

## Exercices : application du cours

### 1.1 Extraire et copier des informations

Une Université utilise un fichier avec la liste des étudiants ainsi que des informations les concernant. On suppose que ces renseignements sont mis sous la forme suivante :

```
D = {nom1: information1, nom2: information2, ... }
```

Les renseignements sont mis sous la forme d'un tableau D.

On souhaite créer une *liste chaînée* avec ces renseignements. Les étudiants sont alors les *maillons* d'une chaîne. Les maillons s'appelleront *Etudiant*. Les valeurs contenues sont *nom* et *information*. Un attribut appelé *suiv* stocke le *pointeur* vers l'étudiant suivant.

#### 1.1.1 Classe etudiant

Ecrire le script Python qui définit la classe *Etudiant*

#### 1.1.2 Le 2e étudiant

Ecrire le script qui affecte les renseignements du dictionnaire au 2e maillon *etudiant2*.

#### 1.1.3 Le premier étudiant

Ecrire le script qui affecte les renseignements du dictionnaire au premier maillon *etudiant1*.

Pointer aussi vers le 2e étudiant en renseignant l'attribut *suiv* du premier étudiant vers le second.

#### 1.1.4 La classe Université

Cette classe est une liste chaînée des étudiants. Cette liste ne comporte qu'un seul attribut, *premier*, qui pointe vers le premier étudiant de la chaîne.

- Ecrire le script qui définit cette classe.
- Ecrire l'instruction qui définit l'instance de la classe *Université*. On appellera cette instance de classe : *formation*. L'élément de tête de *formation* sera l'étudiant1.

#### 1.1.5 Questions de cours

- Quel est l'avantage de stocker les étudiants dans une liste chaînée plutôt que dans un dictionnaire
- Quel est l'avantage de la liste chaînée par rapport à d'autres structures linéaires, comme le tableau par exemple. Illustrez votre propos avec le cas du départ ou de l'arrivée d'étudiants en cours d'année dans cette université.

## Exercice 2

## Créer une liste chaînée - Pythontutor

L'exercice est à la page : [allophysique.com/docs/NSI/structure/page21/#exercices](http://allophysique.com/docs/NSI/structure/page21/#exercices)

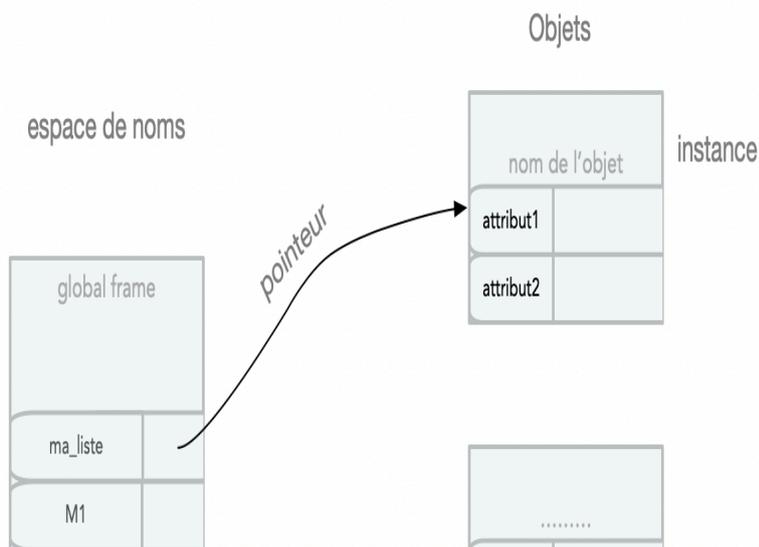
**Question 1 :** Compléter le script :

```

1 class Maillon:
2     def __init__(self):
3         self.val = None
4         self.suiv = None
5
6 class Liste:
7     def __init__(self):
8         self.tete = None
9
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ...
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...

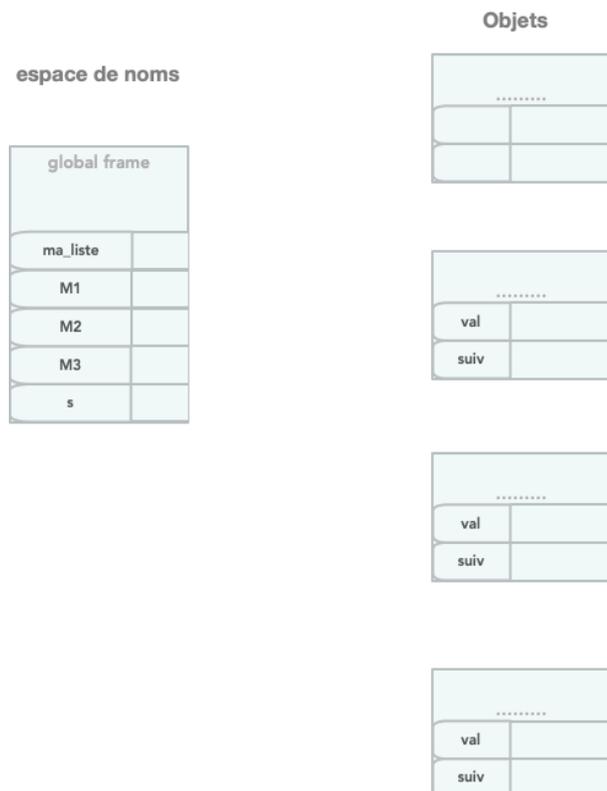
```

