TD/TP Structure de données - Piles et Files

M. Tellene

1 Appréhender la structure de données pile

Exercice 1

Décrire l'état de la pile après chaque ligne du programme suivant :

```
p = Pile()
p.empiler(1)
p.empiler(8)
p.depiler()
p.empiler(12)
p.empiler(0)
p.depiler()
```

Exercice 2

Écrire un programme qui crée une pile. Le programme devra empiler 3 valeurs : 8 puis 1 puis 9. Une fois fait, le programme devra dépiler 1 fois.

Sur votre feuille, décrire l'état de la pile.

Reprenez le programme et empilez la valeur 4 puis dépiler 3 fois.

Sur votre feuille, re-décrire l'état final de la pile.

2 Créer la classe `Pile`

Organisation : aller dans votre dossier personnel, puis dans votre dossier « structure de données » (ou « SD »), dans ce dossier, créer un dossier « pile et file », c'est ce dossier qui contiendra le travail fait lors de ce TP.

On rappelle qu'une pile suit le concept du « dernier entré, premier sorti »ou en anglais **LIFO** pour Last In First Out. En termes de structure de données, une pile est une séquence de $x_0, x_1, ..., x_{n-1}$ de valeurs où il est possible de retirer et d'ajouter un élément du **même côté de la séquence**. Ainsi dans une pile, seul le sommet est accessible.

Créer un fichier **Pile.py** et dans ce fichier, créer une classe `Pile`. Une pile n'est définie que par un attribut `contenu` initialisé avec un tableau vide (cf `[]`).

Une fois fait, créer les méthodes associées à la structure de données de pile. Si vous ne vous en souvenez pas, elles sont données dans la suite.

- `est_vide()` : renvoie `True` si la pile est vide, `False` sinon
- `empiler(x)` : empile la valeur `x` (attention le premier élément du tableau `contenu` doit être le dernier élément à avoir été empilé)
- `depiler()` : dépile (enlève) la valeur au sommet de la pile (cf la dernière valeur à avoir été empilée)
- `vider()` : la pile ne contient plus aucun élément

3 Utiliser la classe `Pile` pour résoudre des problèmes

3.1 Renverser un pile

Écrire une méthode `renverser` qui permet de renverser un objet de la classe `Pile`.

Exemple :

```
print(p)
>>> 12 10 8 6

p.renverser()
print(p)
>>> 6 8 10 12
```

Vous n'utiliserez pas la méthode `reverse` de la classe `list`.

3.2 Validation de parenthèses

Soit une chaîne de caractères ne pouvant contenir que les éléments suivants : `"("`, `")"`, `"["`, `"]"`, `"{"` ou `"}"`. Le but est de déterminer si la chaîne de caractères est bien parenthésée.

Une chaîne de caractères est valide si et seulement :

- une parenthèse ouverte et fermée par le même type de parenthèse
  - Exemple valide : `"()"`
  - Exemple invalide : `"(]"`
- les parenthèses ouvertes doivent être fermées dans le bon ordre
  - Exemple valide : `"([])"`
  - Exemple invalide : `"([)]"`
- toutes les parenthèses fermantes ont une parenthèse ouverte du même type correspondante
  - Exemple valide : `"()[]{()}"`
  - Exemple invalide : `"()[{}])"`

Écrire une fonction prenant en paramètre une chaîne de caractère uniquement composée de parenthèses et renvoyant si celle-ci est bien parenthésée ou non.

3.3 La calculatrice polonaise inverse à pile

L'écriture polonaise inverse des expressions arithmétiques en plaçant l'opérateur après ses opérandes. Cette notation ne nécessite aucune parenthèse ni aucune règle de priorité. Ainsi l'expression polonaise inverse décrite par la chaîne de caractères : `'1 2 3 * + 4 *'` désigne $(1+2\times3)\times4$.

La valeur d'une telle expression peut être calculé facilement en utilisant une pile pour stocker les résultats intermédiaires. Pour cela, on observe un à un les éléments de l'expression polonaise et on effectue les actions suivantes :

- si on voit un nombre, on l'empile
- si on voit un opérateur binaire, on récupère les deux nombres au sommet de la pile, on applique l'opérateur, et on replace le résultat sur la pile

Si l'expression était bien écrite, il y a bien toujours deux nombres sur la pile lorsque l'on voit un opérateur, et à la fin du processus il reste exactement un nombre sur la pile : le résultat.

