Les tableaux

I Introduction

Jusqu'à présent, nous utilisions des variables pour stocker nos informations. Cette approche reste possible tant que le nombre d'information est faible. Dès que l'on dispose d'une grande quantité d'information, un stockage par tableau assure une meilleure organisation.

II Notions de tableaux

Un tableau permet de stocker plusieurs valeurs dans une seule variable et d'y accéder ensuite facilement. En python, on construit un tableau en énumérant ses valeurs entre crochets et séparées par des virgules.

```
table = [2, 3, 5]
```

Ici, on a déclaré une variable table contenant un tableau. Ce tableau contient 3 entiers. Les valeurs contenues dans un tableau sont *ordonnées*. Ici, la première valeur contenue dans le tableau est 2, la seconde est 3 et la troisième est 5. On représente ce tableau par des cases consécutives contenant les valeurs. Pour accéder à une valeur précise dans un tableau, on utilise son **indice**. Cet indice est le numéro de la case de notre tableau. On peut connaître la longueur d'un tableau en utilisant la fonction *len(table)*, qui retournera 3 dans notre cas.

```
table = [2, 3, 5]
print(len(table))
```

```
3
```

Les tableaux sont définis à partir de l'indice 0. Un tableau aura alors des indices allant de 0 à len(table)-1. Si l'on renseigne un indice supérieur à l'indice maximal, python retournera une erreur (*IndexError : list index out of range*).

II.1 Modifier le contenu d'un tableau

Pour renseigner une valeur dans un tableau, il faut utiliser l'*affectation*. On précise alors le tableau dans lequel on effectue l'opération, ainsi que l'indice correspondant la case voulue.

```
table[1] = 1
print(table)
```

Puis, lorsque l'on demande à python d'afficher notre tableau table, on obtient :

```
[2, 1, 5]
```

Il est possible d'ajouter une nouvelle information à la suite dans notre tableau. Pour cela, on fait appel à la méthode *append()* comme ci-dessous :

```
table.append(7)
print(table)
```

```
[2, 1, 5, 7]
```

Info

La taille du tableau est définie dynamiquement en Python. Lorsque l'on a ajouté la valeur 7 à notre table, Python a automatiquement ajouté un espace mémoire. Ce ne sera pas le cas pour la majorité des langages de programmation.

II.2 Parcourir un tableau

Pour parcourir un tableau, la méthode la plus simple est l'utilisation d'une boucle **for** qui permet de passer successivement tous les indices de notre tableau.

```
for i in range(0, 2) :
        table[i]
```

Dans le cas présent, on parcourt les indices 0, puis 1 puis 2. Si on souhaite parcourir la totalité d'un tableau, on utilisera une boucle allant jusqu'à la taille maximale de notre table.

```
for i in range (0, len(table)):
        table[i]
```

Attention

Il est possible de renseigner un indice négatif en python. Lorsque c'est le cas, on parcourt le tableau de droite à gauche. Ainsi, préciser table[-1] retournera la valeur stocké tout à droite, soit 7 dans notre exemple précédent (bien souvent la dernière valeur ajoutée).

III Construire un grand tableau

III.1 Allocation de l'espace mémoire

Lorsqu'il est nécessaire de construire un tableau vraiment grand, il devient difficile de le faire en énumérant chacun des éléments. On utilise alors le symbole * pour associer une multiplicité (et non pas une multiplication dans le cas d'un tableau) :

```
monTableau = [0] * 1000
```

Le tableau « monTableau » créé contient 1000 cases, ayant chacun la valeur 0 associée. Le symbole * permet donc, dans le cas d'un tableau, de réitérer l'opération (ici 1000 fois). Ainsi, maintenant que la taille de monTableau est définie, il peut le parcourir de bout en bout et y associer le carré de l'indice :

```
for i in range (0, 1000) :
        monTableau[i] = i * i
```

Info

Lors de la définition d'un grand tableau, il ne sera plus possible d'ajouter de nouveaux indices. La taille obtenu est donc fixe dans notre cas.

III.2 Concaténer deux tableaux

Il est possible de concaténer deux tableaux avec le symbole mathématique « + ». Tout comme pour le symbole « * », les tables ne seront pas additionnées, mais bien mises l'une à la suite de l'autre.

python

```
T = [1] * 4
U = [2] * 5
T+U
```

```
[1, 1, 1, 1, 2, 2, 2, 2, 2]
```

IV Exemple : Calcul d'une moyenne

Pour effectuer le calcul d'une moyenne, il est possible de créer autant de variable que de notes obtenues. On se rendra vite compte que cette solution n'est pas évolutive, et qu'il est nécessaire de créer une nouvelle variable à chaque nouvelle note. A contrario, on peut définir un tableau qui contiendra la totalité des notes. Commençons avec l'exemple ci-dessous :

```
notes = [14, 12, 17, 11]
```

