# DST Informatique
# Représentation de données géolocalisées sur une carte numérique
# Durée : 2 heures

**Consignes**
*Lire les rappels sur* `Python` *donnés en annexe.*
*Ecrire lisiblement en faisant attention aux parenthèses, aux deux-points, aux virgules, aux guillemets, à l'indentation, à l'écriture des mots Python avec ou sans majuscule* (`def`, `True`, ...).
*Chercher à optimiser l'écriture des fonctions en réutilisant les fonctions précédentes. Ce critère sera pris en compte dans la notation.*

**Présentation du thème du sujet**
Les fichiers contenant des photos possèdent des informations sur celles-ci, elles sont communément appelées données `exif`. Entre autres, on y trouve :

— la date où elle a été prise ;

— les coordonnées GPS de l'endroit où elle a été prise.

Le but de ce sujet est de faire un script qui va placer géographiquement sur une carte numérique les noms de fichiers photos archivés.

**Dans tout le sujet, on supposera les modules Python importés.**

## 1 Géolocalisation de photos

**Question 1**
Qu'affiche le script ci-dessous ?

```
x = "2020:06:12 21:12:40"
y = "2020:06:12 21:32:40"
print(x > y)
```

**Solution**
```
False
```

Lors des questions 2, 3 et 4, les chaînes de caractères contiennent une fois et une seule le caractère `"."` .
*Exemple* : `mot1 = "chat.jpg"`, `mot2 = "chien.jpeg"`, `mot3 = "fichier.ipynb"`

Les chaînes de caractères qui commencent par `"."` sont exclues.
*Exemple* : `".ipynb"`, `".jpg"` sont exclues.

**Question 2**
Écrire une fonction nommée `placeDuPoint` qui a pour paramètre une chaîne de caractères `ch` et qui renvoie l'indice du point dans cette chaîne de caractères.
*Exemple* : dans `"devoir.ipynb"` le point a pour indice 6.

**Solution**
```
def placeDuPoint(ch):
    for k in range(len(ch)):
```

```
        if ch[k] == ".":
            return k
```

**Question 3**
Écrire une fonction `coupeExtension` qui a pour paramètre une chaîne de caractères `ch` et qui renvoie la chaîne de caractères précédant le caractère `"."`.
*Exemple* : la fonction appliquée à `"devoir.ipynb"` renvoie `"devoir"`.

**Solution**
```
def coupeExtension(ch):
    i = placeDuPoint(ch)
    return ch[:i]
```

**Question 4**
Écrire une fonction `jpgToHtml` qui a pour paramètre une chaîne de caractères `ch` représentant le nom d'un fichier avec son extension et qui renvoie le nom avec l'extension `.html`.
*Exemple* : la fonction appliquée à `"photo.jpeg"` renvoie `"photo.html"`.

**Solution**
```
def jpgToHtml(ch):
    return coupeExtension(ch) + ".html"
```

## 2 Lecture de données `exif`

Dans cette partie, il est important de lire attentivement la partie **module exif** de l'**annexe 2**.

On rappelle l'utilisation de la fonction `openImage` :
`my_image = openImage("photo.jpg")`

La variable `my_image` permet alors d'accéder aux données `exif` et on fera référence à cette variable dans toute la suite du sujet.

La question suivante se réfère à la partie **Coordonnées GPS - Règles de Conversion** de l'**annexe 2**.

**Question 5**
Écrire une fonction `convertToDecimale` qui a pour paramètre un tuple `angle` de trois flottants `(degres, minutes, secondes)` et qui renvoie le flottant correspondant.

**Solution**
```
def convertToDecimale(angle):
    degre, minutes, secondes = angle
    return degre + 1 / 60 * minutes +  1 / 3600 * secondes
```

On définit un type `Point` comme étant un tuple `(latitude, longitude)` de deux flottants signés.

**Question 6**
Écrire une fonction `imageToGpsPoint` qui a pour paramètre une variable `my_image` et qui renvoie une variable de type `Point`.

**Solution**

```
def imageToGpsPoint(my_image):
    angle = my_image.gps_latitude
    if my_image.gps_latitude_ref == "N":
        lat = convertToDecimale(angle)
    else:
        lat = - convertToDecimale(angle)
    angle = my_image.gps_longitude
    if my_image.gps_longitude_ref == "E":
        long = convertToDecimale(angle)
    else:
        long = -convertToDecimale(angle)
    return (lat, long)
```

**Question 7**
Écrire une fonction `listeAvantMidi` qui a pour paramètre une liste `lst` de photos (les photos sont représentées par leur nom qui est une chaîne de caractères) et qui renvoie la liste des photos prises avant midi.

