

## Informatique 2 : Projet Sherlock

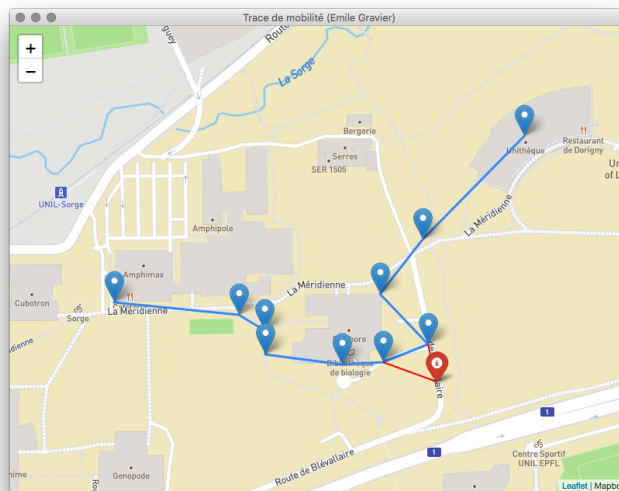
*Instructions pour la réalisation du projet "Collection et analyse de traces de mobilité dans le cadre d'une investigation".*

**Lire attentivement les énoncés des exercices, et tester chaque étapes. Penser à utiliser des copies (superficielles ou profondes) lors du retour ou du passage par références.**

**Pour faciliter la correction du projet et des tests, respecter les noms et signatures (nombre et ordre des paramètres) des fonctions et des méthodes !**

Le but général du projet est d'identifier, à partir de leurs traces de mobilité, les suspect-e-s plausibles pour un crime ayant eu lieu à un endroit donné, à une date et heure données. La méthodologie générale est la suivante<sup>1</sup> :

- Récupérer les différentes informations fournies au programme par l'enquêteur. Ces données prennent la forme d'arguments de la ligne de commande, de fichiers de configuration et de fichiers de données.
- Lister les suspect-e-s et, pour chacun d'entre eux, construire sa trace de mobilité (des informations sous forme date et heure/localisation) à partir des sources de données spécifiées. Les sources sont des photographies géo-tagguées, des tweets géo-taggués, des logs de connexion Wi-Fi, et/ou des logs de téléphone portables.
- Extraire de ces traces de mobilité les localisations de chaque suspect juste avant et juste après le crime (la dernière information de localisation antérieure au crime et la première postérieure au crime).
- Déterminer, à l'aide des données de cartographies de Google Map, si le suspect a pu se rendre sur le lieu du crime depuis sa localisation juste avant le crime et du lieu du crime à sa localisation juste après le crime durant l'intervalle de temps correspondant.



Dans ce document, on décrit, pas à pas, toutes les étapes pour parvenir à ce résultat. Le document est structuré en sections qui décrivent le travail à effectuer pour chaque module. Le numéro et le titre de la séance du cours où les notions nécessaires à l'implémentation du module correspondant sont présentées est indiqué en en-tête.

**⚠ Attention** : Les modules ne sont pas à traiter dans l'ordre ; le module VI peut d'ores et déjà être implémenté, mais pas le module V<sup>2</sup>. Dans un premier temps, commencer par intégrer les morceaux du projet qui ont déjà été effectués en séance d'exercices. Certaines sections contiennent des diagrammes UML donnant de **nombreuses informations** pour guider l'implémentation (**respecter les noms des attributs et**

1. On notera que la méthodologie n'est en rien rigoureuse. Le but du projet est d'appliquer les concepts vus en cours à un exemple concret proche de la thématique de la formation.

2. Lire en entier l'énoncé afin d'identifier les différentes parties du programmes et l'ordre dans lequel vous allez implémenter les différents modules.

**des méthodes**). Ce document est accompagné d'un squelette du programme contenant des commentaires et des **"TODO"** indiquant le code à produire à différents endroits du programme. Les mentions **"🔧 Urgent"** indiquent les instructions minimums qui sont nécessaires pour avoir du code fonctionnel.

Chaque module contient un bloc `main` contenant une série d'instructions de tests. Ces tests doivent fonctionner une fois que le module est correctement complété. Il est indiqué, sous chaque bloc `main`, ce que le programme devrait afficher lorsque le bloc `main` est exécuté. Dé-commenter les instructions une fois que la méthode correspondante a été implémentée.

Pour tester le programme, un mini-jeu de données est fourni (version 2018). Il permet de valider la grande majorité des fonctions à implémenter. Le jeu de données réel sera fourni plus tard dans le semestre. Il convient également de tester vos fonctions pour gérer les erreurs posées par les cas problématiques : valeurs ou types incorrectes, cas particuliers, et exceptions en générales. Pour les exceptions, générer un message sur la sortie erreur si l'utilisateur·trice a spécifié l'argument *verbose* dans la ligne de commande.

**Avant de commencer**, installer les dépendances suivantes avec pip (PyCharm → Preferences → Project → Project interpreter → +) : *exifRead*, *folium*, *googlemaps*, *PyQt5*, *tweepy*. (Ces dépendances sont déjà installées sur les machines des salles de TPs). Pour ceux qui désirent travailler sur leur propre machine, utiliser le fichier *requirement.txt* afin d'installer les dépendances.

## I 🔧 *Le module outil : utils.* Cours 1 : Manipulation avancée des objets de base

Le module *utils* contient des fonctions utiles pour le reste du programme. Deux fonctions `get_if_exists` et `convert_to_degrees` sont utilisées pour l'extraction des données EXIF (voir Section VII). Elles sont fournies (implémentées). La fonction `dict.factory` est utilisée pour le traitement des résultats des requêtes SQL exécutées pour l'extraction des traces de mobilité à partir de logs Wi-Fi (voir Section IX). Elle est fournie. Pour afficher les objets composites (voir Section VI), on a besoin d'une fonction `indent` pour décaler (indenter) le texte automatiquement en y ajoutant une chaîne de caractères en début de ligne.

### 📖 Instructions :

1. Implémenter une fonction `indent` qui prend en paramètre un texte, sous forme de chaîne de caractères, et un espacement (optionnel, qui vaut `"\t"` par défaut), sous forme de chaîne de caractères, et qui ajoute le caractère d'espacement au début de chaque ligne du texte (TP1 exercice 11). Par exemple, `print("zero\n" + indent("one\n" + indent("two\nthree", "\t")))`  affiche :

```
zero
    one
        -two
        -three
```

💡 **Indice :** (1) découper la chaîne de caractères passée en paramètre au niveau des passages à la ligne (`split`), (2) ajouter la chaîne d'espacement au début de chaque chaîne de caractères de la liste ainsi obtenue (`map` ou une liste en compréhension) et enfin (3) assembler les différentes chaînes de la liste en intercalant un passage à la ligne entre elles (`join`).

