M1 BIOINFO
X1BI030

Nantes Université
2022/2023

Projet de travaux pratiques
JeMeGare

Le projet est à effectuer en binôme (ou, *exceptionnellement*, en monôme, après consultation de l'enseignant). L'ensemble du code *correctement commenté* est à déposer sur Madoc le 28 novembre 2022 à 23h55 au plus tard, sous la forme d'une archive compressée contenant les fichiers de code source. On re-précisera en commentaires dans chaque fichier du programme les noms et prénoms de chacun-e des étudiant-e-s.
Attention : en cas de doute sur la paternité du code, un entretien avec chaque membre du binôme sera programmé, au cours duquel chacun-e devra en démontrer la compréhension et la maîtrise.

On souhaite écrire une application permettant de trouver le parking de *Nantes Métropole* le plus proche de l'endroit où l'on se trouve et qui possède encore un minimum de places libres. L'application doit aussi indiquer le coût du parking en fonction du temps de stationnement prévu et de l'heure de la demande. Pour cela, on va utiliser les informations disponibles sur le site de données publiques ouvertes de Nantes Métropole.

1 Mise en œuvre

À partir d'une adresse fournie par l'utilisateur, on va déterminer les coordonnées GPS de la position grâce au fichier des adresses postales de Nantes Métropole. En les comparant avec les coordonnées des parkings de Nantes, on peut déterminer les distances respectives et choisir le parking le plus proche. Le fichier de tarification horaire comporte à la fois les coordonnées GPS des parkings et le coût suivant la durée de stationnement. Le fichier des disponibilités nous permet de déterminer le nombre de places disponibles pour chaque parking et de ne calculer les distances à l'utilisateur que pour les parkings ayant des places libres.

2 Le programme Python `address2coords.py`

On va commencer par écrire un programme Python prenant en entrée le chemin d'accès au fichier des adresses au format JSON ainsi qu'une adresse. Le programme retourne sur la sortie standard la latitude et la longitude correspondant à l'adresse :

```
# ./address2coords.py ~/.jemegare/adresses-postales-nantes-metropole.json "2 chemin de la
↪  houssinière"
-1.555334912694764 47.239366769874266
```

Le programme doit afficher un message d'information sur la sortie d'erreur standard et retourner le code d'erreur 1 s'il est appelé avec le mauvais nombre de paramètres :

```
# ./address2coords.py ~/.jemegare/adresses-postales-nantes-metropole.json

./address2coords.py: recherche des coordonnées associées à une adresse
de Nantes Métropole.
Usage: ./address2coords.py <fichier de coordonnées> <adresse>
```

Le fichier d'adresses est une liste d'entrées au format décrit dans la table 1.

Si l'adresse a été trouvée dans la base, le programme doit retourner le code d'erreur 0 ; sinon, il retourne le code d'erreur 3. La comparaison des adresses doit se faire indépendamment de la casse (majuscules/minuscules).

3 Le programme Python `availabilities.py`

On va maintenant écrire un programme Python prenant en entrées un fichier au format JSON indiquant le nombre de places disponibles pour chaque parking de *Nantes Métropole* ainsi qu'un nombre minimum de places

Table 1 – Format d'une entrée du fichier d'adresses


```
{"datasetid": "244400404_adresses-postales-nantes-metropole",
 "recordid": "42fbdf498aeb8d222905a225ea2fc4a7ed96277c",
 "fields": {"pole": 1.0,
            "adresse": "28 Rue L\u00e9on Blum",
            "nomcom": "LA-MONTAGNE",
            "gid": "90133",
            "codcommune": "440101",
            "lpole": "Sud-Ouest",
            "geo_shape": {"coordinates": [-1.689705939802552, 47.18361201627864],
                          "type": "Point"
                         },
            "commune": "La Montagne",
            "iris2008": "441010101",
            "numero": "28",
            "rivoli": "1251",
            "code_postal": "44620",
            "mot_directeur": "Blum"},
 "record_timestamp": "2022-10-25T08:01:36.381+02:00"}
```


libres et retournant sur la sortie standard la liste des identifiants (le champs « idobj » dans le format JSON décrit par la table 2) des parkings répondant au critère. Pour obtenir la liste des identifiants des parkings disposant d'au moins 300 places disponibles, on écrirait :

```
# ./availabilities.py ~/.jemegare/disponibilites-parking-nantes-metropole.json 300
1044
280
3989
5533
```

