Aide-mémoire pour les modules `tweepy` et `googlemaps`

Romain Tavenard

Dans la suite, vous trouverez des éléments pour vous aider à utiliser les modules Python `tweepy` et `googlemaps` (v. 3.0.2). Pour installer ces modules sur un ordinateur de l'Université (ou sur votre ordinateur si vous disposez d'Anaconda), ouvrez `Anaconda Prompt` et entrez les commandes suivantes dans le terminal :

```
pip install --user tweepy
pip install --user googlemaps==3.0.2
```

Ce document n'est absolument pas exhaustif, mais il a vocation à centraliser des éléments utiles en pratique et parfois insuffisamment documentés.

1 Avant-propos : les clés d'API

Pour utiliser ces modules, la première chose à faire sera de préciser votre clé d'API. Attention, il ne faut jamais écrire dans vos scripts Python ces clés "en dur", mais plutôt les lire dans un fichier annexe. Dans le cadre du projet `Twitter`, par exemple, cela vous permettra de rendre votre code Python sans avoir à divulguer au correcteur vos clés d'API (qui doivent rester secrètes, au même titre qu'un mot de passe). Dans les exemples qui suivent, on supposera donc que ces clés d'API sont stockées dans des variables (dont le contenu a été défini en lisant un fichier de clés, typiquement un fichier JSON).

2 Tweepy

2.1 Obtenir des identifiants pour l'API Twitter

Pour travailler avec l'API Twitter, vous devrez posséder un compte Twitter, puis, étant connecté à ce compte, visiter la page https://apps.twitter.com et y créer une "Application". Une aide détaillée de cette procédure de création d'application est disponible ici. Une fois cette application créée, vous devrez, dans l'onglet "Keys and Access tokens", créer :

— une clé d'API (aussi appelée "Consumer Key") et sa clé secrète ;
— un "Access Token" et son pendant secret.

Une fois ces identifiants créés, vous pourrez vous identifier dans `tweepy` avec les commandes suivantes :

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(key, secret)
acces_api = tweepy.API(auth)
```

2.2 Exemple d'utilisation

Les principales actions que vous pourrez effectuer depuis `tweepy` seront :

— Obtenir la `timeline` (liste de tweets) pour un utilisateur à partir de son identifiant [lien] ;
  — retourne une liste d'objets de type `Status` (voir plus bas)
— Poster un tweet [lien] ;
  — retourne un objet `Status` représentant le tweet envoyé
— Supprimer un tweet à partir de son identifiant [lien]

Voici un exemple d'utilisation de ces trois fonctions (pour plus de détails sur les arguments facultatifs que l'on peut leur fournir, vous pourrez vous référer au lien fourni pour chaque fonction) :

```
# Liste des tweets les plus récents du compte officiel de l'UR2
list_tweets = access_api.user_timeline("univrennes_2")

# Poste un tweet dont le texte est 'Coucou' depuis le compte identifié
tweet = access_api.update_status("Coucou")

# Supprime un tweet à partir de son identifiant (doit être un tweet du compte identifié !)
tweet_id = ...
access_api.destroy_status(tweet_id)
```

Un objet de type `Status` représente un tweet et il possède un nombre important d'attributs (et un niveau de base en anglais devrait suffire à comprendre le sens d'un certain nombre d'entre eux) :

— `author`
— `contributors`
— `coordinates`
— `created_at`
— `destroy`
— `entities`
— `favorite`
— `favorite_count`
— `favorited`

— `geo`
— `id`
— `id_str`
— `in_reply_to_screen_name`
— `in_reply_to_status_id`
— `in_reply_to_status_id_str`
— `in_reply_to_user_id`
— `in_reply_to_user_id_str`
— `is_quote_status`
— `lang`
— `parse`
— `parse_list`
— `place`
— `retweet`
— `retweet_count`
— `retweeted`
— `retweets`
— `source`
— `source_url`
— `text`
— `truncated`
— `user`

Accéder à ces attributs se fait simplement : par exemple, pour ce qui est de l'attribut `created_at` d'une variable `s` de type `Status`, on écrira :

