API REST d'accès aux données

Planche de TD pour un cours dispensé à l'université de Rennes 2

1 Rappel : organisation de votre code

Pour ce TD, vous créerez un nouveau fichier `td_api_rest.py`. Dans ce fichier, votre code sera organisé de la manière suivante :

```
# Imports
import requests
import datetime

# Fonctions
def [...]

# Tests
[...]
```

Notamment, vous définirez vos fonctions en début de fichier et les appels seront listés en fin de fichier. De cette manière, vous pourrez, d'une question à l'autre, réutiliser les fonctions déjà codées au besoin.

2 Énoncé

1. Se rendre sur le site de la STAR et trouver l'API indiquant les prochains passages de métro rennais. Cliquez sur l'onglet "API" pour accéder aux options de requête. Essayez notamment d'ajouter le *facet* `depart` et notez le format de date utilisé (un *facet* est un attribut dont on demande explicitement qu'il soit présent dans la réponse pour tous les résultats retournés).

   Petit point sur :
   UTC, Temps Universel Coordonné, (Coordinated Universal Time) est une échelle de temps adoptée comme base universelle. En France, nous avons une heure d'avance sur ce temps de référence. Pour représenter une date complète utilisable à l'international l'API

de la Star utilise le modèle suivant `"jourThoraire+timezone"` en trois parties avec pour séparateurs `"T"` et `"+"` où :

— jour = `"aaaa-mm-jj"` ,
— horaire = `"hh:mn:sc"`,
— timezone `IN {"00:00", "01:00"...}`

Ainsi :

— si timezone vaut `"01:00"` il s'agit de l'heure "en France",
— si timezone vaut `"00:00"`, il y a une heure de retard

2. Écrire une fonction qui prend en entrée une chaîne de caractères représentant une date dans le format vu à la requête précédente (exemples : `"2021-11-25T09:01:52+01:00"` ou `"2021-11-25T09:01:52+00:00"`) et retourne une date de sortie ayant les caractéristiques suivantes:

— du type date de Python (ce qui permet la réalisation de calculs, de comparaison...) ,
— privé de timezone et
— son horaire doit être celui de fuseau horaire français.

3. En consultant l'interface d'édition de requêtes de l'API de la STAR (celle que vous avez trouvé à la question 1), composez une requête qui permette :

— d'afficher les attributs `depart`, `destination` et `nomarret` pour les résultats retournés (ajout de ces attributs à la liste des *facets*)
— de ne conserver que les passages pour lesquels l'attribut `precision` vaut `Temps réel` (valeur à spécifier dans la catégorie *refine*)
— de forcer les dates à être spécifiées dans le fuseau horaire `Europe/Paris`
— de retourner les 100 prochains passages.

Notez l'URL générée (clic droit sur le lien du bas de la page, puis "Copier le lien").

Pour les questions suivantes, il est conseillé d'importer le module `pprint` qui permet d'afficher de manière claire les dictionnaires :

```
from pprint import pprint

[...]
pprint(mon_joli_dictionnaire)
```

4. Écrire une fonction qui retourne la liste de tous les passages de métro à venir. Cette fonction fera une requête API, en limitant le nombre de résultats à 100 lignes. La liste retournée par cette fonction contiendra des dictionnaires composés de 3 clés : `depart` (contenant l'heure de départ au format `datetime`), `destination` et `nomarret` et vous ne conserverez que les passages pour lesquels l'attribut `precision` vaut `Temps réel`.

Attention : pour certains passages, l'attribut `"depart"` n'existe pas : ces passages doivent donc être ignorés.

5. Écrire une fonction qui prend en entrée une liste de passages tels que ceux retournés par la question précédente et un délai `t` en minutes et qui retourne la liste des passages qui auront lieu dans un délai de `t` minutes après l'instant présent. Tester cette fonction en affichant la liste des prochains passages de métro dans les 10 minutes à venir.

3 Devoir

Cet exercice est à rendre sur CURSUS avant la séance de TD de la semaine prochaine. Le rendu se fera sous la forme d'un unique fichier Python (pas de version `.txt` ou `.pdf`) structuré comme demandé plus haut et contenant le code relatif à cette partie. Ce code devra se suffire à lui-même et devra donc contenir les éventuelles fonctions annexes appelées par ce code.

6. En utilisant le service Open Data de Rennes Métropole, écrivez une fonction qui affiche le nombre total de passages de vélos (même si le nom du jeu de données sous-entend qu'il fournit des infos sur les passages de vélos et de piétons, seuls les vélos sont comptés) devant chacun des compteurs installés dans Rennes (attribut `name`), pour le mois de novembre 2021. Notez que dans l'interface utilisée, pour filtrer une date par mois (c'est-à-dire ne conserver que les enregistrements pour le mois `MM` de l'année `YYYY`), on peut demander que l'attribut `date` soit de la forme : `YYYY/MM`, en donnant les valeurs voulues à `YYYY` et `MM`. Votre fonction devra afficher une sortie de la forme :

```
Le compteur Eco-Display Place de Bretagne a vu passer 48827 vélos en novembre 2021.
Le compteur Rennes Rue d'Isly V1 a vu passer 20703 vélos en novembre 2021.
```