**Solution**

```
def listeAvantMidi(lst):
    liste = []
    for photo in lst:
        im = openImage(photo)
        date = im.datetime
        if date[11:13] < "12":
            liste.append(photo)
    return liste
```

# 3 Module Folium

**Présentation du module folium**
Python possède un module nommé **folium** qui facilite la visualisation des données.
Les données que nous allons manipuler sont des photos ayant des données `exif`, plus particulièrement des coordonnées GPS, puis nous allons mettre en place le script qui permet de placer leurs noms sur une carte numérique. On précise que l'on peut ajouter le script pour les visualiser, mais cela nécessite du code **Html**, ce qui n'est pas abordé dans ce sujet.
Chaque nom de photo sera positionné sur cette carte grâce à ses coordonnées GPS. Un marqueur (nommé **marker** dans le module **folium**) permet de repérer cette photo positionnée sur la carte et une option **popup** permet de nommer ce marqueur. La carte peut être centrée sur des coordonnées particulières et peut être plus ou moins « zoomée ».

Par commodité et pour permettre une lecture fluide du document, nous utiliserons les anglicismes suivants :

- **marker** pour tout marqueur (ou symbole) placé sur la carte numérique à certaines coordonnées GPS ;
- **popup** : option qui permet de donner un nom au **marker** par le biais d'une chaîne de caractères.

Pour la création d'une carte numérique, on procède comme ci-dessous :

```
import folium #importation du module
location = (35.9279, -114.9721) #location est de type Point
#ce tuple représente le couple (latitude, longitude)
#Pour créer une carte vide centrée sur location:
carteTest = folium.Map(location, zoom_start = 20) # l'option de zoom est ici à 20 arbitrairement
#Pour ajouter un marker à la carte créée ici nommée carteTest
#Ce marker est placé aux coordonnées location:
folium.Marker(location, popup = 'Las Vegas').add_to(carteTest)
#popup donne un nom au marker, ici 'Las Vegas', sur la carte
#Pour créer et sauvegarder la carte au format html
carteTest.save('MaCarte.html') #Le nom de sauvegarde est ici Macarte.html
```

Dans tout ce qui suit, le module **folium** sera supposé importé.

On rappelle que chaque carte doit être centrée sur un point dont les coordonnées GPS sont connues.

**Question 8**
Créer une fonction nommée `carteVide` qui a pour paramètre une donnée de type Point nommée `centre` et qui renvoie une carte vide créée avec un zoom de 20, et qui doit être centrée sur `centre`.

**Solution**

```
def carteVide(centre):
    return folium.Map(location = centre, zoom_start = 20)
```

**Question 9**
Écrire le script de la fonction `transfertTocarte`, qui a pour paramètres une carte vide appelée `carte`, un objet image de type `my_image` appelé `im` et une chaîne de caractères (le `popup` à placer) appelée `popupPasse` et qui ajoute à la carte le `marker` aux coordonnées du `Point` et le `popup`. Cette fonction ne renvoie rien.

**Solution**

```
def transfertTocarte(carte, im, popupPasse):
    folium.Marker(imageToGpsPoint(im), popup = popupPasse).add_to(carte)
```

L'exemple de code ci-dessous permet d'ajouter plusieurs markers (ayant des noms et/ou des coordonnées GPS différents) à une carte :

```
centre = (46.87, 4.00261)
c = folium.Map(location=centre, zoom_start=20)
folium.Marker((46.877, 4.00261), popup='LosAngeles1').add_to(c)
folium.Marker((46.87, 4.003), popup='LosAngeles2').add_to(c)
folium.Marker((46.88, 4.003), popup='LosAngeles3').add_to(c)
c.save('maCarte.html') #sauvegarde de la carte
#le nom doit avoir l'extension .html pour obtenir une page web lisible
```

**Question 10**
Écrire une fonction `transfertListeTocarte`, qui prendra en paramètres d'entrée une donnée de

type `Point` appelé `centre`, une liste de photos appelée `listePhotos`. Cette fonction crée une carte centrée sur `centre` et y ajoute chaque nom de la liste `listePhotos`. Le script doit répondre en plus aux exigences suivantes :

— chaque `popup` aura pour `nom` le nom ajouté ;

— la fonction renverra la carte créée.

