10 Dictionnaire

Extrait du programme

Thème : Types construits

- **Contenus :**
  Dictionnaires par clés et valeurs

Capacités attendus :
Construire une entrée de dictionnaire.
Itérer sur les éléments d'un dictionnaire.

Commentaires :
Il est possible de présenter les données `EXIF` d'une image sous la forme d'un enregistrement.
En Python, les p-uplets nommés sont implémentés par des dictionnaires.
Utiliser les méthodes `keys()`, `values()` et `items()`.

I. Introduction et rappels

Nous avions déjà 2 types construits :

- le p-uplet :

```
>>> adaTuple = ("Ada", "Lovelace", 10, 12, 1815)
```

  Les valeurs stockées peuvent être de type différents (ici nous avons des chaînes de caractères, *String*, et des entiers, *integer*).
  Les valeurs sont entre parenthèses [1] séparées par des virgules.
  Les valeurs ne peuvent être modifiées, taille du tuple non plus.

- le tableau :

```
>>> adaTableau = [10, 12, 1815]
```

  Les valeurs stockées doivent être du même type [2] (ici, il n'y a que des entiers, *integer*).
  Les valeurs sont entre crochets séparées par des virgules.
  Les valeurs sont modifiables, mais pas la taille du tableau [3].

La structure de **tableau** est bien pratique mais elle peut aussi s'avérer insuffisante dans biens des cas. En effet, dans le tableau `adaTableau = [10 ,12 ,1815]` on peut oublier si on est dans un système français jour/mois/année ou dans un système anglo-saxons month/day/year. On aimerait pouvoir fixer à chaque valeur une **clé**. Ainsi :

- 10 sera associé à la clé `jour` ;
- 12 sera associé à la clé `mois` ;
- 1815 sera associé à la clé `annee`.

Pour la suite, peu importe l'ordre dans lequel les valeurs sont stockées, si on demande la clé `mois`, on obtient le résultat 12.

1. Techniquement, les parenthèses ne sont pas obligatoires, mais très fortement conseillées.
2. En réalité, la structure **tableau** n'est pas implémenté en Python. A la place, il y a la structure **liste** qui permet de stocker des valeurs de types différents. Cette structure étant hors-programme, on l'évitera.
3. Sauf qu'en réalité, nous n'avons pas de **tableau** mais des **listes** dont nous pouvons modifier la taille sans aucun souci

II. Dictionnaire

a) Définition

Définition :
Un **dictionnaire** est une structure de données définie par un ensemble d'éléments, auxquels on accède par mot **clé**.
Les éléments d'un dictionnaire sont des couples clé-valeur. Un dictionnaire est créé avec des accolades, les différents couples étant séparés par des virgules. La clé et la valeur correspondante d'un élément sont séparées par deux-points.

L'implémentation d'un dictionnaire optimise le coût en temps de la recherche d'un élément comme nous le verrons dans la suite de ce cours.

```
>>> adaDico = {"prenom" : "Ada", "nom" : "Lovelace", "jour" : 10, "mois" : 12, "
    annee" : 1815}
```

Quand le dictionnaire est un peu long, on peut préférer aller à la ligne après chaque couple clé-valeur.

```
>>> adaDico = {"prenom" : "Ada",
               "nom" : "Lovelace",
               "jour" : 10,
               "mois" : 12,
               "annee" : 1815}
```

b) Accès aux valeurs

Pour avoir accès à une valeur, il suffit de l'appeler à l'aide de sa clé :

```
>>> adaDico["jour"]
10
```

Attention ! Comme il n'y a pas de notion d'ordre dans un dictionnaire, l'appel `adaDico[2]` renverra un message d'erreur :

```
>>> adaDico[2]
Traceback (most recent call last):
  File "<console>", line 1, in <module>
KeyError: 2
```

Évidemment, un appel avec une mauvaise clé renverra également un message d'erreur :

```
>>> adaDico["day"]
Traceback (most recent call last):
  File "<console>", line 1, in <module>
KeyError: 'day'
```

c) Modifier un dictionnaire

- On peut modifier une valeur

```
>>> adaDico["nom"]="comtesse de Lovelace"
>>> adaDico
{'prenom': 'Ada', 'nom': 'comtesse de Lovelace', 'jour': 10, 'mois': 12, 'annee
    ': 1815}
```

- On peut rajouter un couple clé-valeur

```
>>> adaDico["nom jeune fille"] = "Byron"
>>> adaDico
{'prenom': 'Ada', 'nom': 'comtesse de Lovelace', 'jour': 10, 'mois': 12, 'annee
    ': 1815, 'nom jeune fille': 'Byron'}
```

