Liste chaînée

Terminale NSI - LPO Tani Malandi

Sources

Cette activité a été construite à l'aide du cours du site Python Carnot sur les listes chaînées : `https://python-carnot.fr/nsi-terminale/listes/cours`.

1 Tableaux et listes en Python

Nous avons appris à utiliser les tableaux, une structure de donnée séquentielle disponible directement pour le programmeur avec une syntaxe adaptée. Revoyons rapidement la création, l'utilisation et la manipulation de tableau en Python :

Créer un tableau	`tableau = [1, 1, 2, 3, 5, 8, 13, 21]`
Lire la 4ème valeur du tableau	`tableau[3]`
Remplacer la 5ème valeur du tableau par -22	`tableau[4] = -22`
Longueur d'un tableau	`len(tableau)`

Ce sont les seules instructions élémentaires que doit satisfaire une structure de donnée appelée tableau. Une partie des autres (pop(), append(), remove(), ...) sont issues d'une autre structure de donnée nommée liste, que le type list en python implémente. Voyons à présent les principales caractéristiques des tableaux et des listes.

1.1 L'interface d'un tableau

Définition Interface

Une interface définit la frontière de communication entre deux entités, comme des éléments de logiciel, des composants de matériel informatique, ou un utilisateur. Elle se réfère généralement à une image abstraite qu'une entité fournit d'elle-même à l'extérieur. Cela permet de distinguer les méthodes de communication avec l'extérieur et les opérations internes, et autorise à modifier les opérations internes sans affecter la façon dont les entités externes interagissent avec elle.
(wikipédia `https://fr.wikipedia.org/wiki/Interface_(informatique)`)

L'interface d'une structure de donnée est l'ensemble des opérations de base permettant d'interagir avec celle-ci. Le programmeur peut (et doit utiliser exclusivement) les opérations définies par l'interface pour interagir avec une structure de donnée. D'un point de vue abstrait, un tableau est une structure de donnée répondant aux critères suivants :

— Il s'agit d'un ensemble de données *homogènes* (c'est-à-dire du même type) rangée de manière *séquentielle* ;

— La taille du tableau, encore appelée *longueur*, est *fixe et connue* au moment de la création du tableau ;

— On peut lire toutes les valeurs d'un tableau en connaissant son rang (ou indice). Les indices sont des entiers compris entre 0 et $N-1$, où N est la longueur du tableau ;

— On peut modifier n'importe quelle valeur du tableau de rang donné ;

— on peut connaître la longueur du tableau ;

— Si un tableau est vide, il est de longueur 0

On constate que les tableaux tels qu'implémentés en python répondent bien à ces critères. Mais les tableaux du langage python peuvent faire bien plus. Voici quelques rappels :

Ajouter des éléments à la fin du tableau	`tableau.append(34)`
Retirer le dernier élément du tableau (s'il existe)	`tableau.pop()`
Insérer (ou retirer) des éléments de rang quelconque du tableau	`tableau.insert(2, 7)`

Ces fonctions ne font pas partie de l'interface d'un tableau, mais plutôt de celle d'une liste. Les tableaux du langage python sont en fait un type hybride ayant à la fois les caractéristiques d'un tableau et d'une liste. D'ailleurs, en python, les tableaux s'appellent des listes.

Nous parlerons de tableau/liste pour désigner ce type hybride du langage python. Nous nous astreindrons cependant à toujours bien distinguer les cas d'utilisation en tant que tableau pur, ou bien en tant que liste pure.

Examinons à présent l'interface d'une liste.

1.2 L'interface d'une liste

Une liste est un type de structure de données aillant les critères suivants :

— Il s'agit d'un ensemble de données *homogènes* (c'est-à-dire du même type) rangée de manière *séquentielle* ;

— La taille d'une liste, encore appelée *longueur*, est *variable* : elle peut changer au cours de la durée de vie de la liste ;

— Il est possible d'insérer et de supprimer des éléments en tête de liste ;

— Il est possible d'insérer et de supprimer des éléments en queue de liste ;

— Il est possible d'insérer et de supprimer des éléments à n'importe quelle position de la liste ;

— Il est possible de parcourir tous les éléments de la liste, du premier jusqu'au dernier ;

— Si une liste est vide, elle ne contient aucun élément, sa longueur est donc nulle.