**Solution**

```
def transfertListeTocarte(centre, listePhotos):
    carte = carteVide(centre)
    for elt in listePhotos:
        im = openImage(elt)
        transfertTocarte(carte, im, coupeExtension(elt))
    return carte
```

**Question 11**

Écrire le code Python qui, à partir d'une liste nommée `listePhotos`, crée une carte sauvée au nom de `"Las Vegas"` et qui ajoute tous les noms par le biais de `markers`. Cette carte sera centrée sur le premier terme de la liste.

*Exemple de liste de photos :*

```
listePhotos = ["Venitian.jpg", "Bellagio.jpeg", "Palazzo.png"]
```

**Solution**

```
image_centre = openImage(listePhotos[0])
centre = imageToGpsPoint(image_centre)
carte = transfertListeTocarte(centre,listePhotos)
carte.save("Las Vegas.html")
```

## 3.1 Coordonnées GPS

On rappelle qu'un objet de type `Point` est un tuple de deux nombres décimaux représentant une `latitude` et une `longitude` dans cet ordre. Ainsi, `gps = (35.9279, -114.9721)` est le tuple `(latitude, longitude)` des coordonnées GPS de Las Vegas en flottants signés.

On donne deux `Point` appelés `inf` et `sup` et on suppose que le segment formé par ces deux `Point` est la diagonale d'un rectangle non aplati où le coin inférieur gauche est `inf` et le coin supérieur droit est `sup`.
On ne souhaite positionner que les noms des photos situées dans ce rectangle.
Dans cette sous-partie, nous parlerons d'une liste de chaînes de caractères qui représentent des noms de photos (type `"photo01.jpg"`) et ayant des données `exif`.

**Question 12**

Écrire une fonction `testRectangle` qui a deux paramètres d'entrée de type `Point` nommés `point1`, `point2` et qui renvoie le booléen :

— `True` si `point1`, `point2` forment un rectangle non aplati, dont `point1` sera le coin inférieur gauche et `point2` le coin supérieur droit ;

— `False` sinon.

**Solution**

```
def testRectangle(point1, point2):
    return point1[0] < point2[0] and point1[1] < point2[1]
```

**Question 13**
Écrire une fonction `milieu` qui a deux paramètres d'entrée de type `Point` nommés `point1`, `point2` et qui renvoie un `Point` qui est le milieu du segment d'extrémités `point1` et `point2`.

**Solution**

```
def milieu(point1, point2):
    return ((point1[0] + point2[0]) / 2, (point1[1] + point2[1]) / 2)
```

On suppose que `inf`, `sup` sont deux objets de type `Point` qui forment un rectangle non aplati.

**Question 14**
Écrire une fonction `intRectangle` qui a trois paramètres d'entrée de type `Point` nommés `point`, `inf`, `sup` et qui renvoie un booléen. Ce booléen sera `True` si `point` est dans le rectangle formé par `inf`, `sup` et `False` sinon. On précise que le bord du rectangle est exclu.

**Solution**

```
def intRectangle(point, inf, sup):
    return inf[0] < point[0] < sup[0] and inf[1] < point[1] < sup[1]
```

## 3.2 Ajout de markers à une carte numérique

On reprend la notion de rectangle présentée dans la question 14.
On dispose d'une liste de noms de photos, les markers seront des noms de photos.
Parmi ces photos on souhaite sélectionner uniquement celles dont les coordonnées GPS sont incluses strictement dans le rectangle délimité par `inf` et `sup`, et positionner leur nom sur une carte numérique.

On rappelle que pour sauvegarder une carte (de nom `"nomDeLaCarte"`), il suffit d'utiliser le code :
`carte.save("nomDeLaCarte" + ".html")`

L'extension `".html"` permet de rendre la carte interprétable par un moteur de recherche.
Le code Python pour l'affichage dans une page web sera alors :
`webbrowser.open("nomDeLaCarte.html")`

**Question 15**
Écrire la fonction `listeTomap` dont les paramètres d'entrée sont :

— une liste de photos au format chaîne de caractères (exemple : `"Bellagio.jpeg"`), nommée `listePhotos` ;

— `inf` et `sup` de type `Point` ;

— `titre` qui sera le nom de la carte sous forme de chaîne de caractères (sans extension).

