valeur, adresse)

Liste de taille 1 : Maillon(5, None)

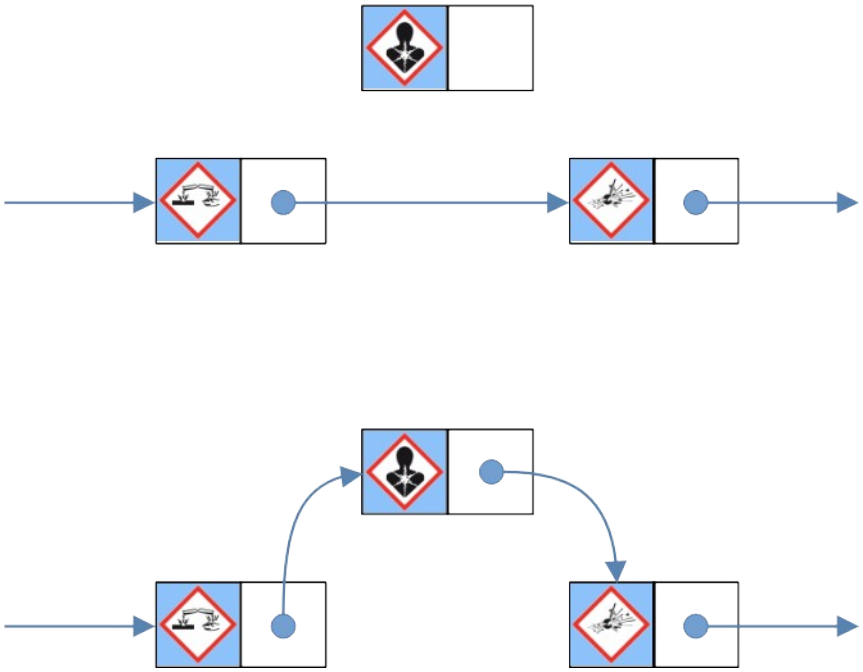
Liste de taille 2 : Maillon(5, Maillon(3, None))

Liste de taille 4 : Maillon(5, Maillon(3, Maillon(9, Maillon(3, None))))

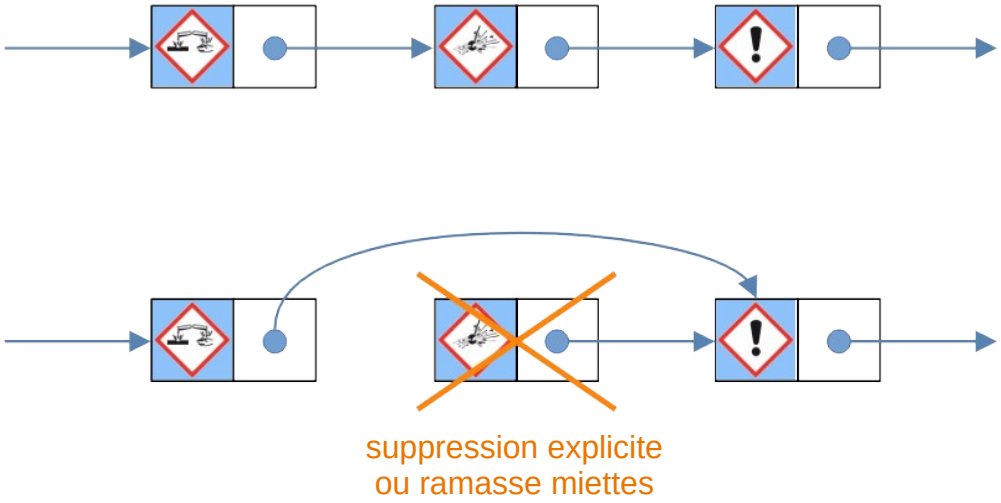
Pour **accéder à un couple** (valeur, adresse), il faudra **balayer** la liste

