Thème 1

Structures de données

Partie 1 : Tableau & Dictionnaire
Partie 2 : Programmation Orientée Objet.
Partie 3 : Liste, Pile, File
Partie 4 : Arbres
Partie 5 : Graphes

P1

Tableau & Dictionnaire

Tableaux (type `list`)

Partie 1 : Tableaux & Dictionnaires

Objectifs :

- Connaître l'interface d'un tableau,
- Créer un tableau,
- Accéder et modifier un élément,
- Ajouter des éléments,
- Parcourir les éléments d'un tableau,
- Connaître les algorithmes de base sur les tableaux.

Définition

Définition

En informatique, un tableau est une structure de données représentant une séquence finie d'éléments auxquels on peut accéder efficacement par leur position, nommée **indice**, dans la séquence.

i Python implemente les tableaux dans le type `list`.
De ce fait, il y a une confusion entre les listes, telles qu'elles sont définies en informatique (nous verrons cette définition plus tard dans l'année) et le type `list` de Python.

Interface

i Dans le domaine des structures de données, **l'interface** désigne l'ensemble des opérations qui sont possibles avec cette structure de donnée.
Elle se distingue de son **implantation** qui désigne la mise en oeuvre pratique dans un langage donné.

Dans l'interface d'un tableau, nous pouvons citer les opérations suivantes :

1. Créer un tableau vide ou non,
2. Mesurer un tableau : combien d'entrées comporte-t-il ?
3. Accéder à une valeur depuis son index.
4. Ajouter / supprimer un élément .
5. Parcourir les éléments

Implémentation

• Initialisation d'un `tableau`

En python, les tableaux sont contenus dans des crochets **[** ... **]**
Les éléments sont séparés par des virgules **,**

[élément1 **,** élément2 **,** élément3 **]**

•• Tableau vide

Il est possible de créer un tableau vide

```
# Créer un tableau vide
mon_tableau = []
```

•• Tableau non vide

Il est aussi possible de l'initialiser avec des valeurs

```
# Créer un tableau à partir de valeurs
>>> mon_tableau = ["Alan", "Ada", "Grace", "Ian"]
["Alan", "Ada", "Grace", "Ian"]
```

Enfin si l'on veut, il est aussi possible de l'initialiser avec les mêmes valeurs

```
# Créer un tableau contenant 5 zéros
>>> mon_tableau = [0]*5
[0, 0 , 0, 0, 0]
```

•• Accéder à un élément

Pour accéder à un élément, on utilise son **`index`** (sa position dans la liste)

⚠ L'index d'une liste commence à 0.

`Tableau` : ["Alan"$_0$, "Ada"$_1$, "Grace"$_2$, "Ian"$_3$]

L'instruction pour accéder à un élément est construite à partir du nom du tableau et de l'index (*la position*) de l'élément souhaité entre crochets.

Exemple

Le tableau s'appelle **`mon_tableau`** et l'élément **`Ada`** est en position **1**.
On accède donc à cet élément avec l'instruction :
mon_tableau[1]

```
>>> mon_tableau = ["Alan", "Ada", "Grace", "Ian"]
>>> print(mon_tableau[1])
"Ada"
```

Les index négatifs

Comment accéder aux derniers éléments d'un tableau

Pour atteindre le dernier élément d'un tableau, il faudrait connaître le nombre total d'éléments du tableau (ne utilisant la fonction `len`) et lui soustraire 1.

Exemple

On considère une liste `notes` qui contient les notes obtenues à différents contrôles par un élève. Pour afficher à la dernière note obtenue, on doit en pratique avoir le code suivant :

```
1 notes = [12, 14, 11, 16, 17, 15, 18, 15, 19, 14, 17]
2 nb_notes = len(notes)
3 print(notes[nb_notes - 1])
```

La variable `nb_notes` ne servant pas par la suite, on pourrait aussi écrire :

```
1 notes = [12, 14, 11, 16, 17, 15, 18, 15, 19, 14, 17]
2 print( notes[len(notes) - 1])
```

Pour alléger l'écriture, Python propose une raccourci, on peut écrire plus directement :

```
1 notes = [12, 14, 11, 16, 17, 15, 18, 15, 19, 14, 17]
2 print( notes[-1])
```

Exercices

Exercice 1

On considère le code suivant :

```
1 tab = ["Alan", "Ada", "Grace", "Ian", "Charles", "Guido"]
```

Que vaut `tab[3]` ?

.....

.....

Exercice 2

Après l'affectation suivante :

```
1 alphabet = [ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
2 'M', 'N','O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' ]
```

Quelle instruction permet d'accéder à la lettre E ?

.....

Quelle instruction permet d'accéder à la lettre Y ?

.....

• Modification d'un élément

On peut modifier les éléments de la liste en affectant une nouvelle valeurs.