- On peut supprimer un couple clé-valeur

```
>>> del(adaDico["nom jeune fille"])
>>> adaDico
{'prenom': 'Ada', 'nom': 'comtesse de Lovelace', 'jour': 10, 'mois': 12, 'annee
    ': 1815}
```

- On peut supprimer un dictionnaire

```
>>> del(adaDico)
>>> adaDico
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'adaDico' is not defined
```

d) Les méthodes

- Avec la méthode `keys()` nous obtenons la liste de toutes les clefs de notre dictionnaire :

```
>>> adaDico.keys()
dict_keys(['prenom', 'nom', 'jour', 'mois', 'annee'])
```

- Avec la méthode `values()` nous obtenons la liste de toutes les valeurs de notre dictionnaire :

```
>>> adaDico.values()
dict_values(['Ada', 'Lovelace', 10, 12, 1815])
```

- Avec la méthode `items()` nous obtenons la liste de toutes les couples (tuples) clé-valeurs de notre dictionnaire :

```
>>> adaDico.items()
dict_items([('prenom', 'Ada'), ('nom', 'Lovelace'), ('jour', 10), ('mois', 12),
      ('annee', 1815)])
```

e) Tester l'appartenance

Nous pouvons tester l'appartenance avec le mot `in` :

- Tester l'appartenance d'une clé :

```
>>> "jour" in adaDico
True
```

- Tester l'appartenance d'une valeur :

```
>>> 3 in adaDico.values()
False
```

- Tester l'appartenance d'un couple clé-valeur :

```
>>> ("prenom","Ada") in adaDico.items()
True
```

f) Parcourir un dictionnaire

- On peut parcourir un dictionnaire à l'aide des clés :

```
>>> for clef in adaDico :
            print(clef)
prenom
nom
jour
mois
annee
```

Un équivalent :

```
>>> for clef in adaDico.keys() :
            print(clef)
prenom
nom
jour
mois
annee
```

- On peut parcourir à l'aide des valeurs :

```
>>> for valeur in adaDico.values() :
            print(valeur)
Ada
Lovelace
10
12
1815
```

Ce qui revient à écrire (mais c'est plus compliqué) :

```
>>> for clef in adaDico :
            print(adaDico[clef])
Ada
Lovelace
10
12
1815
```

- On peut parcourir un dictionnaire suivant ses clefs et ses valeurs :

```
>>> for clef, valeur in adaDico.items() :
...     print("La clé '",clef,"' a pour valeur",valeur,".")
La clé ' prenom ' a pour valeur Ada .
La clé ' nom ' a pour valeur Lovelace .
La clé ' jour ' a pour valeur 10 .
La clé ' mois ' a pour valeur 12 .
La clé ' annee ' a pour valeur 1815 .
```

g) Par compréhension

Un dictionnaire peut être défini par compréhension, comme les tableaux :

```
>>> cube = {n : n ** 3 for n in range(10) if n > 0}
>>> cube
{1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343, 8: 512, 9: 729}
```

h) Un Dictionnaire de dictionnaires

On peut créer un dictionnaire de dictionnaires en associant à une clé, un dictionnaire comme valeur :

```
>>> adaDico = {"prenom" : "Ada",
                "nom" : "Lovelace",
                "naissance" : { "jour" : 10,
                        "mois" : 12,
                        "annee" : 1815},
                "mort" : { "jour" : 27,
                        "mois" : 10,
                        "annee" : 1852}
                }
```

```
>>> adaDico["naissance"]
{'jour': 10, 'mois': 12, 'annee': 1815}
```

```
>>> adaDico["naissance"]["jour"]
10
```

III. Exercices

Exercice 1 (Inspiré par un exemple de José DELAMARE - professeur de SP et d'Informatique - Rouen)
Voici le dictionnaire suivant :

```
user = {"Nom" : "Bond", "password" : "007"}
```

1. Écrire un petit programme qui permet d'afficher toutes les valeurs du dictionnaire.
2. Écrire un petit programme qui donne le nom si le bon mot de passe a été saisi par l'utilisateur.