Cette fonction crée une carte dont le nom est `titre`, et y positionne uniquement le nom des photos sélectionnées. La carte sera centrée sur le milieu du segment d'extrémités `inf` et `sup`.
Cette fonction renvoie `False` si le nombre d'éléments ajoutés à la carte est nul, sinon elle sauvegarde la carte et la renvoie.

**Solution**

```
def listeTomap(listePhotos, inf, sup, titre):
    lst_r = []
    for photo in listePhotos:
        image = openImage(photo)
        location = imageToGpsPoint(image)
        if intRectangle(location, inf, sup):
            lst_r.append(photo)
    if len(lst_r) == 0:
        return False
    else:
        centre = milieu(inf, sup)
        carte = transfertListeTocarte(centre, lst_r)
        carte.save(titre + ".html")
        return carte
```

# 4 Recherche de photos dans un dossier

On suppose que les photos ont été stockées dans un dossier de sauvegarde. Mais avec le temps, ce dossier s'est rempli. Maintenant, il contient des photos mais aussi des dossiers de photos.
Le module `os` permet de gérer ces dossiers et nous allons utiliser en particulier la fonction `os.listdir` qui prend en paramètre une chaîne de caractères (le nom d'un dossier) et qui renvoie la liste de noms de fichiers avec leur extension ou de dossiers.

Par exemple : l'appel de la fonction `os.listdir("Photos")` renvoie la liste des noms des fichiers et des sous-dossiers du dossier appelé ici `Photos`.

**Question 16**
Écrire une fonction `estUnRep` qui a pour paramètre une chaîne de caractères `ch` (qui est soit un nom de fichier contenant donc un point, soit un dossier) et qui renvoie `True` si `ch` est un répertoire et `False` sinon.

**Solution**

```
def estUnRep(ch):
    return "." not in ch
```

**Question 17**
Écrire une fonction `queLesPhotos` qui a pour paramètre une liste de chaînes de caractères nommée `lst` contenant des photos et des répertoires et qui renvoie une nouvelle liste contenant uniquement les photos. On ne modifiera pas `lst` mais on créera une nouvelle liste. On ne regardera pas si les sous-répertoires contiennent aussi des photos.

**Solution**

```
def queLesPhotos(lst):
    listeP = []
    for elt in lst:
        if not estUnRep(elt):
            listeP.append(elt)
    return listeP
```

**Question 18**
Écrire une fonction `listeDesPhotos` qui a pour paramètre une chaîne de caractères nommée `rep` représentant un répertoire et qui renvoie la liste de toutes les photos contenus dans ce répertoire. On ne regardera pas si les sous-répertoires contiennent aussi des photos.

**Solution**

```
def listeDesPhotos(rep):
    return queLesPhotos(os.listdir(rep))
```

**Question 19**
Dans cette question, on souhaite lister toutes les photos d'un répertoire, y compris celles des sous-répertoires de celui-ci.
Ecrire une fonction `toutesLesPhotos` qui a pour paramètre une chaîne de caractères nommée `rep` représentant un répertoire et qui renvoie la liste de toutes les photos qu'il y a dans `rep` et dans tous ses sous-répertoires.
On pourra écrire plusieurs fonctions pour rendre le code lisible.

**Solution**

```
def queLesRep(lst):
    listeR = []
    for elt in lst:
        if estUnRep(elt):
            listeR.append(elt)
    return listeR

def contientUnRep(lst):
    for elt in lst:
        if estUnRep(elt):
            return True
    return False

def toutesLesPhotos(rep):
    lst = os.listdir(rep)
    listeP = []
```

```
    while(contientUnRep(lst)):
        listeP += queLesPhotos(lst)
        listeR = queLesRep(lst)
        lst = []
        for ssrep in listeR:
            lst += os.listdir(ssrep)
    listeP += lst
    return listeP
```

# Annexe 1 : Rappels sur Python

## Rappels sur les chaînes de caractères

```
ch1 = "bon"
ch2 = "jour"
ch3 = ch1 + ch2 # "bonjour"
print("n" in ch1) # True
for k in range(len(ch1)):
    print(ch1[k])
# b
# o
# n
```

## Rappels sur les tuples

Un `tuple` est la donnée d'un n-uplet non mutable (qui ne peut pas être modifié).

```
t = (donnee1, donnee2)
print(t[0]) # affiche donnee1
print(t[1]) # affiche donnee2
val1, val2 = t
print(val1) #affiche donnee1
print(val2) #affiche donnee2
# Pour parcourir un tuple
for elt in t:
    print(elt)
#affiche donnee1
#affiche donnee2
```

Une fonction peut renvoyer un `tuple` dont on peut en extraire le contenu directement.

