TP 4 : Implémentation de conteneurs élémentaires en POO

CPES « Sciences des données, arts et cultures » : Introduction à l'algorithmique

Antoine GAUQUIER
antoine.gauquier@ens.fr

Pierre SENELLART
pierre@senellart.com

07 novembre 2023

Le but de ce TP est d'implémenter un conteneur élémentaire (une liste simplement chaînée) comme une classe Python (donc, en utilisant le paradigme orienté objet) et de comparer ses performances aux conteneurs proposés par Python pour l'implémentation d'une pile.

Consignes pour le DS du 14/11 : Le DS aura lieu le **14/11 de 8h à 10h exactement** : prenez donc vos dispositions pour **ne pas arriver en retard**. Les documents sont autorisés, *limités à 10 feuilles recto-verso* par personne. Aucun appareil numérique n'est autorisé, donc, de façon non-exhaustive : pas d'ordinateur, pas de téléphone, pas de montre connectée, etc. Pas de calculatrice non plus. Le programme de révision du DS inclut tout ce qui a été vu en cours et en TD/TP depuis le début de l'année. Le sujet comportera également des questions avec du code Python à écrire. Comme vous ne pourrez pas tester l'exécution de ce code, les erreurs de syntaxe Python pure ne seront pas pénalisées, tant que le code écrit ressemble à un programme Python.

Remarques sur le DM1 : Les notes du DM1 sont affichées sur Pronote, et vous avez des commentaires individuels sur vos travaux sur l'espace de rendu du DM sur Moodle (y compris pour ceux qui ont envoyé leur DM par mail. Si vous avez toujours des problèmes d'accès à Moodle, envoyez-moi un mail). La moyenne du devoir est de 15.63/20, l'écart-type de 2.83, et les notes vont de 9.5 à 19.75/20. Un corrigé est en ligne sur Moodle.

Nous avons constaté un grand nombre de recopies de contenus de DM entre vous. Vous avez le droit, du moins dans le cadre des DM de ce cours, de discuter des attendus, de vos résultats, voire même de vous entraider. En revanche, la copie, intégrale ou partielle, du travail d'un de vos camarades (et de manière générale, de tout contenu que vous n'avez pas produit vous-même), **constitue une forme de plagiat**. Le plagiat est **interdit**, en particulier par le règlement de PSL, et vous encourrez des sanctions pouvant aller jusqu'à **l'exclusion**. Si jamais ce travail avait vraiment été fait en groupe, l'appropriation d'un travail collectif en précisant votre seul nom sur la copie est également une forme de plagiat. Nous laissons passer pour cette fois, étant donné que nous n'avions pas donné de consignes explicites à ce sujet. Ainsi, **le travail rendu dans le cadre des futurs devoirs maison doit être produit de façon individuelle. Toute recopie du travail d'un autre étudiant entraînera une sanction, choisie en adéquation avec le règlement de PSL**.

Exercice 1: Liste simplement chaînée

Question 1. Créer une classe `Chainon` contenant un simple constructeur permettant d'initialiser les deux attributs (publics) `valeur` et `suivant` d'un chaînon d'une liste simplement chaînée. Par convention, un chaînon sans chaînon suivant aura l'attribut `suivant` positionné à la valeur spéciale `None`.

Question 2. Créer une classe `ListeSimplementChainee` dédiée à l'implémentation d'une liste simplement chaînée. Une telle liste aura deux attributs (privés), `__premier` indiquant le premier chaînon de la liste et `__taille` indiquant le nombre d'éléments de la liste. Ajouter un constructeur à cette classe qui crée une liste vide.

Question 3. Afin de pouvoir facilement échanger `ListeSimplementChainee` et les méthodes des classes standards de Python, implémenter deux méthodes implémentées par ces mêmes classes :

— `append` prenant une valeur en argument et ajoutant cette valeur comme premier élément de la liste.

— `pop` ne prenant pas d'argument et renvoyant le premier élément de la liste, après l'avoir enlevé de la liste.