## II ▶ *Le module principal : sherlock.* Cours 2 : Programmes avancés

Le module principal est contenu dans le fichier *sherlock.py* qui doit être exécuté pour mener à bien l'investigation complète. À la fin du projet, c'est ce fichier qu'il faudra exécuter pour lancer le programme d'investigation. L'enquêteur·trice peut, ou doit, passer un certain nombre d'informations au programme via des arguments de ligne de commande, à savoir :

- Un argument permettant de spécifier si les messages d'avertissement en cas de problèmes doivent être affichés. Il s'agit d'un argument optionnel booléen `-v/-verbose`, qui vaut `False` par défaut (voir cours et TP) ;
- Un argument **obligatoire** spécifiant le nom du fichier (*XML/JSON*) décrivant les suspect·e·s (voir Section IV) ;
- Des arguments **obligatoires** spécifiant les clés pour les *APIs Twitter* et *Google* (voir Sections VIII et XI) ;
- Des arguments **obligatoires** spécifiant les coordonnées GPS (latitude et longitude, de type `float`) du lieu du crime ;
- Un argument **obligatoire** spécifiant, sous forme textuelle, la date et heure du crime. Le format de la date devra être : `JJ/MM/AAAA-hh:mm`, par exemple `01/04/2019 17:30:20`.

### 📖 Instructions :

1. Implémenter, à l'aide du module `argparse`, la récupération des informations à partir de la ligne de commande. À titre d'indication, le message d'aide du module `sherlock` (obtenu en exécutant la commande `"sherlock.py -h"`) devrait être le suivant<sup>3</sup> :

```
usage: sherlock.py [-h] [-v] -s SUSPECT -t TWITTER_API_KEY
                  -u TWITTER_API_KEY_SECRET -g GOOGLE_API_KEY -lat LATITUDE
                  -lng LONGITUDE -d DATE
```

Identifie les suspect.e.s les plus plausibles à partir de leurs traces de mobilité (issues de sources multiples incluant les tweets géo-taggués, les traces Wi-Fi et les flux de photos géo-tagguées) pour un crime spécifié par une date/heure et une localisation

optional arguments:

```
-h, --help            show this help message and exit
-v, --verbose         affiche les détails de l'exécution du programme et les
                      avertissements
-s SUSPECT, --suspect SUSPECT
                      fichier contenant la liste des suspect.e.s et les sources
                      de données de localisation (XML ou JSON)
-t TWITTER_API_KEY, --twitter-api-key TWITTER_API_KEY
                      clé pour l'accès à l'API Twitter (clé privée de l'
                      application Twitter)
-u TWITTER_API_KEY_SECRET, --twitter-api-key-secret TWITTER_API_KEY_SECRET
                      clé secrète pour l'accès à l'API Twitter
-g GOOGLE_API_KEY, --google-api-key GOOGLE_API_KEY
                      clé pour l'accès à l'API Google (clé privée du compte
                      développeur Google)
-lat LATITUDE, --latitude LATITUDE
                      latitude de la scène du crime
-lng LONGITUDE, --longitude LONGITUDE
                      longitude de la scène du crime
-d DATE, --date DATE  date et heure du crime (au format JJ/MM/AAAA hh:mm, par
                      exemple 01/04/2019 17:30:20)
```

**Indice** : Penser à spécifier les types des arguments qui ne sont pas des chaînes de caractères de sorte qu'`argparse` fasse les vérifications et les conversions nécessaires. Réutiliser le code implémenter pendant les séances de TP (TP2).

2. Ecrire la ligne de commande complète permettant d'exécuter le module principal `sherlock` :
  - Clés Twitter API : `Z4bLkruoqSp0JXJfJGTaMQEZO` et `gYyLCa7QiDje76VaTtlylDjGThCBGcp9MIcEglzVq6FJcXIde`
  - Clé Google API : `AIzaSyBsgJp_3ElinD9-T5r2Fbcg0AABR7caito`
  - Crime : (46.520378, 46.520378) le 16 mai 2018 à 9h19 et 23 secondes
  - Déterminer les autres paramètres.

💡 **Indice** : la ligne de commande se présente sous la forme suivante : `"sherlock.py -v -s suspect.xml..."`

Les questions suivantes pourront seulement être traitées plus tard dans l'avancement du projet.

3. Ajouter le paramètre `verbose` (obtenu depuis la ligne de commande) à l'objet de configuration (voir Section III).
4. Définir les clés pour les API Twitter et Google (obtenues depuis la ligne de commande) dans leur modules respectifs (voir Section VIII et XI)
5. Convertir la date du crime (obtenue depuis la ligne de commande) en un objet `datetime`.
 

💡 **Indice** : utiliser la méthode `datetime.strptime` pour créer un objet `datetime`.
6. Créer un objet `Location` décrivant le lieu du crime à partir de ses coordonnées (obtenues depuis la ligne de commande) (voir Section IV)
7. Afficher un message de la forme suivante pour signaler le début de l'analyse. Le nom du lieu du crime aura été obtenu automatiquement à l'aide de la méthode adéquate qui repose sur l'API de *Google Maps* (voir Section XI). Seule 5 décimales seront affichées pour les coordonnées.
 

["Investigation liée au crime du 16/05/2018 à 09:19:23 @ Route Cantonale 16, 1024 Ecublens, Suisse \(46.52038,6.57825\)"](#)
8. Créer la liste des objets `Suspect` à l'aide de la méthode adéquate (voir Section IV) à qui on fournira le nom du fichier de configuration `XML/JSON` (obtenu depuis la ligne de commande).

