Chapitre 10 : Les dictionnaires

I. Les dictionnaires :

Le **dictionnaire** est une autre structure de données Python. Ce n'est **pas un** type de **séquence** (mais peut être facilement adapté au traitement de séquence) et il est **modifiable** .

Pour expliquer ce qu'est réellement le dictionnaire Python, il est important de comprendre qu'il s'agit littéralement d'un dictionnaire.

Dans le monde de Python, le mot que vous recherchez est nommé a key. Le mot que vous obtenez du dictionnaire est appelé a value.

Cela signifie qu'un dictionnaire est un ensemble de paires **clé-valeur** .

Remarque:

- chaque clé doit être **unique** - il n'est pas possible d'avoir plus d'une clé de la même valeur;
- une clé peut être des **données de tout type** (sauf liste): elle peut être un nombre (entier ou flottant), voire une chaîne;
- un dictionnaire n'est pas une liste - une liste contient un ensemble de valeurs numérotées, tandis qu'un **dictionnaire contient des paires de valeurs** ;
- la len()fonction fonctionne également pour les dictionnaires - elle renvoie le nombre d'éléments de valeur-clé dans le dictionnaire;
- un dictionnaire est un **outil à sens unique** - si vous avez un dictionnaire anglais-français, vous pouvez rechercher des équivalents français de termes anglais, mais pas l'inverse.

Nous pouvons maintenant vous montrer quelques exemples pratiques.

Comment faire un dictionnaire?

Si vous souhaitez affecter des paires initiales à un dictionnaire, vous devez utiliser la syntaxe suivante:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" :
"cheval"}
phone_numbers = {'boss' : 5551234567, 'Suzy' : 22657854310}
empty_dictionary = {}
print(dictionary)
print(phone_numbers)
print(empty_dictionary)
```

Dans le premier exemple, le dictionnaire utilise des clés et des valeurs qui sont toutes deux des chaînes. Dans le second, les clés sont des chaînes, mais les valeurs sont des entiers. La disposition inverse (touches → nombres, valeurs → chaînes) est également possible, ainsi que la combinaison nombre-nombre.

La liste des paires est **entourée d'accolades** , tandis que les paires elles-mêmes sont **séparées par des virgules** et les **clés et valeurs par des deux-points** .

Le premier de nos dictionnaires est un dictionnaire anglais-français très simple.

Le second - un très petit annuaire téléphonique.

Les dictionnaires vides sont construits par une **paire d'appareils orthopédiques vides** - rien d'inhabituel.

Le dictionnaire dans son ensemble peut être imprimé avec une seule print() invocation. L'extrait **peut** produire la sortie suivante:

```
{'dog': 'chien', 'horse': 'cheval', 'cat': 'chat'}
{'Suzy': 5557654321, 'boss': 5551234567}
{} # dictionnaire vide
```

Remarque : L'ordre des paires imprimées est différent de celui de l'affectation initiale. Qu'est-ce que ça veut dire?

Tout d'abord, c'est une confirmation que les **dictionnaires ne sont pas des listes** - ils ne préservent pas l'ordre de leurs données, car l'ordre n'a aucun sens (contrairement aux vrais dictionnaires papier). L'ordre dans lequel un

dictionnaire **stocke ses données est complètement hors de votre contrôle** et de vos attentes

En Python 3.6x, les dictionnaires sont devenus des collections **ordonnées** par défaut. Vos résultats peuvent varier en fonction de la version de Python que vous utilisez.

II. Comment utiliser un dictionnaire ?

Si vous souhaitez obtenir l'une des valeurs, vous devez fournir une valeur de clé valide:

```
print(dictionary['cat'])
print(phone_numbers['Suzy'])
```

Obtenir la valeur d'un dictionnaire ressemble à l'indexation, en particulier grâce aux crochets entourant la valeur de la clé.

Remarque:

- si la clé est une chaîne, vous devez la spécifier comme une chaîne;
- **les touches sont sensibles à la casse** : 'Suzy' est quelque chose de différent de 'suzy'.

L'extrait produit deux lignes de texte:

