DIU

Enseigner l'Informatique au Lycée : BLOC1

******

Leçon proposée par :

- DUHEM Aude
- DUVAUCHELLE Sophie
- NICOLAS PATRICE

******

Table des matières :

Orsay, Université PARIS-SUD,

JUIN 2019

TPython : les dictionnaires

Le **dictionnaire** est un **type de données** très puissant et pratique, utilisé pour l'implémentation de nombreux algorithmes efficaces et élégants.

En python, on trouve également l'appellation de **p-uplet nommé** mais la plupart des autres langages appellent cette structure un **tableau associatif** ou un « *mapping* ».

1. Définition et vocabulaire

Un dictionnaire est un tableau qui associe des **clefs** et des **valeurs**. Chaque clef est **unique**. L'association entre une clef et une valeur est appelée une **paire clef-valeur** ou un « *item* ».

2. Création et modification d'un dictionnaire

Un dictionnaire est délimité par des **accolades** ; les **paires clef-valeur** sont séparées par des virgules avec le caractère **deux-points** entre **clef** et **valeur**.

```
>>> dico_vide = {}
{'age': 16, 'nom': 'Besnard', 'prénom': 'Jean', 'classe': '1e 2'}
```

Un dictionnaire ressemble à une **liste** mais il est plus général. Dans une liste, les **indices** sont des nombres entiers ; dans un dictionnaire, les **clefs** peuvent être de (presque) n'importe quel **type** : des nombres, des chaînes de caractères, des tuples…

```
>>> dicoFranAngl = {'un': 'one', 'deux': 'twoo'}
>>> plateauDames = {('A', 1): 'blanc', ('C', 8): 'noir', ('D', 3): 'noir', ... }
```

Comme pour les listes, on peut utiliser la création par **compréhension** :

```
>>> alphabet = {chr(lettre): lettre for lettre in range(65, 90)}
{'X': 88, 'I': 73, 'T': 84, 'Q': 81, 'K': 75, 'D': 68, 'U': 85, 'N': 78, ... }
```

NB. Les clefs ne respectent aucun ordre ! Le dictionnaire est une structure non-ordonnée.

Pour ajouter ou modifier des éléments à un dictionnaire, on utilise des **crochets** :

```
>>> eleve['option1'] = 'anglais'
>>> eleve['prénom'] = 'Jeanne'
{'option1': 'anglais', 'age': 16, 'nom': 'Besnard', 'prénom': 'Jeanne', 'classe': '1e 2'}
```

Pour supprimer un élément, on utilise l'opérateur **del** :

```
>>> del eleve['option1']
{'age': 16, 'nom': 'Besnard', 'prénom': 'Jeanne', 'classe': '1e 2'}
```

3. Accès à une valeur

Les éléments d'un dictionnaire ne sont pas indexés : on utilise les **clefs** pour rechercher les **valeurs** correspondantes. Si la **clef** n'existe pas, on obtient une erreur.

```
>>> eleve['classe']
'1e 2'
>>> eleve['option2']
KeyError: 'option2'
```

4. Taille d'un dictionnaire

La fonction **len()** est utilisée sur les dictionnaires ; elle renvoie le nombre de paires clef-valeur :

```
>>> len(eleve)
3
```

5. Recherche d'une clef

L'opérateur **in** fonctionne sur les dictionnaires comme sur les autres séquences (liste, chaîne, tuple, etc.). Dans le cas qui nous intéresse, il permet de s'assurer de la présence d'une **clef** dans un dictionnaire en retournant **True** ou **False**.

```
>>> dicoFranAngl = {'un': 'one', 'deux': 'two'}
>>> 'un' in dicoFranAngl
True
>>> 'one' in dicoFranAngl
False
```

6. Parcours d'un dictionnaire

L'opérateur **in** permet également **d'«itérer » au travers des cléfs** d'un dictionnaire, au sein **d'une boucle for** (boucle « explicite » ou compréhension – voir paragraphe2 & feuille d'exercices).

```
>>>for cle in dicoFranAngl:
…           print(cle,end='')
un deux
```

En python, les objets de type « dictionnaire » possède **trois méthodes** remarquables :

