Structures de données linéaires

En informatique, une structure de données est une manière d'organiser les données dans le but de faciliter leur traitement.
Pour prendre un exemple de la vie quotidienne, on peut présenter des numéros de téléphone par département, par nom, par profession (comme les Pages jaunes), par numéro téléphonique (comme les annuaires destinés au télémarketing), par rue et/ou une combinaison quelconque de ces classements. À chaque usage correspondra une structure d'annuaire appropriée.
En organisant d'une certaine manière les données, on permet un traitement automatique de ces dernières plus efficace et rapide. On parle de **type abstrait de données (TAD).**
Le fait d'utiliser une structure de données appropriée à un traitement informatique peut également faire baisser de manière significative la complexité d'une application informatique et ainsi contribuer à diminuer le taux d'erreurs.

1. Piles (Stack)

Pour beaucoup d'applications, les seules opérations à effectuer sur les listes sont des insertions et des suppressions aux extrémités. Dans les piles les insertions et les suppressions se font à une seule extrémité, appelée sommet de pile.

Exemples d'utilisation

- ➔ La fonction annuler sur un logiciel
- ➔ Pages web visitées sur un navigateur
- ➔ Dans un microprocesseur une pile est réservée dans la mémoire afin d'y ranger les arguments des sous programmes et les adresses de retour.

Fonctionnement d'une pile

Une pile est une structure de données linéaire (données rangées en ligne) qui a pour principe que le dernier élément entré sera le premier sorti. Les piles sont aussi appelées **LIFO**, pour **Last-In-First-Out**.
Une bonne image pour représenter une pile est une pile d'assiettes : c'est en haut de la pile qu'il faut prendre ou mettre une assiette.
L'implémentation en mémoire de cette structure est très simple, elle est réalisée en affectant une zone mémoire contiguë de taille égale à la taille de la pile souhaitée. Le haut de la pile (position du dernier élément empilé) étant constamment repéré pointeur.

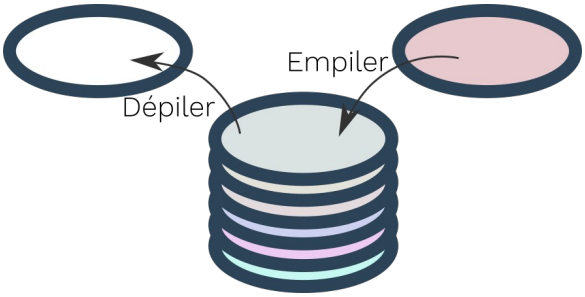


Figure 1: Fonctionnement d'une pile

Interface de manipulation

La manipulation des types de données abstraites s'effectue à partir d'un ensemble de routines constituant l'interface de manipulation. L'interface de manipulation d'une pile est habituellement composée des primitives suivantes :

Nom	Opérations	Implémentation Python à partir d'une liste
`creer_pile()`	Retourne une pile vide	= []
`est_vide(p)`	Teste si la pile **p** est vide. Renvoie True si vide.	== []
`empiler(p,e)`	Empile l'élément **e** sur la pile **p**	.append(e)
`depiler(p)`	Dépiler de **p** un élément	.pop()

Remarque : La méthode pop() a deux effets :
- ✗ elle retire le dernier élément de la liste
- ✗ elle renvoie la valeur de l'élément retiré

2.File (queue)

Une file dite aussi file d'attente (en anglais queue) est une structure de données basée sur le principe du premier entré, premier sorti désigné en anglais par l'acronyme FIFO (first in, first out)

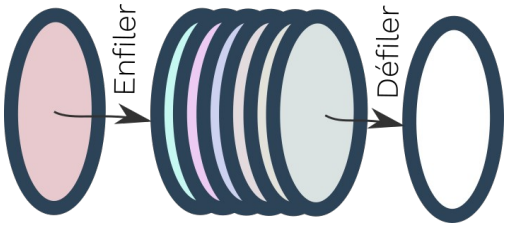


Figure 2: fonctionnement d'une file

Exemples d'utilisation

- ✗ File d'attente d'un serveur d'impression qui traite les documents à imprimer au fur et à mesure où ils arrivent (spool)
- ✗ Mémoire tampon d'une carte réseau (buffer) qui mémorise les bits transmis lorsqu'ils arrivent et les transmet au microprocesseur à la demande sous forme d'octets.

