Dictionnaires

Dans le Programme Officiel :

Contenus	Capacités attendues	Commentaires
Dictionnaires par clés et valeurs	Construire une entrée de dictionnaire. Itérer sur les éléments d'un dictionnaire.	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* ().

Contexte pédagogique :
Les élèves sont déjà familiers des listes (https://github.com/glassus/nsi/blob/master/Premiere/Theme02_Types_construits/01_Listes/Listes.ipynb) et des tuples (https://github.com/glassus/nsi/blob/master/Premiere/Theme02_Types_construits/02_Tuples/Tuples.ipynb).

Le choix pédagogique est de présenter la structure par des exemples pour initier le dialogue, avant de la définir.

Premiers exemples

In [1]:
```
dressing = {"pantalons":3, "pulls":4, "tee-shirts":8}
```

In [2]:
```
dressing["pulls"]
```

Out[2]:
```
4
```

In [3]:
```
vocabulaire = {"navigateur":"browser", "précédent":"back", "suivant":"forwar
d"}
```

In [4]:
```
vocabulaire["suivant"]
```

Out[4]:
```
'forward'
```

remarque orale : un dictionnnaire ne sert pas qu'à compter des objets.

In [5]:
```
AlanTuring = {"naissance":(23,6,1912),"décès":(12,6,1954),"lieu naissance":"
Londres", "lieu décès":"Wilmslow"}
```

In [6]:
```
AlanTuring["décès"]
```

Out[6]:
```
(12, 6, 1954)
```

remarque orale : un dictionnaire peut contenir des objets de types différents (ici, des tuples et des chaines de caractères)

Définition d'un dictionnaire

Un dictionnaire est une donnée composite qui **n'est pas ordonnée**.
Il fonctionne par un système de `clé : valeur` . (en anglais `key : value`)
Chaque paire de `key : value` est appelé un `item` .
Les clés, comme les valeurs, peuvent être de types différents. Chaque clé est **unique**.

Un dictionnaire est délimité par des accolades.
Rappel :

- crochets [ ] -> listes
- parenthèses ( ) -> tuples
- accolades { } -> dictionnaires

```
In [7]: vocabulaire
```

```
Out[7]: {'navigateur': 'browser', 'précédent': 'back', 'suivant': 'forward'}
```

```
In [8]: type(vocabulaire)
```

```
Out[8]: dict
```

Il est possible d'obtenir la liste des clés et des valeurs avec la méthode `keys()` et la méthode `values` .

```
In [9]: dressing.keys()
```

```
Out[9]: dict_keys(['pantalons', 'pulls', 'tee-shirts'])
```

```
In [10]: vocabulaire.values()
```

```
Out[10]: dict_values(['browser', 'back', 'forward'])
```

```
In [11]: vocabulaire.items()
```

```
Out[11]: dict_items([('navigateur', 'browser'), ('précédent', 'back'), ('suivant', 'fo
         rward')])
```

Création d'un dictionnaire vide

On crée un dictionnaire vide par l'instruction :

```
In [12]: monDico = dict()
```

```
In [13]: type(monDico)
```

```
Out[13]: dict
```

ou plus simplement de cette manière :

```
In [14]: unAutreDico = {}
```

In [15]:
```
type(unAutreDico)
```

Out[15]:
```
dict
```

Ajout / Modification d'un élément dans un dictionnaire

Pas besoin d'une méthode `append()` comme pour les listes, il suffit de rajouter une paire `clé : valeur`

In [16]:
```
dressing["chaussettes"] = 12
```

In [17]:
```
dressing
```

Out[17]:
```
{'pantalons': 3, 'pulls': 4, 'tee-shirts': 8, 'chaussettes': 12}
```

On peut évidemment modifier un dictionnaire existant (ce n'est pas un tuple !)

