Ensembles (set)

Laurent Signac – CC-BY-SA – 29-09-20 0922 d5f28dc444c7f9c8dba0

Les type `set` permet en Python de manipuler des ensembles, et dispose de méthodes de manipulations spécifiques (intersection, union…). Le type `set` partage de nombreuses propriétés avec les dictionnaires (un `set` est comme un dictionnaire ne contenant que des clés)

Un ensemble est un type *collection*, *itérable*, *mutable*, n'est *pas ordonné*, et n'est donc *pas une séquence*.

Les éléments d'un ensemble, comme les clés d'une dictionnaire doivent être hachables, donc récursivement non mutables. Naturellement, il ne peut pas y avoir d'élément en double dans un ensemble (si on ajoute un élément qui y est déjà, il ne se passe rien)

La caractéristique principale est que le test d'appartenance d'un élément à l'ensemble est réalisée en complexité $O(1)$ c'est à dire en un temps indépendant de la taille du dictionnaire.

1 Création d'un ensemble

```
s1 = set()      # Ensemble vide
s2 = {"Atchoum", "Prof", "Simplet", "Dormeur",
      "Grincheux", "Joyeux", "Timide"}  # en extension
s3 = {k ** 2 for k in range(1, 10)} # en intention
s4 = set("abcde") # construction à partir de n"importe quel itérable
```

2 Modification d'un ensemble

On ajoute un élément dans un ensemble avec `add` :

```
nains = {"Atchoum", "Prof", "Simplet"}
nains.add("Dormeur")
nains
--> {'Atchoum', 'Dormeur', 'Simplet', 'Prof'}
nains.add("Atchoum") # Il y était déjà
nains
--> {'Atchoum', 'Dormeur', 'Simplet', 'Prof'}
```

On supprime un élément avec `discard` (`remove` est équivalent mais échoue si l'élément n'existe pas, contrairement à `discard`) :

```
nains = {"Atchoum", "Prof", "Thorin", "Simplet"}
nains.discard("Thorin")
nains
--> {'Atchoum', 'Prof', 'Simplet'}
nains.discard("Gimli")
nains
--> {'Atchoum', 'Prof', 'Simplet'}
```

3 Test d'appartenance

On peut tester l'appartenance d'une clé au dictionnaire (opération rapide en $O(1)$) :

```
nains = {"Atchoum", "Prof", "Simplet"}
"Prof" in nains
--> True
```

```
"Balin" in nains
--> False
```

4 Itérations

On peut itérer sur un ensemble :

```
for nom in nains:
    print("{} revient du boulot".format(nom))
```

Vous êtes invités à tester le code précédent. Vous devez être capable de le reproduire de mémoire. Vous devez aussi connaître par cœur les prénoms des 7 nains.

5 Opérations ensemblistes

Les opérations ensemblistes sont déjà disponibles dans la classe `set` :

```
carres = {k ** 2 for k in range(1, 101)} # carrés inférieurs à 10000
cubes = {k ** 3 for k in range(1, 22)} # cubes inférieurs à 10000
```

```
>>> carres & cubes # intesection : À la fois carré et cube
>>> cubes - carres # différence : Est un cube sans être un carré
>>> cubes | carres # union : Est un cube ou un carré
```

Les opérations suivantes ne font pas muter les ensembles, et renvoient un **nouvel ensemble**. Il existe des versions des même opérations faisant muter l'ensemble de départ (`intersection_update`, `update` (union), `difference_update`…)

La page de documentation vous donnera d'autres possibilités (test d'inclusion par exemple)

6 Avoir en tête quand utiliser un ensemble

On utilise un ensemble parce que c'est pratique ou parce que c'est efficace (ou les 2). En particulier, pour éliminer les doublons d'une liste, par exemple, mettre ses éléments dans un ensemble est efficace. Enfin, toutes les opérations ensemblistes disponibles sont aussi implémentées de manières efficace dans CPython.