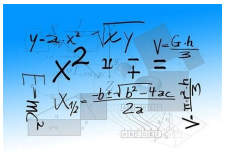

Exercice 25 - Expressions arithmétiques

Indication pour la suite de l'exercice : les chaînes de caractères se manipulent pratiquement comme des listes, en particulier :

chaine = "" → *chaîne vide*
chaine[i] → *caractère de rang i dans chaine*
len(ch) → *longueur de la chaîne de caractères*
car in chaine → *permet de tester si le caractère car apparaît dans la chaine*
ch[deb : fin] → *slicing : extrait la sous-chaîne entre les indices deb et fin exclu*
int(x) → *convertit la chaiune x en entier* *int("367")* → *l'entier 367*
x.isdigit() → *return Trus si x n'est constitué que de chiffres, False sinon*

1) Somme

On considère des expressions sous forme de chaînes de caractères décrivant des sommes de nombres entiers naturels. Ces chaînes ne comprennent que des **chiffres** et le **symbole '+'**.

Ces expressions représentent donc soit un entier naturel, soit une somme d'entiers naturels.

Exemple d'expression : "123+15+1"

Ecrire une **fonction** *est_valide* qui retourne *True* si la chaîne est valide, c'est-à-dire :

- qu'elle ne contient que des caractères autorisés (**chiffres** et "**+**")
- et que les opérateurs "+" sont bien placés : ni au **début**, ni à la **fin**, ni en **doublon**

La fonction retourne *False* sinon.

Exemples d'expressions bien formées : *"123", "123+5", "123+15", "123+15+1"*

Les expressions suivantes sont considérées comme mal formées : "", "12+", "+12", "12++12", "12+x+5"

2) Evaluation

Ecrire une fonction *evalue* qui calcule et retourne la valeur d'une expression arithmétique passée en paramètre. On suppose que l'expression est bien formée.

Exemple : *evalue("123+15+1") retourne 139*
Indications : *on peut utiliser la fonction split*
La méthode split(car) renvoie une liste de tous les mots de la chaîne, en utilisant le caractère car comme séparateur
si ch = "Hydrogene-1.0079-H-1"
ch.split("-") retourne la liste ["Hydrogene", "1.0079", "H", "1"]

3) Liste d'additions

On dispose d'une liste d'expressions arithmétiques sous forme de chaînes de caractères telles que décrites précédemment.

Ecrire un programme qui calcule et affiche la plus grande de ces expressions.

Exemple : si on dispose de la liste des expressions ["123+15", "123+15+1", "123", "123+5"]
le programme affiche : La valeur la plus grande est : 139

4) Expressions arithmétiques

On souhaite généraliser le problème en utilisant les autres opérateurs arithmétiques '-', '*' et '/'.

Ecrire une fonction *calcule* qui reçoit en paramètre deux opérandes et un opérateur et calcule le résultat de l'opération.

calcule(546, 23, '-') retourne la valeur 523

5) Evaluation

On suppose qu'on dispose d'une expression arithmétique sous la forme d'une alternance d'opérandes et d'opérateurs :

Expression = [46, '+', 20, '-', 10, '', 2]*

Ecrire une fonction *evaluation* qui calcule la valeur correspondante d'une expression fournie en paramètre sous forme d'une liste. On suppose qu'on réalisera les opérations au fur et à mesure et qu'on ne tiendra pas compte de la priorité des opérateurs.

Ainsi l'expression *[46, '+', 20, '-', 10, '*', 2]* correspondrait à (46 + 20 -10) * 2

Exemple : evaluation([46, '+', 20, '-', 10, '', 2]) retourne 112*

On utilisera bien entendu la fonction précédente.

6) Expression postfixée

Pour tenir compte de la priorité des opérateurs, on utilise la notation postfixée (ou notation polonaise) d'une expression. L'opérateur n'est pas placé entre les deux opérantes mais après ses opérandes.

Ainsi l'expression 46 20 + 10 2 * - correspond à 46 + 20 – 10 * 2

Pour calculer une expression arithmétique postfixée représentée sous forme de liste, il suffit :

- De créer une liste *pile* avec les entiers successifs que l'on rencontre ;
- lorsque l'on détecte un opérateur, il faut :
  - récupérer les 2 derniers entiers ajoutés dans la *pile*
  - leur appliquer l'opérateur
  - ajouter le résultat dans la *pile*
- et ainsi jusqu'à ce que tous les éléments de la liste de départ soit traité

Ecrire une fonction qui calcule et retourne le résultat d'une expression arithmétique postfixée représentée sous forme de liste.

7) Vérification d'erreur

On veut détecter d'éventuelles erreurs dans les formules, par exemple une division par 0.

Compléter la fonction précédente pour que si elle détecte les erreurs telles que :

- une division par 0
- un nombre d'opérateurs en trop ou insuffisant

Dans ce cas, la fonction retourne "ERR".