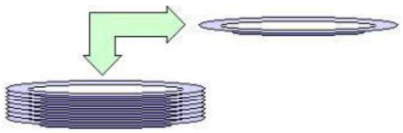
Parmi les listes, il existe 2 structures de données particulières dont les accès mémoire sont optimisés : les piles et les files.

3. Les piles

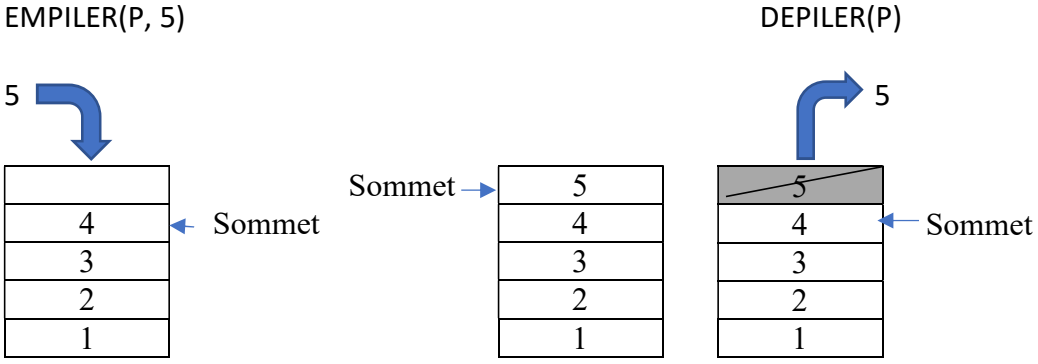
Définition : il s'agit d'une structure des données qui donne accès en priorité aux dernières données ajoutées. La dernière donnée ajoutée sera la première à en sortir. Autrement dit, **on ne peut accéder qu'à l'objet situé au sommet de la pile**. On résume cela par l'expression : **"dernier entré, premier sorti"** c'est-à-dire, en anglais, **"Last In, First Out" (LIFO)**.

Analogie : Pour "visualiser" une pile (informatique), penser à une pile d'assiettes : l'ordre dans lequel les assiettes sont dépilées est l'inverse de celui dans lequel elles ont été empilées, puisque seule l'assiette supérieure est accessible.



Les deux opérations élémentaires dont on a besoin avec cette structure sont :

EMPILER(P, x) qui correspond à l'insertion de la donnée x au sommet de la pile P si la pile n'est pas pleine.
DEPILER(P) qui retire la dernière donnée de la pile Pet la retourne si la pile n'est pas vide.



On peut aussi définir d'autres opérations comme :
PILE_VIDE(P) qui indique si la pile P est vide ou non.
PILE_PLEINE(P) qui indique si la pile P est pleine ou non.

Applications : par exemple, dans un navigateur web, une pile peut servir à mémoriser les pages visitées. L'adresse de chaque nouvelle page visitée est empilée et en cliquant sur un bouton "Afficher la page précédente", on dépile l'adresse de la page précédente.
Autre exemple : dans une console Python, les entrées successives sont stockées dans une pile.

Type abstrait Pile

Type abstrait : Pile
Données : éléments de type T

Opérations
CREER_PILE_VIDE() qui retourne un objet de type Pile.
La pile existe et elle est vide.

EMPILER(P, e)
L'élément est inséré au sommet de la pile P.

DEPILER(P) qui retourne un objet de type T
L'élément situé au sommet de la pile P est enlevé de la pile et est retourné.
EST_VIDE(P) qui retourne un objet de type Booléen.
Retourne Vrai si la pile P est vide et retourne Faux sinon.
EST_PLEINE(P) qui retourne un objet de type Booléen.
Retourne Vrai si la pile P est pleine et retourne Faux sinon.

Conditions :
EMPILER(P, e) est définie si, et seulement si, EST_PLEINE(P) = Faux.
DEPILER(P) est définie si, et seulement si, EST_VIDE(P) = Faux.

Exemple d'application de ce type abstrait :

On considère la suite d'instructions suivante :
P = CREER_PILE_VIDE()
EMPILER(P, 3)
EMPILER(P, 2)
N = DEPILER(P)
EMPILER(P, 5)
EMPILER(P, 7)
EMPILER(P, 9)

Donner une représentation de la pile P à chaque étape et le contenu de la variable N.