```
s.created_at
```

De la même manière, il existe un type de données spécifique pour décrire un auteur de tweet (ou un utilisateur Twitter de manière générale), et ce type fournit les attributs suivants :

— `contributors_enabled`
— `created_at`
— `default_profile`
— `default_profile_image`
— `description`
— `entities`
— `favourites_count`
— `follow_request_sent`
— `followers`
— `followers_count`
— `followers_ids`
— `following`
— `friends_count`
— `geo_enabled`
— `has_extended_profile`
— `id`

- `id_str`
- `is_translation_enabled`
- `is_translator`
- `lang`
- `listed_count`
- `location`
- `name`
- `notifications`
- `profile_background_color`
- `profile_background_image_url`
- `profile_background_image_url_https`
- `profile_background_tile`
- `profile_banner_url`
- `profile_image_url`
- `profile_image_url_https`
- `profile_link_color`
- `profile_sidebar_border_color`
- `profile_sidebar_fill_color`
- `profile_text_color`
- `profile_use_background_image`
- `protected`
- `screen_name`
- `statuses_count`
- `time_zone`
- `timeline`
- `translator_type`
- `url`
- `utc_offset`
- `verified`

Pour un utilisateur stocké dans une variable `u`, on peut accéder aux identifiants de ses *followers* à l'aide de la fonction `followers_ids` :

```
l = u.followers_ids()
```

Pour plus de manipulations sur les utilisateurs Twitter ou les Tweets, référez vous à l'aide en ligne Tweepy.

3 Le module `googlemaps`

Ce module, dont la documentation est accessible à l'adresse https://googlemaps.github.io/google-maps-services-python/docs/ permet d'accéder à plusieurs API Google Maps. Les principales fonctionnalités qu'il offre sont le *geocoding* (récupérer une adresse postale à partir d'un couple longitude/latitude et *vice versa*) et le calcul d'itinéraire.

3.1 Obtenir une clé d'API Google Maps

Pour obtenir une clé d'API Google Maps, vous devrez avoir un compte GMail puis, étant connecté à ce compte, accéder à la page de gestion de vos clés d'API. Sur cette page, dans l'onglet "Identifiants", vous pourrez créer une nouvelle clé d'API.

Une fois la clé créée, vous pourrez vous identifier dans le module `googlemaps` avec la commande suivante :

```
import googlemaps

[...]

acces_api = googlemaps.Client(api_key)
```

3.2 Exemple de fonctions de l'API Google Maps

La documentation du module `googlemaps` liste de manière exhaustive les fonctions que fournit cette API. Dans cette partie, nous allons nous concentrer sur 3 de ces fonctions. Dans tous les cas, vous devrez porter un soin particulier à l'analyse de la structure de données retournée par ces fonctions.

3.2.1 La fonction `directions`

L'un des services les plus important est celui permettant de calculer un itinéraire à partir d'un point d'origine et d'une destination. Cette fonction s'utilise comme suit :

```
directions = acces_api.directions(address, destination)
```

où `address` est l'origine du trajet et `destination` le point d'arrivée. Les paramètres `address` et `destination` peuvent être soit des chaînes de caractères (ex : `"Rennes"`), soit des dictionnaires fournissant la longitude et la latitude du point à considérer (ex : `{"lng": -1.764416, "lat": 48.137123}`). De plus, il est possible de fournir des paramètres facultatifs à cette fonction, comme documenté ici.

3.2.2 La fonction `elevation`

Il est également possible de récupérer l'altitude d'un point (ou d'une série de points) de données à l'aide de la fonction `elevation` qui s'utilise comme suit :

```
altitudes = acces_api.directions(locations)
```

où `locations` est soit une liste de points (au même format que les paramètres `address` ou `destination` de la section précédente) ou bien un seul point de données.

3.2.3 La fonction `geocode`

Il peut également s'avérer utile de récupérer, à partir d'une description textuelle d'un lieu, ses coordonnées GPS. C'est ce que propose la fonction `geocode`qui s'utilise comme suit :

```
coord_gps = acces_api.geocode(address)
```

où `address` est une chaîne de caractères représentant un lieu dont on souhaite connaître les coordonnées GPS. De plus, il est possible de fournir des paramètres facultatifs à cette fonction, comme documenté ici.