---

3. Pour faciliter la correction, utiliser les mêmes noms pour les différents arguments.


- Pour chaque suspect, déterminer, à l'aide de la méthode adéquate qui repose sur l'API de *Google Map* (voir Section XI), si le suspect a pu se rendre sur le lieu du crime depuis sa localisation juste avant le crime et du lieu du crime à sa localisation juste après le crime durant l'intervalle de temps correspondant. **On considèrera que le temps minimal réel (à pied) est la moitié du temps renvoyé par l'API de Google.** Afficher la liste des suspect-e-s plausibles.
- Intercepter toutes les exceptions possibles et afficher un message de la forme suivante (les points de suspension dénotent le message spécifique de l'erreur). En cas d'exception, interrompre l'exécution du programme et afficher, dans tous les cas un message indiquant la fin de l'exécution "Terminé".  
"[Erreur] L'erreur suivante est survenue durant l'exécution du programme : ..."

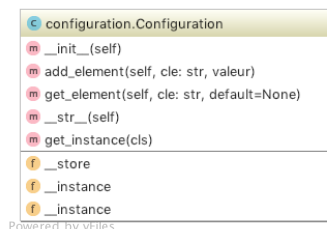
### III Le module de configuration : **configuration**. Cours 6 : UML, Patrons de conception

Le module configuration permet de manipuler les paramètres de configuration globale du programme (p. ex. verbose). On décrit ces paramètres sous la forme de couples clé-valeur (p. ex. "verbose"—True).

#### Instructions :

- Créer une classe **Configuration** contenant : une structure de données adéquate (initialement vide) permettant de stocker les couples clé-valeur, une méthode **add\_element** permettant d'ajouter un élément de configuration, une méthode **get\_element** permettant de récupérer la valeur d'un paramètre à partir de la clé correspondante.  
**Exemple** : Si on effectue **add\_element('N', 4)** avec la structure de données contenant **"verbose": False**, on obtient **"verbose": False, "N": 4**.
- Optionnel** : Ajouter un argument optionnel à **get\_element** spécifiant la valeur de retour (judicieuse) si la clé spécifiée n'est pas présente dans la configuration.  
**Exemple** : En appelant **get\_element('M', 7)** sur la structure de données précédente, on doit obtenir 7.
- Implémenter le patron *Singleton* pour cette classe pour permettre de manipuler la configuration globale de manière cohérente et facilement partout dans le programme.
- Tester.

 **Indice** : réutiliser la correction du TP6, Exercice 5).




### IV Le module **Location**, **LocationSample** et **LocationProvider** : **location**.

#### Cours 5 : Programmation orientée objet

Le module **location** permet de manipuler des données de localisation et définit la classe abstraite **LocationProvider**. La classe **Location** décrit des objets contenant une latitude et une longitude. La classe **LocationSample** décrit des objets contenant un **timestamp** (un objet **datetime** désignant une date et un temps) et un objet **Location**. D'autre part, elle redéfinit les opérateurs de comparaison pour refléter l'ordre chronologique (**ls1 <= ls2** si **ls1** est antérieur à **ls2**) de telle sorte qu'il est possible d'utiliser **sort**, **min**, **max**, **<**, **>**, etc. Les diagrammes UML ci-dessous indiquent la structure générale de ces classes ainsi que les noms des méthodes à implémenter.

#### Instructions :

-  **Urgent** : Définir la classe **Location** et son constructeur. Vérifier les valeurs passées en paramètre et lever une exception en cas de problème (faire ceci pour toutes les méthodes). Réutiliser le code implémenté pendant les séances de TPs (TP3).
- Implémenter les *getters*.
- Redéfinir la méthode **\_\_str\_\_** de sorte à afficher une localisation **exactement** sous la forme suivante et limiter le nombre de décimale à 5).  
**Location [ latitude: 48.85479, longitude: 2.34756]**

4. Définir (sans les implémenter pour l'instant, voir Section XI) les méthodes `get_name` (qui renvoie la description textuelle du lieu correspondant à un objet `Location`) et `get_travel_distance_and_time` (qui renvoie le couple distance et temps de parcours de la localisation pour atteindre le lieu correspondant à un autre objet `Location`). Ignorer (pour l'instant) tous les attributs et méthodes contenant le mot "api".
5. **Urgent** : Définir la classe `LocationSample`. Vérifier les valeurs passées en paramètre et lever une exception en cas de problème (faire ceci pour toutes les méthodes).
6. Implémenter les getters.
7. Redéfinir la méthode `__str__` de sorte à afficher un objet `LocationSample` sous la forme suivante.  
`LocationSample [datetime: 2019-03-03 12:25:00, location: Location [ latitude: 48.85479, longitude: 2.34756]]`
8. **Urgent** : Redéfinir les opérateurs de comparaison.
9. **Optionnel** : La méthode `get_description` renvoie une représentation HTML de l'objet `LocationSample` (timestamps, position, image ou tweet si disponible, etc).



La classe (abstraite) `LocationProvider` décrit des objets permettant de produire une liste d'objets `LocationSample`. Elle spécifie l'existence d'une méthode `get_location_samples` qui renvoie une liste d'objets `LocationSample` triés par ordre chronologique (qui n'est **pas** implémentée, c'est aux classes filles de l'implémenter). Le diagramme UML ci-dessous décrit la classe. La classe **fournit** la fonction `show_location_samples` permettant d'afficher le résultat de `get_location_samples` sur une carte.

#### **Instructions :**

1. Définir la classe abstraite `LocationProvider` et son constructeur (abstrait).
2. Spécifier l'existence de `get_location_samples` (méthode abstraite).
3. Implémenter la méthode `print_location_samples` en utilisant `get_location_samples`. Cette méthode doit retourner une chaîne de caractère décrivant, sous la forme suivante, les objets `LocationSamples` renvoyés par la méthode `get_location_samples`.  
`['LocationSample [timestamp: 2017-03-03 12:25:00, location: Location [latitude: 48.85479, longitude: 2.34756]]', 'LocationSample [timestamp: 2017-03-03 14:56:05, location: Location [latitude: 46.51774, longitude: 6.63223]]']`
4. **Urgent** : Implémenter la méthode `get_surrounding_temporal_location_samples` qui renvoie les objets `LocationSample` de la trace (obtenue via `get_location_samples`) juste avant et juste après le `timestamp` passé en paramètre (la dernière information de localisation antérieure au `timestamp` et la première postérieure au `timestamp`).

💡 **Indice** : On pourra créer la liste des objets `LocationSample` antérieurs au `datetime` passé en paramètre (utiliser `filter` ou une liste en compréhension) et en extraire le `LocationSample` le plus récent. De même, on pourra créer la liste des objets `LocationSample` postérieurs au `datetime` et en extraire le `LocationSample` le plus ancien. Une autre solution consiste à itérer sur la liste des objets `LocationSample` tant que les objets `LocationSample` rencontrés sont antérieurs au `datetime` passé en paramètre. Lorsqu'on rencontre le premier `LocationSample` postérieur au `datetime`, on peut identifier les `LocationSample` juste avant et juste après le `datetime`.

