TD8 : Données du réseau STAR

Le réseau de transports en commun de l'agglomération rennaise (la STAR) fournit des données relatives à ses services en libre accès via une API de données. Dans ce TD, vous effectuerez des requêtes à cette API en utilisant le module `requests`.

Préambule

- Dans votre dossier PythonOpenData créez un sous-dossier TD8.
- Lancer l'éditeur Visual Studio Code.
- Dans Visual Studio Code, ouvrir le dossier PythonOpenData/TD8 et créez un fichier td8.py dans lequel vous écrirez votre code.

Pour ce TD, 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)
```

Exercice 1 : Accès aux données

Q.1.1. Se rendre sur le site de la STAR (`https://data.explore.star.fr/explore/`) et trouver l'API indiquant les prochains passages de métro rennais.

Q.1.2. Lire les informations pour comprendre le sens des données disponibles.

Q.1.3. Cliquer sur l'onglet "API" pour accéder aux options de requête.

Q.1.4. Ajouter le *facet* `depart` et noter 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).

Q.1.5. En utilisant l'interface d'édition de requêtes de l'API de la STAR, composer 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 retourner les 100 prochains passages.

Q.1.6. Noter l'URL générée (clic droit sur le lien du bas de la page, puis "Copier le lien").

Q.1.7. En python, écrire le code permettant d'accéder aux passages de métro renvoyés par cette URL.

Q.1.8. Combien de passages de métro sont renvoyés ? Comparer le nombre d'enregistrements présents dans `records` et le nombre `nhits` annoncé en début de requête.

Exercice 2 : Etude de tous les passages

Les dates fournies par la STAR sont au format ISO, avec précision du fuseau horaire (timezone). Par exemple, `2022-11-25T09:01:52+01:00` correspond bien à l'horaire de 9h01 en France, qui est dans le fuseau horaire UTC+1 (+01 :00).

Les objets Datetime de Python peuvent intégrer la connaissance de la timezone. Ces dates ainsi créées sont dites "aware", par opposition aux dates sans timezone qui sont "naïve".

Notez qu'il n'est pas possible d'effectuer des opérations (comparaisons, différences) entre une date naïve et une date aware. Par exemple, pour comparer une date aware avec la date actuelle, il faudra créer une version aware de datetime.now(), avec :

```
from pytz import timezone
datetime.now(timezone('Europe/Paris'))
```

Q.2.1. Écrire une fonction qui retourne la liste de tous les passages de métro. Cette fonction fera une requête API, en limitant le nombre de résultats à 100 lignes. Vous ne conserverez que les passages pour lesquels l'attribut `precision` vaut `Temps réel`.

La liste retournée par cette fonction contiendra des dictionnaires composés de 4 clés :

- `depart` (contenant l'heure de départ au format `datetime`),
- `destination` contenant la destination,
- `nomarret` contenant le nom de l'arret,
- `ligne` contenant le nom court de la ligne.

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

Q.2.2. 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.

Q.2.3. Tester cette fonction en affichant la liste des prochains passages de métro dans les 10 minutes à venir.

Exercice 3 : Prochains métros dans la station

On souhaite renseigner un panneau d'affichage en entrée d'une station de métro, avec l'horaire de chaque prochain passage, pour chaque ligne de métro, dans chaque direction (voir exemple à la fin du sujet).

Q.3.1. Écrire une fonction qui prend en entrée le nom d'une station de métro, et qui renvoie la liste des passages à cette station. On prendra soin de filtrer la station dès la requête à l'API.

La liste retournée par cette fonction contiendra des dictionnaires composés de 3 clés :

- `depart` (contenant l'heure de départ au format `datetime`),
- `destination` contenant la destination,
- `ligne` contenant le nom court de la ligne.

Q.3.2. Tester cette fonction pour afficher les passages à la station Gares.

Q.3.3. Écrire une fonction qui prend en entrée une liste de passages dans une station, telle que retournée ci-dessus, et qui recherche pour chaque ligne de métro, et chaque direction l'heure du prochain passage à venir.

Les données seront retournées sous la forme d'un dictionnaire dans lequel les clés sont des tuples `(ligne, destination)` et les valeurs la date du premier prochain passage au format datetime.

Q.3.4. Écrire une fonction qui prend en entrée un nom de station, et qui utilise les fonctions précédemment codées, afin d'afficher l'horaire des prochains passages en entrée de la station de métro.
L'affichage pourra être par exemple :

```
*****************************************
Bienvenue à la station Gares
*****************************************
Ligne a, direction La Poterie : prochain métro à 16:58:39
Ligne a, direction J.F. Kennedy : prochain métro à 16:58:04
Ligne b, direction Saint-Jacques - Gaîté : prochain métro à 16:59:02
Ligne b, direction Cesson - Viasilva : prochain métro à 16:58:09
*****************************************
```

Q.3.5. Tester pour plusieurs stations de métro.