Exercice 1

Exercices : application du cours

1.1 Extraire et copier des informations

(exercice sur le modèle de celui `https://inf104.wp.imt.fr/tp5-malloc-et-listes-chainees/` écrit pour le langage C)

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 chainée* avec ces renseigments. Les étudiants sont alors les *maillons* d'une chaine. Les maillons s'appeleront `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 Universite

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

a. Ecrire le script qui définit cette classe.

b. Ecrire l'instruction qui définit l'instance de la classe `Universite`. On appelera cette instance de classe : `formation`. L'élement de tête de `formation` sera l'`etudiant1`.

1.1.5 Questions de cours

a. Quel est l'avantage de stocker les étudiants dans une liste chainée plutôt que dans un dictionnaire

b. Quel est l'avantage de la liste chainée par rapport à d'autres structures lineaires, comme le tableau par exemple. Illustrez votre propos avec le cas du depart ou de l'arrivée d'étudiants en cours d'année dans cette université.

Exercice 2

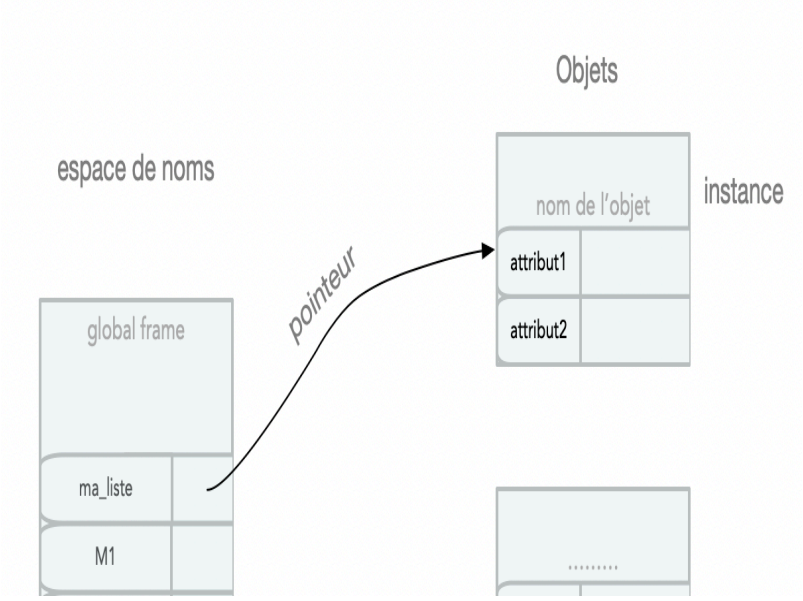
Créer une liste chainée - Pythontutor

L'exercice est à la page : allophysique.com/docs/NSI/structure/page21/#exercices

Question 1 : Compléter le script :

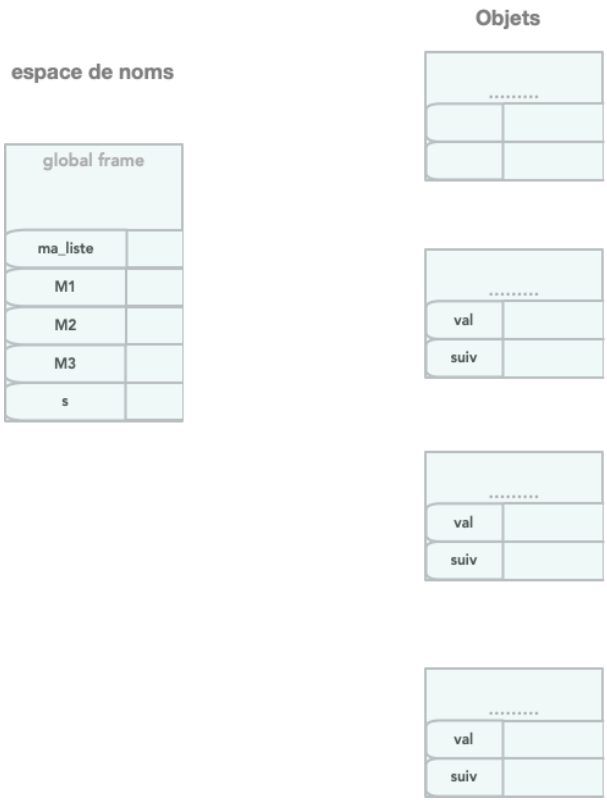
```
class Maillon:
  def __init__(self):
    self.val = None
    self.suiv = None

class Liste:
  def __init__(self):
    self.tete = None

...
...
...
...
...
...
...
...
...
...
...
...
...
```

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 :

```
s = ma_liste.tete
s.val = '1'
print(ma_liste.tete.val)
print(M1.val)
```

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

Exercice 3

Afficher les éléments de liste

Script itératif

```
M = L.tete
while not M.suiv is None:
...
...
...
...
...
...
...
...
...
...
...
...
```