⚠ **Attention** :

- Pour que cette dernière méthode fonctionne correctement, la liste renvoyée par la méthode `get_location_samples` doit être triée par ordre chronologique.
- Gérer le cas où il n'y a pas de `LocationSample` avant ou après (une des valeurs vaut `None` dans ce cas).

**Exemple** : Soit

```
sample1=LocationSample(datetime(2017,3,3,12,25),Location(46.517767,6.63211));
sample2=LocationSample(datetime(2017,3,3,14,56,5),Location(46.517738,6.632233));
sample3=LocationSample(datetime(2017,3,3,16,3),Location(46.517742,6.63222))}
```

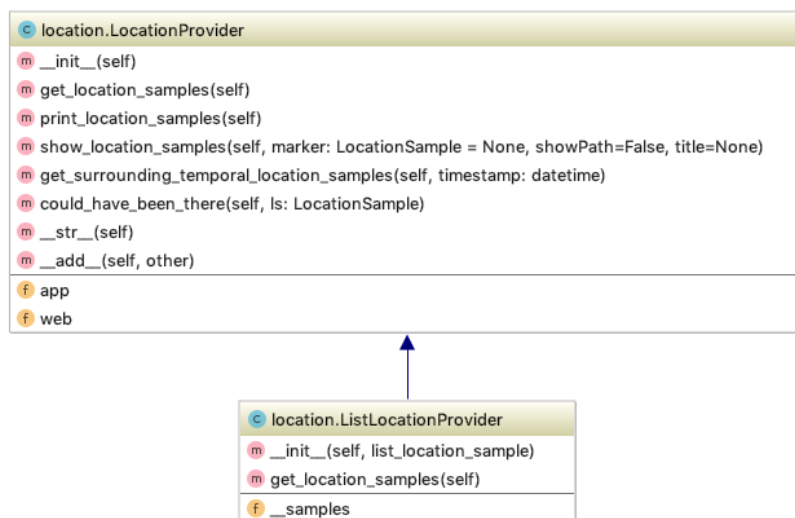
La méthode `get_surrounding_temporal_location_sample(datetime(2017, 3, 3, 13, 45))` doit renvoyer (`sample1`, `sample2`).

5. Définir la méthode `could_have_been_there` qui prend un `LocationSample` en paramètre et à l'aide de la méthode `get_surrounding_temporal_location_samples` détermine si un suspect à pu se rendre sur la scène du crime.

💡 **Indice** : Comparer le temps nécessaire pour rejoindre le `LocationSample` et le temps réellement écoulé.

6. Redéfinir la méthode `__str__` de sorte à afficher un objet `LocationProvider` sous la forme suivante.

`LocationProvider (5 location samples)`



7. Implémenter la classe `ListLocationProvider`, héritant de `LocationProvider`, qui contient (dans un attribut) une liste d'objets `LocationSample` triés et dont la méthode `get_location_samples` renvoie cette liste (utiliser `copy` pour éviter les modifications).

8. Vous pouvez utiliser la classe `ListLocationProvider` pour tester la classe abstraite `LocationProvider`. Tester l'affichage d'une liste de `LocationSample` avec la méthode `show_location_samples`.

## V 🧑‍🤝🧑 **Le module suspects : suspect.** Cours 8 : Formats de fichiers

La classe `Suspect` décrit des objets qui contiennent les informations décrites dans le fichier `XML/JSON`, à savoir un nom de suspect et des sources de `LocationProvider`. On donne un exemple de fichier `XML` ci-dessous.

### 📖 **Instructions** :

1. Définir la classe `Suspect` et son constructeur.
2. Implémenter les getters.

3. **Optionnel** : Redéfinir la méthode `__str__` de sorte à afficher un objet `Suspect` sous la forme suivante.  
[Suspect] Name: jdoe, Location provider: PictureLocationProvider (source: `'../data/pics/jdoe'` (JPG, JPEG, jpg, jpeg), 2 location samples)
4. Implémenter une méthode `create_suspects_from_XML_file` qui prend en paramètre un nom de fichier et qui parse le fichier `XML` et créé la liste de suspect-e-s correspondante. De quel type de méthode s'agit-il (instance / classe / statique)?
  - (a) Extraire l'arbre `XML` du fichier et le parcourir à l'aide du module `xml.etree.ElementTree`.
  - (b) Pour chaque élément suspect dans l'arbre `XML`, créer un objet `Suspect`.
  - (c) Pour chaque élément source dans l'arbre `XML`, créer l'objet `LocationProvider` correspondant (en fonction du type de `LocationProvider`; élément "type"). Tous les paramètres nécessaires peuvent être trouvés dans les sous-éléments de l'élément source.

**⚠ Attention** : les noms de fichiers contenus dans le fichier `XML` sont exprimés relativement au fichier `XML` lui-même. C'est à dire que si le fichier `"data/suspects.xml"` contient un élément `dir` : `"../data/pics/jdoe"`, le chemin réel du répertoire est `"data/../data/pics/jdoe"` (= `"data/pics/jdoe"`). Gérer cette situation en utilisant les fonctions `dirname` et `join` du module `os.path`.

**Exemple :**

- `os.path.dirname(filename)` renvoie le chemin du *directory* correspondant au fichier *filename*.  
`os.path.dirname("data/suspects.xml")` retourne `"data"`.
- `os.path.join(dirname, path)` renvoie la concaténation des deux chemins. `os.path.join("data", "../data/pics/jdoe")` retourne `"data/../data/pics/jdoe"`.
- `os.path.normpath(path)` renvoie le chemin *path* sans les séparateurs inutiles (`"data/../data/"` devient `'data'`). `os.path.normpath("data/../data/pics/jdoe")` retourne `"data/pics/jdoe"`.

