1 Tableaux associatifs : les dictionnaires

Dictionnaire

Un dictionnaire est un conteneur non ordonné d'éléments indexés par des clés (enregistrements du type `clé : valeur`). Le seul moyen d'accéder à une `valeur` particulière est par l'intermédiaire de sa `clé`, chaque clé ne pouvant être présente qu'une seule fois dans la collection.

Remarques :

- à l'instar d'une application mathématique, un dictionnaire est une sorte de correspondance entre un ensemble de clés et un ensemble de valeurs : on parle aussi de *tableau associatif*.
- un dictionnaire est un type de donnée Python à part entière : type `dict` ;
- un dictionnaire est modifiable mais non ordonné : il permet de stocker des couples `clé : valeur` avec des valeurs de tous types, éventuellement hétérogènes et des clés ayant pour seule contrainte le fait d'être d'un type non mutable (plus précisément hashable) qui peut différer au sein d'un même dictionnaire.

Les dictionnaires sont optimisés pour un accès aux valeurs par un *"hachage"* rapide sur les clés. L'utilisateur n'a aucun moyen de savoir dans quel ordre les couples (clés, valeurs) sont placés dans le dictionnaire : leur emplacement est géré par un algorithme spécifique qui optimise l'accessibilité aux valeurs au détriment de leur ordre.
À ce titre, les dictionnaires ne sont pas des séquences (ils ne peuvent pas être parcourus par un indice de position). Ils sont cependant considérés comme des objets *itérables* (on peut donc les parcourir, avec une boucle `for` par exemple).

dict = {*clé 1*:*val 1*, *clé 2*:*val2*, ... , *clé n*:*val n*}

syntaxe d'un dictionnaire : couples *clé* : *valeur*
séparés les uns des autres par des virgules et
entourés d'accolades

Différentes méthodes de construction de dictionnaires sont possibles :

- en évaluant l'expression `dict()` ou `{}`, on obtient le dictionnaire vide ;
- en utilisant la définition avec une saisie complète `{clé 1:val 1,....., clé n :val n}`
- en convertissant une séquence avec la fonction `dict` : par exemple dict([[clé 1,val 1],[clé 2,val 2],...,[clé n,val n]]) ou dict(clé 1 = val 1, ..., clé n = val n) ;
- en ajoutant des paires `clé : valeur` à un dictionnaire existant `dico` : `dico[newclé] = newvaleur` (si la clé existe déjà, la valeur est mise à jour, mais si la clé n'existe pas encore, la paire sera ajoutée dans le dictionnaire).
- en compréhension, par exemple `dico = { x:x**2 for x in range(10)}`

Opérations sur les dictionnaires

longueur d'un dictionnaire (nombre de clés)	`len(dico)`
Appartenance d'une clé *k* à `dico` (vaut **`True`** si *k* est une clé de `dico` et **`False`** sinon)	`k in dico`
Test d'égalité entre deux dictionnaires : renvoie **`True`** s'ils ont les mêmes clés faisant référence aux mêmes valeurs	`dico1 == dico2`
Renvoyer la valeur associée à `clé` (si la clé n'existe pas, lève l'exception `keyError`)	`dico[clé]`
Renvoyer la représentation du dictionnaire (affichage)	`repr(dico)`
Créer (ou modifie) une paire clé/valeur	`dico[clé] = valeur`
Effacer une paire clé/valeur de `dico`	`del dico[clé]`
Effacer tout le contenu du dictionnaire	`dico.clear()`
Construire un dictionnaire en compréhension avec ajout d'une condition facultative sur les éléments de la séquence	`{t1(x):t2(x) for x in séq if c(x)}`
Renvoyer une copie en surface du dictionnaire, indépendante de l'original	`dico.copy()`
Renvoyer la valeur associée à la `clé`, et Renvoyer `None` si la clé est absente	`dico.get(clé)`
Renvoyer la valeur associée à la `clé`, et la valeur de `défaut` si la clé est absente	`dico.get(clé, défaut)`
Renvoyer une vue du dictionnaire, itérable souvent utilisé dans une boucle `for` : `for k,v in dico.items() :`	`dico.items()`
Renvoie un itérable pour décrire les clés du dictionnaire dans une boucle `for` : `for k in dico.keys() :`	`dico.keys()`
Renvoyer un itérable pour décrire les valeurs du dictionnaire dans une boucle `for` : `for v in dico.values() :`	`dico.values()`
Renvoyer la valeur, et supprime la paire clé/valeur (`KeyError` si clé absente)	`dico.pop(clé)`
Renvoyer la valeur, et supprime la paire clé/valeur (évalue `défaut` si clé absente)	`dico.pop(clé,défaut)`
Renvoyer une paire arbitraire (clé,valeur) et la supprime de dico (`KeyError` si `dico` vide)	`dico.popitem()`
Renvoie la valeur si la clé est présente, sinon crée le couple `clé:None`	`dic.setdefaut(clé)`
Renvoyer la valeur si la clé est présente, sinon crée le couple `clé:défaut`	`dic.setdefaut(clé,défaut)`
Mise à jour (fusion) du dictionnaire `dico` à partir du dictionnaire `dico2` (remplace les valeurs si clés homonymes)	`dico.update(dico2)` ou `dico.update(clé 2.1=val2.1,...,)`