Représentation d'une pile avec un tableau :

On peut réaliser une pile capable de contenir n éléments avec un tableau P[0..n] pouvant contenir (n + 1) éléments :
●La première case du tableau (d'indice 0) contient l'indice de la prochaine case vide (c'est l'indice qui correspondra au prochain élément à insérer dans la pile).
●Les cases suivantes du tableau (d'indices 1 à n), contiennent les éléments de la pile ou sont vides. La dernière case non vide du tableau est le sommet de la pile.

Remarques :
●Si P[0] == 1 alors la pile est vide.
●A chaque fois qu'on empile un élément, on augmente P[0] d'une unité.
●A chaque fois que l'on dépile un élément, on diminue P[0] d'une unité, et il n'est pas nécessaire de toucher, dans le tableau, à la donnée qui vient d'être dépilée. Elle est toujours présente "physiquement" dans le tableau, mais on n'y a plus accès par l'intermédiaire des fonctions sur la pile.
●Lorsque P[0] == n + 1 alors la pile est pleine.

Exemple :
On représente une pile P à l'aide d'un tableau de taille fixe 6 : P[0..5]
Lors de la création de la pile vide, toutes les cases du tableau sont initialisées à 0, sauf P[0] qui contient 1.
1) Ecrire le contenu du tableau si l'on empile successivement dans P les éléments 8, 3, 5 :
2) Ecrire le contenu du tableau si on exécute la fonction DEPILER(P).

Exercices sur les piles :

Exercice 1 :
On suppose que l'on a placé dans une pile P la chaine de caractères "BONJOUR". On suppose que l'on dispose des opérations de base sur les piles. Ecrire un algorithme permettant de créer une pile N contenant les éléments de P mais en sens inverse, c'est-à-dire représentant la chaine de caractères "RUOJNOB".

Exercice 2 :

Implémenter en Python les opérations classiques sur les piles à l'aide d'un tableau que l'on "simulera" à l'aide d'une liste. On supposera que le tableau a une taille fixe, par exemple 10.

Exercice 3 :

Proposer une implémentation des opérations classiques de la pile à l'aide des méthodes .pop() et .append() du type liste de Python.

Exercice 4 : Bon parenthèsages ?

Une expression (algébrique ou arithmétique) est correctement parenthésée si d'une part, le nombre de parenthèses et crochets, ouvrants et fermants, est le même et si d'autre part les correspondances ne se croisent pas.
Par exemple, l'expression "[3 + (5 – 7] * 3)" n'est pas correctement parenthésée car la parenthèse fermante ne peut pas venir après le crochet fermant.
1. Ecrire un algorithme qui s'aide d'une pile pour contrôler le bon parenthésage d'une expression.
2. Ecrire en Python une fonction Est_bien_Parenthesee(E) qui renvoie True si l'expression E est correctement parenthésée, et False dans le cas contraire.

Exercice 5 : Calcul d'une expression arithmétique postfixée

On ne considère ici que les expressions numériques contenant les quatre opérations élémentaires (+, −, *, /).
La notation postfixée, appelée aussi Notation Polonaise Inversée (NPI), consiste à mettre les opérations arithmétiques **après** leurs arguments.
Par exemple, l'expression 5 + 3 s'écrit : 5 3 + en NPI.
L'avantage, c'est que la NPI se passe complètement de parenthèses. Par exemple, l'expression 3*((4 + 5) − 6) peut s'écrire en NPI : 3 4 5 + 6 − *.
On se propose de se servir d'une **pile** pour calculer une expression arithmétique exprimée en NPI.
Les données sont lues comme une suite de caractères. Chaque donnée est séparée des autres par un espace. La méthode consiste à utiliser une **pile** pour conserver les **valeurs numériques** au fur et à mesure de leur lecture (de la gauche vers la droite), et à effectuer un traitement quand un **opérateur** est lu. Ce traitement doit dépiler les deux dernières valeurs présentes dans la pile, effectuer l'opération, puis empiler le résultat dans la pile.
Il faut être attentif ici à l'ordre des opérandes dans le cas des opérateurs non commutatifs (− et /).
Le résultat final est contenu dans la pile.

1. Ecrire un algorithme utilisant une pile pour effectuer un calcul en NPI.
2. Implémenter cet algorithme en Python en définissant une fonction calcul(E) où E est l'expression en NPI, écrite sous forme d'une chaine de caractères dans laquelle chaque nombre ou symbole est séparé par un espace.