Opérations sur les listes chaînées

Insertion d’un maillon



Suppression d’un maillon



On travaille sur les pointeurs

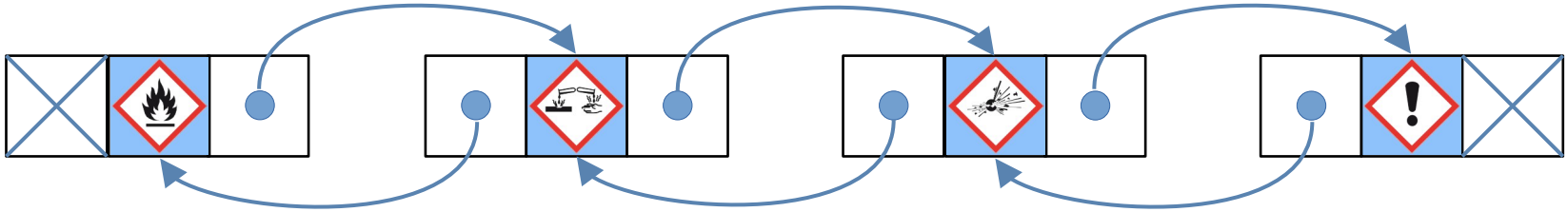
Primitives des listes chaînées

tests, accès en lecture ou en écriture, permettant une utilisation efficace

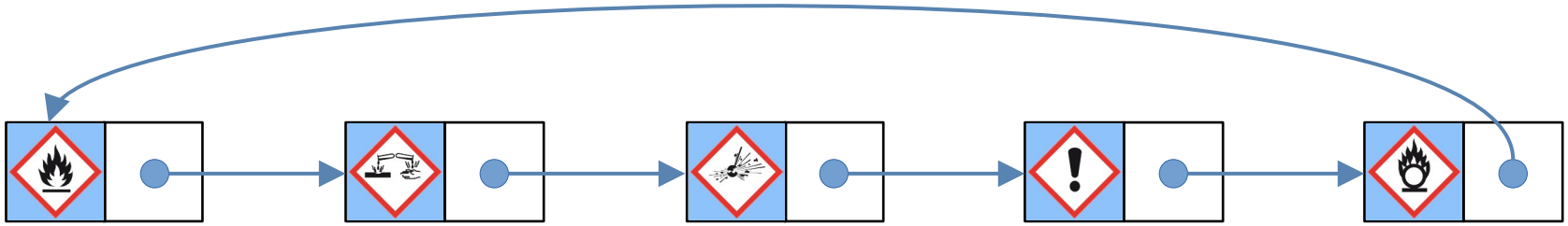
- « Liste est-elle vide ? »
- « Nombre d'éléments »
- « Ajouter en queue »
- « Ajouter en tête »
- « Insertion »
- « Remplacement »
- « Suppression »

- « Placement de l’index sur le premier élément »
- « Placement sur le dernier élément »
- « Placement sur l'élément suivant »
- « Placement sur l'élément précédent »
- « L'élément courant est-il le premier ? »
- « L'élément courant est-il le dernier ? »

