

Florent Gallaire's Blog

Free (libre) software and free (libre) culture (science and law)

[« Quel DPL pour 2013 ?](#)

[Lucas Nussbaum élu DPL pour 2013 »](#)

La syntaxe des sets en Python

Pour comprendre la genèse de la syntaxe des sets, il faut étudier celle des types conteneurs historiques de Python qui sont :

- les listes, délimitées par les crochets [et]
- les tuples, délimités par les parenthèses (et)
- les dictionnaires, délimités par les accolades { et }

Ces types conteneurs sont donc dotés d'une syntaxe légère et facilement utilisable :

```
>>> l = [1, 2, 3]
>>> l
[1, 2, 3]
>>> t = (1, 2, 3)
>>> t
(1, 2, 3)
>>> d = {1: '1', 2: '2', 3: '3'}
>>> d
{1: '1', 2: '2', 3: '3'}
```

Les tuples, qui sont en fait des listes immuables et que l'on ne peut donc pas modifier, sont souvent oubliés car assez peu utilisés, en tout cas de manière consciente. En effet, une simple énumération sans syntaxe spécifique est en fait un tuple :

```
>>> e = 1, 2, 3
>>> e
(1, 2, 3)
```

Si l'on peut donc dire que globalement, les listes et les dictionnaires répondent à la grande majorité des besoins des programmeurs, un quatrième type conteneur s'est lentement mais sûrement fait une place au soleil des pythonistes : les sets.

Un set est un ensemble, c'est-à-dire une collection non ordonnée d'éléments uniques, ce qui se révèle très pratique dans beaucoup d'usages courants. Ce nouveau type conteneur étant fourni en deux saveurs, l'une mutable (comme les listes), et l'autre immuable (comme les tuples).

Suite à la [PEP 218](#), il fut d'abord [introduit dans Python 2.3](#) sous la forme d'un nouveau module `sets` ajouté à la bibliothèque standard, avant de devenir un type *built-in* dans [Python 2.4](#). Cela représentait une amélioration syntaxique non négligeable, plus besoin d'écrire :

```
>>> import sets
>>> s_mutable = sets.Set([1, 2, 3])
>>> s_immutable = sets.ImmutableSet([1, 2, 3])
```

On pouvait dorénavant se contenter de :

```
>>> s_mutable = set([1,2,3])
>>> s_immutable = frozenset([1,2,3])
```

Et de bénéficier en plus d'une implémentation du type `set` en C et non plus en Python, avec la belle augmentation de performance qui va avec.

L'intégration des sets mutables (les plus utiles) dans le *core* du langage Python, au même niveau que les tuples, les listes et les dictionnaires, se heurtait encore à une limitation syntaxique : il fallait écrire `set([1, 2, 3])`. En effet, il n'y a que trois signes de ponctuation [ASCII 7 bits](#) fonctionnant par paire et facilement accessibles sur un clavier, les parenthèse, les crochets et les accolades, qui comme on l'a vu sont déjà utilisés respectivement par les tuples, les listes et les dictionnaires.

Mais que pouvait-on alors faire pour mieux intégrer syntaxiquement les sets à Python ? C'est dans [Python 3.0](#) que la solution a été trouvée : si les tuples et les listes sont des énumérations que l'on ne pourrait distinguer des sets, les dictionnaires sont eux bien différents, et l'on peut donc utiliser les accolades `{ et }` pour une énumération sans risque de confusion :

```
>>> s = {1, 2, 3}
>>> s
{1, 2, 3}
```

Il reste cependant une petite exception qui rend cette solution syntaxique imparfaite, et potentiellement génératrice d'erreurs ou d'incompréhensions, c'est le cas de l'ensemble vide. En effet, `{ }` ne peut pas représenter à la fois le dictionnaire vide **et** le set vide :

```
>>> s = {}
>>> s
{}
>>> isinstance(s, set)
False
>>> isinstance(s, dict)
True
```

De manière logique et rétrocompatible, `{ }` représente donc toujours le dictionnaire vide, et c'est `set()` qui permet de créer un set vide :

```
>>> s = set()
>>> s
set()
>>> s.add(1)
>>> s
{1}
```

Enfin, ce sucre syntaxique de Python 3 a été backporté dans [Python 2.7](#).

Tags: [Libre](#), [Python](#), [Python 3](#), [set](#)

[Tweet](#)

This entry was posted on jeudi, mars 14th, 2013 at 03:45 and is filed under [Uncategorized](#). You can follow any responses to this entry through the [RSS 2.0](#) feed. You can [leave a response](#), or [trackback](#) from your own site.

4 Responses to “La syntaxe des sets en Python”

1.  José dit :
[14 mars 2013 à 05:08](#)


Et en plus on a des opérations magiques sur les sets du genre `issubset`, `intersection`, `difference`...
Elle est pas belle la lib standard ?

[Répondre](#)

2.  [Elessar](#) dit :
[14 mars 2013 à 10:54](#)

Pour info, “set” se dit « ensemble » en français.

[Répondre](#)

3.  [alain](#) dit :
[14 mars 2013 à 19:44](#)


Une ecriture qui aurait pu être sympas pour les ensembles vides:

{,}

à la manière du tuple vide:

(,)

[Répondre](#)

-  [fgallaire](#) dit :
[28 décembre 2016 à 15:31](#)

La syntaxe pour un tuple vide est :

`t = ()`

C’est pour le tuple d’un seul élément que l’on doit utiliser la virgule :

`t = (1,)`

En effet, la syntaxe des tuples fonctionnant aussi sans le parenthèses

`t = (1)`

est équivalent de

`t = 1`

et c’est donc l’affectation d’un scalaire qui intervient dans ce cas, alors que

`t = 1,`

marche très bien.

[Répondre](#)

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit Comment

Florent Gallaire's Blog is proudly powered by [WordPress](#)
[Entries \(RSS\)](#) and [Comments \(RSS\)](#).