Dictionnaires

Laurent Signac – CC-BY-SA – 21-09-21 1741 bc205274852d0cbc74cb

Les type dictionnaire (`dict`) est une implémentation Python du type abstrait *tableau associatif*, basé sur des *tables de hachage*.

Un dictionnaire est un type *collection*, *itérable*, *mutable*, n'est *pas ordonné*, et n'est donc *pas une séquence*.

Un dictionnaire permet d'associer une valeur (objet quelconque) à une clé (qui doit avoir la caractéristique d'être hachable, donc récursivement non mutable). Une fois l'association réalisée, on peut rappeler la valeur en donnant la clé en un temps très court (complexité $O(1)$, c'est à dire en un temps indépendant de la taille du dictionnaire), la plupart du temps.

1 Création d'un dictionnaire

Il existe une multitude de façons de créer un dictionnaire. Voici les principales :

```
d1 = {}      # Dictionnaire vide
d3 = {'clé': 'valeur', 1: (5, 6), 3.1415: 'pi', (2, 3, 5): [1, 2, 3]} # en extension
d5 = {k: k ** 2 for k in range(1, 10)} # en intention
```

2 Rappel / Modification / Ajout des valeurs

Une fois un dictionnaire créé on rappelle une valeur en donnant la clé, on peut modifier de la même manière la valeur associée à une clé, ou bien ajouter une paire clé/valeur :

```
tailles = {"Bilbo": 91, "Frodo": 110, "Sam": 109}
tailles["Frodo"]

--> 110

tailles["Frodo"] = 111
tailles["Frodo"]

--> 111

tailles["Merry"] = 140
tailles

--> {'Bilbo': 91, 'Frodo': 111, 'Sam': 109, 'Merry': 140}
```

3 Test d'appartenance

On peut tester l'appartenance d'une clé au dictionnaire (opération rapide en O(1)) :

```
tailles = {"Bilbo": 91, "Frodo": 110, "Sam": 109}
"Frodo" in tailles

--> True

"Pippin" in tailles

--> False
```

Rem : l'écriture `"Pippin" not in tailles` est préférable à `not "Pippin" in tailles` (PEP 8 [713]).

4 Itérations

On peut itérer sur les clés d'un dictionnaire, sur les valeurs, ou sur les couples clés valeurs. Le plus courant est ce dernier cas :

```
for nom, taille in tailles.items(): # **ne pas** oublier items() !
    print("{} mesure {:02d} cm".format(nom, taille))
```

Vous êtes invités à tester le code précédent. Vous devez être capable de le reproduire de mémoire.

On peut aussi itérer uniquement sur les clés ou uniquement sur les valeurs, mais ces deux cas sont moins courants:

```
for nom in tailles.keys(): # Itération sur les clés
    ...
for taille in tailles.values(): # Itération sur les valeurs
    ...
for nom in tailles: # Itération sur les clés (idem 1er exemple)
    ...
```

5 Accès protégé

Si on tente de récupérer la valeur associée à une clé qui n'est pas dans le dictionnaire, une exception est levée :

```
tailles = {"Bilbo": 91, "Frodo": 110, "Sam": 109}
tailles["Pippin"]

    KeyError: 'Pippin'
```

On pourrait s'en sortir en utilisant le test d'appartenance `in`, mais il existe une solution plus élégante. La méthode `get` des dictionnaires, permet de renvoyer la valeur associée à une clé passée en paramètre, et renvoie une valeur par défaut si la clé n'est pas présente.

```
tailles = {"Bilbo": 91, "Frodo": 110, "Sam": 109}
tailles.get("Pippin")

    --> None

tailles.get("Pippin", 140)

    --> 140
```

6 Avoir en tête quand utiliser un dictionnaire

On utilise un dictionnaire parce que c'est pratique ou parce que c'est efficace (ou les 2). L'efficacité se fait sentir uniquement pour de grands dictionnaires (ou des opérations effectuées de nombreuses fois).

Voici quelques fait à connaître:

- rechercher une valeur dans une dictionnaire est lent
- rechercher une clé dans un dictionnaire est rapide
- avoir la valeur associée à une clé est rapide
- avoir une des clés qui correspond à une valeur est lent
- on peut parfois remplacer une série de `if` par un dictionnaire, ce qui donne une écriture particulièrement concise.

7 Exemple d'utilisation

Un dictionnaire peut être utilisé pour compter le nombre d'apparition de chaque élément d'une liste :

```
lst = [1, 2, 3, 2, 2, 3, 4, 5, 4, 5, 6, 1, 1, 2, 3, 6, 5]
compte = {} # dictionnaire vide
for val in lst:
    compte[val] = compte.get(val, 0) + 1

print(compte)
```

```
{1: 3, 2: 4, 3: 3, 4: 2, 5: 3, 6: 2}
```

Pour réaliser cette tâche, on a la classe `Counter`, qui contient tout ce qu'il faut, et qui est basée sur l'utilisation de dictionnaires