```
chat
5557654321
```

Remarque : vous **ne devez pas utiliser de clé inexistante** . Essayer quelque chose comme ça:

```
print(phone_numbers['president'])   => Erreur
provoquera une erreur d'exécution.
```

Heureusement, il existe un moyen simple d'éviter une telle situation. L' **in** opérateur, avec son compagnon **not in**, peut sauver cette situation.

Le code suivant recherche en toute sécurité certains mots français:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" :
"cheval"}
words = ['cat', 'lion', 'horse']
for word in words:
     if word in dictionary:
          print(word, "->", dictionary[word])
     else:
          print(word, "is not in dictionary")
```

La sortie du code se présente comme suit:

```
cat -> chat
lion is not in dictionary
horse -> cheval
```

- **Parcourir un dictionnaire:**

Les dictionnaires peuvent-ils être **parcourus en** utilisant la for boucle, comme les listes ou les tuples?

Oui et non car un dictionnaire n'est **pas un type de séquence** - la for boucle est inutile avec lui.

Oui, car il existe des outils simples et très efficaces qui peuvent **adapter n'importe quel dictionnaire aux** for **exigences de la boucle** (en d'autres termes, créer un lien intermédiaire entre le dictionnaire et une entité de séquence temporaire).

- **La méthode keys() :**

Le premier d'entre eux est une méthode nommée keys(), possédée par chaque dictionnaire. La méthode **renvoie un objet itérable composé de toutes les clés rassemblées dans le dictionnaire** . Avoir un groupe de clés vous permet d'accéder à l'ensemble du dictionnaire de manière simple et pratique.

Comme ici:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" :
"cheval"}

for key in dictionary.keys():

print(key, "->", dictionary[key]
```

La sortie du code se présente comme suit:

```
horse -> cheval

dog -> chien

cat -> chat
```

- **la fonction sorted() :**

La fonction sorted() , renvoie une liste triée on peut l'utiliser avec les listes , les tuples et les chaines de caractère et les dictionnaires aussi cette fonction ne change pas l'état de l'objet mais renvoie une nouvelle liste triée on peut l'utiliser dans la boucle for pour parcourir une liste trié

```
for key in sorted(dictionary.keys()):
```

- **les méthodes items () et values () :**

Une autre façon est basée sur l'utilisation d'une méthode de dictionnaire nommée items(). La méthode **renvoie des tuples** (c'est le premier exemple où les tuples sont plus qu'un simple exemple d'eux-mêmes) **où chaque tuple est une paire clé-valeur** .

Voilà comment cela fonctionne:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" :
"cheval"}
```

```
for english, french in dictionary.items():

       print(english, "->", french)
```

Notez la manière dont la tuple a été utilisée comme for variable de boucle.

L'exemple imprime:

```
cat -> chat

dog -> chien

horse -> cheval
```

Il existe également une méthode nommée **values(),** qui fonctionne de manière similaire à keys(), mais **renvoie des valeurs** .

Voici un exemple simple:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" :
"cheval"}

for french in dictionary.values():

print(french)
```

Le dictionnaire n'étant pas en mesure de trouver automatiquement une clé pour une valeur donnée, le rôle de cette méthode est plutôt limité.

- **modifier et ajouter des valeurs**

Attribuer une nouvelle valeur à une clé existante est simple - comme les dictionnaires sont entièrement **modifiables** , il n'y a aucun obstacle à les modifier.

Nous allons remplacer la valeur "chat" par "minou", qui n'est pas très précise, mais cela fonctionnera bien avec notre exemple.

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}

dictionary['cat'] = 'minou'

print(dictionary)
```

La sortie est:

```
{'dog': 'chien', 'horse': 'cheval', 'cat': 'minou'}
```

- **Ajout d'une nouvelle clé :**

L'ajout d'une nouvelle paire clé-valeur à un dictionnaire est aussi simple que de changer une valeur - vous n'avez qu'à affecter une valeur à une nouvelle **clé** , **auparavant inexistante** .

