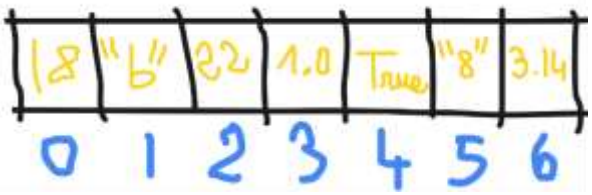
Type de document	Rappels de 1re	Classe	Tle	Durée	2h	Date	09/09/2024
Thème et contenu(s)	Structures linéaires de données : les listes						
Capacités attendues	Savoir parcourir et manipuler une liste – comprendre les différences et les similitudes avec une chaîne de caractères (notion d'immutabilité)						
Prérequis	Programme de classe de 1re dont : structures conditionnelles, boucles						
Description	Cours et exercices de révision autour des listes et des chaînes de caractères						

Note : les tableaux tels qu'on les retrouve dans les langages de programmation « classiques » comme le C n'existent pas nativement en Python, ils nécessitent un import (classe Array de NumPy) que l'on n'abordera pas ici. Par abus de langage volontaire, nous confondrons les termes **liste** et **tableau**

I) Définition et propriétés

I - a) Vue 1 : sous forme de tableau unidimensionnel

Une **structure linéaire de données** est une structure dans laquelle on stocke des éléments de manière séquentielle. Les éléments peuvent être parcourus intégralement en une seule fois. Il n'y a pas de notion de hiérarchie, tous les éléments sont « au même niveau ».



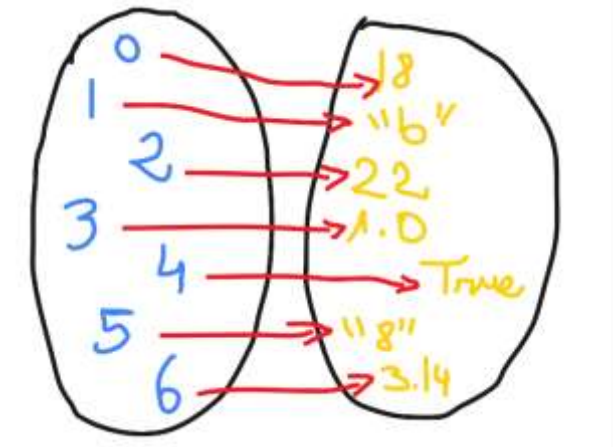
Une **liste à une dimension** est une structure linéaire de données.

Dans une liste « simple », ou tableau, les éléments sont « rangés » les uns à la suite des autres, de manière contigüe. Pour chaque élément i d'une liste, sauf le dernier, on connait même son successeur : i+1. Le premier élément a pour indice 0. Une liste de 7 éléments contient donc des éléments aux indices 0,1,2,3,4,5,6. C'est le cas de la liste ci-dessus. On dit que la taille de cette liste est de 7 (cette liste a une taille de 7).

Attention : il est très fréquent, lorsqu'on programme, de commettre l'erreur de faire référence à un indice qui n'existe pas, mais l'interpréteur Python saura vous remettre dans le droit chemin…

I - b) Vue 2 : sous forme d'ensembles

Une autre manière de reconnaître une structure linéaire de données est de remarquer la particularité suivante : pour chaque élément, il existe une fonction qui lui associe un autre élément (attention il y a des exceptions, par exemple la liste vide.



Ici, les flèches pourraient aller aussi dans l'autre sens également (bijection) : les images ont en effet ici un et un seul antécédent (car je n'ai pas de doublons dans ma liste !), mais ce n'est pas systématique.

I - c) Remarque

Plus tard dans l'année nous verrons d'autres structures linéaires de données (**piles**, **files**, **listes chaînées**) que l'on implémentera avec des listes, ou bien avec des objets. Nous verrons également des structures plus complexes mais très pratiques, telles que les arbres et les graphes, qui **ne sont pas** des structures linéaires de données.

II) Les listes

II - a) La mutabilité

Une des propriétés les plus importantes d'une liste en Python est qu'il s'agit d'un élément **mutable**, c'est-à-dire un élément qui a la possibilité d'être modifié au niveau de sa structure.

II - b) Déclaration d'une liste

Une liste vide se déclare simplement en Python : **maListe = []** et **maListe = list()** sont strictement équivalentes.

Mais…on préfèrera la première possibilité, pour des raisons de simplicité et de lisibilité. Une fois la liste ainsi créée, elle est vide (i.e. sa taille est de 0). On peut aussi initialiser (« remplir ») une liste lors de sa déclaration : **maListe = [18, "b", 22, 1.0, True, "8", 3.14]**

Chaque élément est séparé des autres au moyen d'une virgule. L'espace après la virgule est optionnel mais fortement recommandé (par convention).

Attention : je profite de ce premier cours pour vous prévenir concernant le nommage des listes, ainsi que des variables. **Elles doivent toujours commencer par une minuscule** (par convention, mais j'y tiens !). Pour le reste vous êtes libres d'utiliser le **snake_case** ou le **camelCase**, pour peu que le tout soit homogène. En Python les développeurs préfèrent le **snake_case**.

