TD n° 4

Interroger une API REST

Dans ce TD, nous allons créer un programme qui interroge des serveurs et obtenir des informations et qui les traite.

Pour cela, nous allons nous appuyer sur trois APIs publiques :
- `https://api.gouv.fr/documentation/api-professionnels-bio` : AgenceBIO ;
- `https://api.gouv.fr/documentation/api-recherche-entreprises` : Recherche d'entreprises ;
- `https://nominatim.org/release-docs/develop/api/Search/` : OpenStreetMap ;
- `https://geoservices.ign.fr/documentation/services/api-et-services-ogc/itineraires/api` : IGN itinéraires.

Mise en place de l'environnement de développement

Ce TP nécessite le module python `requests`.

1. Installer le module `requests`.

Linux, Mac (sans anaconda) `pip install requests`
(essayer `pip3` au lieu de `pip` si ça ne marche pas)
Sous Mac, il peut être nécessaire de suivre un tutoriel.
Anaconda `conda install requests`
Windows + Anaconda prompt `pip install requests`
Si rien n'a marché — `pip3 install requests`
— `py -m pip install requests`
— `python -m pip install requests`
— `python3 -m pip install requests`

Nota Bene 4.1: Installation de `requests`

1 Premières requêtes

Dans cette section nous allons nous familiariser avec les principales méthodes fournies par `requests`.

Pour les questions de cette section, vous serez amenés à consulter la documentation de `requests`.

La seule question de cette section qui demande plus d'une (à deux, selon votre style) lignes de code est la question 9. Le programme principal doit vous permettre de tester votre code.

Vous avez deux manières de faire ce TD :
- Soit créer des fonctions (voire des portions de code en programme principale) en utilisant le module `requests` directement
  en partant de `TD_requests_template.py` ;
- Soit encapsuler les fonctions du module `requests` au sein d'une classe `Client`
  en partant de `TD_requests_class_template.py`.

2. Récupérer le fichier de base de votre choix (encapsulation ou non) [1].
3. En utilisant `requests.compat.urljoin` [2], compléter la fonction/méthode `make_url()`, pour qu'elle crée l'url de la ressource d'après le protocole, le nom de domaine et la route.
4. Afficher l'url de la requête effectuée [3] / compléter la méthode `lr_url`
5. Afficher le `code de statut` de la réponse du serveur à la dernière requête effectuée / compléter le code de la méthode `lr_status_code` (la réponse du serveur est stockée dans `__r__`) [4].
6. Afficher les en-têtes de la réponse du serveur à la dernière requête effectuée / compléter le code de la méthode `lr_headers` [5]
7. Afficher le `content-type` de la réponse du serveur.
8. Afficher le **texte** de la réponse du serveur / compléter le code de la méthode `lr_response`, pour qu'elle le renvoie.
9. En utilisant l'attribut `history`[3] de la réponse envoyée par `requests.get`, afficher la liste des **url** et **statuts** par lesquels la requête est passée / modifier la méthode `lr_redirections`, qui renvoie l'historique des redirections et utiliser cette méthode dans le programme principal.
   Vous pourrez ainsi constater les redirection du TD1.

2 Interprétation de réponse et paramètres

Dans cette section, toutes les requêtes s'adressent à l'API AgenceBIO, dont l'url de base est :
- `https://opendata.agencebio.org/api/gouv/operateurs`

10. Allez regarder la documentation de l'API [6] et essayez la dans l'interface fournie : cherchez tous les distributeurs de produits bio dans le Rhône.
11. Quelle était l'url de la requête effectuée par le site Web ?

---

1. L'encapsulation est plus "verbeuse" et ardue au premier abord, mais plus facilement adaptable ensuite.
2. `https://stackoverflow.com/a/23691720`
`urljoin` attend une url de base avec un protocole comme premier paramètre.
3. `https://requests.readthedocs.io/en/latest/user/quickstart/#redirection-and-history`
4. `https://requests.readthedocs.io/en/latest/user/quickstart/#response-status-codes`
5. `https://requests.readthedocs.io/en/latest/user/quickstart/#response-headers`
6. `https://api.gouv.fr/documentation/api-professionnels-bio`

12. Inspirez-vous du code réalisé dans la section 1 pour effecuer la requête de la question 11. Quel est le type de la variable qui contient le texte de la réponse ?