Remarque: il s'agit d'un comportement très différent des listes, qui ne vous permettent pas d'attribuer des valeurs à des indices non existants.

Ajoutons une nouvelle paire de mots au dictionnaire - un peu bizarre, mais toujours valide:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}

dictionary['swan'] = 'cygne'

print(dictionary)
```

L'exemple affiche:

```
{'swan': 'cygne', 'horse': 'cheval', 'dog': 'chien', 'cat': 'chat'}
```

Remarque : Vous pouvez également insérer un élément dans un dictionnaire en utilisant la méthode update(), par exemple:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" :
"cheval"}

dictionary.update({"duck" : "canard"})

print(dictionary)
```

- **Retrait d'une clé :**

Remarque: la suppression d'une clé entraînera toujours la **suppression de la valeur associée** . **Les valeurs ne peuvent exister sans leurs clés** .

Cela se fait avec l'instruction **del**.

Exemple:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" :
"cheval"}

del dictionary['dog']

print(dictionary)
```

Remarque: la **suppression d'une clé non existante provoque une erreur** .

L'exemple affiche:

```
{'cat': 'chat', 'horse': 'cheval'}
```

Pour supprimer le dernier élément d'un dictionnaire, vous pouvez utiliser la méthode **popitem()**:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" :
"cheval"}

dictionary.popitem()

print(dictionary) # outputs: {'cat' : 'chat', 'dog' : 'chien'}
```

Dans les anciennes versions de Python, c'est-à-dire avant 3.6.7, la popitem() méthode supprime un élément aléatoire d'un dictionnaire.

III. Tuples et dictionnaires :

Nous avons préparé un exemple simple, montrant comment les tuples et les dictionnaires peuvent fonctionner ensemble.

Imaginons le problème suivant:

- vous avez besoin d'un programme pour évaluer les notes moyennes des élèves;
- le programme doit demander le nom de l'élève, suivi de son score unique;
- les noms peuvent être entrés dans n'importe quel ordre;
- la saisie d'un nom vide termine la saisie des données;
- une liste de tous les noms, ainsi que le score moyen évalué, doivent ensuite être émis.

Exemple :

```
school_class = {}

while True:

name = input("Enter the student's name (or type exit to stop):
")

if name == 'exit':

break

score = int(input("Enter the student's score (0-10): "))

if name in school_class:

school_class[name] += (score,)
```

```
else:

school_class[name] = (score,)

for name in sorted(school_class.keys()):

adding = 0

counter = 0

for score in school_class[name]:

adding += score

counter += 1

print(name, ":", adding / counter)
```

Maintenant, analysons-le ligne par ligne:

- **ligne 1** : créez un dictionnaire vide pour les données d'entrée; le nom de l'élève est utilisé comme clé, tandis que tous les scores associés sont stockés dans un tuple (le tuple peut être une valeur de dictionnaire - ce n'est pas un problème du tout)
- **ligne 3** : entrez dans une boucle "infinie" (ne vous inquiétez pas, elle se cassera au bon moment)
- **ligne 4** : lisez le nom de l'élève ici;
- **ligne 5-6** : si le nom est exit, quittez la boucle;
- **ligne 8** : demandez l'un des scores de l'élève (un entier compris entre 1 et 10)
- **ligne 10-11** : si le nom de l'élève est déjà dans le dictionnaire, allongez le tuple associé avec le nouveau score (notez l'opérateur + =)
- **ligne 12-13** : s'il s'agit d'un nouvel élève (inconnu du dictionnaire), créez une nouvelle entrée - sa valeur est un tuple à un élément contenant le score entré;
- **ligne 15** : parcourir les noms des élèves triés;
- **ligne 16-17** : initialiser les données nécessaires pour évaluer la moyenne (somme et compteur)

- **ligne 18-20** : nous parcourons le tuple, en prenant tous les scores suivants et en mettant à jour la somme, avec le compteur;
- **ligne 21** : évaluer et imprimer le nom et la note moyenne de l'élève.