Gérer des enregistrements

Complément - niveau intermédiaire

Implémenter un enregistrement comme un dictionnaire

Il nous faut faire le lien entre dictionnaire python et la notion d'enregistrement, c'est-à-dire une donnée composite qui contient plusieurs champs. (À cette notion correspond, selon les langages, ce qu'on appelle un `struct` ou un `record`)

Imaginons qu'on veuille manipuler un ensemble de données concernant des personnes; chaque personne est supposée avoir un nom, un age et une adresse mail.

Il est possible, et assez fréquent, d'utiliser le dictionnaire comme support pour modéliser ces données comme ceci

```
personnes = [
    {'nom': 'pierre',  'age': 25, 'email': 'pierre@foo.com'},
    {'nom': 'paul',    'age': 18, 'email': 'paul@bar.com'},
    {'nom': 'jacques', 'age': 52, 'email': 'jacques@cool.com'},
]
```

Bon, très bien, nous avons nos données, il est facile de les utiliser.

Par exemple, pour l'anniversaire de pierre on fera:

```
personnes[0]['age'] += 1
```

Ce qui nous donne

```
for personne in personnes:
    print 10*"="
    for info, valeur in personne.items():
        print "{} -> {}".format(info, valeur)
```

Un dictionnaire pour indexer les enregistrements

Cela dit, il est bien clair que cette façon de faire n'est pas très pratique; pour marquer l'anniversaire de pierre on ne sait bien entendu pas que son enregistrement est le premier dans la liste. C'est pourquoi il est plus adapté, pour modéliser ces informations, d'utiliser non pas une liste, mais à nouveau... un dictionnaire.

Si on imagine qu'on a commencé par lire ces données séquentiellement dans un fichier, et

qu'on a calculé l'objet `personnes` comme la liste qu'on a vue ci- dessus, alors il est possible de construire un index de ces dictionnaires, (un dictionnaire de dictionnaires, donc).

C'est-à-dire

```
index_par_nom = {personne['nom']: personne for personne in personnes}

print "enregistrement pour pierre", index_par_nom['pierre']
```

Attardons nous un tout petit peu; nous avons construit un dictionnaire par compréhension, en créant autant d'entrées que de personnes. Nous aborderons en détail la notion de compréhension en semaine 4, donc si cette notation vous paraît étrange pour le moment, pas d'inquiétude.

Le résultat est donc un dictionnaire qu'on peut afficher comme ceci:

```
for nom, record in index_par_nom.iteritems():
    print "Nom : {} -> enregistrement : {}".format(nom, record)
```

Dans cet exemple, le premier niveau de dictionnaire permet de trouver rapidement un objet à partir d'un nom; dans le second niveau au contraire on utilise le dictionnaire pour implémenter un enregistrement, à la façon d'un `struct` en C.

Techniques similaires

Notons enfin qu'il existe aussi, en python, un autre mécanisme qui peut être utilisé pour gérer ce genre d'objets composites, ce sont les classes que nous verrons en semaine 5, et qui permettent de définir de nouveaux `types` plutôt que, comme nous l'avons fait ici, d'utiliser un type prédéfini. Dans ce sens, l'utilisation d'une classe permet davantage de souplesse, au prix de davantage d'effort.