---

13. Utiliser la méthode `.json()` [7] fourni par `requests` pour interpréter la chaine de caractère récupérée / modifier la méthode `lr_response`, pour qu'elle permette de choisir entre le contenu textuel de la réponse et le contenu `json` interprété selon son type.
14. Interpréter le `json` obtenu à la question 12 et donner la « raison sociale » du dernier résultat.
15. Pour améliorer la lisibilité du code, utiliser l'argument `params` de `requests.get`, qui liste les paramètres sous forme de dictionnaire [8] / modifier la méthode `get` pour qu'elle prenne un argument optionnel supplémentaire.
    Tester cette méthode et vérifier l'url produite.

3 Mise en pratique : en attendant le projet...

Dans le projet, vous allez essayer de fournir un service qui permettra de choisir *judicieusement* des producteurs. Pour effectuer ce choix, vous aurez besoin de données de base :

— un lieu de référence (le restaurant où les ingrédients seront livrés) ;
— une liste d'ingrédients.

À partir de ces données de base, vous aurez besoin d'autres données à calculer grâce à des APIs pour chaque ingrédient :

— le nom de l'entreprise (producteur) ;
— le ou les produits concernés ;
— le nom du gérant ;
— le nombre de km de l'itinéraire le plus court entre l'adresse d'implantation et le producteur.

3.1 Récupération des données

16. Écrire une fonction/méthode qui permette d'obtenir les coordonnées (latitude, longitude) d'une adresse, en utilisant l'API openstreetmap.

17. En utilisant l'API AgenceBIO, écrire une fonction ou méthode qui à la donnée d'une denrée et des coordonnées d'un point géographique permette de récupérer le producteur (de la denrée) le plus proche (à vol d'oiseau).
    Pour tester cette méthode vous pouvez chercher des `'Pommes de terre'`.

18. En utilisant l'API IGN itinéraire, écrire une fonction/méthode qui trouve la distance la plus courte, en voiture, entre deux points d'après leurs coordonnées géographiques.
    NB : Procéder comme dans la section 2 vous facilitera le choix des paramètres.

19. En utilisant l'API Recherche d'entreprise, écrire une fonction/méthode qui récupère les données d'une entreprise d'après son numéro SIRET [9].

7. `https://requests.readthedocs.io/en/latest/user/quickstart/#json-response-content`
8. `https://requests.readthedocs.io/en/latest/user/quickstart/#passing-parameters-in-urls`
9. Celui-ci peut être fourni comme valeur pour `q`.

Pour ceux qui ont choisi d'encapsuler le client. Celui-ci vous permet de recourir au « patron de conception » suivant : pour chaque nouveau service, créer une sous classe de la classe `Client` qui effectue les requêtes concernées et traite les résultats. Pour l'API openstreetmap, le code sera structuré comme suit.

```
#!/usr/bin/env python3
from client import Client

class OSM_Client(Client):
    def __init__(self):
        super().__init__("nominatim.openstreetmap.org", "https")

    def get_coordinates(self, country=None, city=None, postalcode=None,
        street=None):
        res = False
        payload = {"format":"jsonv2"} ## the format of the response we
            expect : https://nominatim.org/release-docs/develop/api/
            Search/#json-with-address-details
        ### finish creating payload
        if self.get("search", payload) and self.lr_status_code() == 200:
            ## we issue a get request...
            response = self.lr_response() ## get a json object
            if len(response) > 0:
                res = {}
                ## retrieve longitude and latitude in a dictionary
        else:
            print(self.lr().url, self.lr_error())
        return res

if __name__ == "__main__":
    ### test it
    o = OSM_Client()
    print(o.get_coordinates("France", "Oullins", street="29, rue Pierre
        Semard"),o.lr().url)
    print(o.get_coordinates("France", "Lyon", street="18, rue Georges
        Gouy"),o.lr().url)
    print(o.get_coordinates("France", "Rennes", street="5, allée
        Geoffroy de Pontblanc"),o.lr().url)
```

Listing 4.1 – Patron de conception suggéré présenté sur l'API OpenStreetMap

Nota Bene 4.2: *Design Pattern*

3.2 Résolution du problème

20. Utilisez les méthodes/fonctions créées dans la section précédente pour résoudre le problème posé : il s'agit de partir d'une adresse postale et d'obtenir une liste d'entreprise avec le(s) produit(s) concernée(s), la distance par la route, ainsi que le nom et le prénom d'un gérant.

Vous pouvez proposer des optimisations (grouper des produits chez certains fournisseurs, etc.) ou proposer des mécanismes de gestion/correction de certaines erreurs.

Nota Bene 4.3: Optimisation