M1SOL041 - Documents électroniques
Master Langue et Informatique
TD7
2023-2024
Le format JSON (I)

Gaël Lejeune, Oumaima Zine
Sorbonne Université

Rappel de fonctions utilisées fréquemment :

```
def lire fichier(chemin):
     with open(chemin , "r", encoding = "utf-8") as f:
          chaine = f.read()
     return chaine
def decouper mots ( chaine ):
     return chaine .split ()
```

Exercice 1 : Manipulations de base en Json

Récupérez le fichier JSON et le corpus multilingue depuis Moodle ou sur https://github.com/ozine/MLI_7.

Nous allons tout d'abord travailler sur le premier fichier afin de se familiariser avec des **map** Json (tableau associatif ou dictionnaire en Python).

Ouvrez le fichier **fr.json** (attention à le placer au bon endroit) de la façon suivante :

```
import json
with open("fichiers TD4 /fr.json") as f:
dico = json.load(f)
```

Astuce : si vous ne savez pas où vous êtes exactement dans le système de fichier et que vous avez des FileNotFound à répétition, essayez les deux commandes suivantes :

```
import os , glob
print(" Affiche le chemin absolu vers le dossier courant ")
print(os. getcwd ())
print("Ce qu'il y a dans le dossier courant :")
print(glob.glob("./*"))
print("Ce qu'il y a 'en dessous ' du dossier courant ")
print(glob.glob(" ./*/* "))
```

Revenons à nos moutons :

```
print(type(dico))
print(dico.keys())#affiche les clés
```

Effectivement au premier niveau on a un dictionnaire. On peut donc le parcourir, comme on l'a vu dans le cours de Méthodologie :

```
for cle , valeur in dico.items():
print(cle)
print(type(valeur))
```

On peut gérer plusieurs types de variables que nous avons déjà rencontré : chaînes, booléens, nombres, listes (tableaux indicés) et dictionnaires (tableaux associatifs).

Exercice 2 : Ecrire du Json

Écrire dans un fichier Json se fait de la façon suivante :

```
with open("toto.json", "w") as w:
w.write(json.dumps(dico))
```

Plus exactement ici on a sérialisé de façon à transformer la structure de données en quelque chose d'exportable (fichier texte). Avec la méthode load on avait donc désérialisé c'est à dire qu'on avait extrait une structure de données à partir du fichier texte. Allons voir ce qui se passe dans toto.json en gérant l'encoding :

```
with open("toto.json", "r", encoding ="utf-8") as f:
dico = f.read()
print("Dans toto.json :",type(dico))
print(dico[:100])
```

On a une chaîne de caractères. En effet, c'est un fichier texte que l'on a lu et la méthode read nous donne une une suite de caractères. Le fait de l'appeler dico n'y change rien, pas plus que l'extension ".json". On peut toutefois demander à Python de nous trouver la structure de données qu'il y a derrière grâce à eval:

```
dico = eval(dico)
print("Avec eval :", type(dico))
```

Utiliser eval est plus lent mais plus générique, eval va en fait exécuter le code qui lui est passé entre parenthèses. Si c'est une variable, elle est stockée en mémoire.

Json est plus spécifique: il est là pour les données et c'est tout. Mais il offre des options très pratiques pour avoir des données à la fois lisibles pour l'œil humain et (dé)sérialisables. Observez la différence entre toto.json et toto pas lisible.json après avoir exécuté le code suivant (on doit aussi gérer l'encoding en écriture):

```
with open("toto_pas_lisible.json", "w", encoding ="utf-8") as w:
w.write(json.dumps(str(dico), indent = 2))

with open("toto.json", "w", encoding ="utf-8") as w:
w.write(json.dumps(dico , indent = 2))
```

Retournez voir toto.json, on a ajouté l'indentation ce qui est très pratique mais vous verrez que les caractères accentués apparaissent sous leur représentation dite Unicode.

C'est fait pour que le fichier respecte le jeu de caractères Ascii, les caractères accentués sont codés en ascii : "\uXXXX". On peut donc demander gentiment à la librairie de ne pas assurer la compatibilité Ascii et donc de représenter les caractères accentués avec leur vraie graphie :

```
with open("toto.json", "w") as w:
w.write(json.dumps(dico , indent = 2, ensure_ascii =False))
```

Voilà notre toto.json qui reprend à peu de choses près sa forme originelle. On peut se demander pourquoi il est si compliqué de s'assurer de l'encodage des caractères.