Script recursif

```
def affiche(M):
```

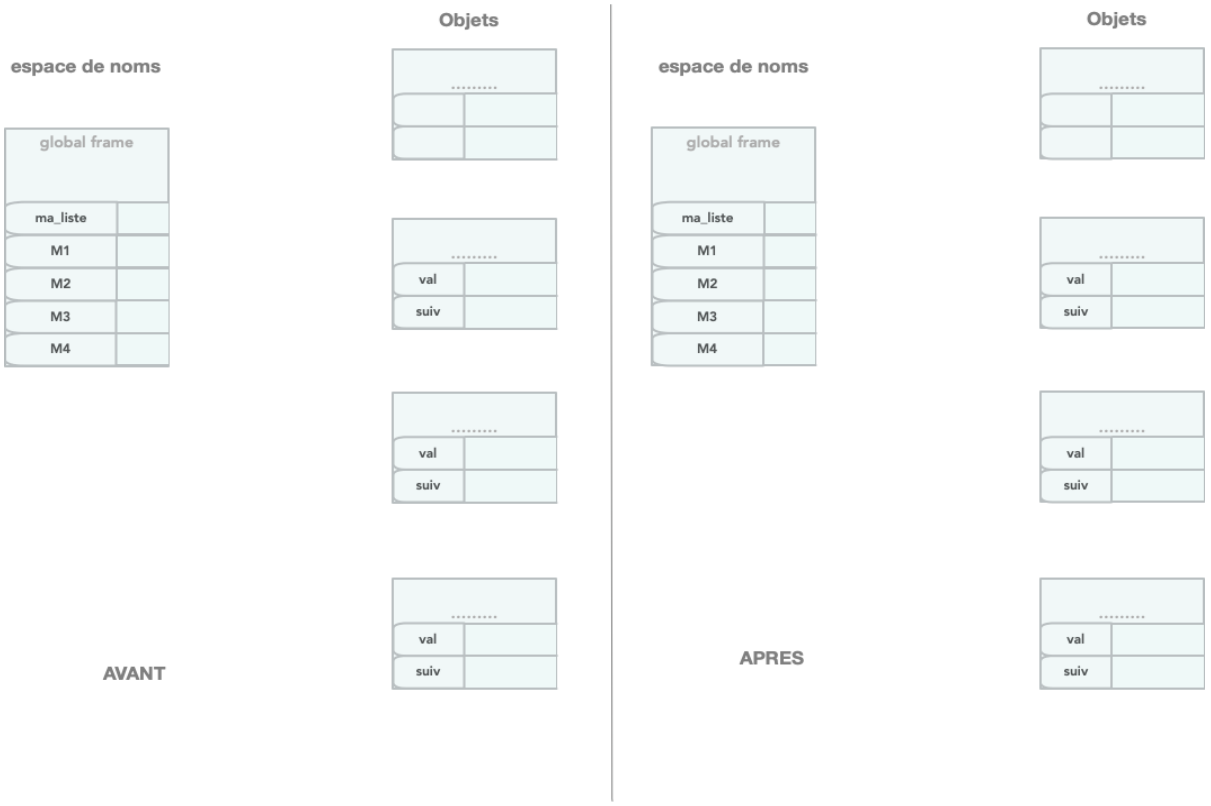
```
    if M.suiv is None:
        return M.val
    else:
...
...
```

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 :



Exercice 5

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 contigües dans la mémoire :

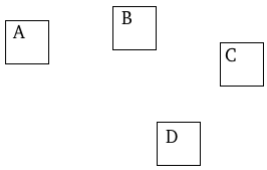


FIGURE 1 – elements dans une liste

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

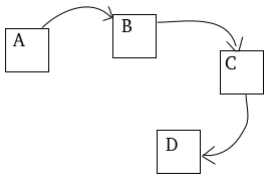


FIGURE 2 – liste chainée grâce aux pointeurs

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 chainée

Exemple avec 3 éléments contenants 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

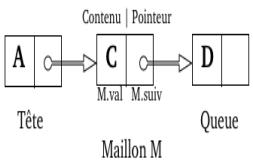


FIGURE 3 – illustration d'une liste chainée

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
- `inserer_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`

```
class Maillon:
  def __init__(self):
    self.val = None
    self.suiv = None
```

Une Liste est une instance de la classe `Liste`

```
class Liste:
  def __init__(self):
    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

```
ma_liste = Liste()
M1, M2, M3 = Maillon(), Maillon(), Maillon()
M1.val = 'A'
M2.val = 'C'
M3.val = 'D'
M1.suiv = M2
M2.suiv = M3
M3.suiv = None
ma_liste.tete = M1
```

5.5.2 Afficher le contenu de l'élément de tête de `ma_liste`

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

5.5.3 Afficher le 2e element

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

5.5.4 Afficher l'élément de rang N :

```
i = 1
M = ma_liste.tete
while i < 27:
  i += 1
  M = M.suiv
print(M.val)
```

5.5.5 Parcourir toute la liste chainée :

```
# script itératif
M = ma_liste.tete
while not M.suiv is None:
  <instruction>
  M = M.suiv
```

```
# script recursif
def parcours(M):
  if M.suiv is None:
    <exit>
  else:
    <instruction>
    parcours(M.suiv)

>>> parcours(ma_liste.tete)
```

5.6 Interêt et inconvenient par rapport aux listes en python

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

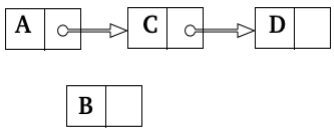


FIGURE 4 – insertion dans une liste

Cette opération est prévue par l'interface d'une liste chainée => O(n) : `inserer_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 necessite 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 créé ainsi une liste *périodique*.

Inconvénient : Pour accéder à un *maillon* de rang `i` dans une liste chainée, 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 asymtotique O(n).