Opérations de base

L'interface de manipulation d'une file est composée habituellement des primitives suivantes :

Nom	Opérations	Implémentation Python à partir d'une liste
`creer_file()`	Retourne une file vide	= []
`est_vide(p)`	Teste si la file **f** est vide. Renvoie True si vide.	== []
`enfiler(p,e)`	Enfile l'élément **e** sur la file **f**	.append(e)
`defiler(p)`	Défiler de **f** un élément	.pop(0)

3.Les listes

Une liste est une structure de données permettant de regrouper des données et dont l'accès est séquentiel. Elle correspond à une suite finie d'éléments repérés par leur rang (index). Les éléments sont ordonnés et leur place a une grande importance.
Une liste est évolutive, on peut ajouter ou supprimer n'importe quel élément.

Notions de listes et de tableaux

Un tableau est un ensemble de taille fixe contenant des valeurs de même type. Le fait que la taille ne varie pas durant son utilisation (interdiction de rajouter des éléments) facilite son implémentation en mémoire en allouant un espace mémoire contiguë.
Contrairement au tableau, la taille d'une liste peut varier au cours du temps. En effet, une liste n'est pas allouée en une seule fois, mais chaque élément est alloué indépendamment, sous la forme d'un maillon.

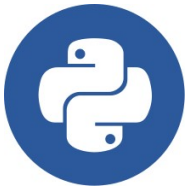
Temps d'accès à une élément

Le temps d'accès à un élément d'un tableau par son index est constant, quel que soit l'élément désiré (O(1)). Cela s'explique par le fait que les éléments d'un tableau sont contigus dans l'espace mémoire. Ainsi, il est possible de calculer l'adresse mémoire de l'élément auquel on veut accéder, à partir de l'adresse de base du tableau et de l'index de l'élément. L'accès est immédiat, comme il le serait pour une variable simple.
Dans une liste, l'accès à un élément nécessite de balayer tous les éléments à partir du premier jusqu'à l'indice voulu. Comparé à un tableau, ceci a pour effet de détériorer le temps d'accès (O(n))

Opérations de base

L'interface de manipulation d'une liste est composée habituellement des primitives suivantes :

Nom	Opérations
`creer_liste()`	Retourne une liste vide
`est_vide(p)`	Teste si la list **`l`** est vide. Renvoie True si vide.
`inserer(l,e,i)`	Insère un élément **`e`** dans la liste **`l`** à l'index **`i`**
`supprimer(l,i)`	Supprime l'élément situé à l'index **`i`** de la liste **`l`**
`lire(l,i)`	Retourne l'élément situé à l'index **`i`** de la liste **`l`**
`modifier(l,e,i)`	Modifie l'élément situé à l'index **`i`** de la liste **`l`** avec la valeur **`e`**
`longueur(l)`	Renvoie le nombre d'élément présent dans la liste **`l`**



Remarque : Les listes sous Python sont en fait implémentées dans des tableaux dont la taille est modifiée au fur et à mesure d'insertion de données. Ces tableaux se nomment des tableaux dynamiques. Ils offres un meilleur temps d'accès lors d'une lecture/ écriture (temps constant) par contre ils nécessitent une recopie entière des données lors d'une nouvelle allocation d'espace du tableau.

4.Les dictionnaires

Un dictionnaire est une structure de données qui permet d'associer une valeur à une clé. Cette clé peut être un mot ou en entier. L'ensemble clé-valeur est appelée entrée ou enregistrement.
On rappelle que les dictionnaires ne comportent pas d'ordre contrairement aux listes. L'accès à un élément s'effectue donc à partir de sa clé et non pas grâce à un index.

Opérations de base

L'interface de manipulation d'une liste est composée habituellement des primitives suivantes :

Nom	Opérations
`creer_dico()`	Retourne une liste vide
`est_vide(d)`	Teste si la list **`d`** est vide. Renvoie True si vide.
`inserer(d,cle, val)`	Insère l'ensemble **`cle`**, **`val`** dans le dico **`d`**
`supprimer(d,cle)`	Supprime l'enregistrement **`cle`** du dico **`d`**
`lire(d,cle)`	Retourne la valeur associée à la clé **`cle`**
`longueur(d)`	Renvoie le nombre d'enregistrement du dico **`d`**