

1. Contexte

La Notation Polonaise Inversée dite aussi **postfixe** permet d'écrire sans parenthèses ni règles de priorités une suite d'opérations arithmétiques qui impliquent des opérateurs et des opérandes.

L'expression $((3 + 15) / (5 - 2))$ s'écrit en NPI :

$$3 \ 15 \ + \ 5 \ 2 \ - \ /$$

Algorithme d'évaluation d'une expression

La structure de calcul en NPI utilise une **pile** de la manière suivante :

- quand on rencontre un **nombre**, on l'**empile**;
- quand on rencontre un **opérateur** (+, -, *, /), on **dépile** les deux nombres au sommet de la pile et on effectue l'opération, le résultat étant de nouveau **empilé**.

Si le calcul est correctement écrit, il y a toujours au moins deux nombres dans la pile quand on rencontre un opérateur, et il reste un seul et unique nombre dans cette pile quand l'expression a été entièrement lue.

2. Exemples d'application

 **À Faire 1** : En utilisant l'algorithme ci-dessus, évaluer l'expression postfixe : $1 \ 2 \ + \ 4 \ \times \ 3 \ +$



Pile d'exécution

 **À Faire 2** : En utilisant l'algorithme ci-dessus, évaluer l'expression postfixe : $8 \ 5 \ - \ 3 \ / \ \times \ 4$



Pile d'exécution

3. Implémentation en Python

Nous allons implémenter dans un module `npi`, les fonctions relatives au calcul des expressions.

 **À Faire 3** : Créer une fonction `evaluation` prenant en argument un calcul en NPI représenté sous la forme d'une chaîne de caractères dont les éléments (opérandes et opérateurs) sont séparés d'un espace, et qui renvoie le résultat du calcul.

Exemple d'utilisation :

```
>>> evaluation('7 3 * 11 + 8 /')
```

```
4
```

Remarque : on se limitera aux opérateurs classiques `+`, `-`, `*`, `/` mais rien n'empêche les plus courageux d'essayer de programmer les opérateurs `sqrt` et `**`.

 **À Faire 4** : Créer une fonction `verifier_infixe` prenant en argument une expression infixé représentée sous la forme d'une chaîne de caractères dont les éléments (parenthèses, opérandes et opérateurs) sont séparés d'un espace, et qui renvoie `True` si l'expression est correctement parenthésée, `False` sinon.

Exemple d'utilisation :

```
>>> verifier_infixe('3 + 2')
```

```
True
```

```
>>> verifier_infixe('(3 + 2)')
```

```
True
```

```
>>> verifier_infixe('(3 + 2')
```

```
False
```

 **À Faire 5** : Créer une fonction `verifier_postfixe` prenant en argument une expression postfixé représentée sous la forme d'une chaîne de caractères dont les éléments (parenthèses, opérandes et opérateurs) sont séparés d'un espace, et qui renvoie `True` si l'expression est correctement exprimée, `False` sinon.

Exemple d'utilisation :

```
>>> verifier_postfixe('3 2 +')
```

```
True
```

```
>>> verifier_postfixe('3 2 + 5 *')
```

```
True
```

```
>>> verifier_postfixe('3 2')
```

```
False
```

```
>>> verifier_postfixe('8 5 - 3 / * 4')
```

```
False
```