UNIVERSITÉ RENNES 2

Contrôle continu

Xavier André & Romain Tavenard

1 Préambule

Pour cet examen de contrôle continu, votre rendu se fera sous la forme d'un fichier Python, structuré comme vu en TD. Il est strictement interdit de communiquer entre vous ou avec l'extérieur. En dehors de cette limitation, vous avez accès à l'ensemble des ressources que vous souhaitez.

Pour chaque question, si vous codez une fonction, vous devrez la tester et laisser les traces de vos tests dans le fichier rendu.

2 Les passages de métros

Les questions de cette partie ne sont pas indépendantes, toutefois, elles sont évaluées de façon indépendante. Vous pouvez donc sauter une question si vous ne parvenez pas à y répondre pour passer à la suite.

Le but de cette partie sera de trouver les prochains passages de métros à une station donnée du réseau rennais (pour toutes les lignes et toutes les directions).

Pour cela, on utilisera l'API disponible à l'adresse : https ://data.explore.star.fr/explore/dataset/tco-metro-circulation-passages-tr/api/

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 ne retiendra que les passages pour lesquels l'attribut `precision` vaut `Temps réel`. La liste retournée par cette fonction contiendra des dictionnaires composés de 3 clés :

— `depart` (contenant l'heure de départ au format `datetime`, converti à l'aide de la fonction `datetime.fromisoformat`) : **si cet attribut n'est pas fourni pour un passage, le passage en question devra être ignoré**,
— `destination` contenant la destination,
— `ligne` contenant le nom court de la ligne.

Testez cette fonction pour afficher les passages à la station `Gares`. **On ne se préoccupera pas ici des problèmes éventuels de fuseau horaire.**

Vous devriez obtenir un résultat de la forme :

```
[{'depart': datetime.datetime(2022, 12, 5, 7, 43, 50, tzinfo=datetime
    .timezone.utc),
'destination': 'La Poterie',
'ligne': 'a'},
{'depart': datetime.datetime(2022, 12, 5, 7, 45, 19, tzinfo=datetime.
    timezone.utc),
'destination': 'La Poterie',
'ligne': 'a'},
{'depart': datetime.datetime(2022, 12, 5, 7, 45, 55, tzinfo=datetime.
    timezone.utc),
'destination': 'Cesson - Viasilva',
'ligne': 'b'},
{'depart': datetime.datetime(2022, 12, 5, 7, 48, 18, tzinfo=datetime.
    timezone.utc),
'destination': 'Cesson - Viasilva',
'ligne': 'b'},
{'depart': datetime.datetime(2022, 12, 5, 7, 43, 17, tzinfo=datetime.
    timezone.utc),
'destination': 'J.F. Kennedy',
'ligne': 'a'},
{'depart': datetime.datetime(2022, 12, 5, 7, 44, 45, tzinfo=datetime.
    timezone.utc),
'destination': 'J.F. Kennedy',
'ligne': 'a'},
{'depart': datetime.datetime(2022, 12, 5, 7, 43, 34, tzinfo=datetime.
    timezone.utc),
'destination': 'Saint-Jacques - Gaîté',
'ligne': 'b'},
{'depart': datetime.datetime(2022, 12, 5, 7, 45, 16, tzinfo=datetime.
    timezone.utc),
'destination': 'Saint-Jacques - Gaîté',
'ligne': 'b'}]
```

2. Écrivez 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`.

3. Écrivez 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
****************************************
```

3 La gestion de calendriers

Les questions de cette partie ne sont pas indépendantes, toutefois, elles sont évaluées de façon indépendante. Vous pouvez donc sauter une question si vous ne parvenez pas à y répondre pour passer à la suite.

Le but de cette partie sera de coder un système simple de gestion d'agendas électroniques.

1. Implémentez une classe `Calendrier` dont le constructeur initialise une liste vide de `RendezVous` nommée `liste_rdv`.
2. Ajoutez à cette classe une méthode `lire_depuis_fichier` qui prend en entrée un nom de fichier JSON, lit le contenu de ce fichier et en stocke le contenu dans `liste_rdv`. Vous devrez créer une classe pour représenter les rendez-vous et chaque rendez-vous aura une date et heure de début, une date et heure de fin (ces dates seront au format `datetime`), un titre et un lieu éventuel. Testez cette fonction avec le fichier `cal01.json` fourni sur CURSUS.
3. Faites en sorte que, si `c` est un `Calendrier`, l'appel à `print(c)` affiche dans le terminal la chaîne de caractères suivante : `"<Calendrier contenant 5 rendez-vous>"` où 5 sera remplacé par le nombre de rendez-vous du calendrier en question.
4. Surchargez, dans la classe `RendezVous`, la méthode `__eq__(self, other)` qui retourne `True` si `self` et `other` sont des rendez-vous identiques, et `False` sinon.
5. Implémentez une méthode `supprime_doublons` qui, pour le calendrier courant, supprime les rendez-vous qui sont en doublons.
6. Implémentez une méthode `union(self, other)` qui retourne un calendrier qui soit l'union des calendriers `self` et `other`, sans doublon.
7. Générez un calendrier qui soit l'union des contenus des fichiers `cal01.json` et `cal02.json` fournis sur CURSUS.