Dictionnaires (type `dict`)

Cours

Un **dictionnaire** est un ensemble **non-ordonné** d'éléments sous la forme **clé-valeur**, qui peuvent être **modifiés** (muables[5]).

En Python les dictionnaires sont du type `dict`.

Les dictionnaires se révèlent très pratiques pour manipuler des structures complexes à décrire et quand les tableaux présentent leurs limites.

Création

Un dictionnaire est déclaré par une série d'éléments sous la forme couples **clés - valeurs** (key-value) séparés par des virgules, et le tout **encadré par des accolades** “{ }”.

```
>>> capitales = {"France": "Paris", "Italie": "Rome", "Espagne": "Madrid"}
>>> capitales
{'France': 'Paris', 'Italie': 'Rome', 'Espagne': 'Madrid'}
>>> type(capitales)
<class 'dict'>
```

Ici les clés de `capitales` sont `'France'`, `'Italie'` et `'Espagne'` ; les valeurs `'Paris'`, `'Rome'` et `'Madrid'`.

Il est aussi possible de créer un dictionnaire vide avec les accolades `{}` :

```
>>> capitales = {}
```

ou avec la fonction `dict()` :

```
>>> capitales = dict()
```

puis de remplir le dictionnaire avec différentes clés (`'France'`, `'Italie'` et `'Espagne'`) et leur valeurs (`'Paris'`, `'Rome'` et `'Madrid'`) :

```
>>> capitales["France"] = "Paris"
>>> capitales["Italie"] = "Rome"
>>> capitales["Espagne"] = "Madrid"
>>> capitales
{'France': 'Paris', 'Italie': 'Rome', 'Espagne': 'Madrid'}
```

Un dictionnaire est affiché sans ordre particulier.

Comme pour les tableaux, on peut aussi créer un dictionnaire par compréhension :

```
>>> d = {x: x**2 for x in range(10)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

Les clés de dictionnaire ne sont pas forcément des chaînes de caractères. Ici ce sont des entiers. On pourrait aussi les convertir en chaines de caractères avec str().

```
>>> d = {str(x):x**2 for x in range (10)}
>>> d
{'0': 0, '1': 1, '2': 4, '3': 9, '4': 16, '5': 25, '6': 36, '7': 49, '8': 64, '9': 81}
```

Il est aussi possible d'utiliser d'autres types de variables comme des p-uplets, mais ⚠️ les tableaux ne peuvent pas être les clés d'un dictionnaire (car muables) :

```
>>> d = {(0, 0): 'X', (0, 1): 'O', (1, 0): 'O', (1, 1): 'X'}
>>> d = {[0, 0]: 'X', [0, 1]: 'O', [1, 0]: 'O', [1, 1]: 'X'}
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Fonction `len()`

La fonction `len()` renvoie la longueur d'un dictionnaire, c'est-à-dire le nombre d'éléments (couple clé :valeur) qu'il contient.

```
>>> capitales = {"France": "Paris", "Italie": "Rome", "Espagne": "Madrid"}
>>> len(capitales)
3
```

Accès aux éléments

Les éléments d'un dictionnaire n'ont **pas d'ordre particulier**, il n'est donc pas possible d'accéder aux éléments par un indice de leur position (comme avec les p-uplets, tableaux et chaines de caractères).

```
>>> capitales[1]
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
KeyError: 1
```

Ce sont les **clés** du dictionnaire qui permettent d'accéder aux valeurs. Pour récupérer la valeur associée à la clé `key` dans un dictionnaire `d`, il suffit d'utiliser la syntaxe suivante `d[key]`[1].

```
>>> capitales["France"]
'Paris'
```

mais un erreur est levée si la clé n'existe pas :

```
>>> capitales['Allemagne']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Allemagne'
```

Le mot clé `in` permet de vérifier si une clé est présente dans un dictionnaire, il renvoie un booléen :

```
>>> "France" in capitales
True
>>> "Allemagne" in capitales
False
```

⚠️ `in` ne s'applique qu' aux clés d'un dictionnaire, pas à ses valeurs :

```
>>> "Paris" in capitales
False
```