Écrire **une fonction (pas une méthode donc à l'extérieur de la classe `Pile`)** prenant en paramètre une chaîne de caractères représentant une expression en notation polonaise inverse composée d'additions et de multiplications de nombres entiers et renvoyant la valeur de cette expression. On supposera que les éléments sont séparés par des espaces. **La fonction ne doit pas renvoyer de résultat si l'expression est mal écrite.**

4 Appréhender la structure de données file

Exercice 3

Décrire l'état de la pile après chaque ligne du programme suivant :

```
f = File()
f.enfiler(1)
f.enfiler(8)
f.defiler()
f.enfiler(12)
f.enfiler(0)
f.defiler()
```

Exercice 4

Écrire un programme qui crée une file. Le programme devra enfiler 3 valeurs : 8 puis 1 puis 9. Une fois fait, le programme devra défiler 1 fois.

Sur votre feuille, décrire l'état de la file.

Reprenez le programme et enfilez la valeur 4 puis défiler 3 fois.

Sur votre feuille, re-décrire l'état final de la file.

5 Créer la classe `File`

On rappelle qu'une file suit le concept du « premier entré, premier sorti »ou en anglais **FIFO** pour First In First Out. En termes de structure de données, une file est une séquence de $x_0, x_1, ..., x_{n-1}$ de valeurs où l'ajout d'éléments se fait par **un côté de la séquence** et le retrait se fait par **l'autre côté**. Ainsi dans une file, le sommet et la fin sont accessibles.

Dans votre dossier « pile et file », créer un fichier **File.py** et dans ce fichier, créer une classe `File`. Une file n'est définie que par un attribut `contenu` initialisé avec un tableau vide (cf `[]`).

Une fois fait, créer les méthodes associées à la structure de données de file. Si vous ne vous en souvenez pas, elles sont données dans la suite.

- `est_vide()` : renvoie `True` si la file est vide, `False` sinon
- `enfiler(x)` : enfile la valeur `x` (attention le premier élément du tableau `contenu` doit être le dernier élément à avoir été enfilé)
- `defiler()` : défile (enlève) la valeur à la fin de la file (cf la dernière valeur à avoir été enfilée)
- `vider()` : la file ne contient plus aucun élément

6 Utiliser la classe `File` pour jouer à la bataille

Considérons le jeu de la bataille. Chaque joueur possède un paquet de cartes et pose à chaque manche la carte prise sur le dessus du paquet. Le vainqueur de la manche récupère alors les cartes posées, pour les placer au-dessous de son paquet.

Écrire une fonction `jeu_bataille` qui permet de jouer à la bataille. Vous pouvez créer des fonctions annexes si vous le souhaitez. Les paquets de deux joueurs doivent être des objets de la classe `File` et les cas de batailles doivent être modélisé par des objets de la classe `Pile`.

Une fonction permet d'initialiser les paquets est donnée ci-dessous vous pouvez l'utiliser :

```python
def initalisation_paquets():
    cartes = [
        ("coeur", 1), ("coeur", 2), ("coeur", 3), ("coeur", 4), ("coeur", 5),
        ("coeur", 6), ("coeur", 7), ("coeur", 8), ("coeur", 9), ("coeur", 10),
        ("coeur", 11), ("coeur", 12), ("coeur", 13), ("trèfle", 1),
        ("trèfle", 2), ("trèfle", 3), ("trèfle", 4), ("trèfle", 5),
        ("trèfle", 6), ("trèfle", 7), ("trèfle", 8), ("trèfle", 9),
        ("trèfle", 10), ("trèfle", 11), ("trèfle", 12), ("trèfle", 13),
        ("pique", 1), ("pique", 2), ("pique", 3), ("pique", 4), ("pique", 5),
        ("pique", 6), ("pique", 7), ("pique", 8), ("pique", 9), ("pique", 10),
        ("pique", 11), ("pique", 12), ("pique", 13), ("carreau", 1),
        ("carreau", 2), ("carreau", 3), ("carreau", 4), ("carreau", 5),
        ("carreau", 6), ("carreau", 7), ("carreau", 8), ("carreau", 9),
        ("carreau", 10), ("carreau", 11), ("carreau", 12), ("carreau", 13)
    ]

    pA = File()
    pB = File()

    while len(pA.contenu) != len(pB.contenu) or len(pA.contenu) == 0:
        for c in cartes:
            if randint(1,2) == 1: pA.enfiler(c)
            else: pB.enfiler(c)
        if len(pA.contenu) != len(pB.contenu):
            print("Paquet non égaux, re-remplissage")
            pA = File()
            pB = File()

    return pA, pB
```