Console Python :

```
>>>#Différentes manières de générer un même dictionnaire
>>> dico1 = {'Newton':1642, 'Gauss':1777, 'Germain':1776, 'Pascal':1623,
'Curie':1867}

>>> dico2 = dict(Newton=1642, Gauss=1777, Germain=1776, Pascal=1623, Curie=1867)

>>> dico3 = dict([('Newton',1642), ('Gauss', 1777) ,('Germain',1776),
('Pascal',1623), ('Curie',1867)])

```

```
>>> dico4 = {nom:date for nom,date in zip(['Newton','Gauss','Germain','Pascal','Curie'],[1642,1777,1776,1623,1867])}
# la commande zip(seq1,seq2) renvoie un itérateur contenant les couples formés à partir des éléments de même indice pris dans les itérateurs seq1 et seq2

>>> dico5 = dict(zip(['Newton','Gauss','Germain','Pascal','Curie'],[1642,1777,1776,1623,1867]))
>>> dico1 == dico2 == dico3 == dico4 == dico5
True

>>> dico3 # affichage d'un dictionnaire par appel
{'Newton': 1642, 'Gauss': 1777, 'Germain': 1776, 'Pascal': 1623, 'Curie': 1867}

>>> 'Newton' in dico3, 'Balzac' in dico3, 'Poe' not in dico3 #test d'appartenance d'une clé à un dictionnaire
(True, False,True)

>>> dico1['Germain'] # renvoie de la valeur associée à une clé, lève une exception KeyError si la clé est absente
1776

>>> dico3.get('Balzac') # variante avec la méthode get qui renvoie None en cas de clé absente
    # cela ne renvoie rien

>>> type(dico3.get('Balzac'))
<class 'NoneType'>

>>> dico3.get('Balzac',"pas trouvé") # possibilité de renvoyer une valeur en cas d'absence de la clé
'pas trouvé'

>>> dico3['Euler'] = 1707 # crée un nouvel enregistrement

>>> del dico3['Gauss'] # supprime un enregistrement

>>> dico3 # affichage du dictionnaire
{'Newton': 1642, 'Germain': 1776, 'Pascal': 1623, 'Curie': 1867, 'Euler': 1707}

>>> for nom in dico3 : print(nom) # itérer sur le dictionnaire revient à itérer sur ses clés
Newton
Germain
Pascal
Curie
Euler

>>> for n,d in dico3.items() : print("{} est né(e) en {}".format(n,d)) # pour itérer sur les paires
Newton est né(e) en 1642
Germain est né(e) en 1776
Pascal est né(e) en 1623
```

```
Curie est né(e) en 1867
Euler est né(e) en 1707

>>> list(dico3.keys()) # pour extraire la liste des clés (possible avec list(dico3)
['Newton', 'Germain', 'Pascal', 'Curie', 'Euler']

>>> list(dico3.values()) # pour extraire la liste des valeurs
[1642, 1776, 1623, 1867, 1707]

>>> dico1.update(dico3) # mise à jour de dico1 avec les enregistrements de dico2 (fusion)

>>> repr(dico1) # renvoie une représentation du  dictionnaire sous la forme d'une chaîne
"{'Newton': 1642, 'Gauss': 1777, 'Germain': 1776, 'Pascal': 1623, 'Curie': 1867, 'Euler': 1707}"

>>> dico1.update(Riemann = 1826, Galois = 1811, Cantor = 1845) # mise à jour avec une liste d'enregistrements

>>> dico1
{'Newton': 1642, 'Gauss': 1777, 'Germain': 1776, 'Pascal': 1623, 'Curie': 1867, 'Euler': 1707, 'Riemann': 1826, 'Galois': 1811, 'Cantor': 1845}
```