Exemple

```
>>> mon_tableau = ["Alan", "Ada", "Grace", "Ian"}
>>> mon_tableau[1] = "Ada Lovelace"
>>> print(mon_tableau)
["Alan", "Ada Lovelace", "Grace", "Ian"]
```

•• Exercices

Exercice 3

On considère le script suivant :

```
tab = [2, 8, 9, 2]
tab[2] = 4
```

Quelle est la valeur de `tab` à la fin de son exécution ?

.....

Exercice 4

On considère le script suivant :

```
tab = [2, 8, 9, 2]
tab[2] = tab[2] + 5
```

Quelle est la valeur de `tab` à la fin de son exécution ?

.....

Exercice 5

On considère le script suivant :

```
tab = [2, 8, 9, 2]
tab[3] = tab[1] + 5
tab[2] = tab[3] + 5
```

Quelle est la valeur de `tab` à la fin de son exécution ?

.....

• Ajouter des éléments

i Il existe plusieurs façons d'ajouter des éléments à un tableau donné.
On peut :

- ajouter un élement,
- étendre un tableau à partir d'un second tableaux,
- concatener 2 tableaux pour en former un troisième.

•• Ajouter un élément

Pour ajouter un **seul élément**, on utilise la méthode `append`. La méthode `append` modifie sur place l'objet tableau qui l'appelle.

```
mon_tableau = ["Alan", "Ada", "Grace", "Ian"]
mon_tableau.append("George")
print(mon_tableau)
["Alan", "Ada", "Grace", "Ian", "George"]
```

•• Ajout par extension de liste

Pour ajouter un tableau d'élément, on utilise la méthode `extend`. La méthode `extend` modifie sur place l'objet tableau qui l'appelle.

```
mon_tableau = ["Alan", "Ada", "Grace", "Ian"]
tableau_2 = ["Charles", "Guido"]
mon_tableau.extend(tableau_2)
print(mon_tableau)
["Alan", "Ada", "Grace", "Ian", "Charles", "Guido"]
```

•• Ajout par concaténation

Il est aussi possible de concaténer 2 listes pour en créer une troisième.

```
tableau_1 = ["Alan", "Ada", "Grace", "Ian"]
tableau_2 = ["Charles", "Guido"]
mon_tableau = tableau_1 + tableau_2
print(mon_tableau)
["Alan", "Ada", "Grace", "Ian", "Charles", "Guido"]
```

•• Exercices

Exercice 6

Qu'affiche le programme python suivant ?

```
tab1 = [3,2,2]
tab2 = [1,5,1]
tab1.append(tab2)
print(tab1)
```

. . . . . .

. . . . . .

• Parcourir un tableau

Il existe deux manières de parcourir les éléments d'une liste :

- Parcourir une liste à partir des `index` des éléments
- Parcourir une liste par itération sur les éléments

•• Parcours à partir de l'index des éléments

i En python, les tableaux (list) fonction taille **len()** donnant le nombre d'éléments contenus dans la liste.
Associer à la fonction `range()`, on obtient un itérable composé de tous les index des éléments de la liste.

Exemple

```
1 mon_tableau = ["Alan", "Ada", "Grace", "Ian", "Charles", "Guido"]
2 for index in range(len(mon_tableau)):
3     print(mon_tableau[index])
```

Dans cet exemple, la longueur de `mon_tableau` est 6.
L'instruction `range(len(mon_tableau))` va être successivement évaluée par `range(6)` puis par un itérable de longueur 6 qui contiendra les valeurs `0, 1, 2, 3, 4` et 5.
Ainsi la variable index prendra successivement l'ensemble de ces valeurs.

•• Itération des éléments

i Comme pour tout les itérables, il est possible de parcourir un tableau en itérant les éléments du tableau, avec les instructions `for v in iterable`.

.

Exemple

```
1 mon_tableau = ["Alan", "Ada", "Grace", "Ian", "Charles", "Guido"]
2 for elmt in mon_tableau:
3     print(elmt)
```

Dans cet exemple, la variable `elmt` prendra successivement les valeurs contenues dans la variable `mon_tableau`, c'est à dire `"Alan"`, `"Ada"`, `"Grace"`, `"Ian"`, `"Charles"` et `"Guido"`

❝ Il est souvent utile de bien nommée la variable créée dans la boucle `for`. Ainsi pour le code précédent, il aurait été préférable d'écrire :

```
1 mon_tableau = ["Alan", "Ada", "Grace", "Ian", "Charles", "Guido"]
2 for informaticien in mon_tableau:
3     print(informaticien)
```

Exercice 7

On considère le code suivant :

```
def feed(t):
    for i in range(len(t)):
        t[i] = t[i] + i
    return t
```

Que renvoie `feed([12, 24, 32])` ?

.....

.....