In [18]:
```
dressing["chaussettes"] = 11
```

In [19]:
```
dressing
```

Out[19]:
```
{'pantalons': 3, 'pulls': 4, 'tee-shirts': 8, 'chaussettes': 11}
```

Suppression d'une clé

On utilise l'instruction `del` (déjà rencontrée pour les listes)

In [20]:
```
del dressing["chaussettes"]
```

In [21]:
```
dressing
```

Out[21]:
```
{'pantalons': 3, 'pulls': 4, 'tee-shirts': 8}
```

Exercice :

Créer une fonction `achat(habit)` qui augmente de 1 le nombre d'habits (pantalon, pull ou tee-shirt) de mon dressing.

In [22]:
```
#correction possible
def achat(habit):
    dressing[habit] = dressing[habit] + 1
```

Utilisation :

In [23]:
```
print(dressing)
achat("pantalons")
print(dressing)
```

```
{'pantalons': 3, 'pulls': 4, 'tee-shirts': 8}
{'pantalons': 4, 'pulls': 4, 'tee-shirts': 8}
```

Test d'appartenance à un dictionnaire

Le mot `in` permet de tester l'appartenance d'une clé à un dictionnaire. Un booléen est renvoyé.

In [24]:
```
"cravates" in dressing
```

Out[24]: False

In [25]:
```
"pulls" in dressing
```

Out[25]: True

Parcours des éléments d'un dictionnaire

itération sur les clés

In [26]:
```
for cle in dressing.keys():
    print(cle)
```

```
pantalons
pulls
tee-shirts
```

itération sur les valeurs

In [27]:
```
for valeur in dressing.values():
    print(valeur)
```

```
4
4
8
```

itération sur les couples (clés, valeurs)

In [34]:
```
for (cle, valeur) in dressing.items():
    print("dans ma penderie il y a", valeur, cle)
```

```
dans ma penderie il y a 4 pantalons
dans ma penderie il y a 4 pulls
dans ma penderie il y a 8 tee-shirts
```

Application 1 : lecture et manipulation des métadonnées EXIF d'une image

On considère l'image liberty.jpg (./liberty.jpg) dont nous allons extraire les données EXIF à l'aide du module exifread (https://pypi.org/project/ExifRead/)

In [46]:
```
import exifread
f = open("liberty.jpg", 'rb')
data = exifread.process_file(f)
```

Question 1 Observer ce que contient la variable `data` . De quel type est-elle ?