Les tableaux/listes de python répondent à tous ces critères (et bien plus encore). Cependant, certaines fonctions de ces interfaces (à la fois pour les tableaux mais aussi les listes) ont un coût caché à l'exécution (qui est imperceptible pour des tableaux/listes ne contenant que peu de données, mais devient non négligeable lorsque l'on travaille sur des données pouvant atteindre plusieurs mégaoctets, voire gigaoctets).

2 Liste chaînée

Les listes en python sont en fait implémentées comme des tableaux dont la taille peut être modifiée dynamiquement. Il existe d'autres manières d'implémenter les listes, avec des coûts différents. La liste chaînée en est une des nombreuses possibilités.

2.1 Interface d'une liste chaînée

Une liste chaînée est une succession de cellules, chacune contenant à la fois une des valeurs de la liste, et un pointeur vers la cellule suivante. Par exemple, une variable $L1$ pointant sur une première cellule contenant la valeur 12, elle-même pointant sur la seconde cellule contenant la valeur 99, etc. L'exemple est illustré figure 1. La dernière cellule ne pointe sur rien, marquant ainsi la fin de la liste chaînée.

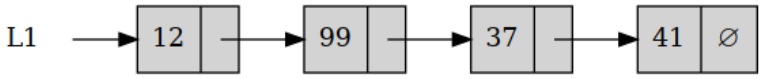

FIGURE 1 – Schéma illustrant une liste chaînée $L1$

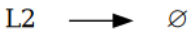

FIGURE 2 – Schéma illustrant une liste chaînée vide $L2$

$L2$ est une liste chaînée vide. Elle est représentée figure 2.

3 Implémentation Python

Comme son nom l'indique, la liste chaînée est implémentée à l'aide d'une séquence d'éléments composés de deux composantes :

— La valeur de cette élément ;

— Un pointeur vers l'élément suivant.

Définition d'un pointeur

En programmation informatique, un pointeur est un objet qui contient l'adresse mémoire d'une donnée ou d'une fonction. C'est l'outil fondamental de l'adressage dit 'indirect '. (wikipédia `https://fr.wikipedia.org/wiki/Pointeur_(programmation)`)

Nous appelerons un tel élément une *cellule*. Il existe deux manières simples de représenter une cellule en python : un tuple `(valeur, pointeur)` ou bien un tableau `[valeur, pointeur]`. Dans le premier cas, on aura une liste *immuable* (qui ne peut pas être modifiée une fois créée), dans le second une liste *variable* (dont la structure et/ou les valeurs peuvent être modifiés après la création). Les deux possibilités ont leurs avantages et leurs inconvénients, comme nous le verrons dans les exercices.

On donne ici une implémentation d'une liste chaînée mutable (cela sera nécessaire pour certains exercices).

```
def liste_vide():
    """
    Renvoie une liste vide.
    """
    return None # On décide de représenter la liste vide par None
```

```
def cellule(tete, queue):
    """
    Construit une liste chaînée en rajoutant l'élément tête au
    *début* de la liste chaînée queue. La liste queue n'est pas
    modifiée par cette opération.
    """
    return [tete, queue]

def tete(liste):
    """
    Renvoie le premier élément de la liste chaînée passée en
    paramètre.
    """
    return liste[0]

def queue(liste):
    """
    Renvoie la liste chaînée liste privée de son premier élément
    (la liste passée en paramètre n'est cependant pas modifiée
    par cette opération).

    Déclenche une erreur si appelé sur une liste vide.
    """
    return liste[1]

def change_tete(cellule, nouvelle_valeur):
    """
    Modifie la valeur de tête de la liste
    """
    cellule[0] = nouvelle_valeur

def change_queue(cellule, nouvelle_cellule):
    """
    Modifie la valeur de la queue de la liste: la cellule
    pointera sur une nouvelle cellule.
    """
    cellule[1] = nouvelle_cellule

def est_vide(liste):
    """
    Teste si une liste chaînée est vide ou non.
    """
    return liste is None

def longueur(liste):
    """
```

```
    Retourne la longueur de la liste chaînée passée en paramètre.
    """
    if liste is None:
        return 0
    else:
        return 1 + queue(liste)

def concatener(l1, l2):
    """
    Renvoie une nouvelle liste chaînée qui sera la concaténation
    de l1 et de l2.

    Les listes l1 et l2 ne sont pas modifiées par cette appel.
    """
    if est_vide(l1):
        return l2
    else:
        return cellule(tete(l1), concatener(queue(l1), l2))
```

