NSI - première

Python 6 - Dictionnaires

qkzk

2021/04/27

Les dictionnaires : le type `dict`

Python propose le type `dict` qui permet d'enregistrer et de manipuler des paires de `clé` et `valeurs`.

Le type `dict`

Créer un dictionnaire

Considérons un exemple simple :

Clé	Valeur
Nom	Duchmol
Prénom	Raymond
age	22
Téléphone	0612345678
email	rd@example.fr

On peut enregistrer ces données de la manière suivante :

```
personne = {
  "nom": "Duchmol",
  "prenom": "Raymond",
  "age": 22,
  "tel": "0612345678",
  "email": "rd@example.fr"
}
```

Les clés sont généralement des chaînes de caractères mais, grosso modo, tout objet **non mutable** peut servir de clé.

Les valeurs sont de n'importe quel type.

Remarquez bien la syntaxe : `{ cle1: valeur1, cle2: valeur2 }`

La syntaxe `d = {}` crée un dictionnaire **vide** de nom `d`.

```
>>> personne
{"nom": "Duchmol", "prenom": "Raymond", "age": 22,
  "tel": "0612345678", "email": "rd@example.fr"}
>>> type(personne)
<class 'dict'>
```

Accéder aux valeurs par leur clé

L'accès à une valeur se fait à l'aide des crochets :

```
>>> personne["nom"]
"Duchmol"
```

La syntaxe `dictionnaire[cle]` donne la valeur correspondant à clé.

Elle provoque une exception `KeyError` si la clé ne fait pas partie des clés enregistrées.

Ajouter ou mettre à jour une valeur par une clé

Les dictionnaires sont *mutables*, on peut modifier ce qu'ils contiennent.

On peut mettre à jour une valeur ou ajouter une nouvelle valeur en utilisant la syntaxe précédente.

Par exemple après son anniversaire :

```
>>> personne["age"] = 23
```

Et s'il souhaite enregistrer sa ville :

```
>>> personne["ville"] = "Lille"
```

Appartenance

Le test d'appartenance `in` permet de tester la présence d'une clé dans le dictionnaire.

```
>>> "nom" in personne
True
>>> "sport" in personne
False
```

Effacer une clé

Il existe différentes méthodes mais la plus courante est d'utiliser `del`

Pour effacer la ville dans le dictionnaire précédent :

```
>>> ville in  personne
True
>>> del personne["ville"]
>>> ville in  personne
False
```

Itérer

On peut écrire des boucles qui parcourent un dictionnaire. Il existe trois méthodes principales pour cela :

1. itérer sur les *clés seulement*,
2. itérer sur les *valeurs seulement*,
3. itérer sur les *clés et les valeurs*.

Toujours avec l'exemple précédent :

Itérer sur les clés seulement

```
>>> for cle in personne:
...  print(cle)
"nom"
"prenom"
"age"
"tel"
"email"
```

Autre manière, plus explicite, utilisant `dict.keys()`

```
>>> for cle in personne.keys():
...  print(cle)
"nom"
"prenom"
"age"
"tel"
"email"
```

itérer sur les *valeurs seulement*,

On utilise cette fois `dict.values()`

```
>>> for valeur in personne.values():
...  print(valeur)
"Duchmol"
"Raymond"
22
"0612345678"
"rd"
```

itérer sur les *clés et les valeurs*.

C'est la méthode la plus courante, on récupère un tuple contenant la clé et la valeur. On peut le détupler dans une syntaxe courte :

```
>>> for cle, valeur in personne.items():
...     print(cle, "\t:", valeur)
"nom"       : "Duchmol"
"prenom"    : "Raymond"
"age"       : 22
"tel"       : "0612345678"
"email"     : "rd@example.fr"
```

Exemples

Considérons la liste de mot suivants :

```
mots = ["person", "woman", "man", "camera", "tv"]
```

Nous allons créer un dictionnaire enregitrant chaque mot et sa longueur.

Pourquoi ? Pour vous montrer, il ne sert à rien, on peut utiliser `len` lorsqu'on en a besoin.

```
mots = ["person", "woman", "man", "camera", "tv"]

longueurs = {}          # on crée un dictionnaire vide

for mot in mots:
    longueurs[mot] = len(mot)
print(longueurs)
```

Ce script va produire l'affichage suivant :

```
{"person": 6, "woman": 6, "man": 3, "camera": 6, "tv": 2}
```

Nous verrons dans le chapitre **liste et dictionnaires par compréhension** qu'on peut tout à fait créer cet objet en une ligne.

