Listes, piles et files

1. Introduction

Dans ce cours, le mot « liste » ne désigne pas les listes que l'on utilise habituellement en Python, mais une structure de données différente définie plus bas.

Une structure de données est une manière d'organiser les données pour les traiter plus facilement. Les listes, les piles (*stack* en anglais) et les files (*queue* en anglais) sont des structures linéaires et séquentielles.
Les tableaux sont des structures indexées. Nous verrons plus tard les arbres, des structures hiérarchiques, et les graphes, des structures relationnelles.
Les **listes Python** ne sont ni des listes, ni des tableaux, ce sont des **tableaux dynamiques**.

2. Les listes

Une liste est une structure abstraite de données permettant de regrouper des données sous une forme séquentielle. Elle est constituée d'éléments d'un même type, chacun possédant un rang.
Une liste est évolutive : on peut ajouter ou supprimer n'importe lequel de ses éléments.
Une liste L est composée de 2 parties :

- sa tête, qui correspond au dernier élément ajouté à la liste ;
- sa queue qui correspond au reste de la liste.

Le langage de programmation Lisp (inventé par John McCarthy en 1958) a été l'un des premiers langages de programmation à introduire cette notion de liste (Lisp signifie "list processing").

Voici les opérations qui peuvent être effectuées sur une liste :

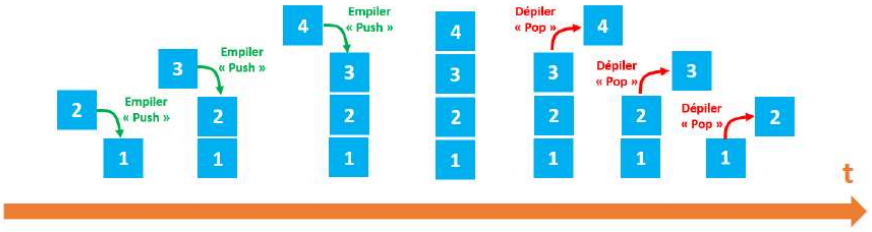
Actions	Instructions
Créer une liste **L** vide	`L = creer_liste_vide()`
Tester si la liste **L** est vide	`liste_vide(L)`
Ajouter un élément **x** en tête de la liste **L**	`ajout(x, L)`
Supprimer la tête **x** d'une liste **L** et renvoyer cette tête **x**	`suppr(L)`
Compter le nombre d'éléments dans une liste **L**	`compte(L)`
Créer une nouvelle liste **L1** à partir d'un élément **x** et d'une liste existante **L**	`L1 = cons(x, L)`

✍ Voici une série d'instructions, compléter ce qui se passe à chacune des étapes :

`L = creer_liste_vide()`	création d'une liste **L** vide
`liste_vide(L)`	renvoie . . . .
`ajout(3, L)`	la liste **L** contient l'élément . . .
`liste_vide(L)`	. . . .
`ajout(5, L)`	la tête de la liste **L** correspond à . . . et la queue contient l'élément . . .
`ajout(8, L)`	. . . .
`t = suppr(L)`	. . . .
`L1 = creer_liste_vide()`	. . . .
`L2 = cons(8, cons(5, cons(3, L1)))`	. . . .

3. Les piles

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. Il est fondé sur le principe du « **dernier arrivé, premier sorti** » : elles sont dites de type **LIFO** (**Last In, First Out**).
On prend souvent l'analogie avec une pile d'assiettes : dans une pile d'assiettes la seule assiette directement accessible et la dernière assiette qui a été déposée sur la pile.



Voici les opérations que l'on peut réaliser sur une pile :

Actions	Instructions
Créer une pile **P** vide	`L = créer_pile_vide()`
Tester si la pile **P** est vide	`pile_vide(P)`
Empiler un nouvel élément **x** sur la pile **P**	`empile(x, P)`
Dépiler l'élément au sommet de la pile **P** et le récupérer	`depile(P)`
Récupérer l'élément au sommet de la pile **P** sans le supprimer	`sommet(P)`
Compter le nombre d'éléments de la pile **P**	`taille(P)`

Soit une pile P composée des éléments suivants : 12, 14, 8, 7, 19 et 22 (le sommet de la pile est 22).
✍ Voici des instructions, compléter ce qui se passe. **À chaque fois on repart de la pile d'origine**.

