W3-S3-C1-ensembles

December 15, 2014

1 Ensembles

1.1 Complément - niveau basique

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

1.1.1 Création en extension

On crée un ensemble avec les accolades, comme les dictionnaires (mais sans utiliser le caractère `:`) et cela donne par exemple

```
In []: heteroclite = {'marc', 12, 'pierre', (1, 2, 3), 'pierre'}
       print heteroclite
```

1.1.2 Création - la fonction `set`

Il devrait être clair à ce stade que, le nom du type étant `set`, la fonction `set` est un constructeur d'ensembles. On aurait donc aussi bien pu faire

```
In []: heteroclite2 = set(['marc', 12, 'pierre', (1, 2, 3), 'pierre'])
       print heteroclite2
```

1.1.3 Un élément dans un ensemble doit être globalement immuable

On a vu précédemment que les clés dans un dictionnaire doivent être globalement immuables. Pour exactement les mêmes raisons, les éléments d'un ensemble doivent aussi être globalement immuables.

```
# on ne peut pas insérer un tuple qui contient une liste
>>> ensemble = {(1, 2, [3, 4])}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Le type `set` étant lui-même mutable, on ne peut pas créer un ensemble d'ensembles

```
>>> ensemble = {{1, 2}}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
```

Et c'est une des raisons d'être du type `frozenset`.

1.1.4 Création - la fonction `frozenset`

Il n'existe pas de raccourci syntaxique comme les {} pour créer un ensemble immuable, qui doit être créé avec la fonction `frozenset`. Toutes les opérations documentées dans ce notebook, et qui n'ont pas besoin de modifier l'ensemble, sont disponibles sur un `frozenset`.

Parmi les fonctions exclues sur un `frozenset`, on peut citer: `update`, `pop`, `clear`, `remove`, `discard`...

1.1.5 Opérations simples

Test d'appartenance

```
In []: (1, 2, 3) in heteroclite
```

Cardinal

```
In []: len(heteroclite)
```

Manipulations

```
In []: ensemble = {1, 2, 1}
       ensemble
```

```
In []: # pour nettoyer
       ensemble.clear()
       ensemble
```

```
In []: # ajouter un element
       ensemble.add(1)
       ensemble
```

```
In []: # ajouter tous les elements d'un autre *ensemble*
       ensemble.update({2, (1, 2, 3), (1, 3, 5)})
       ensemble
```

```
In []: # enlever un element avec discard
       ensemble.discard((1, 3, 5))
       ensemble
```

```
In []: # discard fonctionne même si l'élément n'est pas présent
       ensemble.discard('foo')
       ensemble
```

```
In []: # enlever un élément avec remove
       ensemble.remove((1, 2, 3))
       ensemble
```

```
In []: # contrairement à discard, l'élément doit être présent,
       # sinon il y a une exception
       try:
           ensemble.remove('foo')
       except KeyError as e:
           print "remove a levé l'exception", e
```

La capture d'exception avec `try` et `except` sert à capturer une erreur d'exécution du programme (qu'on appelle exception) pour continuer le programme. Le but de cet exemple est simplement de montrer (d'une manière plus élégante que de voir simplement le programme planter avec une exception non capturée) que l'expression `ensemble.remove('foo')` génère une exception. Si ce concept vous paraît obscur, pas d'inquiétude, nous reviendrons dessus en détail en semaine 6.

```
In []: # pop() ressemble à la méthode éponyme sur les listes
       # sauf qu'il n'y a pas d'ordre dans un ensemble
       while ensemble:
           element = ensemble.pop()
           print "element", element
       print "et bien sûr maintenant l'ensemble est vide", ensemble
```

1.1.6 Opérations classiques sur les ensembles

Donnons nous deux ensembles simples.

```
In []: A2 = set([0, 2, 4, 6])
       print 'A2', A2
       A3 = set([0, 3, 6])
       print 'A3', A3
```

N'oubliez pas que les ensembles, comme les dictionnaires, ne sont **pas ordonnés**.

Remarques * Les notations des opérateurs sur les ensembles rappellent les opérateurs "bit-à-bit" sur les entiers. * Ces opérateurs sont également disponibles sous la forme de méthodes

Union

```
In []: A2 | A3
```

Intersection

```
In []: A2 & A3
```

Différence

```
In []: A2 - A3
```

```
In []: A3 - A2
```

Différence symétrique

On rappelle que $A\Delta B = (A - B) \cup (B - A)$

```
In []: A2 ^ A3
```

1.1.7 Comparaisons

Ici encore on se donne deux ensembles

```
In []: superset = {0, 1, 2, 3}
       print 'superset', superset
       subset =  {1, 3}
       print 'subset', subset
```

Égalité

```
In []: heteroclite == heteroclite2
```

Inclusion

```
In []: subset <= superset
```

```
In []: subset < superset
```

```
In []: heteroclite < heteroclite2
```

Ensembles disjoints

```
In []: heteroclite.isdisjoint(A3)
```

1.1.8 Pour en savoir plus

Reportez vous à la section sur les ensembles dans la documentation python