**Alternative** : implémenter une méthode similaire `create_suspects_from_JSON_file` opérant sur un fichier `JSON`.

- (e) Gérer les exceptions : si une exception survient dans la création d'un suspect (ou d'une source qui lui est associée), ignorer l'erreur et continuer. Afficher simplement un message (sur la sortie d'erreur) de la forme (**uniquement si l'utilisateur·trice a spécifié l'argument `--verbose` dans la ligne de commande ; utiliser l'objet configuration pour ceci, voir Section III**) :

```
Warning: An exception has occurred while processing the following source
([Errno 2] No such file or directory: '../data/../data/pics/jdoex'):
<source>
  <type>Photographs</type>
  <dir>../data/pics/jdoex</dir>
</source>
```



Exemple de fichier `XML` :

```
<?xml version="1.0"?>
<suspects>
  <suspect>
    <name>Patrick Biales</name>
    <sources>
      <source>
        <type>Twitter</type>
        <username>PatrickBiales94</username>
        <token>915546853613211653-4sqbB0Ur7rVda76dMd9jUF2zU2WlsS9</token>
        <token_secret>yLZ3fy2RIHILcAai8vLnHe0IENJUBa6AYNrDGLPfVCvo7</token_secret>
```



```

</source>
<source>
  <type>Photographs</type>
  <dir>../data/pics/biales</dir>
</source>
<source>
  <type>Wi-Fi</type>
  <db>../data/db/wifi.db</db>
  <username>pbiales</username>
</source>
<source>
  <type>Logs</type>
  <file>../data/logs/biales.log</file>
</source>
</sources>
</suspect>
</suspects>

```

**Note :** pour tester plus rapidement les autres parties du code, créer un suspect manuellement plutôt que d'implémenter la méthode de création à partir d'un fichier *XML/JSON*, par exemple :

```

lp1 = PictureLocationProvider( '../data/pics/gravier' )
lp2 = WifiLogsLocationProvider( '../data/db/wifi.db', 'egravier' )
lp = CompositeLocationProvider( lp1, lp2 )
gravier = Suspect( "Emile Gravier", lp )

```

gravier.get\_name() doit retourner "Emile Gravier".

gravier.get\_location\_provider () doit retourner :

```

CompositeLocationProvider (10 location samples)
+   PictureLocationProvider (source: '../data/pics/gravier' (JPG,JPEG,jpg,jpeg), 6
    location samples)
+   WifiLogsLocationProvider (source: '../data/db/wifi.db', user 'egravier', 4
    location samples)

```

## VI Le module de sources composites : composite. Cours 6 : Patrons de conception

Compléter le module *location* en ajoutant la classe **CompositeLocationProvider**. Le modèle de **LocationProvider** composite permet de manipuler plusieurs objets **LocationProvider** comme s'il s'agissait d'un seul. La classe **CompositeLocationProvider** implémente ce modèle, selon le patron de conception composite, pour deux objets **LocationProvider**. Le diagramme *UML* ci-dessous décrit la classe.

### Instructions :

1. Définir la classe **CompositeLocationProvider** et son constructeur.
2. Implémenter la méthode `get_location_samples`.
3. **Optionnel :** Redéfinir la méthode `__str__` de sorte à afficher un objet **CompositeLocationProvider** sous la forme suivante : l'affichage des deux objets **LocationProvider** contenu dans un composite est décalé vers la droite. On utilisera la fonction `indent` définie dans le module `util` (voir Section I).

```

+ CompositeLocationProvider (9 location samples)
+   PictureLocationProvider (source: '../data/pics/jdoe'
    (JPG,JPEG,jpg,jpeg), 2 location samples)
+   CompositeLocationProvider (7 location samples)
+     WifiLogsLocationProvider (source: '../data/db/wifi.db', user
        'John', 0 location samples)
+     TwitterLocationProvider (user 'teaching_isplab' aka 'Teaching
        ISPLab UNIL', 7 location samples)

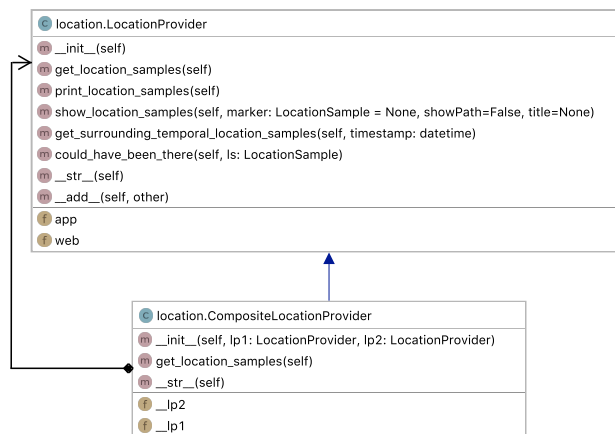
```

4. Définir la méthode `__add__` pour créer un nouveau composite lorsque l'on utilise l'opérateur "+" pour des **ListLocationProvider**.

**⚠ Attention :** Placer judicieusement cette méthode dans la bonne classe du module *location*.

**Exemple :** `lp1 + lp2` renvoie un objet **CompositeLocationProvider** composé de `lp1` et de `lp2`.





## VII Le module de sources Photographies : **picture**. Cours 5 : Programmation orientée objet

Pour certain-e-s suspect-e-s, les enquêteur-trice-s ont retrouvé sur leurs disques durs certaines de leurs photos (prises avec un smartphone) qui contiennent les dates et la localisation auxquelles les photos ont été prises (données *EXIF*<sup>4</sup>). On définit une classe **PictureLocationProvider** pour manipuler de telles sources de données. Les objets de cette classe sont construits à partir de fichiers image au format JPEG extension : .jpg, .jpeg, .JPG, .JPEG—cette liste est stockée comme un attribut de classe (contenu dans le répertoire *directory* passé en paramètre du constructeur. La date/heure et la localisation des photos sont extraites des informations *EXIF* (grâce au module *ExifRead*) ; la majeure partie du code permettant d’extraire les données d’une image est fournie : il ne reste que quelques lignes à compléter. Le diagramme UML ci-dessous décrit la classe.

### Instructions :

1. Définir la classe **PictureLocationProvider** et les éventuels attributs de classe (ainsi que les éventuels getters pour y accéder).

**⚠ Attention :** faire en sorte que la liste des extensions ne puisse pas être altérée via le résultat de la méthode `get_list_valid_extensions` .

2. Compléter la méthode `__extract_location_sample_from_picture` pour renvoyer le triplet **datetime** (**int**), **latitude** (**float**), **longitude** (**float**) extrait de l’image. Le code pour les coordonnées est déjà fourni. Il ne reste qu’à transformer la date/heure de la prise de vue en un **timestamp**. Utiliser la méthode `strptime` du module `datetime`.