On peut alors créer une liste chaînée représentant la liste "N", "S", "I" par la syntaxe (un peu lourde) suivante :

```
>>> l1 = cellule("N", cellule("S", cellule("I", liste_vide())))
>>> l1
["N",[ "S", ["I", None]]]
```

Remarquons qu'il serait tout à fait possible de créer directement la liste chaînée par la syntaxe :

```
>>> ["N", ["S", ["I", None]]]
["N",[ "S", ["I", None]]]
```

Cependant, c'est une très mauvaise idée : en faisant celà, on intégrerait dans nos futurs algorithmes les détails de l'implémentation de notre liste chaînée (notamment le fait qu'elle soit immuable ou bien variable). Cela rendrait toutes modifications futures de ces détails beaucoup plus compliquée. C'est pour cette raison que les informaticiens professionnels se réfugient toujours derrière une interface : que celle-ci soit fonctionnelle comme ici, ou bien utilisant de la programmation orientée objet, elle permet de rendre possible toute modification de l'implémentation qui ne changerait pas l'interface elle-même, offrant par la même occasion une grande souplesse et une grande sécurité au programmeur. *Il est donc possible de changer comment la liste chaînée est programmée sans avoir à changer comment elle est utilisée.*

4 Exercices

Voici une série d'exercices sur les listes chaînées. Afin de pouvoir les réaliser, vous devez récupérer les fonctions de base présentées au dessus (fonctions cellules, tete, ...). Pour cela, télécharger le fichier *listes.py* sur `http://www.numalandi.fr/nsit.html`. Les exercices 1 à 5 sont obligatoires, les suivants (6 à 10) sont facultatifs. L'ensemble des exercices réalisés seront évalués et sont à rendre.

4.1 Exercice 1

Écrire une fonction `liste_chainee(liste_python)` qui prend pour paramètre une liste (un tableau) python, et renvoie la liste chaînée correspondante.

Questions intermédiaires

Transformer une liste python en liste chaînée manuellement

```
liste_py = [1, 3, 8]
```

1. Saisir la liste précédente sur le script
2. Créer une cellule `c0` contenant le premier élément de `liste_py`.(indice `0`)
3. Créer une seconde cellule `c1` contenant le second élément de `liste_py`.(indice `1`)
4. Créer une troisième cellule `c2` contenant le dernier élément de `liste_py`.(indice `2`)
5. Concaténer la cellule `c1` à la suite de `c0` dans une liste `L`.
6. Concaténer la cellule `c2` à la suite de `L`.
7. À l'aide de la fonction afficher `afficher_liste`, afficher la liste chaînée `L`.

Maintenant que vous avez réalisé une version manuelle de la fonction, réaliser la fonction de l'exercice 1.

4.2 Exercice 2

Écrire une fonction `affiche_liste(liste)` qui affiche (à l'aide de `print`) les éléments d'une liste chaînée, séparés par des espaces. (Indice : Quelle fonction permet de savoir si une cellule est vide ?)

Questions intermédiaires

Afficher les éléments d'une liste chaînée manuellement

```
L = cellule("N", cellule("S", cellule("I", liste_vide())))
L2 = L.copy()
```

1. Saisir la liste précédente sur le script
2. Afficher la liste chaînée `L` à l'aide de la fonction `print`.
3. Afficher la tête de la liste chaînée `L` à l'aide de la fonction `print`. (indice : quelle fonction permet d'afficher la tête d'une cellule ?)
4. Afficher la tête de la seconde cellule de la liste chaînée `L` (le caractère "S") à l'aide de la fonction `print`. (indice : Quelle fonction permet d'accéder à la seconde cellule à partir de la première ?)
5. Afficher la tête de la troisième cellule de la liste chaînée `L` (le caractère "I") à l'aide de la fonction `print`.
6. Quelle fonction permet de passer d'une cellule à la suivante ?
7. Quelle boucle permettrait de passer d'une cellule à l'autre ? Pourquoi ? (indice : `for` ou `while` ?)

8. Comment savoir qu'une cellule est la dernière de la liste chaînée ? Quelle fonction permet de vérifier cela ?
9. Avec les éléments précédents, réaliser un code qui affiche les 3 éléments de `L`. Vous utiliserez la variable `L2` afin d'éviter de modifier les valeurs de `L`.

