Interrogation (1h50)

(Calculatrice non autorisée)

Exercice 1 *(10 points)*

Cet exercice composé de deux parties A et B, porte sur les structures de données.

Partie A : Expression correctement parenthésée

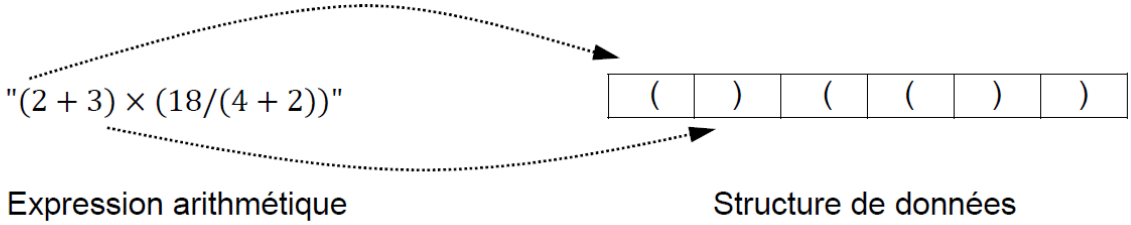
On veut déterminer si une expression arithmétique est correctement parenthésée.

Pour chaque parenthèse fermante ")" correspond une parenthèse précédemment ouverte "(".

Exemples :

- L'expression arithmétique "$(2+3)\times(18/(4+2))$" est correctement parenthésée.
- L'expression arithmétique "$(2+3)\times(18/(4+2$" est non correctement parenthésée.

Pour simplifier les expressions arithmétiques, on enregistre, dans une structure de données, uniquement les parenthèses dans leur ordre d'apparition. On appelle expression simplifiée cette structure.



1. Indiquer si la phrase « les éléments sont maintenant retirés (pour être lus) de cette structure de données dans le même ordre qu'ils y ont été ajoutés lors de l'enregistrement » décrit le comportement d'une file ou d'une pile. Justifier.

Pour vérifier le parenthésage, on peut utiliser une variable `controleur` qui :

- est un nombre entier égal à 0 en début d'analyse de l'expression simplifiée ;
- augmente de 1 si l'on rencontre une parenthèse ouvrante "(" ;
- diminue de 1 si l'on rencontre une parenthèse fermante ")".

Exemple : On considère l' expression simplifiée A : "( )(( ))"
Lors de l'analyse de l'expression A, `controleur` (initialement égal à 0) prend successivement pour valeur 1, 0, 1, 2, 1, 0. Le parenthésage est correct.

2. Écrire, pour chacune des 2 expressions simplifiées B et C suivantes, les valeurs successives prises par la variable `controleur` lors de leur analyse.
   Expression simplifiée B : " ((( )( )"
   Expression simplifiée C : "(( ))("

3. L'expression simplifiée B précédente est mal parenthésée (parenthèses fermantes manquantes) car le `controleur` est différent de zéro en fin d'analyse. L'expression simplifiée C précédente est également mal parenthésée (parenthèse fermante sans parenthèse ouvrante) car le `controleur` prend une valeur négative pendant l'analyse.
Recopier et compléter uniquement les lignes 13 et 16 du code ci-dessous pour que la fonction `parenthesage_correct` réponde à sa description.

```
1 def parenthesage_correct(expression):
2    ''' fonction retournant True si l'expression arithmétique
3    simplifiée (str) est correctement parenthésée, False
4    sinon.
5    Condition: expression ne contient que des parenthèses
6    ouvrantes et fermantes '''
7    controleur = 0
8    for parenthese in expression: #pour chaque parenthèse
9       if parenthese == '(':
10         controleur = controleur + 1
11      else:# parenthese == ')'
12         controleur = controleur - 1
13         if controleur ...  : # test 1 (à recopier et compléter)
14            #parenthèse fermante sans parenthèse ouvrante
15            return False
16    if controleur ... : # test 2  (à recopier et compléter)
17       return True #le parenthésage est correct
18    else:
19       return False #parenthèse(s) fermante(s) manquante(s)
```

Partie B : Texte correctement balisé

On peut faire l'analogie entre le texte simplifié des fichiers HTML (uniquement constitué de balises ouvrantes `<nom>` et fermantes `</nom>`) et les expressions parenthésées :

Par exemple, l'expression HTML simplifiée :
`"<p><strong><em></em></strong></p>"` est correctement balisée.

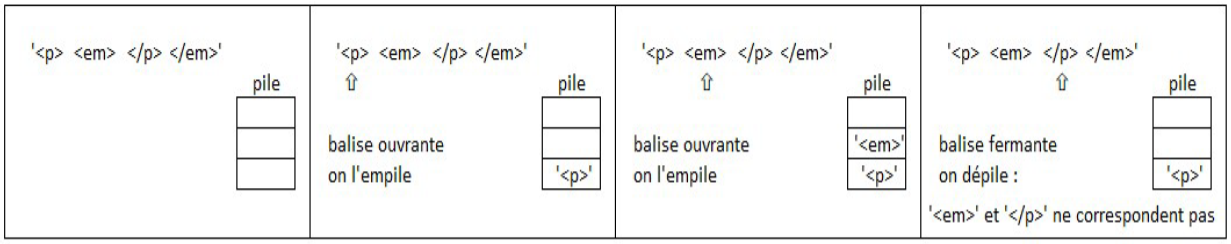
On ne tiendra pas compte dans cette partie des balises ne comportant pas de fermeture comme
 ou <img …>.