**💡 Indice :** (1) afficher la date extraite des données *EXIF* pour en connaître le format (2) créer un objet **datetime** à partir de la date extraite (si elle est non **None**) en utilisant la méthode `strptime`. Vous pouvez utiliser le *debugger* afin de regarder le contenu des variables *date* et *timestamp*.

3. Définir son constructeur. Dans le constructeur, on construit une liste d’objets **LocationSample**, en extrayant, à l’aide de la méthode `__extract_location_sample_from_picture` précédemment complétée, les informations de chaque image du répertoire passé en paramètre (*directory*). Plus précisément, cette méthode renvoie un **datetime**, une **latitude** et une **longitude**, à partir desquels il faut créer un objet **LocationSample**. Si un des éléments renvoyés par la méthode `__extract_location_sample_from_picture` est **None**, ne pas ajouter l’élément à la liste de samples. Utiliser la méthode `scandir` du module `os` pour lister les fichiers du répertoire (voir la slide n°29 du cours n°2). La liste de samples est ensuite stockée grâce à l’attribut défini dans la classe **ListLocationProvider**.

4. Modifier le constructeur pour intercepter les exceptions et continuer l’exécution en passant au fichier suivant en cas de problème. Afficher simplement un message de la forme suivante (uniquement si l’utilisateur-trice a spécifié l’argument `–verbose`) :

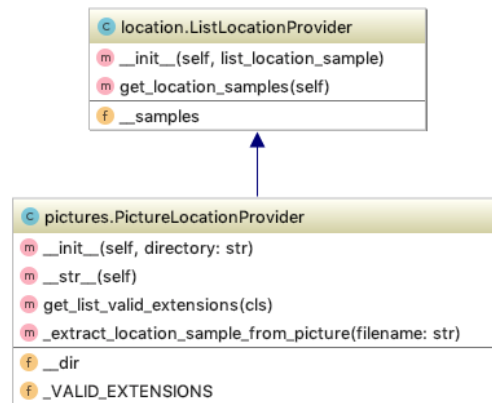
**Warning: Skipping file ‘unil.jpg’ (Missing time and/or location information)**

5. Implémenter la méthode `get_location_samples`.

6. **Optionnel :** Redéfinir la méthode `__str__` de sorte à afficher un objet **PictureLocationProvider** sous la forme suivante :

**PictureLocationProvider** (source: ‘`../data/pics/jdoe`’ (JPG,JPEG,jpg,jpeg), 2 location samples)

4. [https://fr.wikipedia.org/wiki/Exchangeable\\_image\\_file\\_format](https://fr.wikipedia.org/wiki/Exchangeable_image_file_format)



### VIII Le module de sources Twitter : **twitter**. Cours 10 : Programmation réseau, web APIs

Les enquêteur-trice-s ont réussi à faire installer une application Twitter (dégusée en jeu) à certain-e-s suspect-e-s, ce qui leur permet d'accéder aux tweets de ces suspect-e-s ; certains de ces tweets (typiquement ceux postés depuis un smartphone) contiennent une date et une localisation qui sont accessibles via l'*API* de Twitter<sup>5</sup>. On définit une classe **TwitterLocationProvider** permettant de manipuler un compte Twitter comme une source d'objets **LocationSample**, c'est-à-dire un **ListLocationProvider**. Le module *tweepy* (ou autre) permet d'accéder de manière transparente à l'*API* Twitter sans avoir à émettre de requête HTTP directement. Pour accéder à l'*API* Twitter, deux clés sont nécessaires. Elles sont fournies par l'enquêteur via un argument de la ligne de commande (voir Section II). Pour accéder au compte Twitter d'un-e utilisateur-trice particulier (pour ne pas être limité à ses tweets publics) il faut deux **authorization tokens**. Ces derniers sont fournis dans le fichier décrivant les suspect-e-s (voir Section V) et devraient être passés, avec le nom d'utilisateur-trice du compte Twitter du suspect, au constructeur de la classe **TwitterLocationProvider**. Le diagramme UML ci-dessous décrit la classe.

#### Instructions :

1. Définir la classe **TwitterLocationProvider**.
2. Créer deux attributs (de classe, d'instance ?) privés correspondant à la clé de l'*API* et à la clé secrète de l'*API*. Créer les deux setters correspondants.
3. Comme pour la classe **PictureLocationProvider**, créer une méthode privée `_extract_location_sample_from_tweet` qui prendra en paramètre un tweet (au format renvoyé par le module *tweepy*, ou autre), et qui se chargera d'en extraire la date et l'heure de création (sous forme d'un `datetime`), ainsi que la latitude et la longitude. Vérifiez que les paramètres sont bien présents dans le tweet. On utilisera les champs `coordinates` et `created_at` de la réponse renvoyée par l'*API*.

**⚠ Attention :** Utiliser le *debugger* pour parcourir les différents champs renvoyés par l'*API*.

4. Définir son constructeur. Dans le constructeur, construire une liste de **LocationSample** en extrayant, à l'aide de la méthode `_extract_location_sample_from_tweet` précédemment complétée, les informations des tweets obtenu à partir du compte Twitter passé en paramètre. La liste de samples est ensuite stockée grâce à l'attribut défini dans la classe **ListLocationProvider**.

(a) Utiliser les différentes clés pour vous connecter à l'*API* de Twitter.

(b) Commencer par extraire le nom de l'utilisateur-trice (méthode `get.user` de *tweepy*, comme en TP). Créer un attribut nom correspondant à celui-ci. Si une erreur survient lors de l'extraction du nom, afficher un message (uniquement si l'utilisateur-trice a spécifié l'argument `-verbose`) de la forme suivante et assigner une valeur par défaut au nom.