De façon très similaire aux p-uplets et tableaux, le mot clé `in` permet aussi d'écrire une boucle pour parcourir (ou «itérer») toutes les clés d'un dictionnaire :

```
>>> for key in capitales:
...     print("La capitale de", key, "est", capitales[key])
France Paris
Italie Rome
Espagne Madrid
```

Méthodes `.keys()`, `.values()` et `.items()`

Les méthodes `.keys()` et `.values()` renvoient les clés et les valeurs d'un dictionnaire :

```
>>> capitales.keys()
dict_keys(['France', 'Italie', 'Espagne'])
>>> capitales.values()
dict_values(['Paris', 'Rome', 'Madrid'])
```

La méthode `.items()` renvoie tous les couples clé-valeur d'un dictionnaire :

```
>>> capitales.items()
dict_items([('France', 'Paris'), ('Italie', 'Rome'), ('Espagne', 'Madrid')])
```

Les mentions `dict_keys`, `dict_values`, `dict_items` indiquent que nous avons affaire à des objets un peu particuliers. Pour les utiliser il faut par exemple les transformer en tableaux avec la fonction `list()` :

```
>>> list(capitales.values())
['France', 'Italie', 'Espagne']
>>> list(capitales.items())
[('France', 'Paris'), ('Italie', 'Rome'), ('Espagne', 'Madrid')]
```

Modifier un dictionnaire

Modifier ou ajouter un élément

Comme pour les tableaux, on peut modifier une valeur dans un dictionnaire (mais à la différence des tableaux on la désigne par sa clé, pas par un indice) :

```
capitales["Italie"] = "Roma"
>>> capitales
{'Espagne': 'Madrid', 'France': 'Paris', 'Italie': 'Roma'}
```

Dans un tableau, on ne peut pas modifier la valeur d'un indice qui n'existe pas :

```
>>> pays = ['Madrid', 'Paris', 'Roma']
>>> pays[3] = 'Berlin'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

Mais dans un dictionnaire, si on essaie de modifier la valeur d'une clé qui n'existe pas, alors un nouveau couple clé-valeur est créé :

```
>>> capitales["Allemagne"] = 'Berlin'
>>> capitales
{'Allemagne': 'Berlin', 'Espagne': 'Madrid',  'France': 'Paris',  'Italie': 'Roma'}
```

Supprimer des éléments

Comme pour les tableaux, il est possible d'utiliser les méthodes `pop()` et `clear()` (mais pas `remove()`) pour supprimer des couples de clé-valeur :

- `d.pop(key)` [6] supprime du dictionnaire la clé `key` et renvoie la valeur associée :

  ```
  >>> d = {'one':1, 'two':2, 'three' :3}
  >>> d.pop('two')
  2
  ```

- `d.clear()` supprime tous les éléments du dictionnaire :

  ```
  >>> d = {'one':1, 'two' :2, 'three' :3}
  >>> d.clear()
  ```

Comme pour les tableaux, le mot clé `del` permet aussi de supprimer un élément d'un dictionnaire :

```
>>> del capitales['Allemagne']
>>> capitales
{'Espagne': 'Madrid', 'France': 'Paris', 'Italie': 'Roma'}
```

Ou encore le dictionnaire entier avec l'instruction `del capitales`, alors la variable `capitales` n'existe plus.

Dictionnaires muables

Les dictionnaires, comme les tableaux, sont de type muables, donc les mêmes limites s'appliquent.

Copie de dictionnaire

```
>>> d1 = {'one':1, 'two':2, 'three':3}
>>> d2 = d1
>>> d2['three'] = 4
>>> d1
{'one':1, 'two':2, 'three':4}
```

Comme avec les tableaux, `d1` a aussi été modifiée quand on a modifié `d2` ! Les deux variables `d1` et `d2` sont en fait deux noms qui font référence vers le même objet[2].

Pour copier un dictionnaire , il faut créer une copie explicite du dictionnaire initial[3] :

- Avec la fonction `dict(itérable)` qui renvoie un dictionnaire formé des éléments de la variable itérable :

