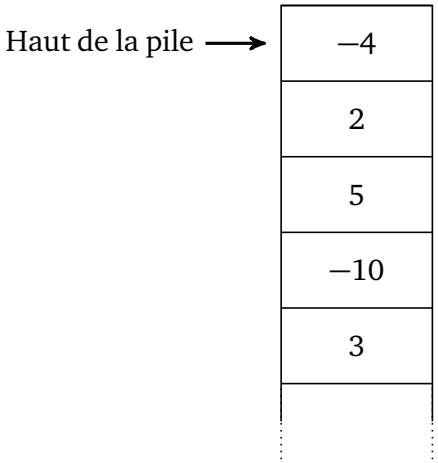
Piles

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. Seule l'élément du dessus est accessible : il est possible de l'enlever (auquel cas la valeur suivante est accessible) ou d'ajouter une valeur à la pile.



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 ajouter 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 n'utiliser que les instructions suivantes

- `P = Pile()` créé une pile vide.
- `P.push(x)` permet d'ajouter l'élément `x` à la pile (empiler).
- `P.pop()` retire le dernier élément de la pile et le renvoie (dépiler). Si la pile est vide, `P.pop` émet une erreur. Une fois l'élément dépilé, 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 ajouté dans l'ordre de la liste.
2. Implémenter une fonction `peek(P)` qui prend en paramètre une pile `P` et qui son renvoie dernier élément, 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.

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**.

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.

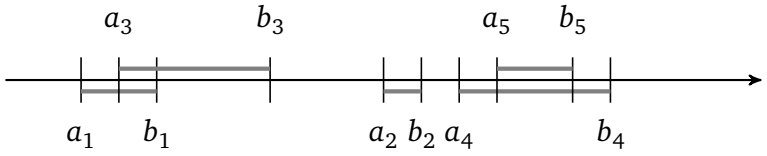
Proposer un algorithme utilisant des piles, qui accepte les expressions bien parenthésées et qui refuse les autres.

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 $(I_k)_{1\leqslant k\leqslant n}$ de $\mathbb{R}$ qui s'écrivent

$$\forall k \in [\![1 ; n]\!], \qquad I_k = [\, a_k \,;\, b_k \,]$$



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

$$I_1 = [0,1] \qquad I_2 = [0\,.\,5,3] \qquad I_3 = [2,2\,.\,5]$$
$$I_4 = [3\,.\,5,3\,.\,75] \qquad I_5 = [4,6] \qquad I_6 = [5,7] \qquad I_7 = [6,10]$$

On trouvera

$$\bigcup_{k=1}^{n} I_k = [\,0\,;\,3\,] \bigcup [3\,.\,5,3\,.\,75] \bigcup [4,10]$$

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 ?