Type de document	Cours et exercices	Classe	1re	Durée	2h	Date	14/11/2022
Thème et contenu(s)	Langages et programmation – Structures linéaires de données						
Capacités attendues	Savoir manipuler une liste et ses éléments : recherche, modification ajout, suppression						
Prérequis	Déclaration de variables, types de base						
Description	Résumé du cours sur les structures linéaires de données						

I) Présentation et définition

On a parfois besoin de stocker un grand nombre d'éléments : les types de base (entier, flottant, booléen, chaîne de caractères) deviennent alors insuffisants. Par exemple, si on veut stocker 50 valeurs, il est fastidieux de créer manuellement 50 variable. On a donc besoin d'objets **conteneurs** pour les **rassembler** et les **manipuler** plus facilement : les **structures linéaires de données (SLD)** font partie de ces conteneurs.

Définition : Une **structure linéaire de données** est une structure dans laquelle on stocke des éléments de manière séquentielle (=en ligne). 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 ».

II) Listes

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

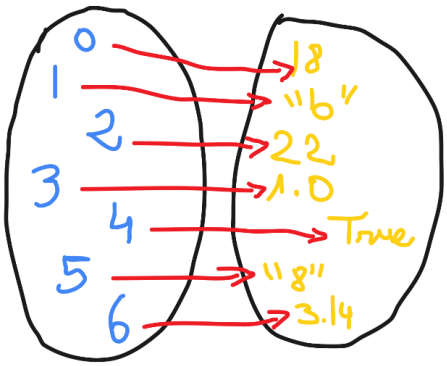
Cette partie du cours s'intéresse à une SLD particulière en Python : les **listes**. Une liste est une collection finie et ordonnée d'éléments (ces éléments pouvant être, pour simplifier, des valeurs de type entier, booléen, flottant ou chaîne de caractères). **Une liste à une dimension est une SLD**.

Dans une liste « simple », ou tableau, les éléments sont « rangés » les uns à la suite des autres, de manière contigüe (pas de « trous » entre les éléments). Pour chaque élément i d'une liste, sauf le dernier qui n'en possède pas, on connait même son successeur : i+1. Le premier élément a pour indice 0.

Sur la liste représentée ci-dessus, on a 7 éléments aux indices 0,1,2,3,4,5,6. On dit que la taille de cette liste est de 7 (elle contient 7 éléments). En bleu nous avons les indices (ils ne changent jamais, ce sont des entiers et on commence à les numéroter à partir de 0) et en jaune les valeurs (les éléments que l'on range dans une liste). Ces valeurs sont, dans le cas présent, de différents types.

II - a) 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.



III) Manipulations

III - a) Déclaration

Une liste vide se déclare simplement en Python : `maListe = []`
La syntaxe est semblable à celle de la création d'une variable, elle suit le schéma `variable = valeur` mais ici valeur est une liste, pas un type de base.
Une liste non vide se déclare en plaçant entre les crochets chaque élément séparé par une virgule.
Ex avec la liste `l` ci-dessous :

```
1  l = [18,"b",22,1.0,True,"8",3.14]
2  print(l)
```

```
[18, 'b', 22, 1.0, True, '8', 3.14]
```

On l'affiche tout simplement au moyen de la fonction `print()`. On utilisera cette liste dans la suite de nos exemples concernant les manipulations.

III - b) Accéder à l'élément d'une liste à un indice donné

On utilise la notation avec les crochets en mettant à l'intérieur l'indice voulu. Par exemple, `l[0]` contient la valeur `18`, `l[6]` contient la valeur `3.14`, etc…Attention : `l[7]` n'existe pas ! Y faire référence provoque une erreur et force le programme à s'arrêter.

```
1  l = [18,"b",22,1.0,True,"8",3.14]
2  print(l[0])
3  print(l[6])
4  print(l[7])
5
```

```
18
3.14
Traceback (most recent call last):
  File "<string>", line 4, in <module>
IndexError: list index out of range
```

III - c) Modifier un élément

La seule condition pour modifier un élément est que cet élément se trouve à un indice qui existe. La syntaxe est simple : `liste[indice] = valeur`
La ligne 4 du code ci-dessous montre la modification d'un élément existant.
Encore une fois, si on fait référence à un indice qui n'existe pas, on obtient une erreur.

```
1  l = [18,"b",22,1.0,True,"8",3.14]
2  print(l)
3  print(l[0])
4  l[0] = 99
5  print(l)
6  print(l[0])
7  l[7] = 12.99
8
```

```
[18, 'b', 22, 1.0, True, '8', 3.14]
18
[99, 'b', 22, 1.0, True, '8', 3.14]
99
Traceback (most recent call last):
  File "<string>", line 7, in <module>
IndexError: list assignment index out of
    range
```