C'est que les caractères que l'on voit à l'écran ne sont obtenus qu'après transformation de binaire (des zéros et de uns) en caractères selon un procédé nommé décodage. C'est la notion d'encoding, abordée en Épistémologie (Figure 1)

	Encoding:Utf-8	Encoding:ISO-8859-1 (latin1)
Interprété comme:Utf-8	Étonnant que ça ait été si délicat à faire.	?tonnant que ?a ait ?t? si d?licat ? faire.
Interprété comme:Latin-1	Ã<89>tonnant que Ã§a ait Ã©tÃ© si dÃ©licat Ã faire.	Étonnant que ça ait été si délicat à faire.

Figure 1: Illustration de problèmes d'encoding

Exercice 3 : Structuration d'un corpus en Json

Nous allons jouer avec le corpus corpus_multi (que vous devez avoir téléchargé sur Moodle). Le but est de parcourir le corpus de manière à construire puis à enregistrer une structure de données qui à chaque fichier associe un dictionnaire comportant les informations suivantes :

- Chemin relatif du fichier
- Langue du fichier
- Taille en caractères et taille en mots

Le format de sortie sera une liste où l'on ajoute à chaque tour de boucle un dictionnaire décrivant les infos ci-dessus pour chaque fichier.

Quelques Astuces :

1. Outils : les librairies re, glob et bien sûr json, les fonctions lire fichier et decouper mots.
2. Au début ne stockez qu'une info (le chemin par exemple) juste pour voir quels sont les obstacles pour la suite
3. Comme le calcul peut être un peu long, vous pouvez commencer à travailler sur seulement 10 fichiers en réduisant la liste de fichiers sortie par glob :

```
import re
for chemin in glob.glob("corpus_multi /*/*/*" )[:10]:
elems = re.split("/", chemin )
langue = elems[1] #l'indice dépend du votre chemin
print(chemin)#juste une fois pour voir
print(elems)#idem
print(langue)#idem
```

Vous devriez obtenir quelque chose comme dans la Figure 2.

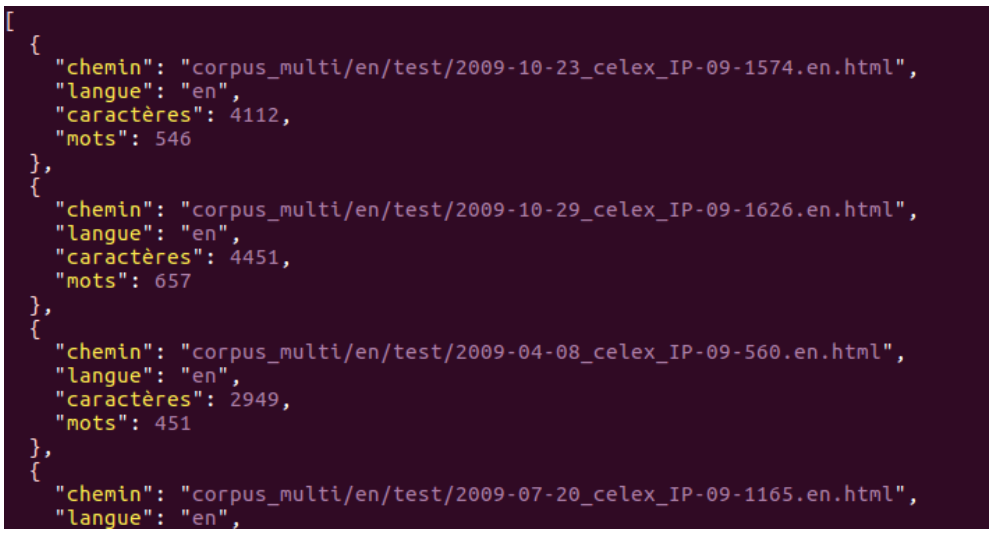

Figure 2: Visualisation de la sortie attendue