

Exercice 1 : Implémentation objet

Écrire une classe `Stack` qui implémente la structure de pile vue en cours. Sauvegarder dans un module `Stack.py`. Celui-ci servira pour les exercices suivants.

Exercice 2 : Les piles c'est renversant

Écrire une fonction `reverse_with_stack` qui utilise une pile et

- en entrée prend un `str`;
- en sortie renvoie un `str` qui est la chaîne passée en paramètre, mais dans l'autre sens.

Exercice 3 : Expressions bien parenthésées

L'expression $(a + b \times (c + (a + b)^2))^3$ est bien parenthésée. $(a - b) \times c$ ne l'est pas.

1. Écrire une fonction `is_valid` qui utilise une pile et
 - en entrée prend in `str` qui est l'expression à tester.
 - en sortie renvoie un `bool` : `True` si elle est bien parenthésée, `False` sinon.
2. Proposer un ensemble de tests unitaires.

Exercice 4 : Égalité de deux piles

1. Écrire une fonction `are_equal` qui
 - en entrée prend 2 piles.
 - en sortie renvoie un `bool` : `True` si les deux piles ont les mêmes éléments (même nombre et même ordre), `False` sinon.

La fonction doit remettre les piles telles quelles.
2. Proposer un ensemble de tests unitaires.

Exercice 5 : Notation polonaise inversée

La Notation Polonaise Inverse (NPI), ou notation post-fixée, est une manière d'écrire les expressions mathématiques en se passant des parenthèses. Elle a été introduite par le mathématicien polonais Jan Lucasiewicz dans les années 1920.

Le principe de cette méthode est de placer chaque opérateur juste après ses deux opérandes.

Par souci de simplicité nous ne considérerons que des expressions mettant en jeu des entiers naturels.

L'expression $2 + 3$ devient en NPI $2\ 3\ +$.

- $2 + 6 - 1$ s'écrit $2\ 6\ +\ 1\ -$
- $5 \times 3 + 4$ s'écrit $5\ 3\ *\ 4\ +$
- $((1 + 2) \times 4) + 3$ s'écrit $1\ 2\ +\ 4\ *\ 3\ +$

Évaluer une expression post-fixée est facile. Pour cela il suffit de lire l'expression de gauche à droite et d'appliquer chaque opérateur aux deux opérandes qui le précèdent. Si l'opérateur n'est pas le dernier symbole on remplace le résultat intermédiaire dans l'expression et on recommence avec l'opérateur suivant.

On peut écrire un programme en Python qui utilise une pile `s` pour évaluer les expressions en NPI en suivant cet exemple

- on considère une chaîne de caractères : `c = '23 6 + 1 -'`;
- on la transforme en une liste : `lst = c.split()`;
- la valeur de `lst` est alors `['23', '6', '+', '1', '-']`;
- on constate que `'23'` et `'6'` ne sont pas des symboles d'opérateurs donc sont convertibles en `int` et on empile ces deux entiers dans `s`;
- le prochain élément de `lst` est `'+'` donc on dépile les deux entiers 6 et 23, on les ajoute, et on empile le résultat 29;
- on continue : on empile 1;
- on arrive à `'-'`, on dépile les 2 entiers 1 et 29 et on les soustrait (dans le bon ordre) et on empile le résultat final 28.

Écrire une fonction `mpi_compute` qui

- en entrée prend un `str` qui est une expression NPI;
- en sortie renvoie un `float` qui est l'évaluation de cette expression (`float` car l'expression peut comporter des divisions).