On peut afficher une note en particulier en sélectionnant l'indice correspondant :

```
Note[2]
```

```
17
```

Pour parcourir l'ensemble des notes et déterminer la moyenne, on utilisera une structure en boucle for :

```
sommeNote = 0                          #Variable contenant la somme des notes
for note in range(0, len(notes)):
        sommeNote += notes[note]       #Addition des notes
print("La moyenne est de ", sommeNote/len(notes))
```

```
La moyenne est de 13.5
```

Il pourrait être intéressant d'afficher la liste des notes. Pour cela, on différencie l'indexe de l'élément avec sa valeur. Chaque élément est traité indépendamment avec la fonction *enumerate()*.

```
for index, item in enumerate(notes):
    print("L'élément à l'indexe :", index,"contient la note ", item,".")
```

```
L'élément à l'indexe : 0 contient la note 14 .
L'élément à l'indexe : 1 contient la note 12 .
L'élément à l'indexe : 2 contient la note 17 .
L'élément à l'indexe : 3 contient la note 11 .
```

Exercice 1

Écrire un programme qui construit un tableau de 100 entiers tirés au hasard entre 1 et 1000, puis l'affiche. On utilisera pour cela la fonction randint() de la bibliothèque random (from random import *).

Exercice 2

Compléter le programme précédent en ajoutant une fonction permettant d'afficher l'entier le plus grand du tableau.

Exercice 3

Écrire un programme qui tire au hasard mille entiers entre 1 et 10 et affiche le nombre de fois que chaque nombre a été tiré.

Exercice 4

Écrire une fonction concatenation(t1, t2) qui créé une nouvelle table contenant t1 puis t2.

Exercice

Exercice 5

La suite de Fibonacci est une séquence infinie d'entier très célèbre. Pour la construire, on commence par positionner 0 et 1, puis on construit l'entier suivant en additionnant les deux entiers précédents. 0, 1, 1, 2, 3, 5, 8, ... Écrire un programme qui construit puis affiche un tableau contenant les 30 premiers termes de la suite. On vérifiera le résultat du dernier élément qui sera 514229.

V Notions sur les tableaux et variables

V.1 Allocation mémoire d'une variable

A ce stade, il est important de comprendre la notion d'espace mémoire et d'adresse mémoire. Une variable est stocké à une adresse précise (son adresse mémoire), et contiendra une valeur donnée (c'est l'espace mémoire). Prenons l'exemple d'une variable x à laquelle on associe la valeur 1 (x = 1). En écrivant maintenant x = x + 1, l'ordinateur détermine l'adresse de la variable x, et écrit la nouvelle valeur 2 dans un nouvel espace mémoire. Ajoutons maintenant la variable y. En écrivant y = x, l'ordinateur associe l'adresse de y à la même adresse que x étant donné que les deux valeurs sont 2. Finalement, en précisant y = x + 1, l'ordinateur alloue un nouvel emplacement mémoire pour y avec la valeur de 3.

python

```
x = 1
print ("A l'emplacement",hex(id(x)),"est la valeur", x)
x = x + 1
y = x
print ("A l'emplacement",hex(id(x)),"est la valeur", x)
print ("A l'emplacement",hex(id(y)),"est la valeur", y)
y = x + 1
print ("A l'emplacement",hex(id(y)),"est la valeur", y)
```

```
A l'emplacement 0x75f5ee35 est la valeur 1
A l'emplacement 0x7d970fb6 est la valeur 2
A l'emplacement 0x7d970fb6 est la valeur 2
A l'emplacement 0x7ac7cec9 est la valeur 3
```

V.2 Allocation mémoire d'un tableau

Prenons maintenant le cas des tableaux. Un créant une variable t = [1, 2, 3] et une variable u = t, l'ordinateur associe automatiquement les deux variables au même espace mémoire, et à la même adresse.

python

```
t = [1, 2, 3]
u = t
print ("A l'emplacement",hex(id(t)),"est la valeur", t)
print ("A l'emplacement",hex(id(t[0])),"est la valeur", t[0])
print ("A l'emplacement",hex(id(t[1])),"est la valeur", t[1])
print ("A l'emplacement",hex(id(t[2])),"est la valeur", t[2])
print ("A l'emplacement",hex(id(u)),"est la valeur", u)
print ("A l'emplacement",hex(id(u[0])),"est la valeur", u[0])
print ("A l'emplacement",hex(id(u[1])),"est la valeur", u[1])
print ("A l'emplacement",hex(id(u[2])),"est la valeur", u[2])
```

```
A l'emplacement 0x7f68dd33f0c8 est la valeur [1, 2, 3]
A l'emplacement 0x8a88e0 est la valeur 1
```

```
A l'emplacement 0x8a8900 est la valeur 2
A l'emplacement 0x8a8920 est la valeur 3
A l'emplacement 0x7f68dd33f0c8 est la valeur [1, 2, 3]
A l'emplacement 0x8a88e0 est la valeur 1
A l'emplacement 0x8a8900 est la valeur 2
A l'emplacement 0x8a8920 est la valeur 3
```

