Chapitre 6 : Structures imbriquées

Il existe en python deux types de structures de données permettant de regrouper des informations : les listes et les dictionnaires. Ces structures de données peuvent contenir des informations de différents types : entiers, flottants, booléens, chaînes de caractères voire d'autres structures (tableaux ou dictionnaires). On parle alors de structures imbriquées (structures de données dans d'autres structures de données).

Si on a une structure imbriquée dans une autre (tableau dans un tableau, dictionnaire dans un dictionnaire, etc), on parle d'une structure imbriquée de niveau 2. Si on a une structure imbriquée dans une autre structure, elle-même imbriquée dans une autre structure, on parle d'une structure imbriquée de niveau 3. Théoriquement, on peut avoir autant d'imbrication que l'on souhaite mais dans la pratique, on dépasse rarement le niveau 2 ou 3.

Les structures de données imbriquées permettent de stocker des informations de manière structurée.

1 Accès aux informations d'une structure imbriquée

Pour accéder aux informations d'une structure de données imbriquée, on va du plus général (structure de données contenant les autres) au plus précis.

Pour accéder à une information stockée dans une structure de données de niveau 2, on utilise :

`variable[...][...]`

Pour accéder à une information stockée dans une structure de données de niveau 3, on utilise :

`variable[...][...][...]`

Si l'on a un tableau `tab` dont chaque case contient un dictionnaire correspondant à une personne (avec un nom et un prénom), alors accéder au nom de la première personne se fait avec :

```
nom = tab[0]["nom"] #nom correspond au nom de la première personne du tableau
```

Remarque : on peut utiliser des variables temporaires. Par exemple :

```
personne = tab[0] #personne correspond à la première personne du tableau
nom = personne["nom"] #nom correspond au nom de la première personne du tableau
```

2 Copie d'une structure imbriquée

La fonction `S.copy()` (où `S` est une structure de données) permet de faire une copie de la structure. On a alors deux structures de données en mémoire. Cependant, si `S` est une structure de données imbriquée, `S.copy()` créera une nouvelle structure mais les structures de données à l'intérieur ne seront pas copiées (on aura des alias).

Par exemple, si `tab` est un tableau de 4 cases contenant chacune un dictionnaire, `tab.copy()` créera un nouveau tableau. Par contre, les cases contiendront les mêmes dictionnaires que les cases de `tab` (en mémoire, on aura 2 tableaux mais seulement 4 dictionnaires au lieu de 8).

Pour faire une vraie copie, il faut alors utiliser `deepcopy(S)` du module `copy`. Dans ce cas, toutes les structures de données (la principale et toutes celles contenues dedans) sont recopiées.

```
from copy import *
#copie est une copie de structure et toutes les structures de données dans
#copie sont des copies de celles dans structure
copie = deepcopy(structure)
```

Attention : La copie via `deepcopy` recopie tous les objets de la structure de données, ce qui peut faire beaucoup ! Il faut donc l'utiliser uniquement quand cela est nécessaire (quand on souhaite vraiment une vraie copie).

3 Parcours de structures imbriquées

Si une structure de données a une forme spécifique (un tableau dont chaque case contient un tableau par exemple), on peut alors parcourir toutes les données de la structure en utilisant des boucles imbriquées.

4 Structures imbriquées et format JSON

Le format JSON est un format de données permettant de représenter des données structurées sous forme d'une chaîne de caractères.

En Python, on peut très facilement passer d'une structure de données à sa représentation sous forme d'une chaîne de caractères grâce à la fonction `dumps`. Le contraire se fait à l'aide de la fonction `loads`. Les fonctions `dumps` et `loads` sont définies dans le module `json`.

L'initialisation d'une structure de données `data` en fonction d'un fichier JSON `fichier.json` se fait selon :

```
from json import dumps, loads
f = open("fichier.json", "r") # Ouverture du fichier fichier.json en lecture
s = f.read() # On lit dans le fichier toutes les informations et on les
             # stocke dans la chaîne de caractères s (au format JSON)
f.close() # On ferme le fichier
data = loads(s) # On initialise data avec les informations de s.
```

De la même manière, l'écriture dans un fichier JSON `fichier.json` des informations contenues dans une structure de données `data` se fait selon :

```
from json import dumps, loads
s = dumps(data) # s est une chaîne de caractères contenant les informations de data
f = open("fichier.json", "w") # Ouverture du fichier fichier.json en écriture
f.write(s) # On écrit la chaîne s contenant toutes les informations (format JSON)
f.close() # On ferme le fichier
```