  ```
  >>> d2 = dict(d)
  ```

- Ou encore utiliser la méthode `.copy()` :

  ```
  >>> d2 = d1.copy()
  ```

Dictionnaire passé en paramètre de fonction

Les mêmes effets qu'avec les tableaux peuvent être observés quand on passe un dictionnaire en paramètre d'une fonction : **la fonction peut en modifier le contenu**.

Illustrons cela par deux fonctions `f(x)` et `g(x)` qui modifient simplement la valeur d'un paramètre `x` et appelons ces fonctions en passant des variables `a` (type `int`) et `d` (type `dict`) en paramètre :

avec une variable de type "immuable"

```
def f(x):
    x = 2
```

avec une variable de type "muable"

```
def g(x):
    x['four']=4
```

Appelons ces fonctions en passant des variables `a` et `d` en paramètre [3]:

avec une variable de type "immuable"

```
>>> a = 1
>>> f(a)
>>> a
1
```

La valeur de `a` n'a pas été modifiée par la fonction `f`.

avec une variable de type "muable"

```
>>> d = {'one':1, 'two':2, 'three':3}
>>> h(d)
>>> d
{'four': 4, 'one': 1, 'three': 3, 'two': 2}
```

La valeur de `d` a été modifiée par la fonction `g` !

Conversion de type (cast)

La fonction `dict(tableau de p-uplets)` transforme un tableau de tuples (comme obtenu par `items()`) en dictionnaire :

```
>>> dict([("one" , 1), ("two" , 2), ("three" , 3)])
{'one': 1, 'three': 3, 'two': 2}
```

Un exemple d'utilisation de dictionnaire : les p-uplets nommés

Un **p-uplet nommé** est un p-uplet, dont les éléments sont appelés via un descripteur au lieu d'un indice. L'intérêt est d'améliorer la lisibilité du code pour réduire les risques d'erreurs. Le type des p-uplets nommés n'existe pas nativement dans Python[4]: , on peut alors utiliser des dictionnaires. Voici un exemple pour montrer la syntaxe :

```
>>> monsieurX = {"nom": "X", "prenom": "Monsieur", "age": 47}
>>> monsieurX["age"]
47
```

Dictionnaire de tableaux, dictionnaires de dictionnaires

De même qu'on peut utiliser des tableaux de tableaux, on peut utiliser des dictionnaires de tuples ou de tableaux :

```
>>> traduction = {"un": ["one", "eins"], "deux": ["two", "zwei"], "trois": ["three", "drei"]}
>>> traduction["un"]
['one', 'eins']
>>> traduction["un"][0]
'one'
```

ou même des dictionnaires de dictionnaires :

```
>>> traduction = {"un": {"Ang":"one", "All":"eins"}, "deux": {"Ang": "two", "All":"zwei"},
"trois": {"Ang":"three", "All":"drei"}}
>>> traduction["un"]
{'Ang': 'one', 'All': 'eins'}
>>> traduction["un"]["Ang"]
'one'
```

1. Ou alors utiliser en utilisant la méthode `.get()` qui permet de récupérer la valeur associée à une clé ou afficher un message si elle n'existe pas :

```
>>> capitale.get('France')
'Paris'
>>> capitale.get('Allemagne', "Cette clé n'existe pas")
Cette clé n'existe pas
```

↩

2. Pour s'en convaincre on peut vérifier les adresses des variables avec id() :

```
>>> id(d1)                      >>> id(d2)
2156942732160                       2156942732160
```

↩

3. Les variables `a` et `d` pourraient s'appeler aussi `x` ce qui donnerait le même résultat. ↩ ↩
4. Il existe un module Python `collection.namedtuple` mais le programme invite à utiliser des dictionnaires dans ce cas. ↩
5. ou *mutable* en anglais. ↩
6. La clé dans `d.pop(key)` est obligatoire, alors que pour les tableaux elle est facultative (par défaut le dernier élément est supprimé) ↩