- **keys()** permet d'itérer sur les clefs (comme précédemment)
- **values()** permet d'itérer sur les valeurs
- **items()** permet d'itérer sur les clefs et les valeurs

```
>>>for valeur in dicoFranAngl.values():
…           print(valeur,end='')
one two
>>>for clef,valeur in dicoFranAngl.items():
…           print(cle,valeur,end=';')
un one;deux two;
```

7. Dictionnaires et listes

Des listes ou des dictionnaires peuvent apparaître comme valeurs dans un dictionnaire. Par contre, ils ne peuvent pas être des clefs.

```
>>> liste = [1, 2, 3]
>>> dico = {'nom': 'Dupond', 'prénom': 'Jean'}
>>> d = dict()
>>> d[liste] = 'oups'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> d[dico] = 'blop'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'dict'
```

Pour aller plus loin...

Pourquoi cette limitation ? Un dictionnaire est mis en œuvre en utilisant une table de hachage et cela signifie que les clefs doivent être hachables.

Une fonction de hachage est une fonction qui prend une valeur (de n'importe quel type) et retourne un entier. Les dictionnaires utilisent ces entiers, appelés valeurs de hachage, pour stocker et rechercher des paires clef-valeur.

Ce système fonctionne très bien si les clefs sont immuables (i.e. figées). Mais si les clefs sont modifiables, comme les listes, de gros problèmes se posent. Par exemple, lorsque vous créez une paire clef-valeur, Python hache la clef et la stocke dans l'emplacement correspondant. Si vous modifiez la clef, puis la hachez à nouveau, elle ira à un endroit différent. Dans ce cas, vous pourriez avoir deux entrées pour la même clef ou vous pourriez ne pas être en mesure de trouver une clef. Quoi qu'il arrive, le dictionnaire ne fonctionnera pas correctement.

Récapitulatif : tuple vs. liste vs. dictionnaire

	TUPLE	LISTE	DICTIONNAIRE
délimité par	`()`	`[]`	`{}`
séquence ordonnée	oui	oui	non
création	• directe	• directe • par boucle FOR • par compréhension	• directe • par boucle FOR • par compréhension
syntaxe de création directe	`t = (elt1, elt2, ... )`	`l = [elt_1, elt_2, ... ]`	`d = {clef1: val1, clef2:val2,... }`
accès aux éléments par	indice (entier positif)	indice (entier positif)	clef (types variés)
suppression d'un élément	impossible	`del l[index]`	`del d[clef]`
opérateur `in`	teste la présence d'un indice	teste la présence d'une valeur	teste la présence d'une clef
modification d'un élément	impossible	`l[index] = valeur`	`d[clef] = valeur`
ajout d'un élément	impossible	`l.append(valeur)`	`d[clef] = valeur`
parcours des éléments	`for element in t:` `for i in range(len(t)):`	`for element in l:` `for i in range(len(l)):`	`for clef in d:`

parcours spécifiques			`for clef in d.keys():`
			`for valeur in d.values():`
			`for clef, valeur in d.items():`

LEXIQUE

- **clef** : un objet qui apparaît dans un dictionnaire comme la première partie d'une paire clef-valeur.
- **dictionnaire** : un tableau qui associe des **clefs** et des **valeurs**
- **fonction de hachage** : une fonction utilisée par une table de hachage pour calculer l'emplacement d'une clef.
- **hachable** : un type qui a une fonction de hachage. Les types immuables comme les entiers, les flottants et les chaînes de caractères sont hachables ; les types modifiables comme les listes et les dictionnaires ne le sont pas.
- *hashtable* **(table de hachage)** : l'algorithme utilisé pour mettre en œuvre des dictionnaires en Python.
- **mappage** : une relation dans laquelle chaque élément d'un ensemble est associé à un élément d'un autre ensemble
- **paire clef-valeur** : une association entre une clef et une valeur
- **recherche** : une opération de dictionnaire qui prend une clef et trouve la valeur correspondante.
- **singleton** : une liste (ou une autre séquence) avec un seul élément.
- **valeur** : un objet qui apparaît dans un dictionnaire comme la seconde partie d'une paire clef-valeur.