W3-S2-E2-marine-dictionnaire

December 15, 2014

1 Fusionner des données

1.1 Exercices

Cet exercice vient en deux versions, une de niveau basique et une de niveau intermédiaire.

La version basique est une application de la technique d'indexation qu'on a vue dans le complément "Gérer des enregistrements". On peut très bien faire les deux versions dans l'ordre, une fois qu'on a fait la version basique on est en principe un peu plus avancé pour aborder la version intermédiaire.

Vous trouverez également en fin de page quelques astuces pour utiliser le framework d'exercices, et notamment pour calculer vous mêmes le résultat attendu.

1.1.1 Contexte

Nous allons commencer à utiliser des données un peu plus réalistes. Il s'agit de données obtenues auprès de MarineTraffic - et légèrement simplifiées pour les besoins de l'exercice. Ce site expose les coordonnées géographiques de bateaux observées en mer au travers d'un réseau de collecte de type *crowdsourcing*.

De manière à optimiser le volume de données à transférer, l'API de MarineTraffic offre deux modes pour obtenir les données * **mode étendu** : chaque mesure (bateau x position x temps) est accompagnée de tous les détails du bateau (`id`, nom, pays de rattachement, etc..) * **mode abrégé** : chaque mesure est uniquement attachée à l'`id` du bateau.

En effet, chaque bateau possède un identifiant unique qui est un entier, que l'on note `id`.

1.1.2 Chargement des données

Nous allons travailler avec une copie locale de ces données; vous avez dès maintenant accès aux deux fichiers * `data/marine-e1-ext.json` - données étendues * `data/marine-e1-abb.json` - données abrégées

Pour charger ces fichiers qui sont au format JSON, la connaissance intime de ce format n'est pas nécessaire, nous allons utiliser le module `json`. Vous pouvez utiliser la cellule qui suit telle quelle, ces détails ne font pas partie de l'exercice parce que cette cellule utilise des notions que nous verrons dans les semaines à venir.

```
In []: # load data from files
       import json

       with open("data/marine-e1-ext.json") as feed:
           extended = json.load(feed)

       with open("data/marine-e1-abb.json") as feed:
           abbreviated = json.load(feed)
```

1.1.3 Format des données

Le format de ces données est relativement simple, il s'agit dans les deux cas d'une liste d'entrées - une par bateau.

Chaque entrée à son tour est une liste qui contient :

```
mode étendu: [id, latitude, longitude, date_heure, nom_bateau, code_pays, ...]
mode abrégé: [id, latitude, longitude, date_heure]
```

sachant que les entrées après le code pays dans le format étendu ne nous intéressent pas pour cet exercice.

```
In []: # une entrée étendue est une liste qui ressemble à ceci
       print extended[7]
```

```
In []: # une entrée abrégée ressemblent à ceci
       print abbreviated[0]
```

On précise également que les deux listes `extended` et `abbreviated` possèdent exactement **le même nombre** d'entrées et correspondent **aux mêmes bateaux** - mais naturellement à des moments différents, et **pas forcément dans le même ordre**.

1.1.4 Exercice - niveau basique

But de l'exercice On vous demande d'écrire une fonction `index` qui calcule, à partir de la liste des données étendues, un dictionnaire qui est : * indexé par l'`id` de chaque bateau, * et qui a pour valeur la liste qui décrit le bateau correspondant.

De manière plus imagée, si :

```
extended = [ bateau1, bateau2, ... ]
```

et si

```
bateau1 = [ id_bateau1, latitude, ... ]
```

on doit obtenir comme résultat de `index` un dictionnaire

```
id_bateau1 -> [ id_bateau1, latitude, ... ]
```

Bref, on veut pourvoir retrouver les différents éléments de la liste `extended` par accès direct (un seul *lookup*) dans l'index.

```
In []: # le résultat attendu
       from corrections.w3_marine_dict import exo_index
       result_index = exo_index.resultat(extended)

       # a quoi ressemble le résultat pour un bateau au hasard
       from pprint import pprint
       for key_value in result_index.iteritems():
           pprint(key_value)
           break
```

Votre code

```
In []: def index(extended):
           "<votre_code>"
```

Validation

```
In []: exo_index.correction(index, extended)
```

Vous remarquerez d'ailleurs que la seule chose qu'on utilise dans cet exercice, c'est que l'id des bateaux arrive en première position (dans la liste qui matérialise le bateau), aussi votre code doit marcher à l'identique avec les bateaux abrégés :

```
In []: exo_index.correction(index, abbreviated)
```

1.1.5 Exercice - niveau intermédiaire

But de l'exercice On vous demande d'écrire une fonction `merge` qui fasse une consolidation des données, de façon à obtenir en sortie un dictionnaire:

```
id -> [ nom_bateau, code_pays, position_etendu, position_abrege ]
```

dans lequel les deux objets `position` sont tous les deux des tuples de la forme

```
(latitude, longitude, date_heure)
```

Voici par exemple un couple clé-valeur dans le résultat attendu. Profitons-en pour découvrir un utilitaire parfois pratique: le module `pprint` pour pretty-printer

```
In []: # le résultat attendu
       from corrections.w3_marine_dict import exo_merge
       result_merge = exo_merge.resultat(extended, abbreviated)

       # a quoi ressemble le résultat pour un bateau au hasard
       from pprint import pprint
       for key_value in result_merge.iteritems():
           pprint(key_value)
           break
```

Votre code

```
In []: def merge(extended, abbreviated):
           "<votre_code>"
```

Validation

```
In []: exo_merge.correction(merge, extended, abbreviated)
```

1.1.6 Les fichiers de données

Pour télécharger les deux fichiers de données : * `data/marine-e1-ext.json` * `data/marine-e1-abb.json`

1.1.7 Quelques mots sur le framework d'exercices

Le format de la page est très étroit, et cela rend assez compliqué le rendu des résultats sous la forme d'un tableau comparatif. Dans le cas de cet exercice, qui utilise des données réelles et bavardes, la mise au point peut nécessiter de travailler un peu autrement, et voici quelques suggestions.

Tout d'abord rappelez-vous que vous pouvez créer un cellule où vous voulez avec le menu `Insert`.

Et ensuite exécuter votre propre code en dehors de la fonction de correction

```
In []: # en supposant que votre fonction index() renvoie un dictionnaire
       mon_index = index(extended)
       for key_value in mon_index.items():
           pprint(key_value)
           break
```

Dans le cas de cet exercice, vous pouvez aussi vous fabriquer vous-mêmes des données plus courtes :

```
In []: extended_sample = [b for b in extended if b[0] <= 220000000]
       print len(extended_sample)
```

```
In []: abbreviated_sample = [b for b in abbreviated if b[0] <= 220000000]
       print len(extended_sample)
```

Puisque comme ceci vous gardez la bonne propriété que les ids correspondent entre les deux listes.
Vous pouvez utiliser la correction avec d'autres données si ça vous est utile.

```
In []: exo_index.correction(index, abbreviated_sample)
```

Bref le message principal ici est que vous avez toute la puissance de l'interpréteur python pour vous aider dans votre travail, n'hésitez pas à vous en servir !