Dans ce chapitre, on s'attarde sur des fonctionnalités avancées de Python pour la manipulation de tableaux ainsi que sur les tableaux à double entrée.

1 Fonctionnalités avancées sur les tableaux

1.1 Ajouter des éléments à un tableau

Jusqu'à maintenant, on manipulait les tableaux, dans l'idée, comme on manipule les tableaux dans la plupart des langages de programmation :

1. création du tableau de taille donnée (`t = [0]*10`) ;
2. accès aux valeurs (`t[2] = 7`).

Il existe cependant d'autres manières de créer des tableaux, dont la première repose sur la méthode `append`. Tester le programme suivant :

```
t = []
t.append(2)
t.append(4)
t.append(8)
```

On constate que la commande `t.append(x)` ajoute à la fin du tableau `t` la valeur `x`. Attention cependant, en utilisant cette commande, la longueur du tableau est modifiée !

1.2 Itérer sur les éléments d'un tableau

Pour parcourir un tableau, la manière habituelle (et commune dans l'idée aux autres langages) est la suivante :

```
for i in range(len(t)):
    print(t[i])
```

Tester le programme suivant :

```
for x in t:
    print(x)
```

On voit que ce programme produit exactement le même affichage. La déférence entre les deux boucles est l'**itérateur**. Dans le premier programme, il s'agissait de `i` l'**indice** de la boucle valant successivement 0, 1, 2 etc. alors que dans le deuxième programme, il s'agît de `x` prenant dans l'ordre toutes les **valeurs** du tableau `t[0]`, `t[1]`, `t[2]` etc.

Cette écriture est plus rapide et plus intuitive à utiliser (on peut y lire en français « pour tout x de t »). Elle ne sera cependant pas applicable si :

— on veut parcourir le tableau dans un ordre particulier ;

— de manière générale dès qu'on a besoin des positions des éléments du tableau.

1.3 Construction de tableaux par compréhension

Dernière méthode pour créer des tableaux, la compréhension. Il s'agit de décrire en une seule commande la taille et le contenu d'un tableau. Tester le programme suivant :

```
t = [3*i+2 for i in range(10)]
print(t)
```

On voit que le tableau `t` contient les valeurs de la forme $3i + 1$ pour tout i entre 0 et 9. Tester le programme suivant :

```
t = [3*i+2 for i in range(10) if i%2 == 0]
print(t)
```

On a réduit les i pour lesquels on insère les valeurs dans le tableaux aux seuls nombres pairs entre 0 et 9.

Ici on a itéré sur un indice dans la construction par compréhension. On peut cependant choisir d'itérer sur les valeurs d'un autre tableau. Exemple à tester :

```
t1 = [4,5,6]
t2 = [x*x for x in t]
print(t2)
```

Cette méthode est utile pour appliquer une certaine transformation (par exemple en statistiques) à une certaine liste de données.

2 Tableaux à double entrée

2.1 Lignes, colonnes, coordonnées

Un tableau à double entrée est une grille de valeurs organisées selon deux directions, en **ligne** et en **colonne**. En voici un exemple de représentation :

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

Afin de ce repérer dans ce tableau, nous allons en numéroter les lignes et les colonnes. Ainsi la valeur J est ici positionnée à l'intersection de la ligne 1 et de la colonne 3, on parle aussi de ses coordonnées (1,3).

	0	1	2	3	4
0	A	B	C	D	E
1	F	G	H	I	J
2	K	L	M	N	O

Attention à la représentation graphique des tableaux : par convention on donne les coordonnées sous la forme **(n° ligne, n° colonne)** dans cet ordre comme dans le plan on donne **(abscisse, ordonnée)** pour décrire les coordonnées d'un point. Cependant, les lignes se comptent verticalement et en descendant alors que les colonnes se comptent horizontalement.

2.2 Implémentation et accès

Sur l'exemple précédent, on peut remarquer que la grille peut être intuitivement coupée en trois tableaux simples `t0 = [A, B, C, D, E]`, `t1 = [F, G, H, I, J]`, et `t2 = [K, L, M, N, O]`. Ces trois tableaux représentent les lignes de la grille.

Dès lors on peut imaginer une implémentation cette grille comme la donnée de ces trois lignes, elle même sous forme de tableau :

```
grille = [t0, t1, t2]
```

Un tableau à double entrée peut donc être implémenté comme **un tableau de tableaux**. Voyons comment accéder aux valeurs de cette grille :

— Écrire `grille[1]` nous donne accès, en lecture ou en écriture, au tableau t1 : la ligne numéro 1 de la grille.

```
grille[1] = t1 = [F, G, H, I, J]
```

— Afin d'accéder, toujours en lecture ou en écriture et par exemple, à la donnée `I`, il faut utiliser `t1[3]` soit encore `grille[1][3]`. Notons ici que 1 et 3 sont précisément les coordonnées de la valeur I de la grille.

```
grille[1][3] = t1[3] = I
```

> Dans un tableau à double entrée `g`, l'appel de `g[i]` donne accès au tableau représentant la ligne `i` et l'appel de `g[i][j]` donne directement accès à la valeur de coordonnées (i, j) de la grille. En revanche, il n'est pas possible d'avoir directement accès à une colonne donnée.

2.3 Création de tableaux à double entrée

Construction de grilles remplies de `0` On veut créer un tableau à double entrée `grille` de taille $n \times m$ (n lignes m colonnes), dans un premier temps disons rempli de `0`. Détaillons étape par étape :

1. On commence par créer le tableau vide `grille` qui va contenir les lignes du tableau.
2. Puis à l'aide d'une boucle `for` de taille n, on ajoute une par une les lignes remplies de m valeurs 0.

Notons aussi qu'une méthode plus rapide est permise par la construction de tableau par compréhension.

```
#etape par etape avec des append
grille = []
for i in range(n):
    ligne = [0]*m
    grille.append(ligne)

#en une seule ligne par comprehension
grille = [[0]*m for i in range(n)]
```

Construction de grille avec formule Si on veut directement remplir les valeurs de la grilles à l'aide d'une formule $f(i,j)$, on peut utiliser les deux méthodes suivantes :

```
#etape par etape avec des append
grille = []
for i in range(n):
    ligne = []
    for j in range(m)
        ligne.append(f(i,j))
    grille.append(ligne)

#en une seule ligne par double comprehension
grille = [[f(i,j) for j in range(m)] for i in range(n)]
```

2.4 Parcours de tableaux à double entrée

La méthode de base pour parcourir un tableau à double entrée est l'utilisation de deux boucles imbriquées :

1. la première, généralement d'indice `i=0...n-1`, sert à traiter chaque ligne de la grille ;
2. la seconde, généralement d'indice `j=0...m-1`, sert à traiter chaque valeur de la ligne courante.

Par exemple, on peut écrire une fonction basique d'affichage :

```
def afficher(g):
    """affiche le contenu de la grille g"""
    for i in range(len(g)):
        for j in range(len(g[i])):
            print(g[i][j], end = " ")
        print()
```

Remarque la hauteur n d'un tableau à double entrée s'obtient à l'aide de `len(grille)` (nombre de ligne). Sa largeur s'obtient à l'aide de `len(grille[i])` (taille d'une ligne) quelque soit le `i` considéré.