Structures de données

Listes, piles et files

Objectifs pédagogiques :

- Types abstraits : liste, pile et file
- Distinguer les modes FIFO (first in first out) et LIFO (last in first out) des piles et des files.

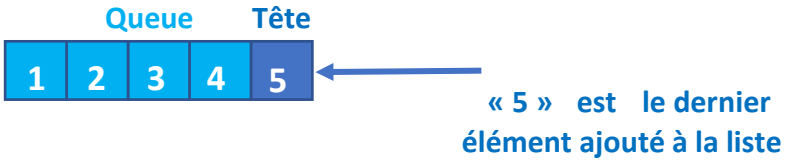
Les **listes** **(attention ; C'est différents des listes en Python, qui sont des tableaux)**, les piles (*stack* en anglais) et les files (*queue* en anglais) sont des **structures abstraites de données** fondamentales en informatique. Elles diffèrent par les conditions d'ajout et d'accès aux éléments qui les constituent.

1. Notions de liste, pile et de file

1.1. 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** (souvent noté *car*), qui correspond au **dernier élément ajouté** à la liste ;
- **sa queue** (souvent noté *cdr*) 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	Instruction
Créer une liste L vide	`L = vide()`
Tester si la liste L est vide	`estVide(L)`
Ajouter un élément x en tête de la liste L	`ajouteEnTete(x,L)`
Supprimer la tête x d'une liste L et renvoyer cette tête x	`supprEnTete(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)`

Exemple
Voici une série d'instructions (les instructions ci-dessous s'enchaînent).

- **L=vide()** => on a créé une liste vide
- **estVide(L)** => renvoie vrai
- **ajoutEnTete(3,L)** => La liste L contient maintenant l'élément 3
- **estVide(L)** => renvoie faux
- **ajoutEnTete(5,L)** => la tête de la liste L correspond à 5, la queue contient l'élément 3
- **ajoutEnTete(8,L)** => la tête de la liste L correspond à 8, la queue contient les éléments 3 et 5
- **t = supprEnTete(L)** => la variable t vaut 8, la tête de L correspond à 5 et la queue contient l'élément 3
- **L1 = vide()**
- **L2 = cons**(8, cons(5, cons(3, L1))) => La tête de L2 correspond à 8 et la queue contient les éléments 3 et 5

Q1.
Voici une série d'instructions (les instructions ci-dessous s'enchaînent), expliquez ce qui se passe à chacune des étapes :

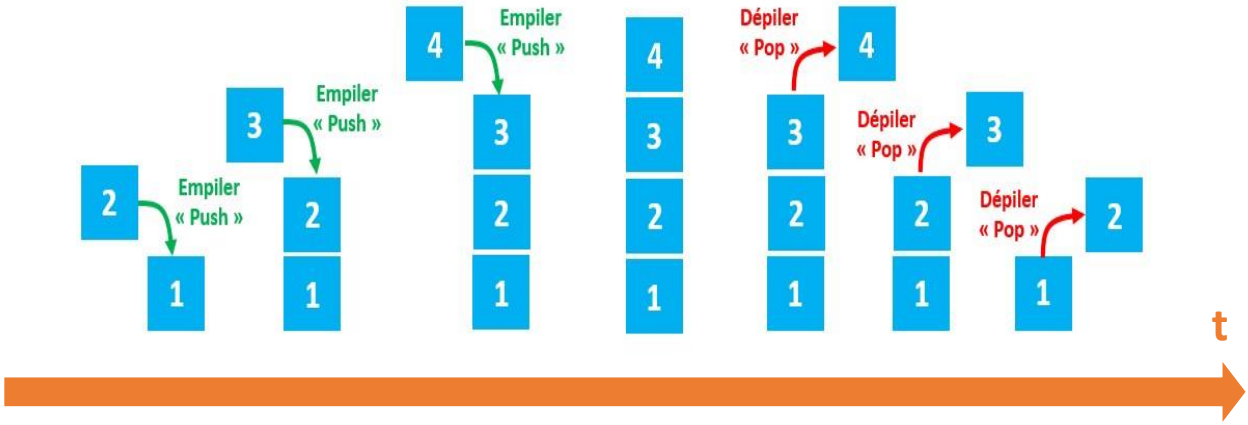
```
L = vide()
ajoutEnTete(10,L)
ajoutEnTete(9,L)
ajoutEnTete(7,L)
L1 = vide()
L2 = cons(5, cons(4, cons(3, cons (2, cons(1, cons(0,L1))))))
```

Q2.
Exercice à partir du fichier python ci-joint (Jupyter Notebook)

Remarque :
les types abstraits de données **piles** et **files**, comme nous allons le voir ci-après, sont des listes possédant des restrictions au niveau de l'ajout ou de la suppression de leurs éléments.
En effet ces actions ne pourront être réalisées que selon certaines modalités aux extrémités de ces deux types de structures abstraites de données.

1.2. Les piles (stacks)

Les **piles** sont fondées sur le principe du « **dernier arrivé, premier sorti** » : elles sont dites de type **LIFO** (**Last In, First Out**).
C'est le principe même de la pile d'assiette : c'est la dernière assiette posée sur la pile d'assiettes sales qui sera la première lavée.



L'insertion d'un élément dans la pile est appelée « **Empiler** » et la suppression d'un élément de la pile est appelée « **Dépiler** ».
Dans la pile, nous gardons toujours trace du dernier élément présent dans la liste avec un pointeur appelé top.