Exercice 2 (Inspiré par un exemple de José DELAMARE - professeur de SP et d'Informatique - Rouen)
Voici le dictionnaire suivant donnant le nom d'un enfant et son âge :

```
personnes = {
    "Jean Bon": 12,
    "César Bistrukla": 10,
    "Marc Assin": 11,
    "John Doeuf": 10,
    "Agathe Leblouze": 11,
    "Guy Dondvelo": 12,
    "Eva Nouissement": 12,
    "Juda Nana": 10,
    "Laurie Culayr": 11,
    "Mélanie Chedanslejardin": 10,
    "Alain Disoir": 10,
    "Jean Foupasune": 10,
    "Sophie Fonsec": 10,
    "Jean Filmongilet": 11,
    "Gédéon Groisdansmabaignoire": 12
    }
```

1. Écrire un petit programme qui permet de faire les phrases du type ci-dessous pour tous les enfants :
   `Jean Bon a 10 ans.`
2. Écrire un petit programme qui permet de faire les mêmes phrases uniquement pour les enfants qui ont 10 ans.
3. Écrire un petit programme qui calcul l'âge moyen des enfants présent dans ce dictionnaire.
4. Écrire un petit programme qui renvoie une phrase du type

   `Avec ses 10 ans, Jean Bon est l'élève le plus âgé du groupe.`

   où évidemment `Jean Bon` sera remplacé par le nom de l'élève le plus âgé et `10` sera remplacé par son âge.

Exercice 3 (**)
Construire par compréhension le dictionnaire de tous les carrés parfaits, du carré de 1 au carré de 100.

Exercice 4 (**)
1. Construire par compréhension le dictionnaire de tous les codes ASCII en clés et de toutes lettres de l'alphabet en minuscules pour les valeurs associés.
2. Écrire une fonction qui prend en argument un code ASCII et qui renvoie le caractère minuscule associé en le recherchant dans le dictionnaire.
3. Écrire une fonction qui prend en argument un caractère minuscule et qui renvoie le code ASCII associé en le recherchant dans le dictionnaire.

IV. Pour aller plus loin : le format JSON

JavaScript Object Notation, **JSON** est un format de données textuelles dérivé de la notation des objets du langage JavaScript.
Un document JSON comprend deux types d'éléments structurels :

- des couples clé-valeur ;
- des listes ordonnées de valeurs.

Ce format est très utilisée pour la gestion de données, tout comme le format **CSV**.
Nous avions vu au chapitre 12 que le format **CSV** a quelques similarités avec le type construit **table** (tableaux à 2 dimensions). Nous constattons ici que le format **JSON**, lui, a quelques similarités avec le type construit **dictionnaire**.

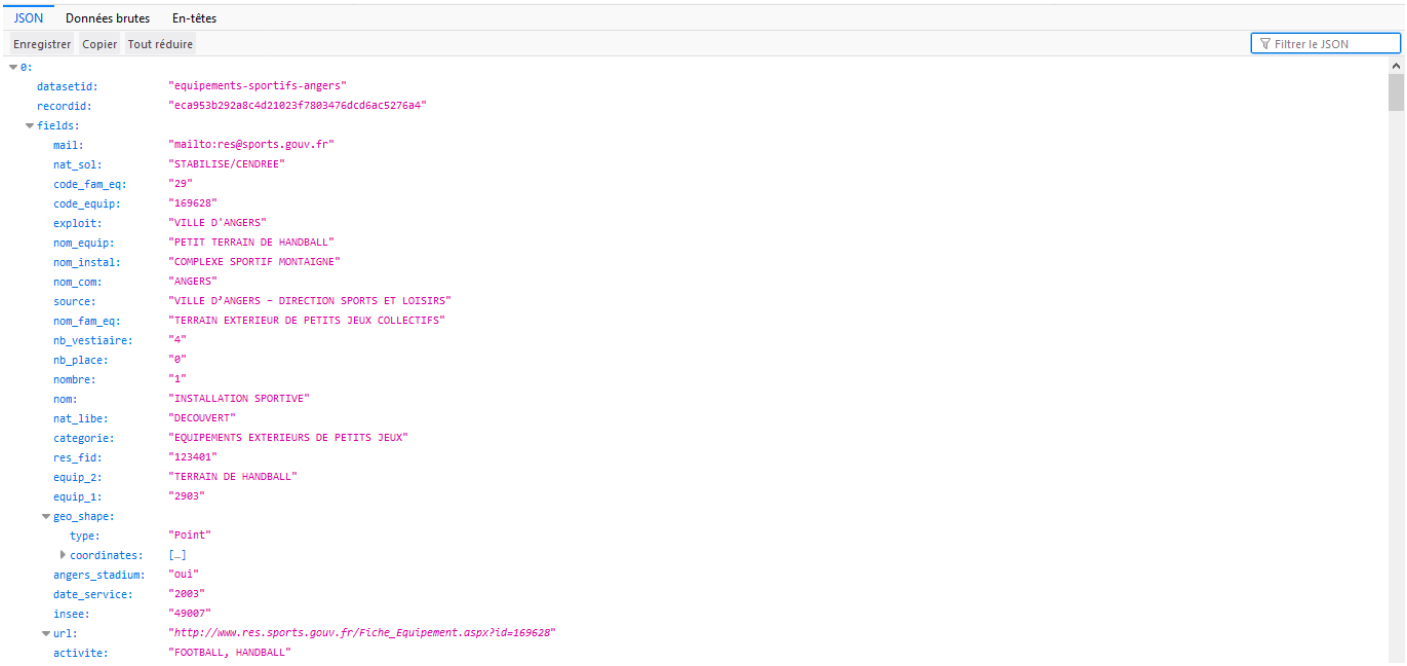
Nous allons travailler à partir du site `https://data.angers.fr/` qui regroupe toutes les données publiques de la ville d'Angers. Intéressons-nous aux équipements sportifs de la ville en téléchargeant le fichier au format **JSON** à l'adresse suivante : `https://data.angers.fr/explore/dataset/equipements-sportifs-angers/export/`
Si on ouvre le fichier avec le bloc-notes, on constate qu'il y a plein d'informations mais cela semble bien désordonné...