III - d) Ajouter un élément

Pour ajouter un élément à une liste existante (vide ou pas) il n'y a qu'une seule solution.
On utilise la syntaxe : `liste.append(element)`
Les éléments s'ajoutent à chaque fois à la fin de la liste. L'exemple ci-dessous montre l'ajout d'un élément à une liste vide, et l'ajout d'un élément à notre liste d'exemple.

```
1  m = []
2  print(m)
3  m.append(458)
4  print(m)
5  l = [18,"b",22,1.0,True,"8",3.14]
6  print(l)
7  l.append("coucou")
8  print(l)
```

```
[]
[458]
[18, 'b', 22, 1.0, True, '8', 3.14]
[18, 'b', 22, 1.0, True, '8', 3.14, 'coucou']
>
>
>
>
```

III - e) Obtenir la taille d'une liste

La taille d'une liste est égale au nombre d'éléments qu'elle contient. On obtient la taille d'une liste en employant la fonction `len()` et en lui passant en paramètre la liste.
On pourrait faire pareil pour une chaîne de caractères. L'exemple suivant illustre les deux cas :

```
1  l = [18,"b",22,1.0,True,"8",3.14]
2  print("la longueur de l est", len(l))
3  s = "abracadabra"
4  print("la longueur de s est", len(s))
```

```
la longueur de l est 7
la longueur de s est 11
>
```

III - f) Parcourir une liste élément par élément

On utilise la même technique que pour chaîne de caractères, avec une boucle `for`. La syntaxe est :
`for <var> in <liste> :`
…
La variable `var` déclarée dans le for prendra successivement comme valeur celle de chaque élément de la liste, en commençant par le premier élément. L'exemple suivant montre que dans ce cas une chaîne de caractères fonctionne **un peu** comme une liste **(uniquement pour son parcours !)**.

```
1  l = [18,"b",22,1.0,True,"8",3.14]
2  print("parcours de la liste l:")
3  for e in l:
4      print(e)
5  s = "hello"
6  print("parcours de la chaîne s :")
7  for c in s:
8      print(c)
9
10
11
12
13
14
```

```
parcours de la liste l:
18
b
22
1.0
True
8
3.14
parcours de la chaîne s :
h
e
l
l
o
```

III - g) Parcourir une liste indice par indice

Il faut pour cela avoir compris les boucles bornées et les itérables : l'idée est de générer un itérable (il faut utiliser `range()`) à partir de la taille de la liste (il faut utiliser `len()`), ce qui équivaut à générer un itérable composé de tous les indices disponibles. La variable du `for` prendra successivement les valeurs des indices de notre liste, en démarrant à 0. À nous d'accéder aux élément avec la syntaxe : `l[indice]`
L'exemple suivant montre un parcours simple d'une liste par indice. J'y affiche l'indice `i` et l'élément correspondant auquel j'accède avec `l[i]`.

```
1  l = [18,"b",22,1.0,True,"8",3.14]
2  for i in range(len(l)):
3      print("indice :",i,"élément :",l[i])
4
5
6
7
```

```
indice : 0 élément : 18
indice : 1 élément : b
indice : 2 élément : 22
indice : 3 élément : 1.0
indice : 4 élément : True
indice : 5 élément : 8
indice : 6 élément : 3.14
```

III - h) Suppression des éléments d'une liste

On peut supprimer un élément à un indice donné, on utilise pour cela la syntaxe :
`liste.pop(indice)`
Dans ce premier cas, on doit s'assurer que l'indice existe.
Le fait d'enlever des éléments à une liste réduit l'ensemble des indices disponibles : si j'ai une liste de 7 éléments, mes indices vont de 0 à 6.