**Question 3 :** On adaptera le schéma de pythontutor de la manière suivante :



Les pointeurs sont représentés par des flèches courbes. Les objets (instances des classes Maillon et Liste) sont représentés par des boîtes.

Représenter en partie le schéma des différents objets et de leurs liens sur l'image suivante :



#### Question 4 :

```

1 s = ma_liste.tete
2 s.val = '1'
3 print(ma_liste.tete.val)
4 print(M1.val)

```

Qu'affiche la console ? Ce résultat était-il prévisible ? Pourquoi ?

Exercice 3

### Afficher les éléments de liste

Script itératif

```

1 M = L.tete
2 while not M.suiv is None:
3     ...
4     ...
5     ...
6     ...
7     ...
8     ...
9     ...
10    ...
11    ...
12    ...
13    ...

```

14 . . .

## Script récursif

```

1 def affiche(M):
2     if M.suiv is None:
3         return M.val
4     else:
5         . . .
6         . . .

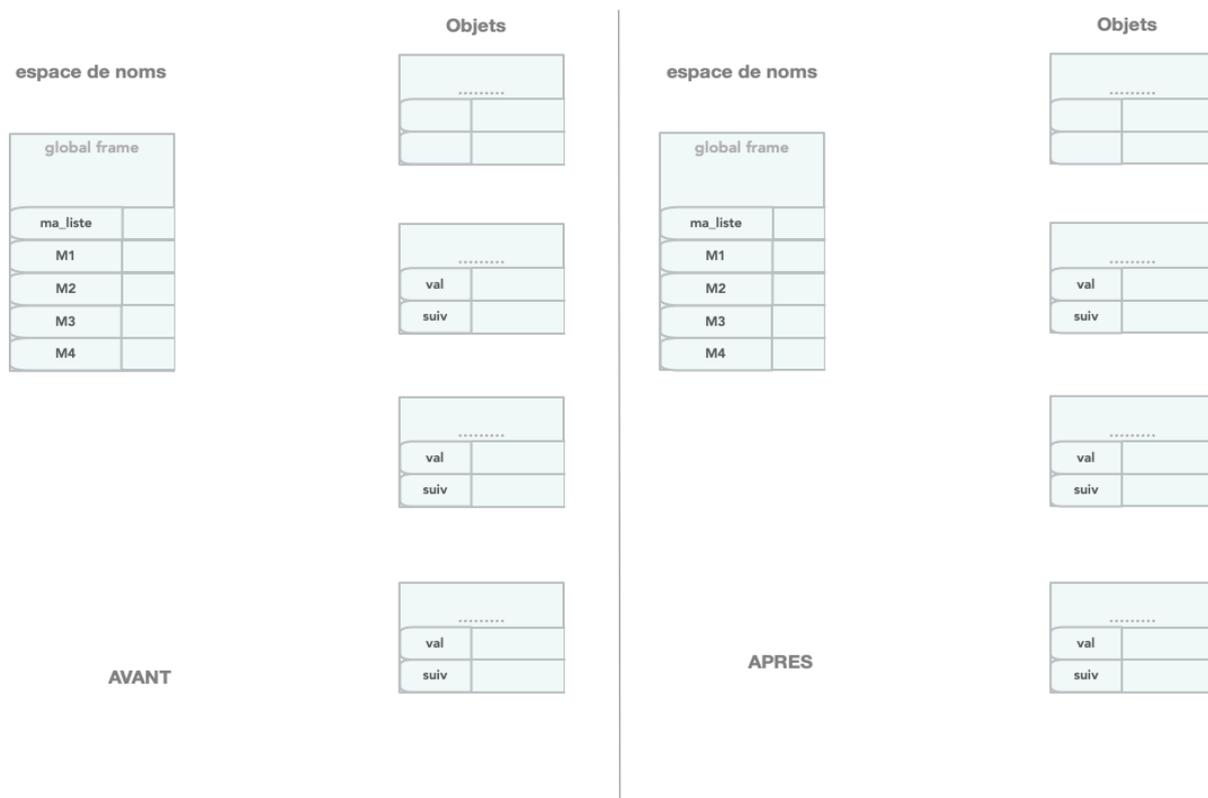
```