In [47]:
```
#solution
data
```

Out[47]:
```
{'Image ImageWidth': (0x0100) Long=3264 @ 18,
 'Image ImageLength': (0x0101) Long=1836 @ 30,
 'Image Make': (0x010F) ASCII=SAMSUNG @ 158,
 'Image Model': (0x0110) ASCII=GT-I9195 @ 166,
 'Image Orientation': (0x0112) Short=Horizontal (normal) @ 66,
 'Image XResolution': (0x011A) Ratio=72 @ 176,
 'Image YResolution': (0x011B) Ratio=72 @ 184,
 'Image ResolutionUnit': (0x0128) Short=Pixels/Inch @ 102,
 'Image Software': (0x0131) ASCII=Shotwell 0.22.0 @ 192,
 'Image YCbCrPositioning': (0x0213) Short=Centered @ 126,
 'Image ExifOffset': (0x8769) Long=208 @ 138,
 'GPS GPSVersionID': (0x0000) Byte=[2, 2, 0, 0] @ 830,
 'GPS GPSLatitudeRef': (0x0001) ASCII=N @ 842,
 'GPS GPSLatitude': (0x0002) Ratio=[44, 51, 24163/1250] @ 934,
 'GPS GPSLongitudeRef': (0x0003) ASCII=W @ 866,
 'GPS GPSLongitude': (0x0004) Ratio=[0, 34, 169537/10000] @ 958,
 'GPS GPSAltitudeRef': (0x0005) Byte=0 @ 890,
 'GPS GPSAltitude': (0x0006) Ratio=58 @ 982,
 'GPS GPSTimeStamp': (0x0007) Ratio=[12, 28, 10] @ 990,
 'GPS GPSDate': (0x001D) ASCII=2019:03:28 @ 1014,
 'Image GPSInfo': (0x8825) Long=820 @ 150,
 'EXIF ExposureTime': (0x829A) Ratio=1/1093 @ 586,
 'EXIF FNumber': (0x829D) Ratio=13/5 @ 594,
 'EXIF ExposureProgram': (0x8822) Short=Aperture Priority @ 242,
 'EXIF ISOSpeedRatings': (0x8827) Short=50 @ 254,
 'EXIF ExifVersion': (0x9000) Undefined=0220 @ 266,
 'EXIF ComponentsConfiguration': (0x9101) Undefined=YCbCr @ 278,
 'EXIF ShutterSpeedValue': (0x9201) Signed Ratio=1/1093 @ 602,
 'EXIF ApertureValue': (0x9202) Ratio=69/25 @ 610,
 'EXIF BrightnessValue': (0x9203) Signed Ratio=76 @ 618,
 'EXIF ExposureBiasValue': (0x9204) Signed Ratio=0 @ 626,
 'EXIF MaxApertureValue': (0x9205) Ratio=69/25 @ 634,
 'EXIF MeteringMode': (0x9207) Short=CenterWeightedAverage @ 350,
 'EXIF LightSource': (0x9208) Short=Unknown @ 362,
 'EXIF Flash': (0x9209) Short=Flash did not fire @ 374,
 'EXIF FocalLength': (0x920A) Ratio=37/10 @ 642,
 'EXIF MakerNote': (0x927C) Undefined=[7, 0, 1, 0, 7, 0, 4, 0, 0, 0, 48, 49,
48, 48, 2, 0, 4, 0, 1, 0, ... ] @ 650,
 'EXIF UserComment': (0x9286) Undefined=User comments @ 748,
 'EXIF FlashPixVersion': (0xA000) Undefined=0100 @ 422,
 'EXIF ColorSpace': (0xA001) Short=sRGB @ 434,
 'EXIF ExifImageWidth': (0xA002) Signed Long=2524 @ 446,
 'EXIF ExifImageLength': (0xA003) Signed Long=1662 @ 458,
 'Interoperability InteroperabilityIndex': (0x0001) ASCII=R98 @ 800,
 'Interoperability InteroperabilityVersion': (0x0002) Undefined=[48, 49, 48,
48] @ 812,
 'EXIF InteroperabilityOffset': (0xA005) Long=790 @ 470,
 'EXIF SensingMethod': (0xA217) Short=One-chip color area @ 482,
 'EXIF SceneType': (0xA301) Short=Directly Photographed @ 494,
 'EXIF ExposureMode': (0xA402) Short=Auto Exposure @ 506,
 'EXIF WhiteBalance': (0xA403) Short=Auto @ 518,
 'EXIF DigitalZoomRatio': (0xA404) Ratio=3 @ 770,
 'EXIF SceneCaptureType': (0xA406) Short=Standard @ 542,
 'EXIF Saturation': (0xA409) Short=Normal @ 554,
 'EXIF Sharpness': (0xA40A) Short=Normal @ 566,
 'EXIF ImageUniqueID': (0xA420) ASCII=S08Q0LEGC01 @ 778}
```

Réponse : `data` est un dictionnaire.

Question 2 Afficher la valeur correspondant à la clé `'Image Make'` .

In [48]:
```
#solution
data['Image Make']
```

Out[48]:
```
(0x010F) ASCII=SAMSUNG @ 158
```

Question 3 Modifier cette valeur pour que l'appareil apparaisse comme étant fabriqué par Apple.

