TP n° 2 : Piles

MP Lycée Camille Jullian

8-15 octobre 2021

Ce TP est destiné à vous faire manipuler un peu la structure de Pile que nous avons vue en cours. Les exercices en sont pour la plupart très abordables, et permettront en fait de réviser également les listes qui seront sous-jacentes à notre implémentation des piles.

Exercice 1

Commencez par simplement programmer l'implémentation des piles vue en cours. On privilégiera a priori la version orientée objet avec définition de classe. Vérifiez que les différentes fonctionnalités (test de pile vide, dépilage, empilage) fonctionnent correctement. On essaiera ensuite d'ajouter à la définition de classe les méthodes suivantes (en n'utilisant bien sûr que les fonctionnalités de base des Piles pour les programmer) :

- une méthode **hauteur** qui renvoie le nombre d'éléments actuellement stockés dans la Pile.
- une méthode **affichage** qui affiche le contenu d'une Pile (le mode d'affichage est laissé au choix du codeur, on essaiera de faire quelque chose de lisible).
- une méthode **videpile** qui vide complètement la Pile.
- une méthode **echange** qui échange les deux objets situés au sommet de la Pile (en supposant qu'il y ait au moins deux objets dans la Pile).
- une méthode **depilebloc(k)** qui dépile d'un seul coup les k derniers objets de la Pile et les renvoie sous forme de liste.
- une méthode **inverse** qui retourne l'ordre des éléments dans une Pile. On pourra ensuite essayer d'en écrire une version qui renvoie la Pile retournée tout en laissant la Pile initiale intacte.
- une méthode **decalage(k)** qui effectue un décalage circulaire de k unités sur la position de tous les éléments de la Pile. Ainsi, si la Pile initiale est $[1, 3, 7, 2, 4, 9]$, la méthode decalage appliquée avec un paramètre $k = 4$ doit renvoyer la Pile $[7, 2, 4, 9, 1, 3]$.

Pour les exercices suivants, on essaiera de tout faire en utilisant des Piles et en se servant uniquement des méthodes programmées ci-dessus.

Exercice 2

Écrire un programme Python qui teste si une chaîne de caractères donnée est un palindrome (on négligera les espaces, et on ne tiendra pas compte non plus des majuscules/minuscules ou des accents, comme par exemple dans le célèbre « Et la marine va venir à Malte ».

Exercice 3

Écrire un programme Python qui effectue le mélange de deux piles : tant qu'aucune des deux piles n'est vide, on dépile aléatoirement l'une des deux piles et on epile l'élément dans une nouvelle pile qui sera renvoyée en fin de programme. On doit vider les deux piles avant de terminer le programme.

Exercice 4 : files

La structure de File est un peu analogue à celle de Pile, mais fonctionne suivant le principe FIFO (First In, First Out) de la file d'attente. Pour implémenter une file d'attente en Python, on peut créer une liste du type $F = [d, f, [0, 0, \ldots, 0]]$, où d représente l'indice de début de la file (initialement égal à 0), f l'indice de fin (initialement à 0 également) et la sous-liste de zéros contient n éléments (longueur maximale de la file d'attente). Pour faire fonctionner la file d'attente, on doit avoir, comme pour les piles, des fonctionnalités permettant de tester si la file est vide, d'enfiler un nouvel élément (qu'on placera à la position f de la liste d'éléments, en augmentant f d'une unité) et de défiler la file (on retire l'élément en position d et on augmente la valeur de d d'une unité). Si jamais la file déborde (d ou f est égal à $n-1$ et on souhaite défiler ou enfiler un élément), aucun problème, on « boucle » en donnant la valeur 0 à d ou à f. Il n'y a que dans le cas où la file contient déjà n éléments qu'on ne peut pas enfiler, et dans le cas où elle est vide qu'on ne peut pas défiler.

Écrire une implémentation en Python de cette structure de données (on pourra essayer de créer une belle classe si on est courageux, mais ce n'est pas obligatoire).

Exercice 5 : problème du cavalier

On souhaite écrire un programme Python permettant de trouver une solution au très classique problème du cavalier : un cavalier est placé dans une position quelconque sur un échiquier (le problème classique se fait sur un échiquier standard à 8×8 cases, mais on peut aussi s'y intéresser sur des échiquiers de taille différente, par exemple sur un 6×6 pour commencer), et on souhaite lui faire parcourir toutes les cases de l'échiquier (à l'aide de déplacements de cavalier, c'est–dire deux cases dans une direction puis une dans une direction perpendiculaire) en passant exactement une fois par chaque case. Si on peut créer un mouvement circulaire où on peut revenir à la case initiale après avoir atteint la dernière c'est encore plus joli. Pour trouver une solution, on pourra procéder ainsi :

- Écrire une fonction Python qui calcule la liste des cases disponibles à partir d'une case donnée (ici une case sera représentée par une liste de deux entiers).
- Créer une Pile contenant les cases déjà parcourues par le cavalier. La pile sera initialisée en contenant uniquement la case de départ souhaitée.
- Créer une deuxième Pile contenant les listes de cases encore disponibles depuis chacune des cases déjà visitées. Cette deuxième Pile (qui est une Pile de listes) contiendra au départ un seul élément : la liste des cases disponibles depuis la case de départ choisie.
- À chaque étape de l'algorithme, on procède comme suit : si la dernière liste de cases disponibles empilée est non vide, on empile dans la liste des cases parcourues son premier élément, et dans la deuxième pile la liste des cases disponibles depuis ce premier élément et qui n'ont pas déjà été parcourues. Si la liste des cases disponibles est vide, on dépile de la liste des cases disponibles la dernière case empilée et on supprime cette même case de la liste des cases disponibles depuis la dernière case désormais atteinte. Bien sûr, on arrête le programme si on a parcouru toutes les cases et on ressort dans ce cas la liste des valeurs de la première Pile. Si jamais on se retrouve à un moment donné avec une première Pile vide, c'est que le programme a échoué à trouver une solution.