Méthodes de programmation

Tableau associatif ou dictionnaire

Anicet E. T. Ebou, ediman.ebou@inphb.ci

01

Définition et création

Définition

Un dictionnaire est une structure de données permettant de stocker les données en les indexant par des clés plutôt que par des entiers.

Les éléments V, au lieu d'être indicés par un entier sont indicés par des clés appartement à un ensemble K.

Un dictionnaire relie donc directement une clé, qui n'est pas nécessairement un entier, à une valeur: pas besoin d'index intermédiaire pour rechercher une valeur comme dans une liste. Par contre, cette clé est **nécessairement d'un type immuable**, qui ne peut pas être modifiée

Création d’un dictionnaire

Un dictionnaire, appelé aussi un « tableau associatif », est une séquence de paires « clé : valeur » séparées par virgules et mises entre accolades.

```
dico = {}  # création d’un dictionnaire vide
type(dico)
alphabet = {‘a’: 1, ‘b’: 2, ‘c’: 3, ‘d’: 4, ‘e’: 5}
print(alphabet)
```

Création d’un dictionnaire

Les **clés** d’un dictionnaire ne sont pas forcément toutes du même type.

Nous pouvons utiliser tout objet non mutable (immuable) pour les clés : des **entiers** (type `int`), des **flottants** (type `float`), des **chaînes de caractères** (type `str`), ou bien des **n-uplets** (type `tuple`)

À l’inverse, une liste ne peut pas être une clé.

Création d’un dictionnaire

Les valeurs peuvent être de n’importe quel type de données.

```
dico2 = {3:[] , ‘cle’:42 , (15 ,15): [1 ,2 ,3 ,4]}
```

On peut également créer un dictionnaire en compréhension comme pour les listes :

```
dico3 = {x: x **2 for x in range(1, 6)}
dico3
```

02

Opérations sur un dictionnaire

Nombres d’éléments

La fonction `len` renvoie le nombre d’éléments associés d’un dictionnaire.

```
alphabet = {‘a’: 1, ‘b’: 2, ‘c’: 3, ‘d’: 4, ‘e’: 5}

len(alphabet)
```

Accès à un élément

On accède aussi à un élément avec la même notation entre crochet que pour les listes, mais en donnant cette fois la clé d'accès en argument: `dictionnaire[clé]`

```
alphabet['a']   #accès à la valeur de la clé 'a'
# erreur quand on demande l'accès à une clé
# non présente dans le dictionnaire
alphabet['x']
```

Ajout et suppression d’un élément

L’opérateur [ ] est nécessaire pour ajouter un élément.

```
alphabet['p'] = 14  # ajoute l'élément de clé 'p' de valeur 14
print(alphabet)
alphabet['p'] = 15  # la clé 'p' existe, on la modifie avec la valeur 15
print(alphabet)
del alphabet['c']   # supprime l'élément de clé 'c'
print(alphabet)
```

Test d’appartenance

Pour tester si une clé est présente dans le dictionnaire, on utilise `clé in dictionnaire` qui renvoie un booléen: `True` si la clé est présente, `False` sinon.

```
‘a’ in alphabet  # renvoie True
‘z’ in alphabet  # renvoie False
```

Parcours d’un dictionnaire

Pour **parcourir les clés** d’un dictionnaire avec une boucle `for`, on utilise la syntaxe: `for cle in dictionnaire:`

Par exemple, avec le dictionnaire précédent, si on veut construire une liste L contenant les valeurs du dictionnaire alphabet:

```
L = []
for cle in alphabet:
    L.append(alphabet[cle])
```

Parcours d'un dictionnaire

On peut aussi le faire en définissant la liste par compréhension.

```
L = [alphabet[c] for c in alphabet]
```

On peut accéder à l'ensemble des clés à l'aide de la méthode `keys()`, et à la liste des couples clé: valeur par la méthode `items()`

```
cles = alphabet.keys()
cles
couples = alphabet.items()
couples
```

Parcours d’un dictionnaire

On obtient à chaque fois un objet de type `dict_keys` et `dict_items` respectivement, qui se comporteront de manière presque similaire à un objet de type list (on peut le parcourir, mais toutefois pas accéder à un élément par son indice).

Copies

Si une valeur est d'un type mutable, on peut alors la modifier sans réaffectation. Tous les exemples précédents montrent bien que les dictionnaires sont des objets mutables en Python. Se poseront donc les mêmes problèmes d'alias et de copie que pour les listes.

On pourra utiliser la méthode `copy()` pour réaliser une copie superficielle d'un dictionnaire, ou bien la fonction `deepcopy` de la bibliothèque `copy` pour réaliser une copie profonde de ce dictionnaire.

Copies

Testez chacun des codes suivants et commentez les lignes.

```
d = {0:['a', 'b']}
e = d
d[1] = ['c', 'd']
d
e
```

```
d ={0:['a', 'b']}
e = d.copy()
d[1] = ['c', 'd']
d[0].append(['e', 'f'])
d
e
```

Copies

Testez chacun des codes suivants et commentez les lignes.

```
from copy import deepcopy

d = {0:['a', 'b']}

e = deepcopy(d)

d[1] = ['c', 'd']

d[0].append(['e', 'f'])

d

e
```