Récursivité

1. Recherche d'un fichier dans une arborescence

1.a. Recherche récursive

Les fichiers sont répertoriés dans une structure de données de type arbre (ou arborescence). Les nœuds de cet arbre sont les dossiers (appelés aussi répertoires ou *directory* en anglais). Les feuilles sont les fichiers. La récursivité est un moyen simple de parcourir un arbre en partant de la racine. On se propose d'écrire une fonction qui effectue la recherche d'un fichier dans une arborescence.

L'entête de cette fonction est :

```
recherche_recursive(repertoire,chemin,fichier)
```

où `repertoire` est le nom du répertoire (ou dossier) à explorer, `chemin` le chemin sous la forme d'une chaîne de caractères (par exemple `dossierA/dossierB/fichier` et `fichier` le nom du fichier (avec son extension).

La fonction doit afficher le chemin des fichiers dont le nom est donné.

Pour écrire cette fonction, on aura besoin du module `os` et des fonctions suivantes :

- ▷ `os.chdir(repertoire)` qui permet de se déplacer dans un répertoire appartenant au répertoire courant.
- ▷ `os.chdir("../")` qui permet de remonter au répertoire parent.
- ▷ `os.listdir(os.getcwd)`, qui renvoie une liste des répertoires et fichiers contenus dans le répertoire courant.
- ▷ `os.path.isdir(nom)`, qui permet de savoir si `nom` est le nom d'un répertoire (et non pas d'un fichier).

[1] Écrire la fonction `recherche_recursive` et la tester, par exemple :

```
recherche_recursive("C:/Anaconda","","pylab.py")
```

[2] Écrire une autre fonction, qui fait la recherche d'un fichier et fournit le résultat sous la forme d'une liste de chemins.

1.b. Solution

```
import os

def recherche_recursive(rep,chemin,fichier):
    chemin += "%s/"%rep
    os.chdir(rep)
    contenu = os.listdir(os.getcwd())
    for c in contenu:
        if os.path.isdir(c):
            recherche_recursive(c,chemin,fichier)
        else:
            if c==fichier:
                chemin = "%s/%s"%(chemin,c)
                print(chemin)
    os.chdir("../")
```

```
def recherche_recursive(rep,chemin,fichier,liste):
    chemin += "%s/"%rep
    os.chdir(rep)
    contenu = os.listdir(os.getcwd())
    for c in contenu:
        if os.path.isdir(c):
            recherche_recursive(c,chemin,fichier,liste)
        else:
            if c==fichier:
                chemin = "%s/%s"%(chemin,c)
                liste.append(chemin)
    os.chdir("../")

liste = []
recherche_recursive("C:/Anaconda","","pylab.py",liste)
print(liste)
```

2. Évaluation d'une expression mathématique

2.a. Objectif

On se propose d'écrire une fonction qui fait l'évaluation d'une expression mathématique simple, fournie sous la forme d'une chaîne de caractères. Cette expression peut contenir des nombres (éventuellement avec un point), les opérateurs * (multiplication) et + (addition). Elle peut aussi contenir des expressions entre parenthèses. Voici un exemple d'expression :

```
2+5*3+2.5*(1+3*6)+7
```

La fonction `eval` permet d'évaluer une expression fournie sous la forme d'une chaîne de caractères. L'objectif est donc d'écrire une fonction faisant la même opération.

2.b. Algorithme

L'expression est une chaîne de caractères stockée dans `expression`. On parcourt `expression` caractère par caratère, en commençant par l'indice `debut`, qui vaut 0 lors du premier appel. On génère une liste, nommée `instructions`, contenant les éléments suivants :

- ▷ `['nombre',valeur]` pour un nombre.
- ▷ `['operateur','*']` > pour l'opération de multiplication.
- ▷ `['operateur','+']` > pour l'opération d'addition.

Le parcours de `expression` s'arrête soit lorsqu'on parvient à la fin de la chaîne, soit lorsqu'on rencontre le caractère de parenthèse fermante.

La seconde étape du traitement consiste à parcourir la liste `instructions` afin d'accomplir d'abord toutes les multiplications (qui sont prioritaires) puis d'accomplir toutes les additions. La réalisation d'une opération (addition ou multiplication) consiste à remplacer les trois éléments (nombre de gauche, opérateur et nombre de droite) par un seul élément de type nombre contenant le résultat de l'opération. Pour effectuer ce remplacement, on pourra procéder par concaténation de trois listes.

À la fin de ce traitement, il ne reste qu'un nombre qui constitue le résultat de l'évaluation.

Lorsque le caractère de parenthèse ouvrante est rencontré, on fait un appel récursif qui consiste à traiter l'expression en débutant au caractère qui suit la parenthèse ouvrante. Le résultat de l'évaluation de cette sous-expression est ensuite insérée dans `instructions`, sous la forme `['nombre',resultat]` puis le parcours de `expression` continue à partir de l'indice pointant la fin de cette sous-expression.

