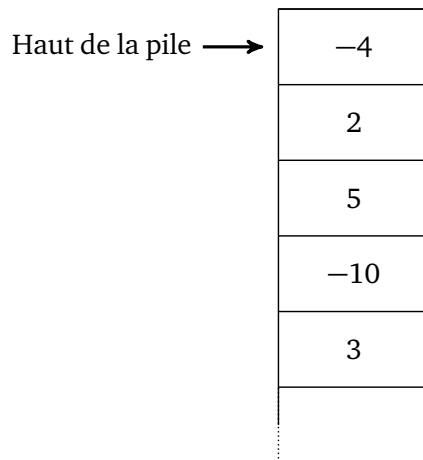


Piles

En Informatique, une pile est une structure de donnée fondée sur le principe « dernier arrivée, premier sorti »¹.

On peut les représenter par une suite d'éléments empilés les uns sur les autres. Seul le dernier élément ajouté est accessible : il est possible de l'enlever.



Les piles sont des exemples de structures de données gérées nativement par les microprocesseurs : elles correspondent des zones de mémoires dont le processeur ne retient que l'adresse du dernier élément. Lorsqu'il ajoute ou retire un élément à la pile, il a simplement à mettre à jour cette adresse.

Un exemple courant de pile est l'historique d'un navigateur web : chaque page visitée est ajoutée à la pile. Quand on clique sur le bouton « Afficher la page précédente », le dernier élément de la liste est retiré et le navigateur s'y rend. Les commandes Undo des éditeurs de texte en sont un autre exemple.

Les piles ne sont pas implémentées nativement en Python. J'ai donc créé une petite librairie pour manipuler ces objets : `pile.py`, dont vous recopierez le contenu dans votre fichier.

Programmer en terme de pile consiste à n'utiliser que les instructions suivantes

- `P = Pile()` crée une pile vide.
- `P.push(x)` permet d'ajouter l'élément `x` à la pile (empiler).
- `P.pop()` retire l'élément au sommet de la pile et le renvoie (dépiler). Si la pile est vide, `P.pop` émet une erreur. Une fois l'élément dépiler, si on ne l'a pas sauvegardé, il est définitivement perdu.
- `P.isempty()` renvoie **True** si la pile est vide, **False** si elle contient au moins un élément.

Exercice 1 — Manipulations de base

1. Implémenter une fonction `initialise(L)` qui prend en paramètre une liste `L` et qui renvoie une pile contenant les éléments de la liste `L` qui auront été empilé dans l'ordre de la liste.
2. Implémenter une fonction `peek(P)` qui prend en paramètre une pile `P` et qui renvoie l'élément qui est au sommet, sans que la pile `P` soit modifiée.
3. Implémenter une fonction `retourne(P)` qui prend en paramètre une pile `P` et qui renvoie une nouvelle pile avec les valeurs de `P` dans l'ordre inverse (sans que la pile `P` soit modifiée).

1. en anglais : LIFO « Last In, First Out »

4. Implémenter une fonction longueur (P) qui prend en paramètre une pile P et qui renvoie sa longueur, c'est-à-dire le nombre de valeurs empilées dans P (sans que la pile P soit modifiée).

Exercice 2 — Expression bien parenthésée

On considère les chaînes de caractères contenant des parenthèses ouvrantes et fermantes. Par exemple "()(())", "(()(())())" ou "()(())". Les deux premières sont correctement parenthésées, mais pas la dernière.

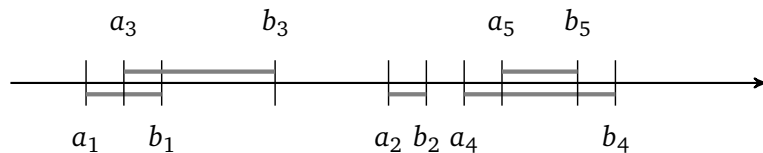
Proposer un algorithme utilisant des piles, qui accepte les expressions bien parenthésées et qui refuse les autres.

Idée de départ : lire la chaîne de caractère. Lorsqu'on rencontre une parenthèse ouvrante, l'empiler.

Exercice 3 — Union d'intervalles

On part d'une famille finie de segments $(I_k)_{1 \leq k \leq n}$ de \mathbb{R} qui s'écrivent

$$\forall k \in \llbracket 1 ; n \rrbracket, \quad I_k = [a_k ; b_k]$$



On cherche à écrire la réunion $\bigcup_{k=1}^n I_k$ sous la forme de la réunion minimale de segments disjoints (qu'on appelle les « composantes connexes » de $\bigcup_{k=1}^n I_k$).

. Par exemple avec

$$\begin{aligned} I_1 &= [0, 1] & I_2 &= [0.5, 3] & I_3 &= [2, 2.5] \\ I_4 &= [3.5, 3.75] & I_5 &= [4, 6] & I_6 &= [5, 7] & I_7 &= [6, 10] \end{aligned}$$

On trouvera

$$\bigcup_{k=1}^n I_k = [0 ; 3] \cup [3.5 ; 3.75] \cup [4, 10]$$

1. Les intervalles sont représentés par des listes à deux éléments $[a, b]$.
Créer deux piles `PileA` et `PileB` contenant respectivement les bornes inférieures et supérieures des intervalles, chaque pile étant triée, la plus petite valeur en premier.
On pourra utiliser la méthode `sort` de Python pour trier des listes.
2. Créer ensuite une pile `Bornes`, puis empiler le premier élément de `PileA`.
Par la suite
 - Déterminer le plus petit élément entre le dessus de `PileA` et le dessus de `PileB`.
 - Si cet élément provient de `PileA`, l'empiler.
 - Si cet élément provient de `PileB`, dépiler un élément de `Bornes`.
 - Déterminer le test à faire ensuite pour savoir si on est toujours à l'intérieur d'une composante connexe. Que faire sinon ?