Table 2 – Format d'une entrée du fichier de disponibilités


```
{"datasetid": "244400404_parkings-publics-nantes-disponibilites",
 "recordid": "a7a24342af2c16a8388e3f0a092ddba253896c5e",
 "fields": {"grp_disponible": 345,
            "grp_nom": "Cit\u00e9 des Congr\u00e8s",
            "grp_statut": 5,
            "grp_identifiant": "009",
            "disponibilite": 345,
            "idobj": "1044",
            "grp_complet": 10,
            "grp_exploitation": 420,
            "location": [47.212901665000004, -1.5439712740000004],
            "grp_horodatage": "2022-10-28T13:03:40+02:00"},
 "geometry": {"type": "Point",
              "coordinates": [-1.5439712740000004, 47.212901665000004]},
 "record_timestamp": "2022-10-28T11:04:00+02:00"}
```


Le programme doit afficher un message d'information sur la sortie standard et retourner le code d'erreur si le nombre de paramètres est incorrect :

```
# ./availabilities.py ~/.jemegare/disponibilites-parking-nantes-metropole.json

./availabilities.py <fichier de disponibilités> <bornemin>
avec <fichier de disponibilités> le chemin vers le fichier JSON de places libres
et   <bornemin> la limite minimum de places libres pour retourner l'identifiant d'un parking
```

Le programme Python `jemegare.py`

On va maintenant écrire un programme Python prenant sur l'entrée standard une liste d'identifiants de parkings et en paramètres le fichier des tarifs au format JSON, la latitude et la longitude de l'utilisateur ainsi que la durée de stationnement prévue (en minutes). Le programme doit retourner la liste des parkings triés en ordre croissant de leur distance à l'utilisateur ainsi que le coût de stationnement pour la durée prévue :

```
# ./availabilities.py ~/.jemegare/disponibilites-parking-nantes-metropole.json 3 |
↪  ./jemegare.py ~/.jemegare/tarification-parkings-nantes-metropole.json
↪  -1.555334912694764 47.239366769874266 35
Bellamy: 1.6 €
Cathédrale: 2.2 €
Talensac: 2.2 €
Tour Bretagne: 2.2 €
[...]
Les Machines: 2.2 €
Les Fonderies: 1.5 €
```

Table 3 – Format d'une entrée du fichier de tarifs


```
{"datasetid": "244400404_parkings-publics-nantes-tarification-horaire",
 "recordid": "5b5de6abea025bb07c6dc948c9ef0acb135b5ca7",
 "fields": {"nuit_50min": 0.9,
            "nuit_30min": 0.6,
            "30min": 1.3,
            "2h": 5.0,
            "3h": 6.7,
            "nuit_20min": 0.4,
            "40min": 2.2,
            "idobj": "1043",
            "nuit_1h30": 1.6,
            "nuit_10min": 0.2,
            "nuit_3h_et": 3.0,
            "1h30": 4.2,
            "nuit_1h": 1.0,
            "50min": 2.5,
            "20min": 0.7,
            "11h": 13.2,
            "10min": 0.5,
            "nom_du_parking": "Aristide Briand",
            "nuit_40min": 0.8,
            "code_court": "ARI1",
            "location": [47.21709359800002, -1.5629364160000137],
            "2h30": 6.0,
            "nuit_2h": 2.0,
            "1h": 2.9,
            "nuit_2h30": 2.6},
 "geometry": {"type": "Point",
              "coordinates": [-1.5629364160000137, 47.21709359800002]},
 "record_timestamp": "2022-01-03T16:14:47.591+01:00"}
```


Pour calculer la distance entre deux points donnés par leurs latitudes et longitudes, on pourra utiliser le module Python `distance.py` disponible sur madoc.

Calcul du coût de stationnement

Le tarif applicable est déterminé à partir de l'heure courante au moment de l'exécution de l'application : le tarif normal est applicable de 8h à 19h ; le tarif de nuit s'applique de 19h à 8h. Le tarif facturé correspond au tarif de la plus petite durée identifiée supérieure à la durée effective.

Exemples :

- Il est 7h59 et l'utilisateur déclare un stationnement de 35 minutes : on applique le tarif de nuit pour 40 minutes (« nuit_40min ») ;
- Il est 23h34 et l'utilisateur déclare un stationnement de 155 minutes : on applique le tarif de nuit pour 3 heures et plus (« nuit_3h_et ») ;
- Il est 15h07 et l'utilisateur déclare un stationnement de 187 minutes : on applique le tarif normal pour 11h (« 11h ») ;

4 Le programme Bash `jemegare.sh`

Le programme `jemegare.sh` est le seul programme appelé directement par l'utilisateur. C'est lui qui se charge ensuite de valider les entrées et d'appeler les autres programmes.