On pourrait indenter les différents champs pour un peu mieux comprendre ce fichier :

```
[
{"datasetid": "equipements-sportifs-angers",
"recordid": "eca953b292a8c4d21023f7803476dcd6ac5276a4",
"fields": {"mail": "mailto:res@sports.gouv.fr",
        "nat_sol": "STABILISE/CENDREE",
        "code_fam_eq": "29",
        "code_equip": "169628",
        "exploit": "VILLE D'ANGERS",
        "nom_equip": "PETIT TERRAIN DE HANDBALL",
        "nom_instal": "COMPLEXE SPORTIF MONTAIGNE",
        "nom_com": "ANGERS",
        "source": "VILLE D'ANGERS - DIRECTION SPORTS ET LOISIRS",
        "nom_fam_eq": "TERRAIN EXTERIEUR DE PETITS JEUX COLLECTIFS",
        "nb_vestiaire": "4",
        "nb_place": "0",
        "nombre": "1",
        "nom": "INSTALLATION SPORTIVE",
        "nat_libe": "DECOUVERT",
        "categorie": "EQUIPEMENTS EXTERIEURS DE PETITS JEUX",
        "res_fid": "123401",
        "equip_2": "TERRAIN DE HANDBALL",
        "equip_1": "2903",
        "geo_shape": {"type": "Point",
                "coordinates": [-0.5344299995, 47.4703199998]},
        "angers_stadium": "oui",
        "date_service": "2003",
        "insee": "49007",
        "url": "http://www.res.sports.gouv.fr/Fiche_Equipement.aspx?id=169628",
        "activite": "FOOTBALL, HANDBALL",
        "geo_point_2d": [47.4703199998, -0.5344299995],
        "date_ref": "2017-12-04", "code_instal": "490070002"},
        "geometry": {"type": "Point",
                "coordinates": [-0.5344299995, 47.4703199998]},
        "record_timestamp": "2019-02-07T11:45:00+01:00"
        },
{"datasetid": "equipements-sportifs-angers",
"recordid": "8cdaae24192da9ce41a2f32e67855ebc8719b49b",
...
        },
]
```

Ce format n'est donc rien d'autre qu'un tableau (les crochets) de dictionnaires (les accolades).

Pour visualiser correctement les données, le plus simple est d'utiliser votre navigateur préféré :



Pour récupérer les données pour un traitement en Python, on peut par exemple utiliser le module `json`. On obtient alors une `liste` de `dictionnaires`.

```
1 import json
2
3 with open('equipements-sportifs-angers.json', encoding='utf8') as equipt:
4     equipements = json.loads(equipt.read())
```

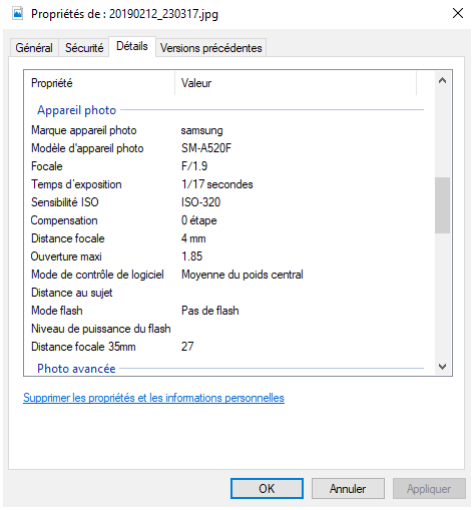
V. Pour aller plus loin : les méta-données EXIF

a) EXIF c'est quoi ?

