TP 14

Travaux Pratiques

Exercices d'application sur les piles

Dans les exercices qui suivent, on utilisera des **piles**. Les piles sont des structures de données basées sur le principe LIFO (Last In First Out : le dernier rentré dans la pile sera le premier à en sortir).

Les opérations élémentaires qu'on peut réaliser sur les piles sont les suivantes :

- savoir si une pile est vide ;
- empiler un nouvel élément sur la pile ;
- récupérer l'élément au sommet de la pile tout en le supprimant. On dit que l'on dépile ;
- accéder à l'élément situé au sommet de la pile sans le supprimer de la pile ;
- on peut connaitre le nombre d'éléments présents dans la pile.

Pour implémenter les piles on utilisera le module `deque`. Chacun des éléments de la pile peut être un objet de type différent.

```
from collections import deque

# Créer une pile vide
pile = deque()

# Tester si une pile est vide
len(pile) == 0

# Ajouer l'élément Truc au sommet de la pile
pile.append("Truc")

# Supprimer (et renvoyer) le sommet d'une pile non vide
sommet = pile.pop()
```

Seules ces fonctions éléméntaires seront utilisées dans les exercices suivants.

Exercice 1 – La parenthèse inattendue Dans cet exercice, on souhaite savoir si une chaîne de caractères est bien parenthésée ou non. Une chaîne bien parenthésée est une chaîne vide ou la concaténation de chaînes bien parenthésées.

▪ **Exemple** Chaînes bien parenthésées :
- `"()"`,`"()()"`,`"(())"` et `"(()())"`.

Chaînes mal parenthésées :
- `")("`,`"(("`,`"()"` et `"())"`.

▪

Question 1 *Implémenter la fonction* `parentheses(s:str)-> list` *renvoie une liste de tuples correpondants aux indices des parenthèses ouvrantes et fermantes.*

Question 2 *Réaliser un programme permettant de savoir si une chaîne de caractères est bien parenthésée. La structure de pile est-elle nécessaire ?*

Question 3 *Adapter le premier programme pour qu'il puisse traiter des chaînes constituées de parenthèses, de crochets, ou d'accolades. Un mot est alors bien parenthésé si la parenthèse fermante qui correspond à chaque parenthèse ouvrante est du même type.*

Question 4 *Adapter le programme pour qu'il puisse traiter des mots constitués de parenthèses et d'autres caractères, qui n'interfèrent pas avec les parenthèses.*

Question 5 *Écrire une version récursive de la fonction* `parentheses`.

Exercice 2 – Inversion

Question 6 *Écrire une fonction qui intervertit les deux éléments situés au sommet d'une pile de taille au moins égale à 2.*

Exercice 3 – Dépile le n^e

Question 7 *Écrire une fonction qui dépile et renvoie le troisième élément d'une pile de taille au moins égale à 3. Les premier et deuxième éléments devront rester au sommet de la pile.*

Exercice 4 – Lire le n^e

Question 8 *Écrire une fonction qui permet de lire (sans l'extraire) le n-ième élément d'une pile. On prévoira le cas où la pile n'est pas de taille suffisante pour qu'un tel élément existe.*

Exercice 5 – Inversion des extrêmes

Question 9 *Écrire une fonction qui prend une pile non vide en argument et place l'élément situé à son sommet tout au fond de la pile, en conservant l'ordre des autres éléments. Quelle est sa complexité en temps et en espace ?*

Exercice 6 – Inversion de la pile

Question 10 *Écrire une fonction similaire à* `reversed`, *qui prend une pile en argument et renvoie une autre pile constituée des mêmes éléments placés dans l'ordre inverse.*

Question 11 *Si l'on s'autorise à détruire la pile fournie, quelle est la complexité en temps et en espace de cette fonction ? Et si on ne s'y autorise pas ?*

Exercice 7 – Tu coupes ?

Question 12 *Écrire une fonction couper qui prend une pile et la coupe en enlevant de son sommet un certain nombre d'éléments (tirés au hasard) qui sont renvoyés dans une seconde pile.*

▪ **Exemple** Si la pile initiale est `[1, 2, 3, 4, 5]`, et que le nombre d'éléments retiré vaut 2, alors la pile ne contient plus que `[1, 2, 3]` et la pile renvoyée contient `[5,4]`. ▪