5.4 Implémenter une file avec Python

NSI terminale - JB Duthoit

5.4.1 Implémenter une file avec un tableau

On veut ici implémenter une file avec un tableau d'une capacité limitée.
On représente la file par un tuple `tableau,taille`, avec `tableau` un tableau et `taille` un tableau contenant une seule valeur, correspondant à la taille de la file.
Pour ajouter `elt` à la file, il suffit d'ajouter `elt` à l'indice `taille` (sauf si elle est pleine...)



Exercice 5.20

1. Créer la fonction `creer_file(c)` qui renvoie un tuple :

```
>>> file = creer_file(5)
>>> file
([None, None, None, None, None], [0])
```

2. Créer la fonction `enfiler(file,elt)` qui enfile `elt` à `file`

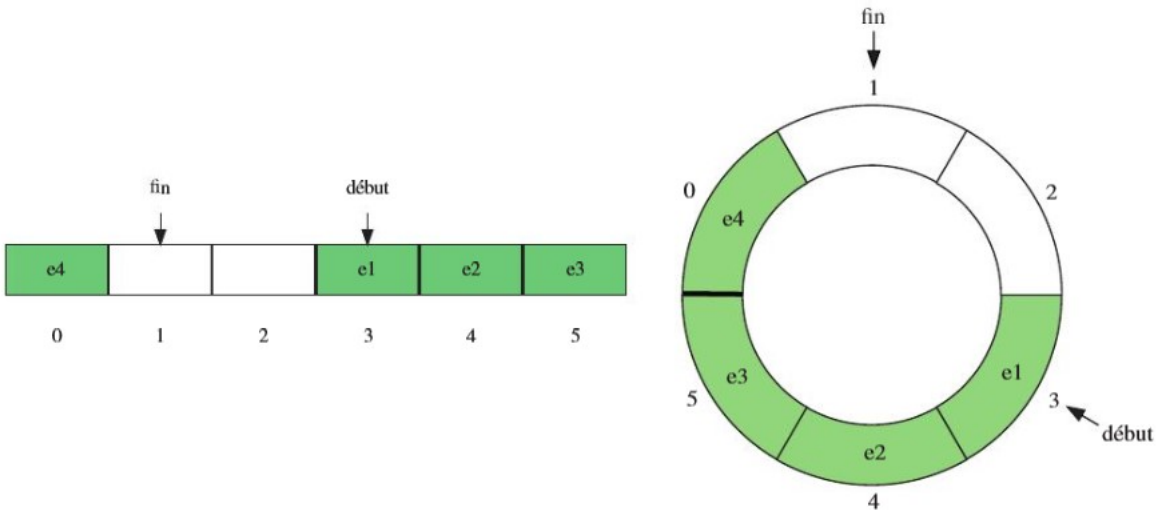
```
>>> enfiler(file,'Alice')
>>> enfiler(file,'Bob')
>>> enfiler(file,'Charlie')
>>> file
(['Alice', 'Bob', 'Charlie', None, None], [3])
```

3. Créer la fonction `defiler(file)` qui défile `file` :

```
>>> defiler(file)
'Alice'
>>> defiler(file)
'Bob'
>>> file
(['Charlie', None, None, None, None], [0])
```

5.4.2 Implémenter une file avec un tableau circulaire

L'implémentation précédente est couteuse car pour défiler un élément, il faut recopier tous les éléments restants.
On considère ici un tableau comme un anneau dont la file occupe les éléments de l'indice `debut` à l'indice `fin - 1`.



Principe du tableau circulaire

Exercice 5.21

1. Créer la fonction `creer_file(c)` qui renvoie un tuple :

```
>>> file = creer_file(5)
>>> file
([None, None, None, None, None], [0],[1])
```

2. Créer la fonction `enfiler(file,elt)` qui enfile `elt` à `file`

```
>>> enfiler(file,'Alice')
>>> enfiler(file,'Bob')
>>> enfiler(file,'Charlie')
>>> file
(['Alice', 'Bob', 'Charlie', None, None], [0],[3])
```

3. Créer la fonction `defiler(file)` qui défile `file` :

```
>>> defiler(file)
'Alice'
>>> defiler(file)
'Bob'
>>> file
([None, None, 'Bob', None, None], [2],[3])
```

5.4.3 Implémenter une file avec les tableaux dynamiques python

Voici comment implémenter facilement une file avec les méthodes des tableaux dynamiques python (type `list`) :

```
ma_file = []
# Pour créer une file vide
ma_file.append(ma_valeur)
```

```
# permet d'enfiler une valeur
ma_file.pop(0)
# permet de défiler une valeur
len(ma_file)
# renvoie la longueur de ma_file
```

Exercice 5.22
Implémenter une file avec un tableau dynamique. On créera pour cela la fonction `creer_file()`, `est_vide(file)`, `enfiler(file,elt)` et `defiler(file)`.

L'objectif de l'exercice suivant est d'implémenter des files avec des objets. L'objet sera ici implémenté avec des tableaux dynamiques.

Exercice 5.23
Créer la classe `File` qui dispose des méthodes suivantes :

- `estvide()` qui renvoie `True` si la file est vide, `False` sinon.
- `enfiler(elt)` qui permet d'enfiler un élément à la file
- `defiler()` qui défile un élément et le renvoie.
- `affiche()` qui affiche la file

5.4.4 Construire une file avec deux piles

Principe

Soient `p1` et `p2` deux piles vides.