*Exemple* :

```
def retourneTuple(x, y):
    return (x+y, x-y)
som, diff = retourneTuple(10, 3)
print(som) # 13
print(diff) # 7
```

## Rappels sur le slicing

```
mot = "Bonjour"
print(mot[ : 3]) # "Bon"
print(mot[3 : ]) # "jour"
```

# Annexe 2 : Présentation des données `exif`

## Module `exif`

Chaque fichier de photo numérique contient des informations appelées données `exif` (`exif` signifie « EXchangeable Image File » ou fichier d'échange de données). Ces informations, créées par l'appareil photo lors de la prise de vue, sont stockées dans le fichier généré par l'appareil.
Voici deux exemples d'informations accessibles :

— Date de la prise de vue.

— Coordonnées GPS du lieu de la prise de vue.

Les données `exif` sont accessibles dans un script Python en important le module `exif`, puis en accédant à `Image` dans ce module. À l'issue du script suivant, la variable `my_image` permet d'accéder aux données `exif` :

```
from exif import Image
with open("photo.jpg", "rb") as imageFile:
    my_image = Image(imageFile)
```

Afin de faciliter le codage, on donne ci-dessous la fonction `openImage` qui sera utilisée lors de ce sujet :

```
def openImage(nom) : #nom est une chaîne de caractères qui représente ici une
↪  photo
    with open(nom, "rb") as imageFile:
        my_image = Image(imageFile)
    return my_image
#exemple d'appel de cette fonction :
    my_image = openImage("photo.jpg")
```

Suite à l'appel de la fonction, la variable `my_image` permet d'accéder aux données `exif`.
La liste des données `exif` qui nous intéressent est :

```
my_image.datetime #une date au format chaîne de caractères
my_image.gps_longitude # un tuple de trois nombres
my_image.gps_longitude_ref # une chaîne de caractères 'E' ou 'W'
my_image.gps_latitude # un tuple de trois nombres
my_image.gps_latitude_ref # une chaîne de caractères 'N' ou 'S'
```

## Coordonnées GPS - Règles de conversion

Le module `folium`, dont on a besoin ici, utilise des coordonnées `gps` au format `tuple` `(latitude, longitude)` où les deux valeurs sont en décimales signées suivant l'orientation Nord-Sud ou Est-Ouest. Les données `gps` issues des données `exif` d'une image sont au format `tuple` `(degres, minutes, secondes)`. Il faut donc procéder à une conversion.
On donne la règle de conversion suivante :

- 1 degré = 1
- 1 minute = 1/60
- 1 seconde = 1/3600

L'orientation pour la latitude est `'N'` ou `'S'` et pour la longitude `'E'` ou `'W'`.
Les valeurs décimales sont signées : négatives si on est `'W'` ou `'S'` et positives sinon.
L'accès à cette orientation est donnée par le code Python ci-dessous :

```
my_image.gps_latitude_ref # donne 'N' ou 'S'
my_image.gps_longitude_ref # donne 'E' ou 'W'
```

*Un exemple* :

```
my_image.gps_latitude # donne par exemple (36,10,11.78)
my_image.gps_longitude # donne par exemple (115,8,23.38)
my_image.gps_latitude_ref # donne par exemple 'N'
my_image.gps_longitude_ref # donne par exemple 'W'
```

Ce qui donne au format décimal avec l'orientation Nord et Ouest : `36.169939,-115.13983`

## Date de prise de vue

La date de prise de vue est une chaîne de caractères de longueur 19, le format est :
`année:mois:jour heure:minute:seconde`

avec un seul espace entre jour et heure

`"1967:06:17 11:15:00"`
`"2020:12:24 23:59:59"`

Pour accéder à l'information de date, il suffit d'écrire le script :

```
var_date=my_image.datetime
#var_date sera une chaîne de caractères qui contient les informations de
 ↪  date

```