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

- on peut savoir si une **pile est vide** (pile_vide?)
- on peut **empiler** un nouvel élément sur la pile (push)
- on peut récupérer l'élément au sommet de la pile tout en le supprimant.
  On dit que l'on **dépile** (pop)
- on peut **accéder à l'élément situé au sommet de la pile sans le supprimer** de la pile (sommet)
- on peut connaitre **le nombre d'éléments présents dans la pile** (taille)

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 ;
- etc …

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.
Stack Overflow est aussi connu comme étant un site web proposant des questions et réponses sous la forme d'un forum d'entre-aide sur un large choix de thèmes concernant la programmation informatique.
Il y a de forte chance que vous trouviez la réponse à un problème informatique même ardu sur Stack Overflow !

Exemples

Soit une pile P composée des éléments suivants : 12, 14, 8, 7, 19 et 22 (le sommet de la pile est 22) **Pour chaque exemple ci-dessous on repart de la pile d'origine** :

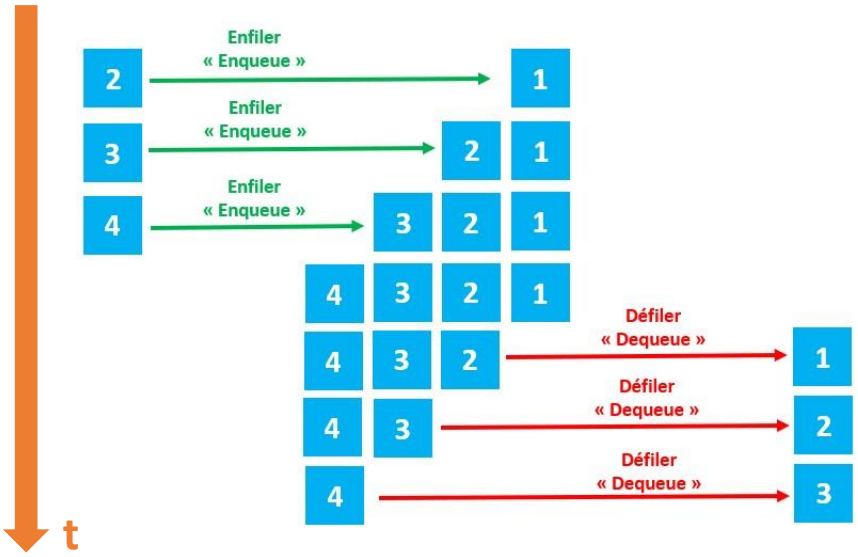
- **pop(P)** renvoie 22 et la pile P est maintenant composée des éléments suivants : 12, 14, 8, 7 et 19 (le sommet de la pile est 19)
- **push**(P,42) la pile P est maintenant composée des éléments suivants : 12, 14, 8, 7, 19, 22 et 42
- **sommet**(P) renvoie 22, la pile P n'est pas modifiée
- si on applique pop(P) 6 fois de suite, **pile_vide?(**P) renvoie vrai
- Après avoir appliqué pop(P) **une fois, taille(P) renvoie 5**

Q3.

Soit une pile P composée des éléments suivants : 15, 11, 32, 45 et 67 (le sommet de la pile est 67).
Quel est l'effet de l'instruction pop(P)

1.3. Les files (queues)

Les **files** sont fondées sur le principe du « **premier arrivé, premier sorti** » : elles sont dites de type **FIFO** (**First In, First Out**). C'est le principe de la file d'attente devant un guichet.



L'insertion d'un élément dans une file s'appelle uneopération de mise en file «**Enfiler** » et la suppression d'un élément s'appelle une opération de retrait de la file « **Défiler** ». Dans la file, nous maintenons toujours deux pointeurs, l'un pointant sur l'élément qui a été inséré en premier et qui est toujours présent dans laliste avec le pointeur en avant et l'autre pointant sur l'élément inséré en dernier avec le pointeur arrière.

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) ;
- etc…

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

- on peut savoir si une **file est vide** (file_vide?)
- on peut **ajouter** un nouvel élément à la file (ajout)
- on peut **récupérer l'élément situé en bout de file tout en le supprimant** (retire)
- on peut **accéder à l'élément situé en bout de file sans le supprimer de la file** (premier)
- on peut connaitre le **nombre d'éléments présents** dans la file (taille)

Exemples :

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). Pour chaque exemple ci-dessous 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)** la file F est maintenant composée des éléments suivants : 12, 14, 8, 7, et 19 (le premier élément rentré dans la file est 19 ; le dernier élément rentré dans la file est 12)
- **premier(F)** renvoie 22, la file F n'est pas modifiée
- si on applique **retire(F) 6 fois de suite**, file_vide?(F) **renvoie vrai**
- Après avoir appliqué **retire(F)** une fois, **taille(F) renvoie 5**.

Q4

Soit une file F composée des éléments suivants : 1, 12, 24, 17, 21 et 72 (le premier élément rentré dans la file est 72 ; le dernier élément rentré dans la file est 1).
Quel est l'effet de l'instruction ajout(F,25)

1.4. Piles vs files : synthèse

Pile	File
Les objets sont insérés et supprimés à 1 seule extrémité	Les objets sont insérés et retirés aux 2 extrémités.
Dans les piles, un seul pointeur est utilisé. Il pointe vers le haut de la pile.	Dans les files, deux pointeurs différents sont utilisés pour les extrémités; le tète et la fin.
Dans les piles, le dernier objet inséré est le premier à sortir.	Dans les files, l'objet inséré en premier est le premier qui sera supprimé.
Les piles suivent l'ordre Last In First Out (**LIFO**)	Les files suivent l'ordre First In First Out (**FIFO**)
Les opérations de pile s'appellent « **Empiler »** et « **Dépiler »**.	Les opérations de file sont appelées « **Enfiler »** et « **Défiler »**.
Les piles sont visualisées sous forme de collections verticales.	Les files sont visualisées sous forme de collections horizontales.