Reprenons l'exemple précédent. Dans un premier temps, l'expression est parcourue jusqu'à la parenthèse ouvrante et la liste d'instructions suivante est générée :

```
['nombre',2]
['operateur','+']
['nombre',5]
['operateur','*']
['nombre',3]
['operateur','+']
['nombre',2.5]
['operateur','*']
```

L'appel récursif de la fonction d'évaluation traite la sous-expression `1+3*6)+7` jusqu'à la parenthèse fermante, ce qui a pour effet de générer la liste d'instructions suivante :

```
['nombre',1]
['operateur','+']
['nombre',3]
['operateur','*']
['nombre',6]
```

Cette liste d'instructions est ensuite évaluée. On commence par évaluer les multiplications, ce qui conduit à :

```
['nombre',1]
['operateur','+']
['nombre',18]
```

puis les additions sont évaluées, ce qui conduit à :

```
['nombre',19]
```

La fonction d'évaluation retourne la valeur (ici 19). La fonction qui traite la première liste d'instructions reprend son cours avec la liste :

```
['nombre',2]
['operateur','+']
['nombre',5]
['operateur','*']
['nombre',3]
['operateur','+']
['nombre',2.5]
['operateur','*']
['nombre',19]
```

et elle parcourt l'expression à partir du caractère suivant la parenthèse fermante, ce qui l'amène finalement à la liste :

```
['nombre',2]
['operateur','+']
['nombre',5]
['operateur','*']
['nombre',3]
['operateur','+']
['nombre',2.5]
['operateur','*']
['nombre',19]
['operateur',+]
['nombre',7]
```

Pour finir, l'évaluation de ces instructions (multiplications puis additions) conduit au résultat final.

2.c. Implémentation

Voici le prototype de la fonction qui évalue l'expression :

```
def calcul(expression,debut=0):
    # code à écrire
    return (resultat,indice)
```

`debut` indique l'indice du caractère où l'analyse doit commencer. Pour appeler la fonction, on écrira `calcul(expression)`, ce qui conduit à `debut=0` (valeur par défaut).

Cette fonction renvoie un doublet `(resultat,indice)`, où `resultat` est le résultat de l'évaluation, c'est-à-dire un nombre, et `indice` l'indice de la chaîne de caractères où il faut poursuivre le parcours. Dans l'exemple précédent, lorsque l'évaluation de la sous-expression entre parenthèses est terminée, l'indice où il faut poursuivre pointe le caractère $\star$ qui suit la parenthèse fermante.

La fonction `calcul` comporte un appel récursif à elle-même après la rencontre d'une parenthèse ouvrante.

[3] Écrire un première version de la fonction `calcul`, qui se limite à traiter une expression sans parenthèses et qui génère simplement la liste d'instructions. Tester cette fonction.

[4] Compléter le code de la fonction `calcul` afin qu'elle fasse l'évaluation de la liste d'instructions, d'abord les multiplications, puis les additions.

[5] Compléter le code de la fonction afin qu'elle traite la rencontre d'une parenthèse ouvrante et déclenche alors l'appel récursif. Après cet appel, il faut insérer le résultat renvoyé par la fonction `calcul` dans la liste d'instructions puis poursuivre le parcours de l'expression à l'indice renvoyé par la fonction `calcul`.

2.d. Solution

```
def calcul(expression,debut=0):
    print(expression[debut:])
    chiffres = ['0','1','2','3','4','5','6','7','8','9','.']
    operateurs = ['*','+']
    instructions = []
    nombre = False
    chaine_nombre = ''
    i = debut
    N = len(expression)
    boucle = True
    while i < N and boucle:
        c = expression[i]
        if c in chiffres:
            if nombre==False:
                nombre = True
                chaine_nombre = c
                if i==N-1:
                    valeur = float(chaine_nombre)
                    instructions.append(['nombre',valeur])
            else:
                chaine_nombre += c
        else:
            if nombre:
                nombre = False
```

```
            valeur = float(chaine_nombre)
            instructions.append(['nombre',valeur])
        if c in operateurs:
            instructions.append(['operateur',c])
        elif c==')':
            boucle = False
            i -= 1
        elif c=='(':
            i += 1
            resultat = calcul(expression,i)
            instructions.append(['nombre',resultat[0]])
            i = resultat[1]
        else:
            print("Erreur de syntaxe")
            return
    i += 1
indice = i
i = 0
# traitement des multiplications
while i < len(instructions):
    ins = instructions[i]
    if ins == ["operateur","*"]:
        ins1 = instructions[i-1]
        ins2 = instructions[i+1]
        if ins1[0]!='nombre' or ins2[0]!='nombre':
            print("Erreur de syntaxe")
            return
        x1 = ins1[1]
        x2 = ins2[1]
        resultat = x1*x2
        n = len(instructions)
        instructions = instructions[0:i-1]+[["nombre",resultat]]+instructions[i+2:n]
    else:
        i += 1
# traitement des additions
i = 0
while i < len(instructions):
    ins = instructions[i]
    if ins == ["operateur","+"]:
        ins1 = instructions[i-1]
        ins2 = instructions[i+1]
        if ins1[0]!='nombre' or ins2[0]!='nombre':
            print("Erreur de syntaxe")
            return
        x1 = ins1[1]
        x2 = ins2[1]
        resultat = x1+x2
        n = len(instructions)
        instructions = instructions[0:i-1]+[["nombre",resultat]]+instructions[i+2:n]
    else:
        i += 1
if len(instructions)!=1:
    print("Erreur de syntaxe")
    return
if instructions[0][0]!='nombre':
    print("Erreur de syntaxe")
    return
resultat = instructions[0][1]
```

```
    return (resultat,indice)


e = "2+5*5+2*10*(2+3*5*(2+1))+(8+2)*7"


print(calcul(e))
--> (1037.0, 32)
```