Dictionnaires

Complément - niveau basique

Ce document résume les opérations courantes disponibles sur le type `dict`. On rappelle que le type `dict` est un type **mutable**.

Création en extension

On l'a vu, la méthode la plus directe pour créer un dictionnaire est en extension comme ceci

```
annuaire = {'marc': 35, 'alice': 30, 'eric': 38}
print annuaire
```

Création - la fonction dict

Comme pour les fonctions `int` ou `list`, la fonction `dict` est une fonction de construction de dictionnaire - on dit un constructeur - dont on a aussi vu dans la vidéo l'usage habituel à base d'une liste de tuples (`clé`, `valeur`)

```
annuaire = dict([('marc', 35), ('alice', 30), ('eric', 38)])
print annuaire
```

Remarquons qu'on peut aussi utiliser cette autre forme d'appel à `dict` pour un résultat équivalent

```
annuaire = dict(marc=35, alice=30, eric=38)
print annuaire
```

Remarquez ci-dessus l'absence de quotes autour des clés comme `marc`. Il s'agit d'un cas particulier de passage d'arguments que nous expliciterons plus longuement en fin de semaine 4.

Accès atomique

Pour accéder à la valeur associée à une clé, on utilise la notation à base de crochets `[]`

```
print 'la valeur pour marc est', annuaire['marc']
```

Cette forme d'accès ne fonctionne que si la clé est effectivement présente dans le dictionnaire. Dans le cas contraire une exception `KeyError` est levée. Si vous n'êtes pas sûr si la clé est présente, vous pouvez utiliser `get` qui accepte une valeur par défaut

```
print 'valeur pour marc', annuaire.get('marc', 0)
print 'valeur pour inconnu', annuaire.get('inconnu', 0)
```

Le dictionnaire est un type **mutable**, aussi on peut **modifier la valeur** d'une clé

```
annuaire['eric'] = 39
print annuaire
```

Ou encore, exactement de la même façon, **ajouter une entrée**

```
annuaire['bob'] = 42
print annuaire
```

Enfin pour **détruire une entrée**, on peut utiliser l'instruction `del` comme ceci

```
del annuaire['marc']
print annuaire
```

Pour savoir si une clé est présente ou non, il est conseillé d'utiliser l'opérateur d'appartenance `in` comme ceci

```
# forme recommandée
print 'john' in annuaire
```

Notez qu'avec des versions plus anciennes de python, vous pourrez trouver aussi du code qui utilise la méthode `has_key` qui n'est plus recommandée

```
# forme obsolete
annuaire.has_key('john')
```

Accès à toutes les entrées

La méthode la plus fréquente pour "balayer" tout un dictionnaire est à base de la méthode `iteritems`; voici par exemple comment on pourrait afficher le contenu

```
for nom, age in annuaire.iteritems():
    print "{}, age {}".format(nom, age)
```

On remarque d'abord que les entrées sont listées dans le désordre; ceci est dû à l'action de la fonction de hachage, que nous avons vue dans la vidéo précédente.

On remarque aussi que le nom de la méthode est un peu étrange. En fait, il existe aussi une méthode `items` qui remplit le même rôle, et dont le nom est plus simple à retenir. Cependant, nous vous conseillons d'utiliser systématiquement la méthode `iteritems` qui, étant implémentée comme un itérateur et non comme une liste, est préférable dès que la taille du dictionnaire devient importante.

Nous reviendrons sur la notion d'itérateurs en semaine 3, mais voici un premier aperçu de la

différence entre ces deux méthodes

```
print annuaire.items()
```

```
print annuaire.iteritems()
```

Comme vous l'avez peut-être deviné, la différence principale entre ces deux approches est que `items` construit **vraiment** un objet liste (qui peut être très grosse, et donc prendre de la place, et du temps à calculer), alors que l'itérateur rendu par `iteritems` est de taille constante, et se trouve construit très rapidement.

À nouveau ce concept sera approfondi en semaine 3, mais si vous ne devez retenir qu'une seule méthode pour "balayer" un dictionnaire, retenez **`iteritems`**

On peut obtenir la liste des clés et des valeurs avec

```
# utiliser de préférence ceci dans un "for"
print 'clés', annuaire.iterkeys()
print 'valeurs', annuaire.itervalues()
```

qui sont comme iteritems la version "itérateur" de `keys` et `values`

```
# utiliser ceci pour inspecter un dictionnaire dans un terminal
print 'clés', annuaire.keys()
print 'valeurs', annuaire.values()
```

Complément - niveau intermédiaire

La fonction `len`

On peut comme d'habitude obtenir la taille d'un dictionnaire avec la fonction `len`

```
print '{} entrées dans annuaire'.format(len(annuaire))
```

La méthode `update`

On peut également modifier un dictionnaire avec le contenu d'un autre dictionnaire avec la méthode `update`

```
annuaire.update({'jean':25, 'eric':70})
annuaire.items()
```

La méthode `setdefault`

Cette méthode (https://docs.python.org/2/library/stdtypes.html#dict.setdefault) permet, en un seul appel, de retourner la valeur d'une clé et de créer cette clé au besoin, c'est à dire si elle n'est pas encore présente

```
print 'avant', annuaire
print 'set_default eric',annuaire.setdefault('eric', 50)
print 'set_default inconnu',annuaire.setdefault('inconnu', 50)
print 'après', annuaire
```

Pour en savoir plus sur le type `dict`

Pour une liste exhaustive reportez-vous à la page de la documentation python ici

https://docs.python.org/2/library/stdtypes.html#mapping-types-dict (https://docs.python.org/2/library/stdtypes.html#mapping-types-dict)

Dictionnaires avec ordre

Comme cela devrait être clair à présent, un dictionnaire est non ordonné, dans ce sens qu'il ne se souvient pas de l'ordre dans lequel les éléments ont été insérés.

```
d = dict()
for i in ['a', 7, 3, 'x']:
    d[i] = i
for k, v in d.iteritems():
    print 'dict', k, v
```

Signalons également l'existence dans le module `collections` (https://docs.python.org/2/library/collections.html) de la classe `OrderedDict` (https://docs.python.org/2/library/collections.html#collections.O rderedDict) , qui est une extension du type `dict` mais qui possède cette bonne propriété:

```
from collections import OrderedDict
d = OrderedDict()
for i in ['a', 7, 3, 'x']:
    d[i] = i
for k, v in d.iteritems():
    print 'OrderedDict', k, v
```

Complément - niveau avancé

Vues sur un dictionnaire

Signalons enfin les méthodes (https://docs.python.org/2/library/stdtypes.html#dict.viewitems)

- `viewitems`
- `viewkeys`
- `viewvalues`

qui retournent des objets de type ***vue***, qui sont dynamiquement mis à jour:

```
# on crée un dictionnaire
```

```
d = dict(a=1, b=2, c=3)
# et une vue sur ce dictionnaire
view_values = d.viewvalues()
for v in view_values:
    print v


# si on modifie le dictionnaire, la vue est modifiée également
del d['b']
for v in view_values:
    print v
```

Reportez vous à la section sur les vues de dictionnaires (https://docs.python.org/2/library/stdtypes.html#dictionary-view- objects) pour plus de détails.