

W2-S7-C1-for-sur-plusieurs-variables

December 15, 2014

1 Plusieurs variables dans une boucle for

1.1 Complément - niveau basique

Nous avons vu précédemment (séquence ‘Les tuples’, complément ‘Sequence unpacking’) la possibilité d’affecter plusieurs variables par filtrage à partir d’un seul objet, comme ceci :

```
In []: item = (1, 2)
      a, b = item
      print 'a', a, 'b', b
```

D’une façon analogue, il est possible de faire une boucle `for` qui itère sur une seule liste mais qui ‘agit’ sur plusieurs variables, comme ceci :

```
In []: entrees = [(1, 2), (3, 4), (5, 6)]
      for a, b in entrees:
          print 'a', a, 'b', b
```

Chaque itération retourne un tuple, et les variables `a` et `b` vont être affectées à, respectivement, le premier et le deuxième élément du tuple. Cette mécanique est massivement utilisée en python.

1.2 Complément - niveau intermédiaire

1.2.1 La fonction zip

Voici un exemple très simple qui utilise la technique qu’on vient de voir.

Imaginons qu’on dispose de deux listes de longueurs égales, dont on sait que les entrées correspondent une à une, comme par exemple :

```
In []: villes = ["Paris", "Nice", "Lyon"]
      populations = [2*10**6, 4*10**5, 10**6]
```

Afin d’écrire facilement un code qui “associe” les deux listes entre elles, python fournit une fonction *built-in* baptisée `zip`; voyons ce qu’elle peut nous apporter sur cet exemple :

```
In []: zip(villes, populations)
```

On le voit, on obtient en retour une liste composée de tuples. On peut à présent écrire une boucle `for` comme ceci

```
In []: for ville, population in zip(villes, populations):
      print population, "habitants a", ville
```

Qui est, il nous semble, beaucoup plus lisible que ce que l’on serait amené à écrire avec des langages plus traditionnels.

Tout ceci se généralise naturellement à plus de deux variables.

```
In []: for i, j, k in zip(range(3), range(100, 103), range(200, 203)):
        print 'i', i, 'j', j, 'k', k
```

Remarque lorsqu'on passe à `zip` des listes de tailles différentes, le résultat est tronqué et la liste retournée a la taille de la plus petite liste en entrée.

1.2.2 La fonction `enumerate`

Une fonction utile permet d'itérer sur une liste avec l'indice dans la liste, il s'agit de `enumerate`

```
In []: for i, ville in enumerate(villes):
        print i, ville
```

Cette forme est plus simple et plus lisible que les formes suivantes qui sont équivalentes, mais qui ne sont pas pythoniques

```
In []: for i in range(len(villes)):
        print i, villes[i]
```

```
In []: for i, ville in zip(range(len(villes)), villes):
        print i, ville
```