Principe pour enfiler

1. Avant d'enfiler `elt`, on déplace les éventuels éléments présents dans `p2` vers `p1` par dépilages et empilages succésifs.
2. On enfile `elt` dans `p2` (qui est vide)
3. On déplace chaque élement de `p1` vers `p2` par dépilages et empilages succésifs.

Principe pour défiler

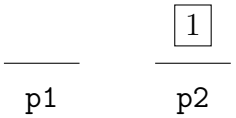
1. Pour défiler, il suffit de dépiler sur `p2`.

Exemple pour enfiler 1, puis 2 puis 3 :

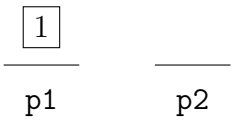
1. Initialisation :



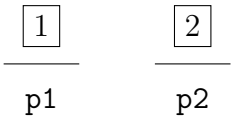
2. On enfile 1 par empilage de 1 sur `p2` :



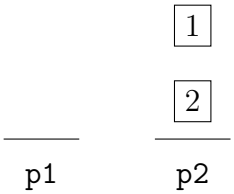
3. Pour enfiler 2 :
   a) On déplace 1 sur `p1` :



b) On enfile 2 par empilage de 2 sur `p2` :

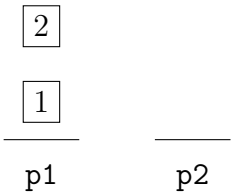


c) On déplace 1 sur `p2` :

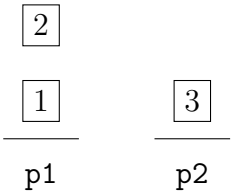


4. Pour enfiler 3 :

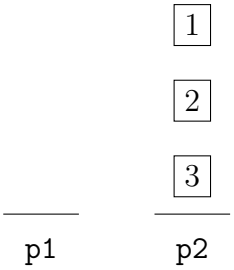
a) On déplace les éléments de `p2` sur `p1` :



b) On enfile 3 par empilage de 3 sur `p2` :



c) On déplace les éléments de `p1` sur `p2` :



5. Pour défiler, il suffit de dépiler successivement sur `p2` !

Exercice 5.24

On considère une classe `Pile` construite précédemment, et disposant des méthodes `est_vide()`, `empiler(elt)`, `depiler()`.
On considère deux piles `p1` et `p2`.
Avec ces deux piles, on souhaite, en utilisant le principe précédent, créer une file.
On créé une file vide par la création des deux piles vides `p1` et `p2`

1. Créer la pile p1 et la pile `p2`.
2. Créer la fonction `enfiler(elt)`, avec `elt` l'élément à enfiler, en utilisant seulement les piles `p1` et `p2`
3. Créer la fonction `defiler()`, en utilisant seulement les piles `p1` et `p2`
4. Tester votre file !

Exercice de synthèse 5.25

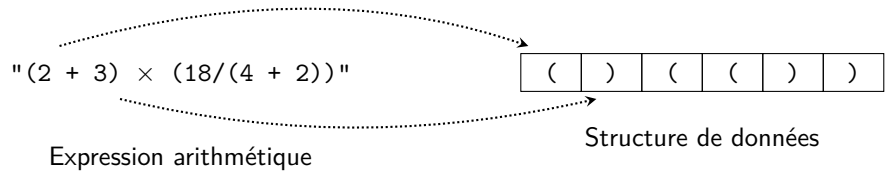
Cet exercice composé de deux parties A et B, porte sur les structures de données.

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) × (18/(4 + 2))"` est correctement parenthésée.
- L'expression arithmétique `"(2 + 3) × (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
 8     controleur = 0
 9     for parenthese in expression: #pour chaque parenthèse
10         if parenthese == '(':
11             controleur = controleur + 1
12         else:# parenthese == ')'
13             controleur = controleur - 1
14             if controleur ... : # test 1 (à recopier et compléter)
15                 #parenthèse fermante sans parenthèse ouvrante
16                 return False
17     if controleur ... : # test 2 (à recopier et compléter)
18         return True #le parenthésage est correct
19     else:
20         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.

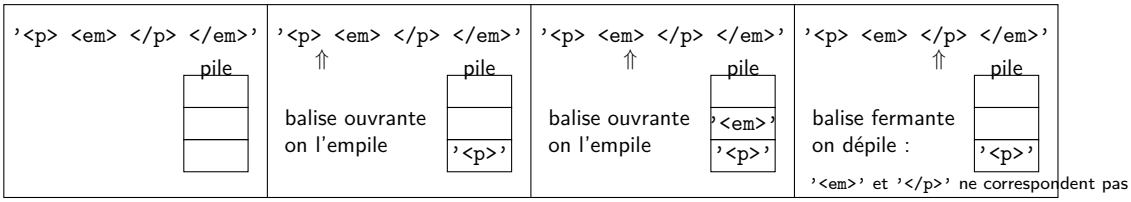
On ne tiendra pas compte dans cette partie des balises ne comportant pas de fermeture comme `<br>` ou `<img ...>`.

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.