Appel de la fonction pour afficher TOUS les éléments de L :

Exercice 4

## Insertion d'un élément

Compléter le schéma pour représenter les éléments AVANT l'insertion, puis APRES l'insertion :



## Listes

### 5.1 Principe

Une liste chaînée est une structure linéaire, constituée de cellules. Une cellule (ou *maillon* contient :

- une valeur (de n'importe que type),
- et un pointeur qui dirige soit vers la cellule suivante, soit vers None, auquel cas la cellule est la *dernière* de la séquence.

Il s'agit d'une structure *linéaire*, mais dans laquelle les éléments n'occupent pas *à priori*, des positions contiguës dans la mémoire :

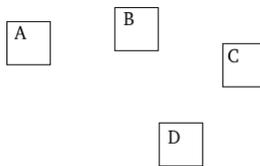


FIGURE 1 – elements dans une liste

Pour relier ces éléments ensemble, dans une même structure de données, il faut alors utiliser des *pointeurs*.

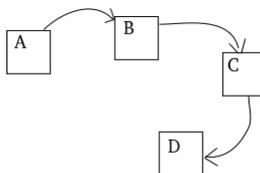


FIGURE 2 – liste chaînée grâce aux pointeurs

L'élément A pointe sur B, qui pointe sur C, qui pointe sur D. D ne pointe sur ... rien !

### 5.2 Représentation d'une liste chaînée

Exemple avec 3 éléments contenant les valeurs 'A', 'C', et 'D', mises dans 3 maillons M1, M2 et M3.

- M1 pointe vers M2
- M2 pointe vers M3
- M3 pointe sur ...rien

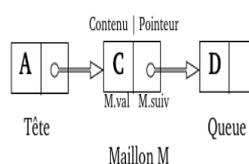


FIGURE 3 – illustration d'une liste chaînée

### 5.3 Interface

On trouve en général les opérations suivantes pour l'interface d'une *Liste* :

- `est_vide` : renvoie `True` si Liste vide
- `taille` : renvoie le nombre de *maillons* de la séquence
- `get_dernier_maillon` :
- `get_maillon_indice` : n'existe pas avec listes python
- `ajouter_debut` : n'existe pas avec listes python
- `insérer_apres` : n'existe pas avec listes python
- `ajouter_fin` :
- `supprimer_apres` : n'existe pas avec listes python
- ...

### 5.4 Implémentation

L'implémentation est plus naturelle en *programmation orientée objet* (voir le chapitre POO).

Un maillon est une instance de la classe `Maillon`

```
1 class Maillon:
2     def __init__(self):
3         self.val = None
4         self.suiv = None
```

Une Liste est une instance de la classe `Liste`

```
1 class Liste:
2     def __init__(self):
3         self.tete = None
```

Son attribut `tete` est de type `Maillon` , ou bien vaut `None` si la liste est vide.