II - c) Manipuler une liste

Note : *vous allez sans doute vous rendre compte que l'on utilise des fonctions, mais aussi des choses qui y ressemblent mais qui sont précédées d'un point « . ».* Ce sont des **méthodes**. Elles jouent le rôle de **fonctions** en programmation objet. Nous y reviendrons beaucoup plus en détail cette année

Soit une liste l :

- Récupérer l'élément d'indice i (il faut qu'il existe !) : **l[i]**
- Obtenir la longueur de l : **len(l)**
- Ajouter un élément elem à l (l'élément se mettra à la fin de l) : **l.append(elem)**
- Renvoyer l'indice de la première occurrence de l'élément elem : **l.index(elem)**
- Supprimer la première occurrence de l'élément elem de la liste : **l.remove(elem)**
- Supprimer un élément à un indice donné i : **del l[i]**
- Supprimer un élément à un indice donné i et le renvoyer : **l.pop(i)**

II - d) Parcourir une liste

Il y a deux manières simples (d'autres existent mais on les verra plus tard).

- On peut le faire élément par élément (on itère donc sur des éléments et non pas des indices), grâce à une boucle **for** :
  for i in maListe:
  <<code ici>>

Ici, i représente un élément de la liste

- On peut le faire en itérant sur des indices. **N'oubliez pas : on ne peut itérer que sur quelque chose d'itérable, c'est-à-dire (pour simplifier) une liste de valeurs. Cela signifie que nous devons générer une liste des indices à parcourir.** Nous utilisons pour cela la fonction **range()**. Si nous connaissons la longueur n d'une liste l, nous ferons donc :
  for i in range(n):
  <<code ici>>
  Si nous ne connaissons pas la longueur de la liste, nous remplaçons n par len(l).
  for i in range(len(l)):
  <<code ici>>

Ici, i représente un entier correspondant à un indice

Note : La démarche serait un peu différente avec un **while** *car on n'aurait pas besoin de créer* un itérable

III) Exercices

Ces exercices doivent être faits **d'abord sur papier**, puis vérifiés sur machine. Conservez une trace totale (papier et machine) de ce que vous faites, c'est-à-dire que vous devez toujours écrire, pour chaque question, votre code à la suite des questions précédentes, sans rien effacer. Prenez l'habitude de commenter succinctement mais précisément votre code.

III - a) Échauffement

1. Créez et initialisez la même liste qu'en II - b)
2. Affichez cette liste
3. Affichez sa longueur
4. Utilisez une boucle for pour afficher chaque élément de la liste (itérez sur les éléments)
5. Utilisez une boucle for pour afficher chaque élément de la liste (itérez sur les indices)
6. Utilisez une boucle while pour parcourir cette liste et affichez chaque élément
7. Que vaut maListe[0] ? Affichez la valeur
8. Que vaut maListe[7] ? Affichez la valeur. Que remarquez-vous ?
9. Ajoutez False à cette liste
10. Refaites les deux questions précédentes. Que remarquez-vous ?
11. Affichez l'indice de 3.14
12. Supprimez la valeur située à l'indice 7 sans la retourner
13. Parcourez la liste et affichez chaque élément suivi de son type
14. Créer une liste nommée maListe2 égale à maListe
15. Ajoutez "toto" à maListe
16. Ajoutez "titi" à maListe2
17. Affichez maListe et maListe2. Que remarquez-vous ? Gardez cette remarque sous le coude…

III - b) (Re)découvrir l'immutabilité

1. Déclarez une variable contenant la chaîne de caractères « Sacré Graal ».
2. Affichez la longueur de cette chaîne, puis son dernier caractère
3. Remplacez-le par un autre caractère. Que se passe-t-il ?
4. Pourquoi, en Python, une **<str>** n'est pas une **<list>**, même si elle y ressemble ?
5. Proposez une solution alternative pour ce remplacement

III - c) Autres manipulations de listes et instructions Pythoniques

1. Créez une liste d'au moins 10 éléments
2. Affichez son dernier élément
3. Affichez les éléments des indices de 4 à 8 (faites-le de deux manières)
4. Affichez tous les éléments sauf le premier (faites-le de deux manières)
5. Affichez tous les éléments sauf le dernier (faites-le de deux manières)
6. Affichez tous les éléments sauf les 2 premiers et les 3 derniers (faites-le de deux manières)
7. Illustrez la manière dont se comportent les bornes en Python et comparez avec les maths
8. Créez une liste des 10 premiers entiers, et refaites pareil en 1 ligne
9. En une ligne, multipliez-les par deux et placez le résultat dans une nouvelle liste
10. Affichez le type de chaque élément de cette liste
11. Écrivez une fonction qui prend en paramètre une liste d'entiers et renvoie la même liste de
12. Strings. Faites pareil en une seule ligne. Comment s'appelle le fait de forcer la modification du type d'une variable ?
13. Recherche : comment travailler de manière indépendante sur deux listes A et B, B étant une copie de A ?
14. Comment afficher l'adresse mémoire d'un élément ? Créez explicitement deux chaînes de caractères identiques et examinez leurs adresses. Que se passe-t-il ?
15. Recherche : outre les listes, quels sont les autres types construits Python que vous connaissez ?
16. Quelles sont leurs principales propriétés ?