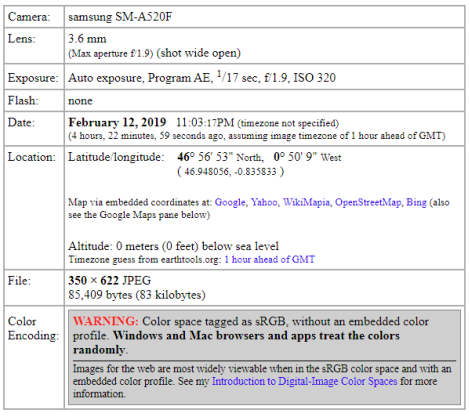
EXIF (*Exchangeable image file format*) correspond à un ensemble de métadonnées générées lors de la prise de vue d'un appareil photographique. Ces données sont stockées dans les fichiers images de formats `.jpg`, `.raw` ou `.tiff`. Ce format de stockage des données a été établi en 1995 par la Japan Electronic Industry Developpement Association. Elles permettent de sauvegarder, entre autres :

- les données techniques de l'image (poids, résolution, dimensions, date ...)
- les paramètres de configuration de l'appareil (références du boîtier, ouverture, distance focale, sensibilité...)
- le crédit photo (nom de la photo, auteur, droits d'usage...)
- des informations géographiques (coordonnées GPS, lieu, ...)

b) Comment lire ces données ?

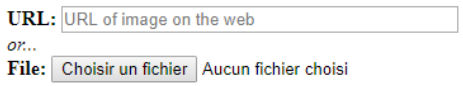


- Sur un ordinateur :
  Faire un clic droit sur l'image puis Propriétés / Détails.
  Notez qu'il est possible de supprimer les propriétés et les informations personnelles en cliquant sur le lien en bas de l'encart d'affichage.



Camera:	samsung SM-A520F
Lens:	3.6 mm (Max aperture f/1.9) (shot wide open)
Exposure:	Auto exposure, Program AE, 1/17 sec, f/1.9, ISO 320
Flash:	none
Date:	**February 12, 2019** 11:03:17PM (timezone not specified) (4 hours, 22 minutes, 59 seconds ago, assuming image timezone of 1 hour ahead of GMT)
Location:	Latitude/longitude: **46°** 56' 53" North, **0°** 50' 9" West (46.948056, -0.835833) Map via embedded coordinates at: Google, Yahoo, WikiMapia, OpenStreetMap, Bing (also see the Google Maps pane below) Altitude: 0 meters (0 feet) below sea level Timezone guess from earthtools.org: 1 hour ahead of GMT
File:	**350** × **622** JPEG 85,409 bytes (83 kilobytes)
Color Encoding:	WARNING: Color space tagged as sRGB, without an embedded color profile. **Windows and Mac browsers and apps treat the colors randomly**. Images for the web are most widely viewable when in the sRGB color space and with an embedded color profile. See my Introduction to Digital-Image Color Spaces for more information.

- Depuis un navigateur web :
  Des sites permettent d'obtenir les métadonnées en ligne, soit en pointant une image stockée sur l'ordinateur, soit en donnant l'URL de l'image.



  Par exemple le site `http://exif.regex.info/exif.cgi` donne l'intégralité des métadonnées EXIF (algorithme d'encodage, mode de codage de la couleur...)

c) Avec le langage Python

On peut travailler sur des images numériques (et donc des photos) on va récupérer le sous module `Image` du module `PIL`.

```
#pour travailler sur des images numériques
from PIL import Image

#pour charger la photo
photo = Image.open("ada.png")

#pour afficher la photo
photo.show()

#ne pas oublier de fermer le fichier
photo.close()
```

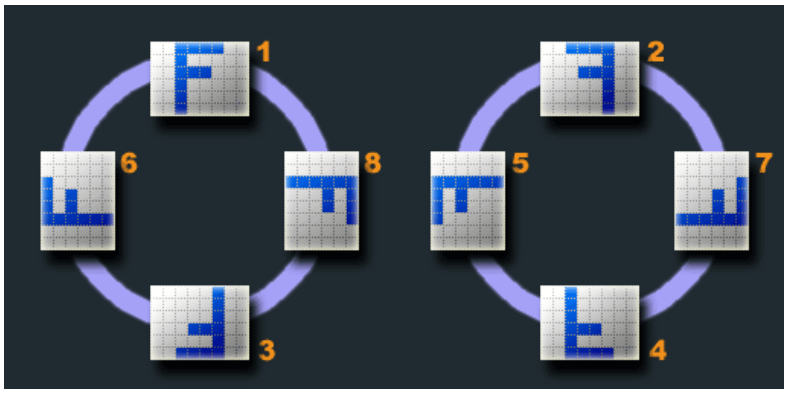
Pour les images numériques au format `.jpg`,`.raw` ou `.tiff` on peut récupérer les méta-données `EXIF` :

```
#pour travailler sur des images numériques
from PIL import Image

#pour charger la photo
photo = Image.open("chouchou0.jpg")

#pour afficher la photo
#photo.show()


#Récupération des métadonnées EXIF
info = photo._getexif()
for clef, valeur in info.items():
    print(clef,":",valeur)

#ne pas oublier de fermer le fichier
photo.close()
```