In [51]:
```
#solution
data['Image Make']="(0x010F) ASCII=APPLE @ 158"
```

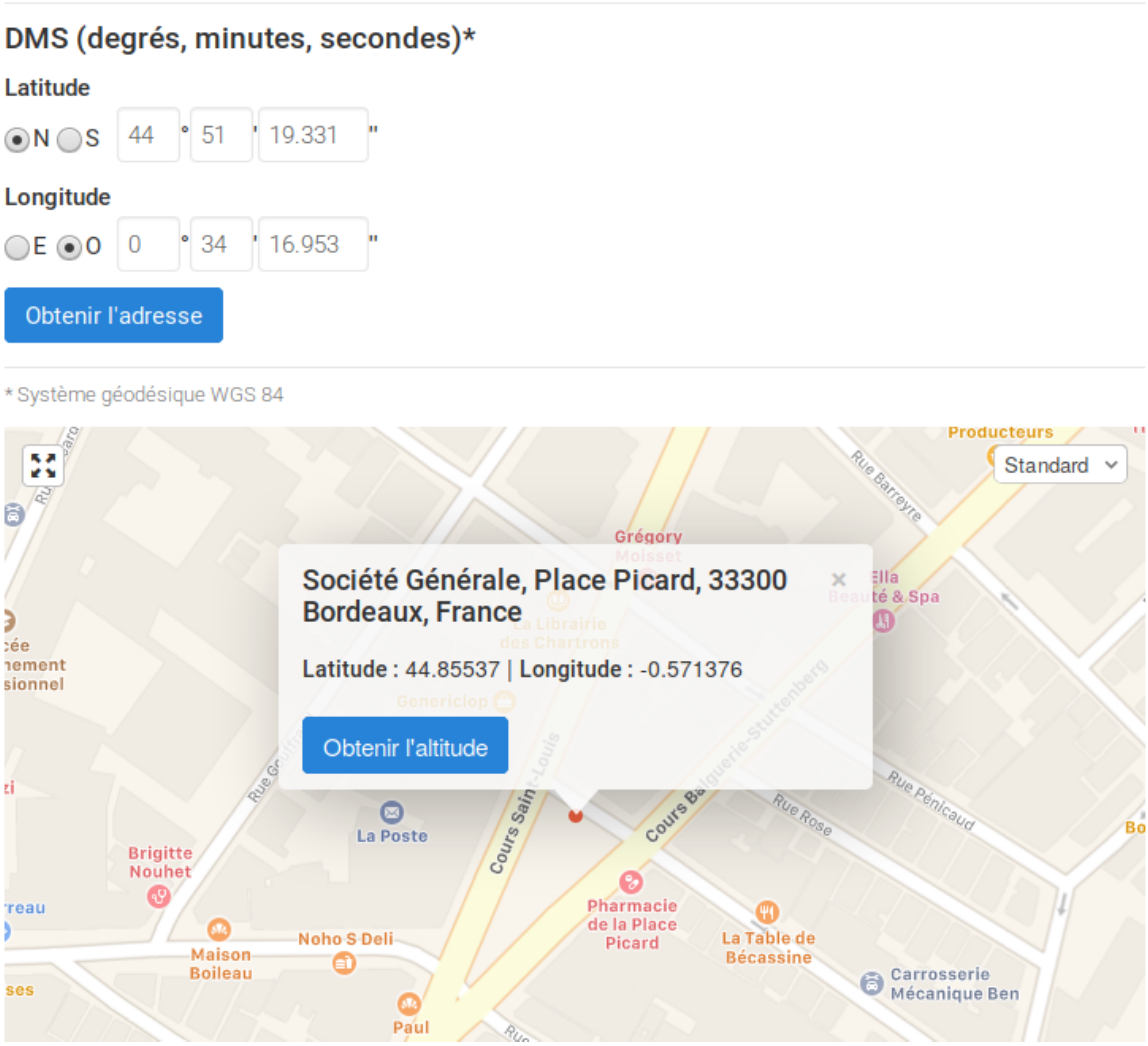
Question 4 Repérer dans le dictionnaire les renseignements relatifs à la latitude et la longitude enregistrées dans les métadonnées de cette image.
Attention, les données sont au format Degré Minutes Secondes. La dernière valeur donnée est donnée sous forme de quotient, il faut le calculer ! Vous pourrez utiliser le site https://www.coordonnees-gps.fr/ (https://www.coordonnees-gps.fr/)

In [56]:
```
#réponse :
print(data['GPS GPSLatitudeRef'])
print(data['GPS GPSLatitude'])
print(data['GPS GPSLongitudeRef'])
print(data['GPS GPSLongitude'])
```

```
N
[44, 51, 24163/1250]
W
[0, 34, 169537/10000]
```

En DMS, la latitude est 44°51'19.3304" Nord.
la longitude est 0°34'16.9537" Ouest.