   Résultat attendu :

   ```
   N
   S
   I
   ```

10. Afin d'afficher les éléments séparés par un espace et non un retour à la ligne, vous utiliserez la fonction `print` de la façon suivante : `print(..., end=" ")` Où ... doit être remplacé par la tête de la cellule et `end=" "` permet de remplacer le retour à la ligne par un espace.

Maintenant que vous avez réalisé une version manuelle de la fonction, réaliser la fonction de l'exercice 2.

4.3 Exercice 3

Écrire une fonction `nieme_element(liste, n)` qui retourne l'élément d'indice n de la liste chaînée fournie en paramètre.

Questions intermédiaires

Renvoyer la valeur d'une cellule située à un rang connu.

```
L = cellule("S", cellule("A", cellule("L", cellule("U", cellule("T", liste_vide()))))
indice = 2
```

1. Recopier les deux lignes précédentes dans votre script python.
2. Accéder à la valeur (tête) de la cellule en position `1` (rappel : le début est à `0`) et afficher là.
3. Accéder à la valeur de la cellule en position `indice` et afficher là.
4. Changer la valeur de `indice`, la variable vaut maintenant `4`.
5. Est-ce que le code permet encore d'accéder à la valeur de la cellule ? Si ça n'est pas le cas, modifier le code pour qu'il le permette et afficher là.

Maintenant, pour répondre à l'exercice, il faut transformer la réponse à la dernière question en une fonction. Il au lieu d'afficher la valeur, il faut la retourner à l'aide de `return`.

4.4 Exercice 4

Écrire une fonction `occurences(valeur, liste)` retournant le nombre d'occurences de valeur dans la liste chaînée `liste`.

Questions intermédiaires

Trouver le nombre de fois qu'une valeur est dans une liste.
Utiliser les réponses des questions de l'exercice 3.

```
L = cellule("C", cellule("O", cellule("U", cellule("C", cellule("T", liste_vide()))))
valeur = "C"
indice = 1
```

1. Recopier les deux lignes précédentes dans votre script python.
2. Dans une condition `if-else`, afficher le message `"C'est bon"` si la tête de la première cellule de `L` vaut `valeur` et sinon afficher `"C'est pas bon"`.
3. Modifier le code de la question précédente pour que le test soit effectué pour la cellule de rang `2`.
4. Modifier maintenant le code pour qu'il fonctionne avec la valeur de rang `indice`.
5. Créer une variable initialisée à `0`, ajouter lui `1` à chaque fois que la tete de la cellule vaut `valeur`.

Il vous faut transformer le code de la dernière question en une fonction pour répondre à l'exercice 4.

4.5 Exercice 5

Écrire une fonction `trouve(valeur, liste)` retournant l'indice de la première occurence de `valeur` dans `liste`. Si `valeur` n'est pas dans `liste`, renvoie `None`.

4.6 Exercice 6 - Pour aller plus loin

Écrire une fonction `concaténer_en_place(l1, l2)` qui insèrera une cellule de valeur `valeur` à la position donnée par `indice`.

Renvoie la tête de `liste` (important si on insère en position 0, car la tête de `liste` sera alors modifiée).

4.7 Exercice 7 - Pour aller plus loin

Écrire une fonction `inserer_ordre(liste, valeur)` qui insère `valeur` dans la liste (supposée ordonnée) `liste`, de façon à ce que la liste reste ordonnée. Attention, aucune vérification de l'ordonnancement initial de liste est effectué. La liste sera modifiée par cette opération. La fonction renvoie la nouvelle valeur de la liste (indispensable si on insère en première position car la cellule initiale change alors).

4.8 Exercice 8 - Pour aller plus loin

En se servant de l'exercice précédent, écrire une fonction `tri_par_insertion(liste)` qui utilise cet algorithme pour trier la liste chaînée fournie en paramètre. Attention, cette liste ne doit pas être modifiée par cet appel, il faut retourner une nouvelle liste, triée.

4.9 Exercice 9 - Pour aller plus loin

Écrire une fonction `dernière_cellule(liste)` qui renvoie la dernière cellule de la liste `liste`. On suppose que la liste n'est pas vide.

4.10 Exercice 10 - Pour aller plus loin

Utiliser la fonction de l'exercice précédent pour écrire la fonction `concaténer_en_place(l1, l2)` qui concatène les listes $l1$ et $l2$ en reliant la fin de la première liste à la seconde. La liste $l1$ sera modifiée par cette opération (mais pas la liste $l2$). Renvoyer la toute première cellule de la concaténation.