Par contre, les couples (et surtout les clefs) du dictionnaire sont difficiles à comprendre :

```
36864 : b'0220'
37121 : b'\x01\x02\x03\x00'
37377 : (571769, 65536)
36867 : 2019:09:19 16:26:26
36868 : 2019:09:19 16:26:26
37378 : (149094, 65536)
37379 : (393216, 65536)
37380 : (0, 65536)
37381 : (149094, 65536)
37383 : 2
37384 : 0
37385 : 24
37386 : (216269, 65536)
40961 : 65535
40962 : 3264
271 : SAMSUNG
272 : SM-G531F
40963 : 1836
274 : 1
531 : 1
41996 : 0
41495 : 2
282 : (72, 1)
283 : (72, 1)
33434 : (155, 65565)
33437 : (22, 10)
41729 : b'\x01'
34850 : 2
34853 : {1: 'N', 2: ((47, 1), (27, 1), (59, 1)), 3: 'W', 4: ((0, 1), (33, 1), (5, 1)), 6: (0, 10)}
34855 : 50
296 : 2
41986 : 0
40960 : b'0100'
41987 : 0
305 : GIMP 2.10.8
306 : 2019:09:29 19:26:26
41988 : (65536, 65536)
41989 : 31
41990 : 0
41992 : 0
41993 : 0
41994 : 0
34665 : 212
37500 : b'\x07\x00\x02\x00\x07\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x0c\x00\x07\x00\x01\x00\x00\x00\x00\x00\
      x00\x00@\x00\x07\x00\x01\x00\x00\x00\x00\x00\x00\x00P\x00\x07\x00\x01\x00\x00\x00\x01\x00\x00\x00`\x00\
      x07\x00\x04\x00\x00\x00none\x00\x01\x07\x00\x01\x00\x00\x00\x00\x00\x00\x00\x01\x00\x07\x00\x04\x00\x00
      \x000100\x00\x00\x00\x00'
```

Par exemple, la clé 274 permet d'obtenir l'orientation de l'appareil suivant le schéma ci-dessous :




```python
#pour travailler sur des images numériques
from PIL import Image

#pour charger la photo
photo = Image.open("chouchou0.jpg")

#pour afficher la photo
#photo.show()


#Récupération des métadonnées EXIF
info = photo._getexif()

#orientation de l'appareil
orientation = info[274]

#et si on retournait l'image
#Rotation de l'image suivant les cas
position="paysage"
if orientation == 3:
    photo=photo.rotate(180, expand=True)
elif orientation == 6:
    photo=photo.rotate(270, expand=True)
    position="portrait"
elif orientation == 8:
    photo=photo.rotate(90, expand=True)
    position="portrait"
photo.save("image_reorientee.jpg")


#ne pas oublier de fermer le fichier
photo.close()
```

La clé 34853 permet quant à elle d'obtenir les coordonnées GPS du lieu où a été pris la photo :

```
#pour travailler sur des images numériques
from PIL import Image

#pour charger la photo
photo = Image.open("chouchou0.jpg")

#pour afficher la photo
#photo.show()


#Récupération des métadonnées EXIF
info = photo._getexif()

#Récupération de la localisation GPS
gps = info[34853]
#gps est à nouveau un dictionnaire
lat_dir = gps[1] #N ou S pour North ou South
lat = gps[2]
lon_dir = gps[3] #W ou E pour West ou East
lon = gps[4]

#Conversion décimale de la latitude et de la longitude
latitude = lat[0][0]/lat[0][1] + (lat[1][0]/lat[1][1])/60 + (lat[2][0]/lat[2][1])
    /3600
longitude = lon[0][0]/lon[0][1] + (lon[1][0]/lon[1][1])/60 + (lon[2][0]/lon[2][1])
    /3600
#On change le signe de la longitude si besoin
if lon_dir == 'W':
    longitude = -1*longitude

print("les coordonnées GPS sont")
print("latitude :" , latitude)
print("longitude :" , longitude)

#ne pas oublier de fermer le fichier
photo.close()
```

