Utilisation avancée des tableaux

I Itérer sur les éléments d'un tableau

Lorsque l'on veut répéter une opération pour chaque élément **e** d'un tableau **t**, il est possible d'utiliser l'instruction **for** directement sur le tableau lui-même :

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

La boucle énumère tous les éléments du tableau : elle effectue un tour pour chaque élément de **t** et l'élément inspecté est associé à la variable **e**. Le programme précédent est donc équivalent à ce que nous avions précédemment vu lors de la récupération par l'indice :

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

On utilise aussi parfois une boucle **for** en fournissant directement le tableau des éléments à traiter tel que dans l'exemple ci-dessous :

```
print("Les directions possibles sont :")
for d in ["Nord", "Sud", "Ouest", "Est"]:
        print("*",d)
```

```
Les directions possibles sont :
* Nord
* Sud
* Ouest
* Est
```

Attention

Une itération directe sur les éléments d'un tableau ne tient plus en compte l'indice de chacun des éléments. Cette approche n'est donc utile que lorsqu'il n'est pas nécessaire de connaître la position i d'un élément d'un tableau.

II Construire un tableau par compréhension

Nous avions vu précédemment que pour initialiser un grand tableau, il était nécessaire de l'allouer dans un premier temps avec une valeur arbitraire, puis effectuer son remplissage (ici par 3*i + 1) :

```
t = [0] * 100
for i in range(len(t)):
        t[i] = 3 * i + 1
```

Python propose une syntaxe pour effectuer simultanément l'allocation en mémoire et le remplissage par une boucle :

```
[3 * i + 1 for i in range(100)]
```

Cette nouvelle approche mélange les crochets définissant notre tableau, ainsi que la structure de la boucle **for**. On appel cela la *notation par compréhension*. Le parcours de la variable **i** n'est pas limité à un intervalle d'entier construit avec **range**. On peut alors parcourir un autre tableau déjà construit tel que :

```python
t = [2 * i for i in range(5)]
print([x * x for x in t])
```

```
[0, 4, 16, 36, 64]
```

On peut même choisir de ne conserver que certaines valeurs en ajoutant une condition booléenne à la compréhension du tableau :

```python
print([x * x for x in range(10) if i % 2])
```

```
[1, 9, 25, 49, 81]
```

Ici, on ne conserve que les résultats impaire grâce à l'opérateur *modulo*.

III Tableaux à plusieurs dimensions

Les tableaux en Python peuvent contenir toute sorte de valeurs. On peut même construire un tableau contenant lui-même des tableaux :

```
t = [[0, 1, 2], [1, 0, 3], [3, 9, 2], [0, 0, 1]]
```

Pour accéder à un entier contenu dans ce tableau de tableaux, on commence par accéder à l'un des trois tableaux, puis l'on accède à l'un de ses éléments :

```python
print(t[2][1])
```

```
9
```

Ces tableaux sont appelés *tableaux à plusieurs dimensions*. Dans le cas précédent, nous avons deux dimensions, la première de taille 4 et la seconde de taille 3. On peut représenter ce tableau graphiquement :

0 1 2
1 0 3
3 9 2
0 0 1

La première dimension est représentée verticalement, et la seconde dimension est représentée horizontalement. Pour modifier un élément, on effectuer une affectation comme on le ferait avec un tableau à une seule dimension :

```
t[3][1] = 7
```

Lorsque le nombre de ligne est égal au nombre de colonne, on appel ce tableau une matrice. Néanmoins, la taille des tableaux dans un tableau principal n'est pas forcément la même, comme dans l'exemple ci-dessous :

```
t = [[1, 2], [5, 6, 7, 8, 9], [4, 8]]
```

III.1 Construction

Pour construire un tableau de tableau, on peut utiliser la *construction par compréhension*.

```
t = [[0] * 5 for i in range(3)]
```

Il est important de comprend qu'on a ici évalué trois fois l'expression [0] * 5, pour chaque valeur de i, en obtenant *à chaque fois un nouveau tableau.*

Attention

Il faut faire la distinction avec l'expression

```
t = [[0] * 5] * 3
```

Dans ce cas, on créé un tableau dont les éléments sont les trois même tableaux. On peut vérifier cela en effectuant une affectation à un sous-tableau tel que :

```
t[0][1] = 7
```

L'affectation sera alors propagée sur la totalité des sous tableaux, retournant alors le résultat `[[0, 7, 0, 0, 0], [0, 7, 0, 0, 0], [0, 7, 0, 0, 0]]`

III.2 Parcours d'un tableau à plusieurs dimensions

Pour parcourir tous les éléments d'un tableau à plusieurs dimensions, on va naturellement utiliser des boucles imbriquées, chaque boucle prenant une dimension. Si l'on souhaite effectuer la somme des éléments de notre tableau t précédent (*de dimension* 3×5), on réalisera la boucle imbriquée suivante :

```
a = 0
for i in range(3):
        for j in range(5):
                a += t[i][j]
```

On peut aussi utiliser l'approche de la section précédente :

```
a = 0
for l in t:
        for x in l:
                a += x
```

Ici, la variable l parcourt les lignes du tableau à deux dimensions, et la variable x parcourt les éléments de chaque ligne. Il n'est même plus nécessaire de connaître la longueur des différents éléments.

Savoir-faire

A retenir

On peut parcourir les éléments d'un tableau **t** avec la construction **for x in t :** où **x** est une variable qui va recevoir successivement tous les éléments de **t**. Plutôt que d'allouer un tableau pour le remplir ensuite par une boucle, on peut utiliser avantageusement la **construction par compréhension**. Elle prend la forme **[e for x in t]**, où **e** est une expression qui dépend d'une variable **x** qui parcourt les éléments de **t**. Il est possible de définir des **tableaux à deux dimensions**, aux éléments desquels on accède via **deux indices**, ou même des tableaux de dimensions trois ou plus. Techniquement, on réalise ceci en définissant un **tableau dont chaque case contient un tableau**, correspondant donc aux lignes de longueur variable ou non.

Exercice

Exercice 1

Écrire un programme qui construit un tableau **t** de taille 11 x 11 tel que chaque élément t[i][j] contient i x j.

Exercice

Exercice 2

Écrire un programme qui crée un tableau à deux dimensions de taille 30 x 30 contenant des entiers tirés au hasard entre 1 et 9999, puis l'affiche. Modifier ensuite le programme pour afficher la valeur maximale et son emplacement [i][j].

Exercice

Exercice 3

Écrire un programme qui transforme un tableau **t** de 64 cases en un tableau **m** à deux dimensions de taille 8 x 8, tel que m[i][j] = t[8 * i + j] pour tout i >= 0 et j < 8.