Dictionnaires

Cours 6

Utilisation de dictionnaires en boîte noir - v1.1

Lycée La Martinière Monplaisir, 41 Rue Antoine Lumière, 69372 Lyon

1 Tableau associatif

En informatique, un tableau associatif (aussi appelé *dictionnaire* ou *table d'association* est un type de données associant à un ensemble de *clefs*, un ensemble correspondant de *valeurs*. Chaque clef est associée à une seule valeur (au plus).

1.1 Exemples de la vie courante

- annuaire : les noms sont les clés et les valeurs sont les numéros de téléphone et les adresses ;
- dictionnaire : les mots sont les clés et les définitions sont les valeurs ;
- inventaire : les produits sont les clés et les quantités sont les valeurs.

2 Le type `dict` en Python

Les dictionnaires en Python correspondent à une implémentation d'un tableau associatif. C'est une structure muable (comme les listes) et itérable. Depuis la version de Python 3.7, cette structure conserve l'ordre (depuis la version 3.6 de fait, mais c'était alors considéré comme un détail d'implémentation, c'est maintenant une propriété documentée et à conserver dans les prochaines versions). En général, cet ordre a une importance moindre (d'où son implémentation tardive).

2.1 Déclaration littérale

Pour déclarer un dictionnaire vide, on a la choix entre les accolades `{}` et la fonction `dict()`.

```
>>> d1 = {} ; d2 = dict()
>>> type(d1) ; type(d2)
<class 'dict'>
<class 'dict'>
```

Si on souhaite directement avoir un tableau rempli, le format est le suivant : `{clef1: valeur1, clef2: valeur2, ... }`.

```
>>> lettres = {'m': 1, 's': 4, 'p': 2, 'i': 4}
>>> lettres
{'m': 1, 's': 4, 'p': 2, 'i': 4}
>>> type(lettres)
<class 'dict'>
```

2.2 Lecture de valeur, ajout de clef et modification de valeur

Le format pour traiter ces 3 cas est le même : `d[clef]`.

```
>>> lettres = {'m': 1, 's': 4, 'p': 2}
>>> lettres['i'] = 4 # création d'un nouveau couple clef, valeur dans lettres
>>> lettres['m'] # accès en lecture de la valeur correspondant à la clef 'm'
1
>>> lettres['s'] = 2 # changement de la valeur correspondant à la clef 's'
>>> print(lettres)
{'m': 1, 's': 2, 'p': 2, 'i': 4}
```

2.3 Suppression d'une clef

Comme pour une liste, on utilise le mot clé `del` :

```
>>> print(lettres)
{'m': 1, 's': 2, 'p': 2, 'i': 4}
>>> lettres = {'m': 1, 's': 4, 'p': 2, 'i': 4, 'o': 2}
>>> del lettres['o']
>>> print(lettres)
{'m': 1, 's': 4, 'p': 2, 'i': 4}
```

2.4 Itérations sur les clefs et les valeurs

Dans une boucle for, on peut directement itérer sur les clés, sur les valeurs ou sur les deux à la fois.

```
>>> for clef in lettres.keys():
...     print(clef)
m
s
p
i
>>> for valeur in lettres.values():
...     print(valeur)
1
4
2
4
>>> for cle, val in lettres.items():
...     print(cle, val)
m 1
s 4
p 2
i 4
```

2.5 Test d'appartenance avec le mot clef `in`

On peut tester si une clef apparaît dans un dictionnaire grâce au test `cle in dico`.

```
>>> 'k' in lettres
False
>>> 'i' in lettres
True
```

2.6 Copie d'un dictionnaire

Comme pour les listes, on a une méthode `d.copy()`. On rappelle que la commande d'affectation `d2 = d1` ne crée pas une copie valide, mais juste donne 2 noms différents à un même objet, ce qui est très rarement ce qui est souhaité.

```
>>> d1 = {'m': 1, 's': 4, 'p': 2, 'i': 4}
>>> d2 = d1 ; d3 = d1.copy()
>>> d1['m'] = 0 ; d2['o'] = 2
>>> print(d1) ; print(d2) ; print(d3)
{'m': 0, 's': 4, 'p': 2, 'i': 4, 'o': 2}
{'m': 0, 's': 4, 'p': 2, 'i': 4, 'o': 2}
{'m': 1, 's': 4, 'p': 2, 'i': 4}
```

2.7 Restriction sur les clefs et les valeurs

Il n'y a pas de restriction sur les valeurs, mais il existe une restriction sur les clefs : il faut qu'elles soient strictement immuables (ne pas être ou contenir de listes ou de dictionnaire par exemple). Les principaux types de clefs qui sont pratiques à utiliser sont :

- les chaines de caractères ;
- les entiers ;
- les n-uplets comprenant des chaines de caractères et/ou des entiers.