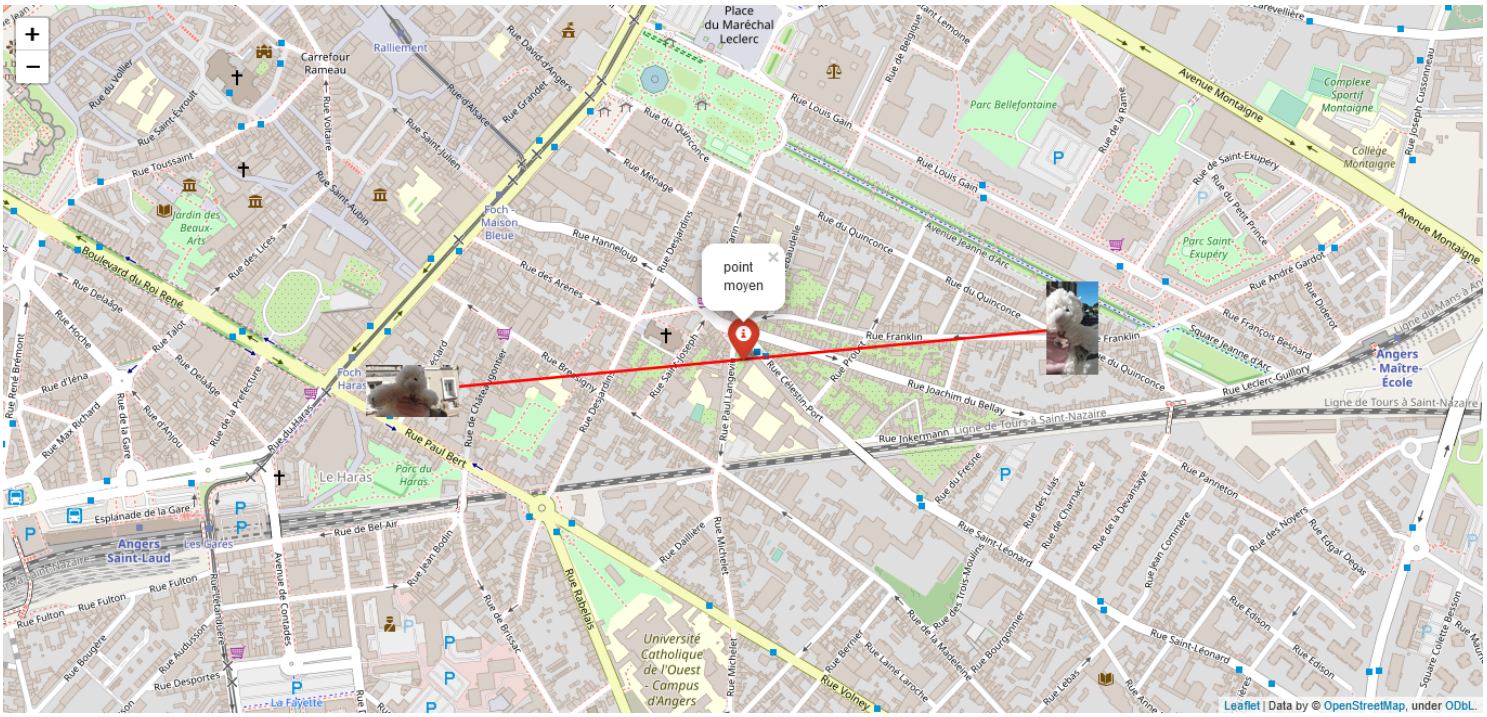
Nous obtenons les coordonnées GPS suivantes :

```
les coordonnées GPS sont
latitude : 47.46638888888889
longitude : -0.5513888888888889
```