Ce qui donne :



En effet, cette photographie a été prise Place Picard, à Bordeaux, où se trouve une Statue de la Liberté.

Discussion avec les élèves sur le crédit à appporter aux données EXIF, car on vient de voir qu'elles étaient modifiables !

Application 2 : résolution du pydéfi "le seul et unique"

D'après le défi (https://callicode.fr/pydefis/PokePlusRare/txt) proposé par L.Signac .

Le seul et unique

N'en attrapez qu'un seul !

Vous avez réussi à mettre la main sur la liste des positions de tous les Pokémons du globe. Cette liste, relativement longue, précise pour chaque lieu le nom du Pokémon présent, suivi de ses coordonnées (deux nombres entiers séparés par une virgule)

Afin d'asseoir votre réputation de chasseur, vous vous êtes mis en tête de capturer le Pokémon unique, qui n'est représenté qu'une seule fois dans la liste.

Pour valider le défi, indiquez les coordonnées (deux nombres entiers séparés par une virgule) auxquelles vous pourrez le capturer.

Entrez par exemple : `-30, 67` si le Pokémon unique a ces coordonnées.

Ce problème est tiré de c0d1ng UP 2017

Travail préalable

Nous allons travailler avec les quelques données suivantes, présentées sous forme d'une liste de listes :

In [29]:
```
L = [['givrali', '75', '46'],
 ['branette', '35', '153'],
 ['kyogre', '23', '-10'],
 ['roserade', '-87', '91'],
 ['balignon', '44', '-155'],
 ['keunotor', '32', '163'],
 ['givrali', '-22', '124'],
 ['kyogre', '-54', '-26'],
 ['pyronille', '11', '-102']]
```

Question 1

Créer un dictionnaire `pokemon` dont la clé sera le nom du Pokémon, et la valeur le tuple de ses coordonnées. Si le Pokémon est en double dans la liste, sa valeur sera alors la chaîne de caractère `"doublon"` .

In [30]:
```
#solution possible

pokemon = {}

for personnage in L :
    nom = personnage[0]
    if nom in pokemon :
        pokemon[nom] = "doublon"
    else :
        pokemon[nom] = (personnage[1],personnage[2])
```

In [31]:
```
pokemon
```

Out[31]:
```
{'givrali': 'doublon',
 'branette': ('35', '153'),
 'kyogre': 'doublon',
 'roserade': ('-87', '91'),
 'balignon': ('44', '-155'),
 'keunotor': ('32', '163'),
 'pyronille': ('11', '-102')}
```

Question 2

Les données d'entrée du pydéfi ont été enregistrées dans le fichier input_defi.txt (./input_defi.txt). Les lignes suivantes permettent de parcourir chacun des personnages de la liste.

In [38]:
```
#code donné aux élèves
fich = open("input_defi.txt", "r")
for ligne in fich.readlines():
    personnage = ligne[:-1] #on enlève le caractère de retour à la ligne \n
    personnage = personnage.split(",") # on sépare les trois éléments de per
sonnage, qui deviennent une liste
```

In [33]:
```
personnage
```

Out[33]:
```
['tauros', '-32', '-114']
```

Tous les personnages ont été parcourus : seul le dernier ("tauros" s'affiche). Grâce au code ci-dessous et à la question 1, créer de la même manière qu'à la question 1 un dictionnaire qui contiendra tous les pokemons.

In [35]:
```
#solution possible
pokemon = {}
fich = open("input_defi.txt", "r")
for ligne in fich.readlines():
    personnage = ligne[:-1]
    personnage = personnage.split(",")
    nom = personnage[0]
    if nom in pokemon :
        pokemon[nom] = "doublon"
    else :
        pokemon[nom] = (personnage[1],personnage[2])
```

Question 3

Répondre au défi : quelles sont les coordonnées du seul pokémon unique de cette liste ?

In [37]:
```
# solution possible 1 : en regardant l'intégralité du dictionnaire pokemon
(pas si grand...)

# solution possible 2 :

for (cle, valeur) in pokemon.items() :
    if valeur != "doublon" :
        print(valeur)
```

```
('74', '-73')
```