`depile(P)` renvoie 22 et la pile P est maintenant composée des éléments suivants : 12, 14, 8, 7 et 19
`empile(49, P)` . . . .
`sommet(P)` . . . .
on applique `depile(P)` 6 fois de suite
`pile_vide(P)` . . . .

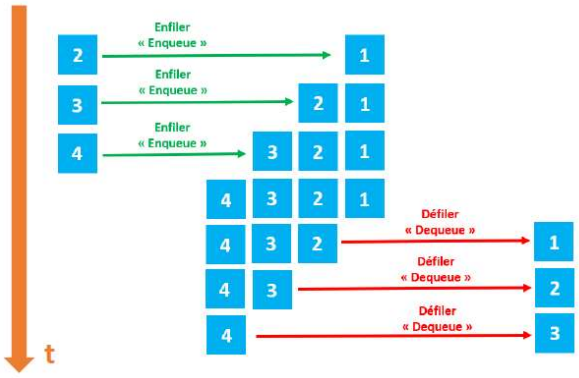
De nombreuses applications s'appuient sur l'utilisation d'une pile. En voici quelques-unes :

- dans un navigateur web, une pile sert à mémoriser les pages web visitées ; l'adresse de chaque nouvelle page visitée est empilée et l'utilisateur dépile l'adresse de la page précédente en cliquant sur le bouton « Afficher la page précédente »
- l'évaluation des expressions mathématiques en notation post-fixée (ou polonaise inverse) utilise une pile ;
- la fonction « Annuler la frappe » (Undo en anglais) d'un traitement de texte mémorise les modification apportées au texte dans une pile ;
- la récursivité (une fonction qui fait appel à elle-même) utilise également une pile ;
- …

Remarque : en informatique, un **dépassement de pile** ou **débordement de pile** (en anglais, ***stack overflow***) est un bug causé par un processus qui, lors de l'écriture dans une pile, écrit à l'extérieur de l'espace alloué à la pile, écrasant ainsi des informations nécessaires au processus.

4. Les files

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. On prend souvent l'analogie de la file d'attente devant un magasin pour décrire une file de données.



Les files sont basées sur le principe **FIFO** (**First In First Out**) : le premier qui est rentré sera le premier à sortir. Ici aussi, on retrouve souvent ce principe FIFO en informatique.

Voici les opérations que l'on peut réaliser sur une file :

Actions	Instructions
Créer une file **F** vide	`F = créer_file_vide()`
Tester si la file **F** est vide	`file_vide(F)`
Ajouter un nouvel élément **x** à la file **F**	`ajout(x, F)`
Récupérer l'élément en bout de file tout en le supprimant	`retire(F)`
Accéder à un élément en bout de file sans le supprimer de la file	`premier(F)`
Compter le nombre d'éléments de la file **F**	`taille(F)`

Soit une file F composée des éléments suivants : 12, 14, 8, 7, 19 et 22 (le premier élément rentré dans la file est 22 ; le dernier élément rentré dans la file est 12).
✍ Voici des instructions, compléter ce qui se passe. **À chaque fois on repart de la file d'origine**.

`ajout(F,42)` la file F est maintenant composée des éléments suivants : 42, 12, 14, 8, 7, 19 et 22 (le premier élément rentré dans la file est 22 ; le dernier élément rentré dans la file est 42)
`retire(F)` . . . .

`premier(F)` . . . .
on applique `retire(F)` 6 fois de suite
`file_vide(F)` . . . .
on applique `retire(F)` une fois
`taille(F)` . . . .

En général, on utilise une file pour mémoriser temporairement des transactions qui doivent attendre pour être traitées. Voici quelques exemples d'applications :

- les serveurs d'impression, qui doivent traiter des requêtes dans l'ordre dans lequel elles arrivent, et les insère dans une file d'attente (ou une queue) ;
- les requêtes entre machines sur un réseau ;
- certains moteurs multi-tâches, dans un système d'exploitation, qui doivent accorder du temps machine à chaque tâche, sans en privilégier une plus qu'une autre ;
- un algorithme de parcours en largeur utilise une file pour mémoriser les nœuds visités ;
- on utilise des files pour créer toutes sortes de mémoires tampons (buffers en anglais) ;
- …

Source : https://pixees.fr/informatiquelycee/n_site/nsi_term_structDo_liste.html