On peut utiliser une autre bibliothèque, `folium`, qui permet de générer une page `HTML` récupérant une carte **Open Street Map** et d'y rajouter des informations supplémentaires (icon, parcours, ...).
On peut ainsi, mettre nos photos sur une carte.
Voici un exemple de programme.

```
#pour travailler sur des images numériques
from PIL import Image
#pour créer une carte
import folium


def coordGPS(photo,i):
    '''
    fonction qui :
    reoriente la photo si nécessaire
    récupère la longitude et la latitude
    récupère le jour, mois, année dela prise de photo
    '''

    #Récupération des métadonnées EXIF
    info = photo._getexif()

    ##############################################################
    #Récupération de l'orientation
    orientation = info[274]

    #Rotation de l'image suivant les cas
    position="paysage"
    if orientation == 3:
        photo=photo.rotate(180, expand=True)
    elif orientation == 6:
        photo=photo.rotate(270, expand=True)
        position="portrait"
    elif orientation == 8:
        photo=photo.rotate(90, expand=True)
        position="portrait"
    photo.save("image"+str(i)+".jpg")

    #Format de l'icône
    if position=="paysage":
        formatIcon = (86,48)
    else:
        formatIcon = (48,86)

    ##############################################################
    #Récupération de la localisation GPS
    gps = info[34853]
    #gps est à nouveau un dictionnaire
    lat_dir = gps[1] #N ou S pour North ou South
    lat = gps[2]
    lon_dir = gps[3] #W ou E pour West ou East
    lon = gps[4]

    #Conversion décimale de la latitude et de la longitude
    latitude = lat[0][0]/lat[0][1] + (lat[1][0]/lat[1][1])/60 + (lat[2][0]/lat[2][1])/3600
    longitude = lon[0][0]/lon[0][1] + (lon[1][0]/lon[1][1])/60 + (lon[2][0]/lon[2][1])/3600
    #On change le signe de la longitude si besoin
    if lon_dir == 'W':
        longitude = -1*longitude

    ##############################################################
    #récupération de la date et heure de prise de vue
    #récupération du temps
    date = info[36867]
    annee, mois, jourHeure, minute, seconde = date.split(':')
    jour, heure = jourHeure.split(' ')


    #on ferme le fichier photo ouvert
    photo.close()

    #On renvoie les données
    return ([latitude,longitude],formatIcon,heure,minute,jour,mois,annee)



##########################################################################
### récupération de toutes les informations pour toutes les photos ###
##########################################################################
nom = input("Quel est votre nom ? ")
prenom = input("Quel est votre prénom ? ")
nbPhotos = int(input("Combien de photos avez-vous ? "))


#pour le calcul du centre de la carte
#on va calculer les coordonnées du point moyen
xMoy =0
yMoy =0

#pour stocker toutes les coord GPS de toutes les photos
ptPhotos = []

#pour stocker toutes les formats pour les icones
forIcones = []

#pour stocker les textes concernant la date de prise de vue
textDates = []
```

```
 93
 94 for i in range(nbPhotos):
 95     #ouverture de la photo
 96     photo = Image.open("chouchou"+str(i)+".jpg")
 97
 98     #récupération des données
 99     info_photo = coordGPS(photo,i)
100
101     #coord GPS
102     ptPhotos.append(info_photo[0])
103     #pour le calcul du point moyen
104     xMoy = xMoy + info_photo[0][0]
105     yMoy = yMoy + info_photo[0][1]
106
107     #format des icones
108     forIcones.append(info_photo[1])
109
110     #textes sur les dates de prise de vue
111     heure = str(info_photo[2])
112     minute = str(info_photo[3])
113     jour = str(info_photo[4])
114     mois = str(info_photo[5])
115     annee = str(info_photo[6])
116     textDates.append(nom+" "+prenom+"\n à : "
117                      +heure+"h "+minute+"min\n le"
118                      +jour+"/"+mois+"/"+annee)
119
120 #finalisation du calcul du point moyen
121 xMoy = xMoy/nbPhotos
122 yMoy = yMoy/nbPhotos
123
124
125
126 ##############################################################################
127 ### affichage de toutes les données sur une carte openStreetMap #####
128 ##############################################################################
129 #création de la carte centrée
130 carte= folium.Map(
131     #carte centrée sur le point moyen
132     location=([xMoy,yMoy]),
133     #zoom de 16
134     zoom_start=16)
135
136
137 #on rajoute des infos sur la carte
138
139 #placement de tous les marqueurs avec les photos/icons
140 for i in range(nbPhotos):
141     folium.Marker(
142         #coordonnées GPS de la photo
143         ptPhotos[i],
144         #texte sur la date dans le popup
145         popup = textDates[i],
146         #création d'un îcone avec la photo
147         icon = folium.CustomIcon("image"+str(i)+".jpg", icon_size=forIcones[i])
148         ).add_to(carte)
149
150 #pas du tout obligatoire mais juste pour tracer un icone
151 #création marqueur pour les infos du temps de parcours
152 folium.Marker(
153     [xMoy,yMoy], #ccordonnées GPS
154     popup="point moyen",
155     icon=folium.Icon(color='red', icon='info-sign')
156     ).add_to(carte)
157
158 #ajout du parcours à vol d'oiseau
159 folium.PolyLine(
160         ptPhotos,
161         color="red",
162         weight=2.5,
163         opacity=1
164         ).add_to(carte)
165
166
167 ##############################################################################"
168 carte.save('carte_chouchou.html')
```


point
moyen
Leaflet | Data by © OpenStreetMap, under ODbL.