•• Les méthodes

Dans toute cette section, on considèrera la liste **`nombres`**. Les instructions sont considérées être effectuées les une à la suites des autres.

```
nombres = [17, 38, 10, 25, 72]
```

1. Ordonner la liste

```
# La méthode sort permet de classer les éléments de la suite
nombres.sort()
print(nombres)
# Affiche  [10, 17, 25, 38, 72]
```

2. Inverser les éléments d'un tableau

```
nombres.reverse()
print(nombres)
# Affiche  [72, 38, 25, 17, 10]
```

3. Supprimer des éléments

```
nombres.remove(38)
print(nombres)
# Affiche [72, 25, 17, 10]
# L'instruction pop enlève et renvoi le dernier élément de la liste
print(nombres.pop())
# Affiche 10
print(nombres)
# Affiche la liste [72, 25, 17 ]
```

4. Information sur un élément

```
# On peut récupérer la position d'un élément
print(nombres.index(17))
# Affiche  2
liste = [72, 25, 17, 25 ]
compte le nombre d'occurrence d'un élément
print(liste.count(25))
# Affiche : 2
```

Les compréhensions de listes

•• Syntaxe de base

Un tableau par compréhension (ou liste en intension) est une expression qui permet de générer un tableau de manière très compacte. Cette notation reprend la définition mathématique d'un ensemble en intension :

$$\{x^2 | x \in [\![2, 10]\!]\} \Longrightarrow \text{[x**2 for x in range(2, 11)]}$$

Un tableau en compréhension `tab` a la syntaxe minimale suivante :

```
1    tab = [expr for elmt in iterable]
```

⚠ **La paire de crochets**, les mots-clefs `for` et `in` sont obligatoires.

- `iterable` est, assez souvent, un objet de type `list`, un `range` ou une chaine de caractères,
- `elmt` est l'élément courant qui parcourt l'élément `iterable` ;
  `elmt` est appelé **variable de contrôle** de la liste en compréhension,
- `expr` est une expression qui dépend en général de `elmt`, mais pas forcement, et dont la valeur est placée dans `tab`

Exemple

On veut créer un tableau (`tab`) contenant les carrés des entiers préalablement stockés dans un tableau (`entiers`)

```
1  entiers = [0,1,2,3,4,5]
2  tab = [elmt*2 for elmt in entiers]
3  print(tab)
4  # Affiche [0,1,4,9,16,25]
```

Description : On créer un tableau `L` contenant le double (`elmt*2`) de tous les éléments se trouvant dans la listes `entiers` (`for elmt in entiers`).

•• Exercices

Exercice 8

Déterminer le contenu de la variable `L` après exécution du code ci-dessous.

```
1  entiers = [5, 8, 12, 9]
2  L = [ elmt**2 for elmt in entiers]
```

.....

.....

Exercice 9

Compléter le code ci-dessous pour que la variable `L` contienne les 5 premiers multiple de 3.

```
entiers = [0, 1, 2, 3, 4]
L = [ ..... for ..... in entiers]
```

.....

.....

Exercice 10

Soit `lettres = ["A","b","C","d"]`.
Écrire un programme qui remplace chaque lettre de la liste par son code ASCII.
On pourra utiliser l'instruction `ord` pour obtenir le code ASCII d'un caractère

.....

.....

•• Filtrer un tableau

On peut rajouter une condition, pour ne garder que certains éléments de la liste initiale.

```
L = [expr for elmt in t if condition]
```

Exemple

On se donne la liste des entiers inférieurs ou égaux à 20 et on veut filtrer uniquement ceux qui sont pairs.

```
entiers = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
L = [elmt for elmt in entiers if elmt % 2 == 0]
print(L)
# Affiche [0,2,4,6,8,10,12,14,16,18,20]
```

Description : On créer un tableau `L` contenant les éléments (`elmt`) se trouvant dans la listes `entiers` (`for elmt in entiers`) à condition que le reste de leur division euclidienne par 2 soit nul (`elmt % 2 == 0`).

Exercice 11

Que va contenir la variable `mon_tab` après l'exécution de ce programme.

```
l = [1, 7, 9, 15, 5, 20, 10, 8]
mon_tab = [p for p in l if p > 10]
```

.....

.....

Exercice 12 Multiple

Compléter le code ci-dessous pour que la liste `L` contienne les entiers inférieurs ou égaux à 100 qui sont des multiples 5.

```
1  L =  [ m for m in range(...) if  ..............  ]
```

Exercice 13 FizzBuzz

Décrire en compréhension la liste `fizzbuzz` des entiers inférieurs à 100 qui sont multiples de 5 ou de 7 mais pas de 5 et 7.

. . . . . .

. . . . . .

. . . . . .

Exercice 14 Bissextile

Depuis l'ajustement du calendrier grégorien, l'année n'est bissextile (elle aura 366 jours) que dans l'un des deux cas suivants :

- si l'année est divisible par 4 et non divisible par 100 ;
- si l'année est divisible par 400.

Décrire par compréhension la liste des années bissextile de 1600 à aujourd'hui (2022).

. . . . . .

. . . . . .

. . . . . .

. . . . . .