

## Objectifs

### ☑ Connaissances visées

- ☐ *Listes, piles, files : structures linéaires.*
- ☐ *Dictionnaires, index et clé.*

### ☑ Compétences à développer

- ☐ *Distinguer des structures par le jeu des méthodes qui les caractérisent.*
- ☐ *Distinguer les modes FIFO (first in first out) et LIFO (last in first out) des piles et des files.*
- ☐ *Choisir une structure de données adaptée à la situation à modéliser.*
- ☐ *Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.*

## Séquence 2 – Implémentation d'une liste chaînée en P.O.O.

Les 'listes' implémentées nativement dans le langage Python – qui sont aussi appelées 'tableaux dynamiques' dans d'autres langages – sont des structures de données linéaires abstraites qui peuvent parfaitement remplir le rôle d'une liste chaînée avec une grande efficacité.

Il peut paraître alors superflu d'implémenter une structure linéaire de données comme une liste chaînée selon un paradigme 'objet'.

Toutefois cet exercice permet de mieux appréhender le mode de fonctionnement des 'listes' de Python, et de se confronter à la façon dont des valeurs sont 'gérées' en mémoire.

<b>Ressources matérielles et/ou logicielles</b>	☐ Éditeur et interpréteur Python (Ex : Pyzo)
<b>Ressources documentaires</b>	☐ « Ce qu'il faut retenir » (remis à la fin de la séquence 1)
<b>Modalités de réalisation</b>	☐ Alternances de phases de recherche individuelle ou en binômes et de mise en commun
<b>Durée estimée</b>	☐ 2h00
<b>Trace écrite à conserver dans le classeur</b>	☐ Le présente fiche de séquence ☐ Le code écrit en Python.

## Consignes

1. Écrire une classe appelée 'Element' qui permettra de créer les éléments d'une liste chaînée. Chaque objet 'Element' comportera deux attributs : un attribut 'valeur' et un attribut 'suivant'
2. Écrire une classe appelée 'ListeChaine' qui permettra de créer des listes chaînées. Chaque objet 'ListeChaine' comportera un seul attribut : 'start' qui prendra la valeur 'None' par défaut.
3. Écrire une méthode appelée 'estVide()' qui renverra 'true' si un objet 'ListeChaine' ne contient aucun élément 'Element' (donc si la valeur de 'start' est 'None') et 'false' dans le cas contraire.
4. Écrire une méthode appelée 'getElement()' qui renverra 'None' si un objet 'ListeChaine' est vide ou la valeur de l'objet 'Element' de l'attribut 'start'
5. Écrire une méthode appelée 'addElement()' qui ajoutera un objet 'Element' comme valeur de l'attribut 'start'.
6. Écrire une méthode appelée 'longIter()' qui renverra le nombre de valeurs d'un objet 'ListeChaine', en programmation itérative.
7. Écrire une méthode appelée 'listeValeurs()' qui renverra les valeurs de la liste, sous la forme d'une chaîne de caractères. Les valeurs seront séparées les unes des autres par une valeur passée à un paramètre 'separateur', avec un caractère 'espace' de chaque côté. Si la liste est vide, la méthode renverra la chaîne "None".
8. Écrire une méthode appelée 'ajouteApres()' qui permettra d'ajouter une valeur passée en paramètre, après une autre valeur également passée en paramètre.
9. Écrire une méthode 'cherche()' qui recherche la présence d'une valeur passée en paramètre dans la liste chaînée. Cette méthode renverra 'True' si la valeur est présente, sinon 'False'
10. Écrire une méthode 'retire()' qui retire de la liste chaînée une valeur passée en paramètre. Cette méthode renvoie 'False' si la valeur passée en paramètre n'existe pas dans la liste chaînée.
11. Tester la classe 'ListeChaine' :
  - a) en créant un objet,
  - b) en ajoutant des valeurs à cet objet,
  - c) puis en testant s'il est vide,
  - d) en déterminant sa longueur
  - e) en affichant la liste des valeurs,
  - f) en insérant une valeur après une autre,
  - g) en cherchant une valeur,
  - h) et en retirant une valeur.
12. Déterminer la complexité de chacune de ces méthodes.