LES TABLEAUX : DU MODELE ABSTRAIT AUX LISTES PYTHON

0. Introduction : pourquoi regrouper des données ?

Pour stocker les âges de trois personnes, on pourrait utiliser plusieurs variables :

age1 = 17
age2 = 16
age3 = 18

Cette solution devient peu pratique lorsque le nombre de valeurs augmente. On a besoin d'une structure permettant de regrouper plusieurs éléments, de les organiser et de les retrouver facilement.

Dans ce chapitre, nous étudions le modèle du tableau, dont tous les éléments sont de même type. En Python, nous utiliserons des listes pour représenter ces tableaux. Nous nous limiterons aux opérations nécessaires à l'étude de cette structure.

1. Le tableau : une structure de données abstraite

1.1. Définition générale

Dans ce chapitre, on étudie des tableaux dont tous les éléments sont de même type.

Exemple conceptuel : une suite de températures mesurées pendant quatre jours.

Indice	**0**	**1**	**2**	**3**
Température	18	21	19	22

L'ordre compte : l'élément d'indice 0 est le premier, celui d'indice 1 est le deuxième, etc. Dans le modèle utilisé ici, les indices commencent à 0.

CC BY NC ND

1.2. Les propriétés essentielles

Ordre : les éléments occupent des positions distinctes et leur ordre est significatif.

Indice : chaque position est repérée par un indice qui permet d'accéder à l'élément correspondant.

Type des éléments : dans le modèle classique étudié, les éléments sont de même type (par exemple, tous des entiers ou toutes des températures).

Taille : le tableau contient un certain nombre d'éléments. Dans le modèle classique du tableau, cette taille est souvent fixée à la création.

1.3. Opérations envisagées sur un tableau

Sans encore parler de Python, on peut identifier plusieurs opérations utiles :

- Consulter l'élément situé à un indice donné ;
- Modifier la valeur d'un élément ;
- Connaître le nombre d'éléments ;
- Parcourir les éléments dans l'ordre.

Cette description générale permet de séparer le modèle de données des outils concrets qu'un langage de programmation met à disposition.

À toi de jouer :

Avant de passer à Python, observe le tableau conceptuel suivant : `[14,17,12,19]`.

1°) Quelle est la valeur de l'indice 0 ? 14

2°) Quelle valeur se trouve à l'indice 2 ? 12

3°) Combien d'éléments contient le tableau ? 4

4°) Si l'on échange les deux premières valeurs, le tableau est-il inchangé ? Expliquer-en une phrase.

Non, le tableau n'est pas inchangé car l'ordre des valeurs change.

Il devient `[17,14,12,19]`.

CC BY NC ND

2. L'implémentation en Python

2.1. Créer un tableau et accéder à ses éléments

On crée un tableau en plaçant ses éléments entre crochets et en les séparant par des virgules.

```
t = [18, 21, 19, 22]
print(t[0]) # Affiche 18
print(t[2]) # Affiche 19
```

En Python, les indices commencent à 0. Pour un tableau de n éléments, les indices positifs vont de 0 à n − 1.

Premier élément : indice 0.

Deuxième élément : indice 1.

Dernier élément : indice n − 1.

2.2. Connaître la longueur

La fonction len() renvoie le nombre d'éléments d'un tableau.

```
t = [18, 21, 19, 22]
print(len(t)) # Affiche 4
```

2.3. Indices négatifs

Python permet aussi d'utiliser des indices négatifs, qui comptent depuis la fin du tableau.

```
t = [18, 21, 19, 22]
print(t[-1]) # Affiche 22
print(t[-2]) # Affiche 19
```

L'indice −1 désigne le dernier élément, −2 l'avant-dernier, etc. Cette possibilité est propre à la manière dont Python permet d'accéder aux tableaux ; elle ne fait pas partie de la définition abstraite présentée plus haut.

À toi de jouer :

On considère le tableau : `t = [7, 12, 5, 9, 14]`

1. Sans exécuter le programme, donne les valeurs de `t[0]`, `t[3]` et `t[-1]`.

`t[0]=7 ; t[3]=9 ; t[-1]=14`

CC BY NC ND

2. Quelle instruction permet d'obtenir le nombre d'éléments de t ? len(t)

3. Quel indice permet d'accéder à la valeur 5 ? t[2]

3. Les opérations sur les tableaux en Python



3.1. Modifier un élément

On peut remplacer la valeur d'un élément en utilisant son indice et une affectation.

```
t = [18, 21, 19, 22]
t[1] = 20
print(t) # Affiche [18, 20, 19, 22]
```

On peut aussi calculer une nouvelle valeur à partir de l'ancienne.

```
t[0] = t[0] + 1
# Équivalent à : t[0] += 1
```

3.2. Parcourir une partie d'un tableau

Supposons que pda[i] représente le nombre de personnes âgées de i années. Pour additionner les effectifs des âges de 10 à 19 ans inclus :

```
n = 0
for i in range(10, 20):
     n = n + pda[i]
```

La variable n est un accumulateur : elle conserve progressivement le résultat. range(10, 20) produit les entiers de 10 à 19 ; la borne de fin est exclue.

On peut écrire n += pda[i], ce qui est équivalent à n = n + pda[i].

3.3. Parcourir tout un tableau

```
t = [2, 5, 8, 3]
for i in range(len(t)):
   print(t[i])
```

La boucle parcourt les indices 0, 1, 2 et 3. L'expression range(len(t)) permet d'adapter le parcours à la longueur du tableau.

cc BY NC ND

3.4. Créer et remplir un tableau

On peut créer un tableau contenant plusieurs fois la même valeur grâce à l'opérateur *.

t = [0] * 100

Cette instruction crée un tableau de 1 00 éléments initialisés à 0. On peut ensuite remplir ses cases avec une boucle.

```
t = [0] * 100
for i in range(100):
   t[i] = i**2
```

Après la boucle, t[0] vaut 0, t[1] vaut 1, t[2] vaut 4 et t[3] vaut 9,…

3.5. Créer et remplir un tableau par compréhension

La notation par compréhension permet de créer un tableau en utilisant des crochets [] et une boucle for pour ajouter des éléments.

```
t=[2*i for i in range (10)]

print(t)

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

4. Tableau et fonctions



4.1. Renvoyer un tableau

Une fonction peut construire un tableau et le renvoyer avec return.

Exemple : créer un tableau contenant les carrés des entiers de 0 à n − 1.

```
def carres(n):
      t = [0] * n
      for i in range(n):
         t[i] = i**2
      return t

resultat = carres(4)
print(resultat) # Affiche [0, 1, 4, 9]
```


CC BY NC ND

Bilan du chapitre

Opération en Python	Exemple	Rôle
Créer un tableau	`t = [2, 5, 8]`	Regrouper des valeurs
Accéder à un élément	`t[1]`	Lire une case
Modifier un élément	`t[1] = 10`	Changer une valeur
Connaître la longueur	`len(t)`	Obtenir le nombre d'éléments
Parcourir	`for i in range(len(t)):`	Examiner les éléments successivement
Initialiser	`t = [0] * 100`	Créer un tableau initialisé à 0
Renvoyer un tableau	`return t`	Transmettre un tableau en résultat

BY NC ND