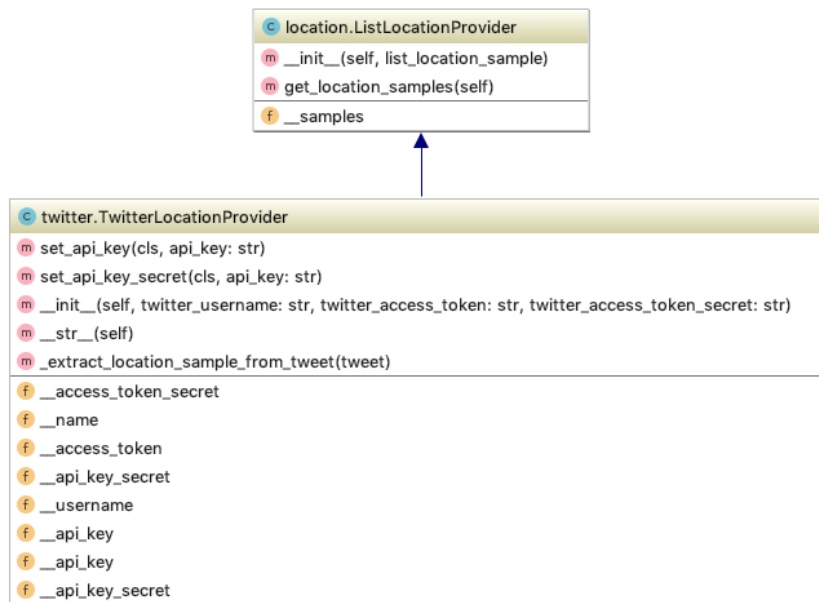
**Warning:** Could not extract teaching.isplab's Twitter account information ( details sur l'erreur)

(c) Récupérer les tweets postés par l'utilisateur-trice (méthode `user.timeline`). Pour chaque tweet, extraire la date et l'heure ainsi que les données de géolocalisation à l'aide de la méthode `_extract_location_sample_from_tweet`. Interceptor les exceptions et continuer l'exécution en passant au tweet suivant en cas de problème. Afficher simplement un message de la forme suivante (uniquement si l'utilisateur-trice a spécifié l'argument `-verbose`) :

**Warning:** Skipping tweet (Missing time and/or location information (842372886627258369))

5. Informations détaillées sur l'*API* de Twitter : [https://dev.twitter.com/rest/reference/get/statuses/user\\_timeline](https://dev.twitter.com/rest/reference/get/statuses/user_timeline).

5. Redéfinir la méthode `__str__` de sorte à afficher un objet `TwitterLocationProvider` sous la forme suivante : `TwitterLocationProvider (user 'teaching_isplab' aka 'Teaching ISPLab UNIL', 7 location samples)`



## IX Le module de sources Wi-Fi : wifi. Cours 9 : Bases de données

Les enquêteur-trice-s ont obtenu accès aux logs de connexion du réseau Wi-Fi de l'UNIL (via RADIUS/EAP). À chaque fois qu'un smartphone se connecte au réseau secure-unil (il se connecte automatiquement dès lors qu'un point d'accès est à portée s'il est configuré pour utiliser ce réseau), un événement est créé dans les logs. L'événement contient l'identifiant UNIL de l'utilisateur-trice et l'identifiant du point d'accès Wi-Fi. La localisation de ces points d'accès est connue. Toutes ces données sont stockées dans une base de données au format *SQLite*.

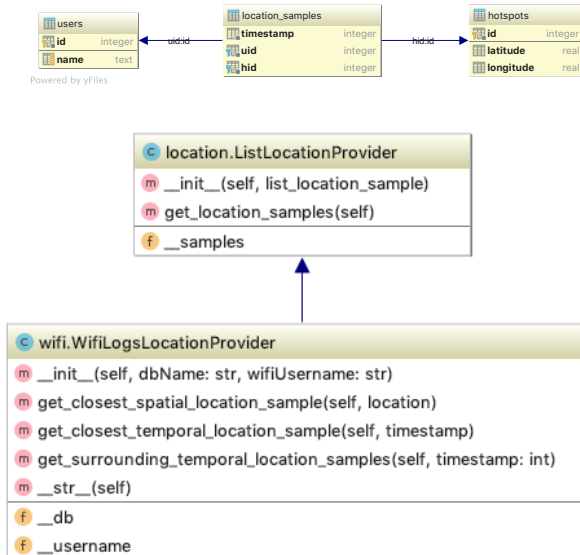
La base de données contient trois tables décrites dans le diagramme UML suivant. La table `users` contient la liste des utilisateur-trice-s spécifiés par un identifiant unique et un nom d'utilisateur-trice (unique également, par exemple `jdoe`). La table `hotspots` contient la liste des points d'accès Wi-Fi spécifiés par un identifiant unique, une latitude et une longitude. La table `location_samples` contient les événements de connexion spécifiés par un **timestamp** et une référence (via les clés étrangères correspondantes) vers l'utilisateur-trice et le point d'accès concerné. On définit une classe `WifiLogsLocationprovider` pour manipuler de tels logs comme une source d'objets `LocationSample`, c'est-à-dire un location provider. Un tel objet est construit à partir du nom du fichier de base de données et le nom d'utilisateur-trice UNIL du suspect.

### Instructions :

1. Définir la classe `WifiLogsLocationProvider`.
2. Définir son constructeur. Dans le constructeur, construire une liste d'objets `LocationSample` à partir des données contenues dans la base de données passée en paramètre. Pour ce faire, utiliser une commande *SQL* adéquate (utiliser l'instructions `SELECT ... JOIN ON ...` pour combiner les données des différentes tables). La liste de samples est ensuite stockée grâce à l'attribut défini dans la classe `ListLocationProvider`. On peut utiliser la fonction `dict_factory` (du module `utils`) comme `row_factory` pour la connexion afin de pouvoir accéder directement aux résultats de la requête à l'aide des noms de colonnes de la base de données *SQLite* (c.f. TP).  
**💡 Indice :** (1) Ecrire une requête *SQL* pour récupérer les données voulues en effectuant deux opérations `JOIN` et un filtre `WHERE` (pour sélectionner l'utilisateur-trice dont on veut récupérer les logs ; le nom de l'utilisateur-trice est passé en paramètre au constructeur). (2) Créer une connexion à la base de données stockée dans le fichier dont le nom est passé en paramètre au constructeur. (3) Récupérer un curseur et exécuter la requête. (4) Récupérer les résultats, créer un objet `LocationSample` pour chaque enregistrement et l'ajouter à la liste de samples.
3. Modifier la requête *SQL* utilisée pour trier directement les résultats par ordre chronologique.
4. Implémenter la méthode `get_location_samples`.