Listes doublement chaînées

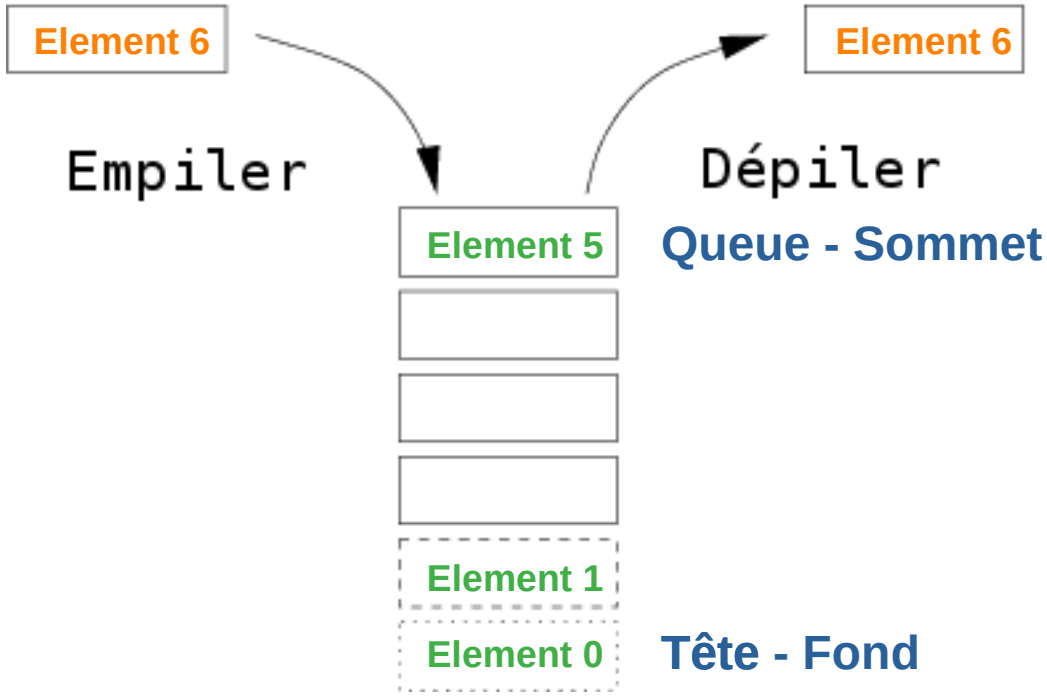


Listes cycliques



Les piles

LIFO : Last In First Out



Primitives

- « Empiler »
- « Dépiler »
- « La pile est-elle vide ? »
- « Nombres d'éléments »
- « Valeur de l'élément au sommet ? »
- « Vider la pile »
- « Dupliquer l'élément au sommet »
- « Permuter les deux premiers éléments »

Applications :

- Pile du microprocesseur
- Pile d'appel des algorithmes récursifs
- Mémorisation des pages web visitées
- Annulation de la frappe
- Mémorisation d'un chemin (trajet, recherche en profondeur)
- Inversion de tableau

Les files

FIFO : First In First Out

Primitives

- « Enfiler »
- « Défiler »
- « La pile est-elle vide ? »
- « Nombres d'éléments »

Applications :

- Mémoires tampons
- Mémoriser temporairement des transactions qui doivent attendre pour être traitées
- File d'attentes des serveurs d'impression
- Gestion des tâches par un système d'exploitation
- Algorithme de parcours en largeur

Abstraction des structures de données

Une structure de donnée est **spécifiée par son interface**

Pour une même interface, **plusieurs implémentations** plus ou moins coûteuses, **possibles**

Piles et files de Python

class `collections`**.deque**([*iterable*[, *maxlen*]])

Renvoie un nouvel objet *deque* initialisé de gauche à droite (en utilisant `append()`) avec les données d'"*iterable*. Si *iterable* n'est pas spécifié, alors la nouvelle *deque* est vide.

Les *deques* sont une généralisation des piles et des files (*deque* se prononce « *dèque* » et est l'abréviation de l'anglais *double-ended queue*) : il est possible d'ajouter et retirer des éléments par les deux bouts des *deques*. Celles-ci gèrent des ajouts et des retraits utilisables par de multiples fils d'exécution (*thread-safe*) et efficients du point de vue de la mémoire des deux côtés de la *deque*, avec approximativement la même performance en *O(1)* dans les deux sens.

Bien que les les objets `list` gèrent des opérations similaires, ils sont optimisés pour des opérations qui ne changent pas la taille de la liste. Les opérations `pop(0)` et `insert(0, v)` qui changent la taille et la position de la représentation des données sous-jacentes entraînent des coûts de déplacement de mémoire en *O(n)*.

- append(x)
- appendleft(x)
- pop(x)
- popleft(x)