Modifions maintenant la valeur contenue dans u[1] = 5. Étant donné que les tableaux t et u pointent à la même adresse, modifier une valeur du tableau de u revient à modifier la même donnée dans t.

```python
t = [1, 2, 3]
u = t
u[1] = 5
print ("A l'emplacement",hex(id(t)),"est la valeur", t)
print ("A l'emplacement",hex(id(t[0])),"est la valeur", t[0])
print ("A l'emplacement",hex(id(t[1])),"est la valeur", t[1])
print ("A l'emplacement",hex(id(t[2])),"est la valeur", t[2])
print ("A l'emplacement",hex(id(u)),"est la valeur", u)
print ("A l'emplacement",hex(id(u[0])),"est la valeur", u[0])
print ("A l'emplacement",hex(id(u[1])),"est la valeur", u[1])
print ("A l'emplacement",hex(id(u[2])),"est la valeur", u[2])
```

```
A l'emplacement 0x7f288537a0c8 est la valeur [1, 5, 3]
A l'emplacement 0x8a88e0 est la valeur 1
A l'emplacement 0x8a8960 est la valeur 5
A l'emplacement 0x8a8920 est la valeur 3
A l'emplacement 0x7f288537a0c8 est la valeur [1, 5, 3]
A l'emplacement 0x8a88e0 est la valeur 1
A l'emplacement 0x8a8960 est la valeur 5
A l'emplacement 0x8a8920 est la valeur 3
```

V.3 Tableaux et fonctions

Une fonction a la capacité de modifier le contenu d'un tableau qui lui est passé en argument. Ce n'est pas le cas d'une variable.

```python
def f(a, b):
        a = a + 1
        b[a] = 7
a = 1
t = [1, 2, 3]
print("t vaut", t)
print("a vaut", a)
f(a, t)
print("t vaut", t)
print("a vaut", a)
```

```
t vaut [1, 2, 3]
a vaut 1
t vaut [1, 2, 7]
a vaut 1
```

De même qu'une fonction peut recevoir un tableau en argument, une fonction peut aussi renvoyer un tableau comme résultat. Prenons l'exemple d'une fonction permettant de définir les carrés des n premiers entiers.

```python
def carré(n):
        t = [0] * n
        for i in range (n):
                t[i] = i * i
        return t
u = carré(5)
print ("A l'emplacement",hex(id(u)),"est la valeur", u)
print ("A l'emplacement",hex(id(t)),"est la valeur", t)
```

```
[0, 1, 4, 9, 16]
NameError: name 't' is not defined
```

La variable t initialisé lors de l'exécution de la fonction carré(n) est supprimé par le ramasse-miettes. Néanmoins, les différentes valeurs sont stockés dans le tableau retourné par notre fonction, à l'adresse de c.

Savoir-faire

A retenir

Un tableau permet de regrouper plusieurs valeurs sous la même forme d'une séquence ordonnée dans laquelle chaque valeur est associée à un indice. Les indices permettent d'accéder aux valeurs contenues dans un tableau, pour les consulter ou les modifier. On peut utiliser une boucle pour examiner tour à tour les différentes valeurs contenues dans un tableau.

Exercice 6

Écrire une fonction somme(tab) qui calcule et renvoie la somme des éléments d'un tableau d'entiers. En déduire une fonction moyenne(tab) qui calcule et renvoie la moyenne des éléments du tableau tab, supposé non vide.

Exercice 7

Écrire une fonction produit(tab) qui calcule et renvoie le produit des éléments d'un tableau d'entiers. Si le tableau contient 0, la fonction devra renvoyer 0 sans terminer le calcul.

Exercice 8

Écrire une fonction échange(tab, i, j) qui échange dans le tableau les éléments aux indices i et j.

Exercice 9

Écrire une fonction miroir(tab) qui reçoit un tableau en argument et le modifie pour échanger le premier élément avec le dernier, le second avec l'avant-dernier, etc. On pourra se resservir de la fonction échange() de l'exercice précédent.

Exercice 10

Pour mélanger les éléments d'un tableau aléatoirement, il existe un algorithme très simple qui procède ainsi : on parcourt le tableau de la gauche vers la droite et, pour chaque élément à l'indice i, on l'échange avec un élément situé à un indice tiré aléatoirement entre 0 et i (inclus). Écrire une fonction mélange(tab) qui réalise cet algorithme. On pourra se resservir de la fonction échange() de l'exercice précédent. Cet algorithme s'appelle le *mélangeur de Knuth*).