Tester que ces méthodes fonctionnent comme attendu.

Question 4. Ajouter à `ListeSimplementChainee` la méthode magique `__len__` sans autre argument que `self` et renvoyant le nombre d'éléments de la liste. Cette méthode est appelée sur un objet `o` quand on utilise la fonction standard Python `len(o)`. Tester.

Question 5. Afin de pouvoir utiliser la syntaxe « `for x in l:` » où `l` est une liste, nous devons implémenter un *itérateur* de liste simplement chaînée. Un itérateur est une classe dont les instances indiquent une position dans la liste chaînée. Définir une classe `Itérateur` avec une propriété `_chainon` stockant un chaînon et trois méthodes :

— un constructeur, qui initialise l'itérateur en le faisant pointer vers un chaînon passé en argument ;

— une méthode magique `__iter__` sans argument renvoyant l'itérateur lui-même ;

— une méthode magique `__next__` qui renvoie la valeur contenue dans le chaînon et fait pointer l'itérateur vers le chaînon suivant ; si le chaînon est `None`, la méthode renvoie l'exception `StopIteration` avec `raise StopIteration`

Enfin, ajouter à la classe `ListeSimplementChainee` une méthode magique `__iter__` sans argument renvoyant un itérateur vers le premier chaînon de la liste.

Question 6. Tester l'itération sur votre liste chaînée : créer une liste, lui ajouter quelques éléments, puis faire une boucle `for` qui affiche chacun des éléments de la liste tour à tour.

Question 7. Comme `Chainon` et `Iterateur` ne servent qu'à l'intérieur de `ListeSimplementChainee`, il est coutume d'inclure leur définition à l'intérieur de celle de `ListeSimplementChainee`. On fait référence à une classe `B` définie à l'intérieur d'une autre classe `A` avec le nom `A.B`. Faire ce changement est vérifier que tout fonctionne toujours.

Exercice 2: Comparaison de piles

On va, dans cet exercice, comparer la performance de 4 classes pour implémenter une structure abstraite de *pile* :

- la classe ListeSimplementChainee ;
- les listes Python traditionnelles ;
- la classe `array.array` du module standard `array` – cette classe nécessitant de spécifier un type d'éléments, on l'utilisera pour stocker des entiers sur 32 bits (en passant la valeur `'l'` au constructeur de la classe) ;
- la classe `collections.deque` du module standard `collection`.

Ces 4 structures de données implémentent toutes les méthodes `append`, `pop`, `__len__` et `__iter__` qu'on a ajoutées à `ListeSimplementChainee`, qui pourront donc s'utiliser exactement de la même manière.

Question 1. Définir une classe `Pile` dont le constructeur prend en argument une structure de données vide de l'un de ces 4 types et la stocke dans une propriété privée `__conteneur`. Ce conteneur servira à stocker les éléments de la pile. Ajouter à cette classe `Pile` les méthodes :

- `push` qui ajoute une valeur en haut de la pile ;
- `pop` qui enlève la valeur du haut de la pile et la renvoie ;
- `__len__` qui renvoie le nombre d'éléments sur la pile.

Question 2. Définit une fonction Python `compare_piles` prenant un entier `n` en argument et qui calcule le temps nécessaire (en utilisant une des méthodes vues en TP) à construire successivement des piles basées sur les 4 structures de données à comparer, ajoute les `n` premiers entiers naturels l'un après l'autre à la pile puis enlève tous les éléments de la pile un par un.

On évitera au tant que possible la duplication de code. On pourra utiliser `type(c).__name__` pour afficher le nom de la classe du conteneur, ainsi que le temps calculé.

Question 3. Utiliser cette fonction pour comparer les performances sur, par exemple, la tâche d'empiler puis dépiler 100 000 éléments. Quel structure de données est la plus efficace pour cette tâche ? Comparer les performances observées et essayer de les comprendre. Les structures de données moins efficaces ont-elles d'autres avantages ?