Le programme `jemegare.sh` accepte 4 paramètres : l'adresse actuelle de l'utilisateur, la durée (en minutes) de stationnement prévue, le nombre minimum de places libres que doit encore offrir un parking pour être éligible et le nombre maximum de parkings à proposer en résultats. Pour obtenir les trois parkings les plus proches de la Faculté des Sciences et Techniques offrant au moins 4 places libres pour un stationnement de 37 minutes, on écrirait :

```
# ./jemegare.sh "2 chemin de la houssinière" 37 4 3
Bellamy: 1.6 €
Cathédrale: 2.2 €
Talensac: 2.2 €
```

Si le nombre de paramètres est incorrect, le programme doit afficher un message d'information et retourner le code d'erreur 1 :

```
# ./jemegare.sh "2 chemin de la houssinière" 37 4

./jemegare.sh <adresse> <durée> <bmin> <nombre>
avec:   <adresse> l'adresse de la position actuelle
        <durée> la durée de stationnement prévue (en minutes)
        <bmin> le nombre minimum de places libres
et      <nombre> le nombre maximum de parkings à proposer
```

On souhaite que tous les programmes et fichiers de données JSON soient stockés dans un même répertoire. Par défaut, cela doit être `~/.jemegare`. Il doit cependant être facile de modifier cela dans le code. Au démarrage, le programme doit s'assurer que ce répertoire existe. Si ce n'est pas le cas, il doit le créer. Le programme doit ensuite s'assurer de l'existence des trois fichiers de données au format JSON : le fichier d'adresses, le fichier de disponibilités et le fichier de tarifs. Si ce n'est pas le cas, il doit les télécharger (on pourra utiliser la commande `wget` pour cela).

Les fichiers de tarifs et de disponibilités sont régulièrement mis à jour. On s'assurera que le fichier de tarifs a été téléchargé il y a moins de 30 jours (on pourra utiliser la commande `find` avec l'option `-mtime`) et que le fichier de disponibilités l'a été moins d'une heure auparavant (on pourra utiliser la commande `find` avec l'option `-mmin`). Si ce n'est pas le cas, on re-téléchargera ces fichiers.

Si l'adresse donnée par l'utilisateur n'est pas trouvée, on affiche un message d'erreur sur la sortie d'erreur standard et l'on retourne le code d'erreur 6 :

```
# ./jemegare.sh "2 rue de la houssinière" 37 4 3
L'adresse 2 rue de la houssinière n'a pas été trouvée.
```

Annexe : accès aux champs d'une entrée JSON

Tous les fichiers JSON téléchargés sont constitués d'une seule liste d'entrées de type *dictionnaire*. Exemple avec le fichier `structure-json.json` :


```
[{"datasetid": "244400404_parkings-publics-nantes-disponibilites",
  "recordid": "0949949178d9b5685b25bd2247f003ff571a07f2",
  "fields": {"grp_disponible": 59,
             "grp_nom": "Feydeau",
             "grp_statut": 5,
             "grp_identifiant": "001",
             "disponibilite": 59,
             "idobj": "4320",
             "grp_complet": 10,
             "grp_exploitation": 420,
             "location": [47.21407529499999, -1.552558781000016],
             "grp_horodatage": "2022-10-28T16:38:41+02:00"},
  "geometry": {"type": "Point",
               "coordinates": [-1.552558781000016, 47.21407529499999]},
  "record_timestamp": "2022-10-28T14:39:00+02:00"},
 {"datasetid": "244400404_parkings-publics-nantes-disponibilites",
  "recordid": "f11f4fb495ac0b012bd3003a45e42591a190a61a",
  "fields": {"grp_disponible": 296,
             "grp_nom": "Tour Bretagne",
             "grp_statut": 5,
             "grp_identifiant": "003",
             "disponibilite": 296,
```

```
            "idobj": "299",
            "grp_complet": 10,
            "grp_exploitation": 643,
            "location": [47.21784288800001, -1.5582500169999776],
            "grp_horodatage": "2022-10-28T16:38:41+02:00"},
  "geometry": {"type": "Point",
               "coordinates": [-1.5582500169999776, 47.21784288800001]},
  "record_timestamp": "2022-10-28T14:39:00+02:00"}]
```


Il est possible d'itérer sur un tel fichier en lisant chaque entrée comme un dictionnaire, certaines entrées de ce dictionnaire pouvant elles-mêmes être des dictionnaires.

Exemple : avec le programme Python suivant :

```python
#!/usr/bin/env python3
import sys
import json

if len(sys.argv) < 2:
    print("Donnez le nom du fichier JSON en paramètre",file=sys.stderr)
    exit(1)

try:
    with open(sys.argv[1]) as jsonfile:
        data = json.load(jsonfile)
except IOError:
    print(f"{sys.argv[0]}: Impossible d'ouvrir le fichier {sys.argv[1]}")
    exit(1)

for entry in data:
    print(entry['recordid'])
    print(entry['fields']['grp_nom'])
    print(entry['geometry']['coordinates'][1])
```

on obtient le résultat :

```
#./exemple_json.py structure-json.json
0949949178d9b5685b25bd2247f003ff571a07f2
Feydeau
47.21407529499999
f11f4fb495ac0b012bd3003a45e42591a190a61a
Tour Bretagne
47.21784288800001
```