Afin de vérifier qu'une expression HTML simplifiée est correctement balisée, on peut utiliser une pile (initialement vide) selon l'algorithme suivant :

On parcourt successivement chaque balise de l'expression :

- lorsque l'on rencontre une balise ouvrante, on l'empile ;
- lorsque l'on rencontre une balise fermante :
  - si la pile est vide, alors l'analyse s'arrête : le balisage est incorrect ,
  - sinon, on dépile et on vérifie que les deux balises (la balise fermante rencontrée et la balise ouvrante dépilée) correspondent (c'est-à-dire ont le même nom) si ce n'est pas le cas, l'analyse s'arrête (balisage incorrect).

Exemple : État de la pile lors du déroulement de cet algorithme pour l'expression simplifiée `"<p><em></p></em>"` qui n'est pas correctement balisée.



État de la pile lors du déroulement de l'algorithme

4. Cette question traite de l'état de la pile lors du déroulement de l'algorithme.

a. Représenter la pile à chaque étape du déroulement de cet algorithme pour l'expression `"<p><em></em></p>"` (balisage correct).

b. Indiquer quelle condition simple (sur le contenu de la pile) permet alors de dire que le balisage est correct lorsque toute l'expression HTML simplifiée a été entièrement parcourue, sans que l'analyse ne s'arrête.

5. Une expression HTML correctement balisée contient 12 balises.
Indiquer le nombre d'éléments que pourrait contenir au maximum la pile lors de son analyse.

Exercice 2 *(10 points)*

Cet exercice porte sur les structures de données (files et la programmation objet en langage python)

Un supermarché met en place un système de passage automatique en caisse.
Un client scanne les articles à l'aide d'un scanner de code-barres au fur et à mesure qu'il les ajoute dans son panier.
Les articles s'enregistrent alors dans une structure de données.

La structure de données utilisée est une file définie par la classe `Panier`, avec les primitives habituelles sur la structure de file.
Pour faciliter la lecture, le code de la classe `Panier` n'est pas écrit.

```
class Panier():
  def __init__(self):
      """Initialise la file comme une file vide."""

  def est_vide(self):
      """Renvoie True si la file est vide, False sinon."""

  def enfiler(self, e):
      """Ajoute l'élément e en dernière position de la file,
    ne renvoie rien."""

  def defiler(self):
      """Retire le premier élément de la file et le renvoie."""
```

Le panier d'un client sera représenté par une file contenant les articles scannés.
Les articles sont représentés par des tuples `(code_barre, designation, prix, horaire_scan)` où

- `code_barre` est un nombre entier identifiant l'article ;
- `designation` est une chaine de caractères qui pourra être affichée sur le ticket de caisse ;
- `prix` est un nombre décimal donnant le prix d'une unité de cet article ;
- `horaire_scan` est un nombre entier de secondes permettant de connaitre l'heure où l'article a été scanné.

1. On souhaite ajouter un article dont le tuple est le suivant `(31002, "café noir", 1.50, 50525)`.
   Ecrire le code utilisant une des quatre méthodes ci-dessus permettant d'ajouter l'article à l'objet de classe `Panier` appelé `panier1`.

2. On souhaite définir une **méthode** `remplir(panier_temp)` dans la classe `Panier` permettant de remplir la file avec tout le contenu d'un autre panier `panier_temp` qui est un objet de type `Panier`.

Recopier et compléter le code de la méthode `remplir` en remplaçant chaque `.....` par la primitive de file qui convient.

```
def remplir(self, panier_temp):
    while not panier_temp.....:
        article = panier_temp.....
        self.....(article)
```

3. Pour que le client puisse connaitre à tout moment le montant de son panier, on souhaite ajouter une **méthode** `prix_total()` à la classe `Panier` qui renvoie la somme des prix de tous les articles présents dans le panier.
   Ecrire le code de la méthode `prix_total`. **Attention, après l'appel de cette méthode, le panier devra toujours contenir ses articles**.

4. Le magasin souhaite connaitre pour chaque client la durée des achats. Cette durée sera obtenue en faisant la différence entre le champ `horaire_scan` du dernier article scanné et le champ `horaire_scan` du premier article scanné dans le panier du client. Un panier vide renverra une durée égale à zéro. On pourra accepter que le panier soit vide après l'appel de cette méthode.
   Ecrire une **méthode** `duree_courses` de la classe `Panier` qui renvoie cette durée.