5. **Optionnel** : Redéfinir la méthode `get_surrounding_temporal_location_sample` de telle sorte que les calculs soient effectués directement via la requête *SQL* (et non par du code Python).
6. **Optionnel** : Redéfinir la méthode `__str__` de sorte à afficher un objet `WifiLogsLocationProvider` sous la forme suivante :

`WifiLogsLocationProvider` (source: `'../data/db/wifi.db'`, user `'alice'`, 3 location samples)



## X Le module de sources Logs de téléphone : logs. Cours 8 : Analyse Syntaxique

Les enquêteur-trice-s ont obtenu accès aux logs des téléphones portables de certain-e-s suspect-e-s. À chaque fois qu'une action est effectuée avec le smartphone, l'événement est inséré dans les logs du téléphone. De nombreux événements y figurent mais ceux qui sont intéressants pour l'enquête sont ceux qui sont inscrit lorsque certaines applications demandent des données de géolocalisation. Ainsi, certaines lignes de logs contiennent l'heure de la demande ainsi que les coordonnées courante de l'utilisateur-trice. Chaque fichier contient des logs (un par ligne). Chaque ligne débute par l'heure exacte ou le log a été créé. Le texte y figurant ensuite dépend de l'action effectuée. Les logs de requête de coordonnées ont plusieurs particularités :

- Ils contiennent le nom de l'application effectuant la requête
- Ils contiennent des coordonnées
- Ils contiennent la source de ces coordonnées (GPS, WIFI, ...)

Aucune de ces particularités n'est exclusive mais si un log les contient toutes, il s'agit bien d'un log de requête de coordonnées. Attention toutefois à plusieurs éléments. Certains logs n'ont pas pu obtenir le timestamp, il est donc fixé à UNKNOWN. D'autres n'ont pas pu obtenir de coordonnées, elles sont donc également fixée à UNKNOWN. Quant aux source de coordonnées, certaines sont trop peu précises, par conséquent, seule les données provenant d'une source GPS seront considérées (3ème ligne de l'exemple ci-dessous).

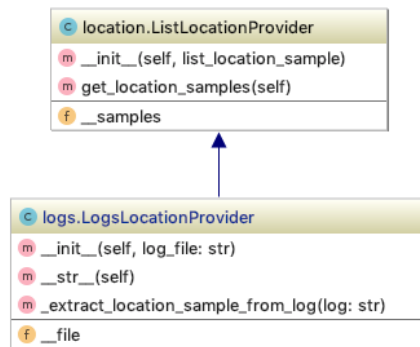
```

[2019-03-21T13:08:21.149] unplugging device , status: on battery , battery level: 31%
[2019-03-21T13:08:22.393] airplane mode disable
[2019-03-21T13:08:23.035] Location request from app facebook , returned coordinates:
(46.7483762870913, 6.5883562761524354) , source: GPS
[2019-03-21T13:08:25.663] plugged device , status: on charge , battery level: 17%
[2019-03-21T13:08:27.180] unplugging device , status: on battery , battery level: 56%
[2019-03-21T13:08:28.840] device volume changed , current volume strength: 2/10
  
```

### Instructions :

1. Définir la classe `LogsLocationProvider`.
2. Définir son constructeur. Dans le constructeur, définir un attribut contenant le nom du fichier de logs. Construire une expression régulière répondant aux critères ci-dessus. Utiliser ensuite cette expression pour filtrer les logs qui font partie du langage qu'elle définit.

3. Extraire de ces logs valides les données temporelles et de géolocalisation. Créer ensuite un `LocationSample` et l'ajouter dans la liste. Pour ce faire, définir une méthode `_extract_location_sample_from_log` qui prend un log en argument et retourne un datetime, une latitude et une longitude.
4. **Optionnel** : Redéfinir la méthode `__str__` de sorte à afficher un objet `LogsLocationProvider` sous la forme suivante :  
**LogsLocationProvider** (source: `../data/logs/jdoe.log`, 2 location samples)



## XI Le module de cartographie Google Maps : **googlemaps**. Cours 10 : Programmation réseau, web APIs

On enrichit la classe `Location` du module `location` (voir Section IV) grâce au module de cartographie Google Maps (via son API) auquel on accède via le module `googlemaps`. Ce module ajoute la possibilité de se connecter à l'API de Google Maps afin d'obtenir l'adresse textuelle ou formatée correspondante à des coordonnées géographiques (reverse geocoding, comme en TP), ainsi que d'obtenir la distance et le temps nécessaire pour effectuer un trajet (à pied ou en utilisant un autre mode de transport) entre deux points.

### Instructions :

1. Ajouter deux attributs de classe correspondant à la clé de l'API et au client (c'est-à-dire l'objet `Client` du module `googlemaps`) qui effectuera le lien avec l'API.
2. Créer la méthode `set_api_key`, qui en plus de modifier la clé, crée et initialise l'objet `Client`.
3. Ajouter une méthode privée (pour usage interne) `__check_api_init` qui permet de vérifier que le client a été initialisé, avec une explication de la marche à suivre si ce n'est pas le cas. Par exemple :  
**Exception: You must specify a Google API key. Use Location.set\_api\_key (...)**
4. Implémenter la méthode d'instance `get_name` qui retourne, en utilisant l'API Google reverse geocoding<sup>6</sup>, le nom correspondant aux coordonnées contenues dans un objet `Location` (explorer le champ `formatted_address` contenu dans la réponse de l'API Google). Renvoyer une valeur par défaut et afficher un message d'erreur en cas de problème. Vérifier que l'API est correctement initialisée.  
**Optionnel** : spécifier la langue dans laquelle exprimer le nom du lieu.

Implémenter la méthode d'instance `get_travel_distance_and_time` qui retourne, en utilisant l'API Google distance matrix<sup>7</sup>, le couple distance/temps de parcours depuis l'objet `Location` considéré vers un autre objet `Location` passé en paramètre. Utiliser le paramètre adéquat pour obtenir les résultats dans le système international (pour les unités de mesure de la distance par exemple).

**Optionnel** : ajouter un paramètre optionnel pour spécifier le mode de transport.

## Conclusion

Grâce à ces modules, vous avez maintenant la possibilité de collecter et d'analyser les traces de mobilité des différents suspect-e-s. Récupérer la liste des samples pour écarter les suspect-e-s innocents, n'ayant pas pu avoir le temps de se rendre sur le lieu du crime. À vous de jouer !

6. Information détaillée sur l'API : <https://developers.google.com/maps/documentation/geocoding/intro#ReverseGeocoding>

7. Information détaillée sur l'API : <https://developers.google.com/maps/documentation/distance-matrix/intro>