Sequence unpacking

Complément - niveau basique

Remarque préliminaire nous avons vainement cherché une traduction raisonnable pour ce trait du langage, connue en anglais sous le nom de ***sequence unpacking*** ou encore parfois ***tuple unpacking***, aussi pour éviter de créer de la confusion nous avons finalement décidé de conserver le terme anglais à l'identique.

Déjà rencontré

L'affectation dans python peut concerner plusieurs variables à la fois. En fait nous en avons déjà vu un exemple en Semaine 1, avec la fonction `fibonacci` dans laquelle il y avait ce fragment :

```
for i in range(2, n + 1):
    f2, f1 = f1, f1 + f2
```

Nous allons dans ce complément décortiquer les mécanismes derrière cette phrase qui a probablement excité votre curiosité :)

Un exemple simple

Commençons par un exemple simple à base de tuple. Imaginons qu'on dispose d'un tuple `couple` dont on sait qu'il a deux éléments :

```
couple = (100, 'spam')
```

On souhaite à présent extraire les deux valeurs pour les affecter à deux variables. Une solution naïve consiste bien sûr à faire simplement :

```
gauche = couple[0]
droite = couple[1]
print 'gauche', gauche, 'droite', droite
```

Cela fonctionne naturellement très bien, mais n'est pas très pythonique - comme on dit ;)

On préfèrera la formulation équivalente suivante :

```
(gauche, droite) = couple
print 'gauche', gauche, 'droite', droite
```

La logique ici consiste à dire, affecter les deux variables de sorte que le tuple `(gauche, droite`

`)` soit égal à `couple`

Remarquons que les parenthèses ici sont optionnelles, on peut tout aussi bien écrire :

```
gauche, droite = couple
print 'gauche', gauche, 'droite', droite
```

Autres types

Cette technique fonctionne aussi bien avec d'autres types. Par exemple :

```
liste = [1, 2, 3]
[gauche, milieu, droit] = liste
print 'gauche', gauche, 'milieu', milieu, 'droit', droit
```

En fait c'est même encore plus flexible que ça puisqu'on pourrait même écrire :

```
liste = [1, 2, 3]
gauche, milieu, droit = liste
print 'gauche', gauche, 'milieu', milieu, 'droit', droit
```

C'est presque magique, on a comparé un tuple avec une liste !

En réalité, les seules contraintes fixées par `python` sont que

- le terme à droite du signe `=` est un ***iterable*** (tuple, liste, string, etc.),
- le terme à gauche soit écrit comme un tuple ou une liste,
- les deux termes ont la même longueur.

La plupart du temps le terme de gauche est écrit comme un tuple. C'est pour cette raison que les deux termes ***tuple unpacking*** et ***sequence unpacking*** sont en vigueur.

La façon ***pythonique*** d'échanger deux variables

Une caractéristique intéressante de l'affectation par ***sequence unpacking*** est qu'elle est sûre; on n'a pas à se préoccuper d'un éventuel d'ordre d'évaluation, les valeurs **à droite** de l'affectation sont **toutes** évaluées en premier, et ainsi on peut par exemple échanger deux variables comme ceci :

```
a = 1
b = 2
a, b = b, a
print 'a', a, 'b', b
```

En profondeur

Le ***sequence unpacking*** ne se limite pas au premier niveau dans les structures, on peut extraire des données plus profondément imbriquées dans la structure de départ; par exemple avec en

entrée la liste :

```
structure = ['abc', [(1, 2), ([3], 4)], 5]
```

Si on souhaite extraire la valeur qui se trouve à l'emplacement de 3, on peut écrire :

```
(a, (b, ((trois,), c)), d) = structure
print 'trois', trois
```

Ou encore, sans doute un peu plus lisible :

```
(a, (b, ([trois], c)), d) = structure
print 'trois', trois
```

Naturellement on aurait aussi bien pu écrire ici quelque chose comme :

```
trois = structure[1][1][0][0]
print 'trois', trois
```

Affaire de goût évidemment. Mais n'oublions pas une des phrases du zen de python $\textit{Flat is better than nested}$, ce qui veut dire que ça n'est pas parce que vous pouvez faire des structures imbriquées complexes que vous devez le faire. Bien souvent, cela rend la lecture et la maintenance du code complexe, j'espère que l'exemple précédent vous en a convaincu.

Plusieurs occurrences d'une même variable

On peut également utiliser **plusieurs fois** la même variable dans la partie gauche de l'affectation.

```
entree = [1, 2, 3]
a, a, a = entree
print 'a', a
```

Attention toutefois, comme on le voit ici, python n'impose pas que les différentes occurrences de `a` correspondent à des valeurs identiques (en langage savant, on dirait que cela ne permet pas de faire de l'unification). De manière beaucoup plus pragmatique, l'interpréteur se contente de faire comme s'il faisait l'affectation plusieurs fois de gauche à droite, c'est-à-dire comme si il faisait :

```
a = 1; a = 2; a = 3
```

Cette technique n'est utilisée en pratique que pour les parties de la structure dont on n'a que faire dans le contexte. Dans ces cas-là, il arrive qu'on utilise le nom de variable `_`, dont on rappelle qu'il est légal, ou tout autre nom comme `ignored` pour manifester le fait que cette partie de la structure ne sera pas utilisé, par exemple :

```
entree = [1, 2, 3]
```

```
_, milieu, _ = entree
print 'milieu', milieu

ignored, ignored, right = entree
print 'right', right
```