### 5.5 Exemple de scripts

#### 5.5.1 Creation d'une liste

```
1 ma_liste = Liste()
2 M1, M2, M3 = Maillon(), Maillon(), Maillon()
3 M1.val = 'A'
4 M2.val = 'C'
5 M3.val = 'D'
6 M1.suiv = M2
7 M2.suiv = M3
8 M3.suiv = None
9 ma_liste.tete = M1
```

### 5.5.2 Afficher le contenu de l'élément de tête de ma\_liste

```
1 print(ma_liste.tete.val)
```

### 5.5.3 Afficher le 2e element

```
1 print(ma_liste.tete.suiv.val)
```

### 5.5.4 Afficher l'élément de rang N :

```
1 i = 1
2 M = ma_liste.tete
3 while i < 27:
4     i += 1
5     M = M.suiv
6 print(M.val)
```

### 5.5.5 Parcourir toute la liste chaînée :

```
1 # script itératif
2 M = ma_liste.tete
3 while not M.suiv is None:
4     <instruction>
5     M = M.suiv
```

```
1 # script recursif
2 def parcours(M):
3     if M.suiv is None:
4         <exit>
5     else:
6         <instruction>
7         parcours(M.suiv)
8
9 >>> parcours(ma_liste.tete)
```

## 5.6 Intérêt et inconvénient par rapport aux listes en python

L'interface d'une liste chaînée fournit des méthodes plus efficaces que la *Pile*, la *File* ou le *tableau*, lorsque l'on veut par exemple : **insérer**, ou **supprimer** un élément dans la séquence.

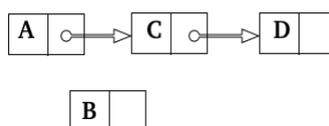


FIGURE 4 – insertion dans une liste

Cette opération est prévue par l'interface d'une liste chaînée =>  $O(n)$  : `insérer_apres(i)`, où  $i$  est l'indice de l'élément après lequel on veut *insérer*.

Avec une liste python, qui implémente la Pile (voir cours) cela nécessite de décaler toutes les valeurs de rang supérieur à  $i$ . C'est une opération qui est évaluée en  $O(n_2)$  pour sa complexité asymptotique.

Il s'agit du même problème lorsque l'on veut *supprimer après  $i$* .

Un autre avantage est la possibilité de faire pointer le dernier élément sur le premier de la liste. On crée ainsi une liste *périodique*.

**Inconvénient** : Pour accéder à un *maillon* de rang  $i$  dans une liste chaînée, il faut remonter la liste depuis le premier élément (celui de tête), jusqu'à celui de rang  $i$ . Et cela se fait avec une complexité asymptotique  $O(n)$ .

## 5.7 Correction des exercices en ligne

```
1 # correction de l'exercice 1
2 class Maillon:
3     def __init__(self):
4         self.val = None
5         self.suiv = None
6
7 class Liste:
8     def __init__(self):
9         self.tete = None
10
11 ma_liste = Liste()
12 M1, M2, M3 = Maillon(), Maillon(), Maillon()
13 M1.val = 'Premier'
14 M2.val = 'Troisieme'
15 M3.val = 'Quatrieme'
16 M1.suiv = M2
17 M2.suiv = M3
18 M3.suiv = None
19 ma_liste.tete = M1
20
21 print(ma_liste.tete.val)
22 print(ma_liste.tete.suiv.val)
23 print(ma_liste.tete.suiv.suiv.val)
24
25 s = ma_liste.tete
26 s.val = '1'
27 print(ma_liste.tete.val)
28 print(M1.val)
```

```
1 # Correction de l'exercice 2 en ligne question 2.1
2 M = L.tete
3 while not M.suiv is None:
4     print(M.val)
5     M = M.suiv
6 print(M.val)
```

Pour le script recursif, on pourra s'aider de l'exemple du script recursif qui ajoute 2.

```
1 # Correction de l'exercice 1.2 en ligne
2
3 def affiche(M):
4     if M.suiv is None:
5         return M.val
6     else:
7         return str(M.val) + '=>' + affiche(M.suiv)
8
9
10 # Correction question 2.3 en ligne
11 >>> print(affiche(L.tete))
12 'A=>B=>C=>D'
```

```
1 # Correction Exercice 3 en ligne
2 M4 = Maillon()
3 M4.val='Deuxieme'
4 M4.suiv = M2
5 M1.suiv = M4
```