

TP 12

Dictionnaires

Exercice 1 – Évaluation d'une expression parenthésée

Dans un éditeur l'ajout d'une parenthèse fermante surnuméraire est signalé (mise en surbrillance). La construction d'un vérificateur de parenthèses repose sur l'utilisation de piles.

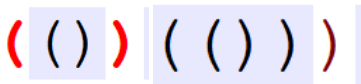


FIGURE 1 – Éditeur généraliste Notepad++



FIGURE 2 – Éditeur Python Pyzo

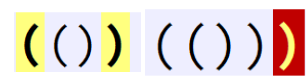


FIGURE 3 – Éditeur LaTeX TeXstudio

FIGURE 4 – Expressions bien et mal parenthésées dans différents éditeurs

Nous allons considérer 3 types de ponctuations symétriques :

- les parenthèses (et) ;
- les crochets [et] ;
- les accolades { et } ;

On souhaite savoir reconnaître 4 cas différents :

- fermeture sans ouverture préalable ;
- fermeture par le mauvais type de parenthèse ;
- une ouverture sans fermeture ;
- l'expression est bien parenthésée.

Nous allons commencer par nous occuper dans un premier temps uniquement d'un seul type de ponctuations : les parenthèses.

Question 1 Écrire une fonction `verifPar0(texte)` qui prend en entrée une chaîne de caractères et renvoie :

- le nombre d'ouvertures non-fermées ;
- -1 dès qu'on rencontre une fermeture sans ouverture préalable.

Une pile sera initialisée au début de la fonction : on ajoutera l'ouverture dans la pile qu'on dépilera lorsqu'il y a fermeture de la parenthèse avec le même type.

Question 2 Tester la fonction avec `"(()())"`, `"((())"` et `"((())())"`.

Nous ajoutons ici les autres types de ponctuations : il faut donc vérifier si la fermeture correspond bien à l'ouverture dans ce cas-là.

Question 3

Écrire une fonction `verifPar1(texte)` qui prend en entrée une chaîne remplie de caractères et renvoie :

- le nombre d'ouvertures non-fermées ;
- -1 dès qu'on rencontre une fermeture sans ouverture préalable ;
- c dès qu'on rencontre une erreur de type au caractère c.

Question 4 Tester la fonction avec `"{[()]}"`, `"{[(())}"` et `"{[(())}]"`.

Exercice 2 – Évaluation d'une expression postfixée

Jean-Pierre Becirspahic

La notation postfixée d'une expression algébrique consiste à placer les opérateurs après son ou ses opérandes. Par exemple, l'addition de a et de b sera écrite $a \ b \ +$ en notation postfixée, la racine carrée de a sera écrite $a \ \sqrt{}$.

L'intérêt majeur de cette notation est qu'une expression postfixe n'est jamais ambiguë : alors que l'expression infixe $1 + 2 \times 3$ peut avoir deux significations : $(1 + 2) \times 3$ ou $1 + (2 \times 3)$, ce n'est jamais le cas d'une expression postfixe, ce qui rend l'usage des parenthèses superflu : $1 \ 2 \ + \ 3 \ \times$ ne peut être compris que de cette façon : $(1 \ 2 \ +) \ 3 \ \times$ et $1 \ 2 \ 3 \ \times \ +$ de cette façon : $1 \ (2 \ 3 \ \times) \ +$. Nous allons montrer comment, à l'aide d'une pile, on peut évaluer une expression algébrique postfixe. Dans cet exercice, les expressions algébriques seront représentées par les listes qui pourront contenir des nombres (de type `int` ou `float`) ou des chaînes de caractères représentant des opérateurs unaires ou binaires (comme par exemple `sqrt` ou `+`). Par exemple, l'expression $\frac{1+2\sqrt{3}}{4}$ sera représentée par la liste `[1, 2, 3, 'sqrt', '*', '+', 4, '/']`. On suppose donné deux dictionnaires répertoriant pour l'un les opérateurs unaires, pour l'autre les opérateurs binaires, et qui associent à chaque chaîne de caractère la fonction correspondante. On peut par exemple définir ces deux dictionnaires à l'aide du script suivant, et les compléter en suivant le même modèle.

```
from numpy import sqrt, exp, log
op_uni = {'sqrt': sqrt, 'exp': exp, 'ln': log}
def add(x, y):
    return x + y

def sous(x, y):
    return x - y

def mult(x, y):
    return x * y

def div(x, y):
    return x / y

op_bin = {'+': add, '-': sous, '*': mult, '/': div}
```

L'évaluation d'une expression postfixe consiste à utiliser une pile initialement vide et à parcourir les éléments de la liste représentant l'expression à évaluer en appliquant les règles suivantes :

- si l'élément est un nombre, il est empilé ;
- si l'élément est un opérateur unaire, le sommet de la pile est dépilé, l'opérateur lui est appliqué et le résultat ré-empilé ;
- si l'élément est un opérateur binaire, deux éléments de la pile sont dépilés, l'opérateur leur est appliqué et le résultat ré-empilé.

Si l'expression postfixe est correcte sur le plan syntaxique (et mathématique), à la fin du traitement de la liste la pile ne contient plus qu'un seul élément égal au résultat de l'évaluation de l'expression. On suppose donnés les deux dictionnaires `op_uni` et `op_bin`.

Question 5 Rédiger une fonction qui évalue une expression postfixe donnée sous forme de liste `evalue(lst:liste) -> float`. Dans un premier temps, on pourra supposer que l'expression est syntaxiquement correcte.

Question 6 Rédiger une seconde fonction d'évaluation qui détecte les erreurs de syntaxe. On l'a notera `evalue2(lst:liste)`.