OBJECTIF BAC

11 Évaluer une expression avec une pile

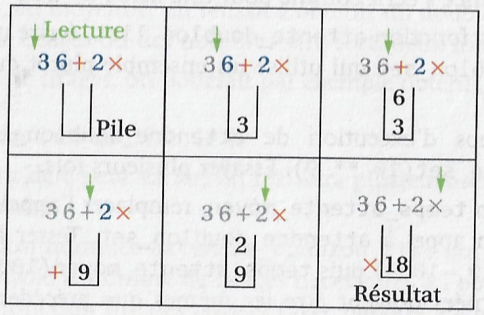
45 min

Nous allons voir comment une pile peut être utilisée en pratique pour évaluer un calcul en notation postfixe qui a la particularité de ne pas nécessiter de parenthèses...

Il existe plusieurs façons de noter les expressions arithmétiques, entre autres :
- la notation infixe (usuelle), utilisant des parenthèses, l'opérateur est indiqué **entre** les opérandes : $(3 + 6) \times 2$;
- la notation postfixe, qui ne nécessite pas de parenthèse, l'opérateur est mentionné **après** les opérandes : 3 6 + 2 ×.

Une expression en notation postfixe peut facilement être évaluée à l'aide d'une pile, en même temps qu'elle est analysée. Pour cela, on lit l'expression de gauche à droite (voir figure) :
- si c'est un nombre, on l'empile ;
- si c'est un opérateur, on l'applique aux deux nombres qui sont au sommet de la pile et on empile le résultat.

À la fin de la lecture de l'expression, si celle-ci était syntaxiquement correcte, la pile ne contient plus qu'un élément : le résultat.



a. Écrire une class `Stack` qui contient les opérations `is_empty`, `pop`, `push`, et lève une exception `EmptyStackError` si on essaie de faire `pop` sur une pile vide. Lors de l'affichage, une pile devra s'afficher verticalement, le dernier élément ajouté au sommet.

b. Écrire une fonction `calcul` qui prend en paramètre une pile (type `Stack` de la question a.) et une opération donnée sous forme d'un caractère : +, −, ...), dépile les deux éléments au sommet de la pile, leur applique l'opération et empile le résultat.

REMARQUE

Ce n'est pas strictement nécessaire, mais vous pouvez trouver utile d'importer le module `operator` qui contient les fonctions `add`, `sub`... (faire `import operator` puis `help(operator)` → FICHE 1).

c. Écrire une fonction nommée `evaluation_lst` qui prend en paramètre une liste contenant des nombres et des symboles arithmétiques (sous forme de caractères) et évalue le résultat. Pour l'instant, on ne fait pas de gestion d'erreur (on suppose que la liste contient des éléments corrects).

d. Dans la suite, nous allons avoir besoin d'un outil qui indique si une chaîne de caractères correspond à un nombre (`float`). Écrire la fonction `is_float` qui prend en paramètre une chaîne de caractères et indique en renvoyant un booléen si cette chaîne correspond à un nombre.

e. Ajouter la gestion des erreurs à la fonction `evaluation_lst` :
- si un élément n'est ni un nombre ni un symbole d'opération, lever une exception `InvalidExpressionError` ;
- si, au moment de faire une opération, il n'y a pas assez d'éléments dans la pile, lever une exception `InvalidExpressionError` ;
- si, à la fin du calcul, la pile ne contient pas exactement un élément (le résultat), lever une exception `InvalidExpressionError`.

On supposera toutefois que la liste passée à `evaluation_lst` ne contient pas autre chose que des chaînes de caractères ou des nombres.

f. À présent, écrire une fonction `evaluation` qui prend en paramètre une chaîne de caractères contenant l'expression postfixe, chaque élément étant séparé des autres par un espace, et évalue cette expression.

Tester cette fonction.

```
# Opérations simples
evaluation("3 5 *") -> 15
evaluation("3 5 * 2 +") -> 17
evaluation("8 1 1 + /") -> 4
evaluation("8 1 -") -> 7

# Prise en compte des nombres à virgule
evaluation("4 2.5 *") -> 10
evaluation("0.5 2 /") -> 0.25

# Exception InvalidExpressionError pour cause de pile vide
evaluation("*")
evaluation("1 1 + +")

# Exception InvalidExpressionError pour cause de pile
# non vide à la fin
evaluation("2 1 + 3 2 -")
evaluation("1 1 + ")

# Exception InvalidExpressionError pour cause de
# symbole inconnu
evaluation("15 4 %")
evaluation("BONJOUR")

# Le cas de plusieurs espaces consécutifs est-il géré ?
evaluation("1   1  +    ") -> 2 ?
```