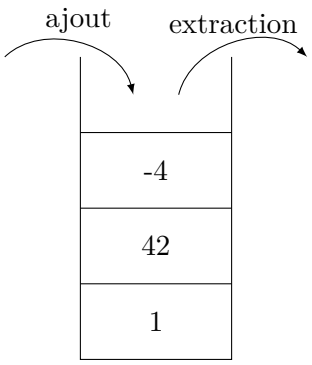
TP : piles
Informatique pour tous

Une **pile** (pensez à une pile d'assiettes) est une structure de donnée (c'est à dire contenant plusieurs valeurs) disposant des opérations suivantes:

- Ajout d'un élément au dessus de la pile.
- Extraction (c'est à dire suppression et renvoie de sa valeur) de l'élément du dessus de la pile.
- Test pour savoir si la pile est vide.

On remarque que l'élément supprimé est toujours celui qui a été ajouté en dernier (c'est pourquoi les piles sont aussi appelées structures FIFO: First In First Out).
Par exemple, à partir d'une pile vide et en rajoutant 1 et 27 puis en supprimant le dessus (c'est à dire 27), et enfin en ajoutant 42 puis -4, on obtient une pile contenant 1, 42, -4 qu'on peut représenter de la façon suivante:



On peut implémenter une pile en Python avec une liste `L`, où le dessus de la pile est le dernier élément de `L`: on ajoute un élément `e` avec `L.append(e)`, on extrait le dernier élément avec `L.pop()` et on peut tester si la pile est vide en regardant si `len(L)` est non nul.
Dans la suite, **on utilisera seulement ces opérations** sur les listes.
Par exemple, la pile ci-dessus est représentée par `L = [1, 42, -4]`.
Remarque: `L.pop()` donne une erreur si `L` est vide.

I Petites questions

1. Écrire une fonction `inverse` ayant une pile en argument (c'est à dire, ici, une liste) et renvoyant une pile contenant les mêmes éléments mais inversés (il suffit donc d'inverser les éléments de la liste).
2. En utilisant la question précédente, écrire une fonction `copie` renvoyant une copie d'une pile en argument (en conservant l'ordre).

II Expression bien parenthésée

On souhaite déterminer si une chaîne de caractères est bien parenthésée (c'est à dire si chaque parenthèse ouvrante est bien fermée par une parenthèse fermante).
Par exemple $(2 * (3 + 4))$ est bien parenthésé mais $(2 * 3 + 4))$ et $(2*)3 + 4)($ ne le sont pas.
On utilise une pile `P` initialement vide de la façon suivante, en parcourant la chaîne de caractères de gauche à droite:

- à chaque fois qu'une parenthèse ouvrante `"("` est rencontrée, on rajoute un élément à `P` (par exemple 0, peu importe la valeur de l'élément rajouté).
- à chaque fois qu'une parenthèse fermante `")"` est rencontrée, on extrait l'élément du dessus de la pile. Si ce n'est pas possible, on en déduit que le texte n'est pas correctement parenthésé.
- on ignore tous les autres caractères (autres que `"("` et `")"`)
- à la fin de la chaîne de caractères, on en déduit qu'elle est bien parenthésée seulement si la pile est vide (c'est à dire qu'il y a eu autant de parenthèses ouvrantes que fermantes).

1. Écrire une fonction `parenthesee` réalisant cet algorithme.

III Notation polonaise inversée

La notation polonaise inversée a été utilisée assez longtemps par les calculatrices. Elle consiste à écrire d'abord les opérandes, puis les opérateurs. Par exemple, $3 + 4 \times 5$ s'écrit $3\ 4\ 5\ \times\ +$ en notation polonaise inversée (à chaque fois que l'on croise un opérateur, on l'applique sur les deux dernières valeurs).
On peut calculer la valeur d'une telle expression avec une pile, en parcourant l'expression de gauche à droite:
À chaque fois que l'on trouve un opérateur (`'+'`, `'*'`, `'-'` ou `'/'`), on retire les deux valeurs du dessus de la pile puis on rajoute à la pile le résultat de l'opérateur appliqué sur ces deux valeurs.
À chaque fois que l'on trouve une valeur, on la met sur la pile.
Une fois l'expression parcourue en totalité, la pile contient un seul élément: la valeur totale de l'expression.

1. Écrire une fonction `npi` renvoyant la valeur d'une expression en notation polonaise inversée (qui est donnée sous forme de liste). Vérifier que `npi([3, 4, 5, '*', '+'])` donne 23.