Autres méthodes du type `dict`

D'usage peu fréquent en NSI

Parmi les autres méthodes dont nous ne nous servirons pas souvent :

- `clear` : vide le dictionnaire,
- `copy` : comme son nom l'indique,
- `pop` : renvoie la valeur associée à une clé et la retire
- `update` : permet de regrouper deux dictionnaires en un seul en écrasant les clés présentes

La méthode `get`

Plus utile est la méthode `get`

```
>>> d = {1: "a", 2: "b"}
>>> d.get(1)
"a"
>>> d.get(3)
>>> d.get(3, "PAS PRESENT !!!!")
"PAS PRESENT !!!!"
```

Lorsqu'on appelle `d.get(cle)`, python renvoie la valeur associée à la clé *si* celle-ci est présente... et sinon il ne renvoie rien.

On peut préciser une valeur par défaut, comme dans `d.get(3, "PAS PRESENT !!!!")`

où l'on obtient la valeur `"PAS PRESENT !!!!"`

Cela permet donc d'éviter les erreurs de type `KeyError` lorsqu'on ne connait pas avec précision les valeurs contenues.

Exercices

Exercice 1

On considère les données suivantes, enregistrées dans le dictionnaire `d` :

Nom	Téléphone
Patrick	0612345678
Marie	0687654321
Hanae	0765432198
Piotr	0777666555

```
repertoire =  {"Patrick" : "0612345678",
               "Marie  " : "0687654321",
               "Hanae  " : "0765432198",
```

```
            "Piotr  " : "0777666555"}
```

1. Comment accéder au téléphone de `Piotr` ?
2. Comment savoir si “Fanny” est enregistrée dans le répertoire ?
3. Modifier le numéro de Patrick, il se termine par un 9 et non un 8.
4. Ajouter “Raoul” dont le numéro est “0789898989”
5. Supprimer “Marie” du repertoire.

Exercice 2

1. Créer à la main en une seule instruction le dictionnaire correspondant au tableau suivant :

nombre	carré
1	1
2	4
3	9
4	16
5	25
6	36
7	49

2. Créer le même dictionnaire en partant d’un dictionnaire *vide* et en ajoutant les nombres et leur carré instruction par instruction.
3. Recommencer en utilisant une boucle.

Exercice 3

En utilisant le dictionnaire précédent, produire l’affichage suivant : Votre programme doit faire 2 lignes, pas plus.

```
1       1
2       4
3       9
4       16
5       25
6       36
7       49
```

Remarque : lorsqu’on affiche la chaîne de caractères `"\t"` on obtient une tabulation.

Exercice 4

Convertir les deux listes suivantes en un dictionnaire en utilisant une seule boucle.

```
keys = ['Ten', 'Twenty', 'Thirty']
values = [10, 20, 30]

{'Ten': 10, 'Twenty': 20, 'Thirty': 30}
```

Exercice 5 - données enfouies

```
sampleDict = {
   "class":{
      "student":{
         "name":"Mike",
         "marks":{
            "physics":70,
            "history":80
         }
      }
   }
}
```

Comment accéder à la valeur correspondant à `"history"` ?

Exercice 6

Changez la clé `"city"` en `"location"` dans le dictionnaire suivant :

```
sampleDict = {
  "name": "Kelly",
  "age":25,
  "salary": 8000,
  "city": "New york"
}
```

sortie voulue :

```
{
  "name": "Kelly",
  "age":25,
  "salary": 8000,
  "location": "New york"
}
```

Exercice 7

Changez le salaire de Brad en 8500 depuis le dictionnaire suivant :

```
sampleDict = {
     'emp1': {'name': 'Jhon', 'salary': 7500},
     'emp2': {'name': 'Emma', 'salary': 8000},
     'emp3': {'name': 'Brad', 'salary': 6500}
}
```

Sortie attendue

```
sampleDict = {
     'emp1': {'name': 'Jhon', 'salary': 7500},
     'emp2': {'name': 'Emma', 'salary': 8000},
     'emp3': {'name': 'Brad', 'salary': 8500}
}
```

Exercice 8 - Compter les lettres d'une chaîne de caractère

Nous allons apprendre à compter les lettres d'une chaîne de caractère.

Voici la chaîne en question, utilisée fréquement pour remplir des zones de texte lorsqu'on développe une interface :

`lorem ipsum dolor sit amet consectetur adipiscing elit`

Nous allons produire le dictionnaire suivant :

```
{' ': 7, 'i': 6, 'e': 5, 't': 5, 'o': 4, 's': 4, 'l': 3, 'r': 3, 'm': 3,
 'c': 3, 'p': 2, 'u': 2, 'd': 2, 'a': 2, 'n': 2, 'g': 1}
```

Il contient le nombre d'apparition d'un caractère dans la chaîne précédente.

Voici l'algorithme :

```
créer un dictionnaire vide
Pour chaque lettre de la chaîne :
  si la lettre n'est pas présente dans le dictionaire:
      l'ajouter avec la valeur 1
  sinon:
      augmenter sa valeur de 1
```

1. Implémenter cet algorithme
2. Reprendre et l'implémenter dans une fonction `compteur`
3. Recommencer en utilisant un module appelé `collections` et sa fonction `Counter`.