Si j'enlève deux éléments, il m'en reste donc 5, mes indices vont désormais de 0 à 4. Il n'y a rien à l'indice 5, tenter d'y accéder me renvoie une erreur.
L'exemple de code suivant illustre ceci.

```
1  l = [18,"b",22,1.0,True,"8",3.14]
2  print(l)
3  l.pop(0)
4  print(l)
5  l.pop(1)
6  print(l)
```

```
[18, 'b', 22, 1.0, True, '8', 3.14]
['b', 22, 1.0, True, '8', 3.14]
['b', 1.0, True, '8', 3.14]
Traceback (most recent call last):
  File "<string>", line 7, in <module>
IndexError: pop index out of range
```

On peut aussi supprimer un élément par valeur (on donne la valeur de l'élément que l'on veut supprimer), avec la syntaxe :
`liste.remove(valeur)`
Si l'élément n'existe pas (ou plus) dans la liste, on aura une erreur, comme ci-dessous :

```
1  l = [18,"b",22,1.0,True,"8",3.14]
2  print(l)
3  l.remove("8")
4  print(l)
5  l.remove(1.0)
6  print(l)
7  l.remove("8")
```

```
[18, 'b', 22, 1.0, True, '8', 3.14]
[18, 'b', 22, 1.0, True, 3.14]
[18, 'b', 22, True, 3.14]
Traceback (most recent call last):
  File "<string>", line 7, in <module>
ValueError: list.remove(x): x not in list
>
```

IV) Exercices de base corrigés

IV - a) Parcours et modification

1. Déclarer une liste vide `k`
2. Déclarer une liste `l` composée de 4 éléments `Str`
3. Placer dans une variable la longueur de `l`
4. Utilisez cette variable dans une boucle bornée pour parcourir `l` par indice
5. Mettre la chaîne de caractères « `toto` » à l'indice 3 de `l`
6. Parcourir `l` par élément

```
1  k = [] #liste k vide
2  l = ["yo", "ya", "you", "ye"] #liste l
3  v = len(l) #v contient la taille de l
4  for i in range(v): #parcours par indice
5      print(l[i])
6  l[3] = "toto" #on met "toto" à l[3]
7  for e in l: #parcours par élément
8      print(e)
9
```

```
yo
ya
you
ye
yo
ya
you
toto
>
```

IV - b) Ajout d'éléments dans des listes déjà existantes

Créer une liste vide l, y ajouter les valeurs « coucou », « toto », « tutu ». Créer une liste vide m. Avec une boucle, ajoutez chaque élément de l dans m. Faites-le de deux manières : par indice et par élément.

```
l = []
l.append("coucou")
l.append("toto")
l.append("tutu")
m = []
for e in l:
    m.append(e)
#par indice on aurait eu
#for i in range(len(l)):
#    m.append(l[i])
print(m)
```

```
['coucou', 'toto', 'tutu']
>
```

IV - c) Suppression

Recréer la liste de ce support de cours et supprimer l'élément à l'indice 0, l'élément « 3.14 », et afficher la liste.

```
l = [18,"b",22,1.0,True,"8",3.14]
l.pop(0)
l.remove(3.14)
print(l)
```

```
['b', 22, 1.0, True, '8']
>
```

IV - d) Échanger des valeurs dans une liste

Créer une liste à deux éléments et échanger les valeurs. Afficher la liste avant et après.

```
k = ["salut", "coucou"]
print(k)
a = k[0]
k[0] = k[1]
k[1] = a
print(k)
```

```
['salut', 'coucou']
['coucou', 'salut']
>
```

V) Exercice facile corrigé

V - a) Remplissage d'une liste

1. Créer une boucle qui fait 5 itérations et qui à chacune demande à l'utilisateur d'entrer un nombre entier (on suppose que l'utilisateur respecte cela).
2. Placer chaque élément au fur et à mesure dans une liste que vous aurez pris soin de déclarer au préalable.
3. Parcourir la liste par élément et afficher chaque élément.
4. Créer une nouvelle liste avec tous éléments **impairs** de la première
5. Afficher cette nouvelle liste
6. Redéclarer la première liste pour qu'elle ne contienne que ces éléments et l'afficher

```python
l = [] #on crée une liste vide
for i in range(5): #on itère 5 fois
    a = input("entre un nombre entier stp : ") #entrée clavier
    a = int(a) #conversion en entier
    l.append(a) #ajout de cet élément à la liste
for j in range(len(l)):
    #pas demandé mais j'améliore l'affichage
    print("élément à l'indice i :", l[j])
m = []
for e in l:
    print(e,e%2)
    if e%2 != 0:
        m.append(e)
print(m)
l = m
print("liste avec éléments impairs : ", l)

```

```
entre un nombre entier stp : 14
entre un nombre entier stp : 8
entre un nombre entier stp : 13
entre un nombre entier stp : 75
entre un nombre entier stp : 2
élément à l'indice i : 14
élément à l'indice i : 8
élément à l'indice i : 13
élément à l'indice i : 75
élément à l'indice i : 2
14 0
8 0
13 1
75 1
2 0
[13, 75]
liste avec éléments impairs :  [13, 75]
```