W2-S5-C1-tuple-et-virgule

December 15, 2014

1 La construction de tuples

1.1 Complément - niveau intermédiaire

1.1.1 Les tuples et la virgule terminale

Comme on l'a vu on peut construire un couple indifféremment comme ceci

```
In []: couple1 = 1, 2
       couple2 = (1, 2)
       couple1 == couple2
```

En réalité, la parenthèse est superflue, il se trouve toutefois qu'elle est largement utilisée pour améliorer la lisibilité des programmes.

Ajoutons que la dernière virgule est optionnelle - en général, c'est-à-dire pour les tuples à au moins 2 éléments. C'est-à-dire qu'on peut aussi bien écrire aussi:

```
In []: couple3 = 1, 2,
       couple4 = (1, 2,)
       couple3 == couple4
```

Qui est bien équivalent aussi aux deux premières formes

```
In []: couple1 == couple3
```

1.1.2 Conseil pour la présentation sur plusieurs lignes

En général d'ailleurs, la forme avec virgule terminale est plus pratique. Considérez par exemple l'initialisation suivante; on veut créer un tuple qui contient des listes (naturellement un tuple peut contenir n'importe quel objet python), et comme c'est assez long on préfère mettre un élément du tuple par ligne:

```
In []: mon_tuple = ([1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9],
                   )
```

L'avantage lorsqu'on choisit cette forme (avec parenthèses, et avec virgule terminale), c'est que d'abord il n'est pas nécessaire de mettre un backslash à la fin de chaque ligne; parce que l'on est à l'intérieur d'une zone parenthésée, l'interpréteur python "sait" que l'instruction n'est pas terminée et va se continuer sur la ligne suivante.

Deuxièmement, si on doit ultérieurement ajouter ou enlever un élément dans le tuple, il suffira d'enlever ou d'ajouter toute une ligne, sans avoir à s'occuper des virgules; si on avait choisi de ne pas faire figurer la virgule terminale, alors pour ajouter un item dans le tuple après le dernier, il ne faut pas oublier d'ajouter une virgule à la ligne précédente. Cette simplicité se répercute au niveau du getionnaire de code source, ou les différences dans le code sont plus faciles à visualiser.

Signalons enfin que ceci n'est pas propre aux tuples. La virgule terminale est également optionnelle pour les listes, ainsi d'ailleurs que pour tous les types python où cela fait du sens, comme les dictionnaires et les ensembles que nous verrons bientôt. Et dans tous les cas où on opte pour une présentation multi-lignes, il est conseillé de faire figurer une virgule terminale.

1.1.3 Tuples à un élément

Pour revenir à présent sur le cas des tuples à un seul élément, c'est un cas particulier; parmi les 4 syntaxes qu'on a vues ci-dessus, on obtiendrait dans ce cas

```
In []: simple1 = 1
       simple2 = (1)
       simple3 = 1,
       simple4 = (1,)
```

Il est bien évident que la première forme ne crée pas de tuple; en fait la seconde non plus, python lit ceci comme une expression parenthésée, et en fait ces deux formes créent un entier simple:

```
In []: type(simple2)
```

Les deux autres formes créent par contre toutes les deux un tuple à un élément comme on cherchait à le faire:

```
In []: type(simple3)
```

```
In []: simple3 == simple4
```

Pour conclure, disons donc qu'il est conseillé de **toujours mentionner une virgule terminale** lorsqu'on construit des tuples

1.1.4 Parenthèse parfois obligatoire

Dans certains cas vous vous apercevrez que la parenthèse est obligatoire. Par exemple on peut écrire

```
In []: x = (1,)
       (1,) == x
```

Mais si on essaie d'écrire le même test sans les parenthèses:

```
In []: 1, == x
```

python lève une erreur de syntaxe. Encore une bonne raison pour utiliser les parenthèses.

1.1.5 Addition de tuples

Bien que le type tuple est immuable, il est tout à fait légal d'additionner deux tuples puisque l'addition va produire un **nouveau** tuple.

```
In []: tuple1 = (1, 2,)
       tuple2 = (3, 4,)
       print 'addition', tuple1 + tuple2
```

Ainsi on peut également utiliser avec un tuple l'opérateur +=

```
In []: tuple1 = (1, 2,)
       tuple1 += (3, 4,)
       print 'apres ajout', tuple1
```

1.1.6 Construire des tuples élaborés

Malgré la possibilité de procéder par additions successives, la construction d'un tuple peut s'avérer fastidieuse.

Une astuce utile consiste à penser aux fonctions de conversion, pour construire un tuple à partir de - par exemple - une liste

Ainsi il est tout à fait possible par exemple de faire ceci

```
In []: # on fabrique une liste pas à pas
       liste = range(10)
       liste[9] = 'Inconnu'
       del liste [2:5]
       print 'la liste', liste

       # on convertit le résultat en tuple
       mon_tuple = tuple(liste)
       print 'le tuple', mon_tuple
```

1.1.7 Digression sur les noms de fonctions prédéfinies

Remarque. Vous avez peut-être observé que nous avons choisi de ne pas appeler notre tuple simplement `tuple`. C'est une bonne pratique en général d'éviter les noms de fonctions prédéfinies par python.

Ces variables en effet sont des variables "comme les autres". Imaginez qu'on ait en fait deux tuples à construire comme ci-dessus, voici ce qu'on obtiendrait si on n'avait pas pris cette précaution

```
In []: liste = range(10)
       tuple = tuple(liste)
       autre_liste = range(100)
       autre_tuple = tuple(autre_liste)
```

Il y a une erreur parce que nous avons remplacé (ligne 2) la valeur de la variable `tuple`, qui au départ référençait vers la fonction de conversion, par un objet tuple. Ainsi en ligne 4, lorsqu'on appelle à nouveau `tuple`, on essaie d'exécuter un objet qui n'est pas 'appelable' (not callable en anglais).

D'un autre coté, l'erreur est relativement facile à trouver dans ce cas. En cherchant toutes les occurences de `tuple` dans notre propre code on voit assez vite le problème. De plus, je vous rappelle que votre éditeur de texte **doit** faire de la coloration syntaxique, et que toutes les fonctions built-in (dont `tuple` et `list` font parties) sont colorées spécifiquement (par exemple, en violet sous IDLE). En pratique, avec un bon éditeur de texte et un peu d'expérience, cette erreur est très rare.