teaching sciences for innovation

Proposition de corrigé

Concours : Concours Commun INP

Année : 2022

Filière : TSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

En cas de doute, n'hésitez pas à nous contacter à : corrigesconcours@upsti.fr.

Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

Représentation de données géolocalisées sur une carte numérique

Corrigé UPSTI

Partie I – Géolocalisation de photos

Question 1 Qu'affiche le script suivant ? Vous expliquerez votre résultat.

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

Le code affiche `False` car la date `x` n'est pas ultérieure à la date `y`.

Techniquement, le script compare les chaînes de caractères en comparant l'unicode de chaque caractère. La date étant écrite sous le format `'aaaa:mm:jj hh:mm:ss'`, la comparaison unicode revient à la comparaison temporelle.

Le code aurait renvoyé `True` si la date `x` était strictement ultérieure à la date `y`.

Question 2 Écrire une fonction nommée `placeDuPoint` qui prend en entrée une chaîne de caractères et qui retourne l'indice du point dans cette chaîne de caractères.

Plusieurs solutions sont proposées

```
def placeDuPoint(chaine):
    # solution avec les chaînes de caractères
    return chaine.find(".")
```

```
def placeDuPoint(chaine):
    # solution avec une boucle for
    for i in range(len(chaine)):
        if chaine[i] == ".":
            return i
```

```
def placeDuPoint(s):
    #solution avec une boucle while
    i = 0
    while s[i] != '.':
        i += 1
    return i
```

Question 3 Écrire une fonction `coupeExtension` qui prend en entrée une chaîne de caractères et qui retourne la chaîne de caractères précédant le caractère `'.'`.

On utilise le slicing pour ne garder que le début de la chaîne de caractères jusqu'à l'indice du `'.'`.

```
def coupeExtension(nom_fichier):
    id_point = placeDuPoint(nom_fichier)
    return nom_fichier[:id_point]
```

Question 4 Écrire une fonction `jpgtohtml` qui prend en entrée une chaîne de caractères représentant le nom d'un fichier avec son extension et qui retourne le nom avec l'extension `.html`.

```
def jpgTohtml(nom_fichier):
    nom_sans_extension = coupeExtension(nom_fichier)
    return nom_sans_extension + ".html"
```

Partie II – Lecture de données exif

```python
from exif import Image

def openImage(nom): #nom est une chaîne de caractère qui représente ici une photo
    with open (nom, 'rb') as imageFile:
        return Image(imageFile)
my_image = openImage('photo.jpg')
```

Question 5 Écrire une fonction `convertTodecimale` qui prend en entrée un tuple (degré, minutes, secondes) et retourne le flottant correspondant.

```python
def convertTodecimale(angle_tuple):
    degre, minutes, secondes = angle_tuple
    return degre + 1 / 60 * minutes +  1 / 3600 * secondes
```

Question 6 Écrire une fonction `imageToGpsPoint` qui prend en entrée une variable `my_image` et qui retourne une variable de type `Point`.

```python
def imageToGpsPoint(im):
    latitude = convertTodecimale(im.gps_latitude)
    if im.gps_latitude_ref == 'S':
        # la latitude est négative, on prend l'opposé
        latitude *= -1
    longitude = convertTodecimale(im.gps_longitude)
    if im.gps_longitude_ref == 'W':
        # la longitude est négative, on prend l'opposé
        longitude *= -1
    return (latitude, longitude)
```

Question 7 Écrire une fonction `listeAvantMidi` qui prend en entrée une liste de photos (les photos sont représentées par leur nom qui est une chaîne de caractères) et qui retourne la liste des photos prises avant midi.

```python
def listeAvantMidi(photos):
    avant_midi = []
    for photo in photos:
        my_image = openImage(photo)
        var_date = my_image.datetime
        heure = var_date[11:13] # on ne sélectionne que l'heure
        if heure < "12":
            avant_midi.append(photo)
    return avant_midi
```

Partie III – Module folium

```
import folium
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
```

Question 8 Créer une fonction nommée `carteVide` qui prend en entrée une donnée de type Point nommé `centre` et qui retourne une carte vide créée avec un zoom de 20, et qui doit être centrée sur `centre`.

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

Question 9 Écrire le script de la fonction `transfertTocarte`, qui prend en entrée une carte vide appelée `carte`, un objet image de type `my_image` et une chaîne de caractères (le `popup` à placer) appelée `popupPassé` et qui ajoute à la carte le `marker` aux coordonnées du `Point` et le `popup`. Cette fonction retourne une carte ainsi modifiée.

Les accents ne sont pas autorisés dans le nommage des variables dans la convention PEP8. Il est fortement recommandé de ne pas en utiliser même si la plupart des environnements de développement python le permettent.

```
def transfertTocarte(carte, im, popupPassé):
    folium.Marker(imageToGpsPoint(im), popup = popupPassé).add_to(carte)
    return 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 Créer une fonction `transfertListeTocarte`, qui prendra en paramètre d'entrée un point 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 retournera la carte créée

```
def transfertListeTocarte(centre, listePhotos):
    carte = carteVide(centre)
    for l in listePhotos:
        image = openImage(l)
        transfertTocarte(carte, image, l)
    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 le liste.

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

III.1 Coordonnées GPS

Question 12 Écrire une fonction `testRectangle` qui a deux paramètres d'entrée de type `Point` nommés `point1`, `point2` et qui retourne le booléen :

- Vrai si `point1`, `point2` forment un rectangle non aplati, dont `point1` sera le coin inférieur gauche et `point2` le coin supérieur droit ;
- Faux sinon.

Remarque : on suppose que le rectangle non aplati demandé a des côtés horizontaux et verticaux, ce qui n'est pas précisé dans le texte.

```
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 retourne un `Point` qui est le milieu du segment `point1 - point2`

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

Question 14 On suppose que `inf`, `sup` sont deux objets de type `Point` qui forment un rectangle non aplati.
Écrire une fonction `intRectangle` qui a trois paramètres d'entrée de type `Point` nommés `point`, `inf`, `sup` et qui retourne un booléen. Ce booléen sera vrai si `point` est dans le rectangle formé par `inf`, `sup` et faux sinon. On précise que le bord du rectangle est exclu.

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

III.2 Ajout de markers à une carte numérique

Question 15 Créer 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 `inf - sup`.
Cette fonction retourne `False` si le nombre d'éléments ajoutés à la carte est nul, sinon elle sauvegarde la carte et la retourne.

```
def listeTomap(listePhotos, inf, sup, titre):
    centre = milieu(inf, sup)
    carte = carteVide(centre)
    aucune_photo = True
    for  photo in listePhotos:
```

```
        image = openImage(photo)
        location = imageToGpsPoint(image)
        if intRectangle(location, inf, sup):
            transfertTocarte(carte, image, photo)
            aucune_photo = False
    if aucune_photo :
        return False
    carte.save(titre + '.html')
    return carte
```

Ensuite l'exécution du code suivant permet d'obtenir une carte comme celle proposée en figure ci-dessous.

```
listeTomap(listePhotos, (47.8, 2), (48, 2.1) , 'manouvellecarte')
```

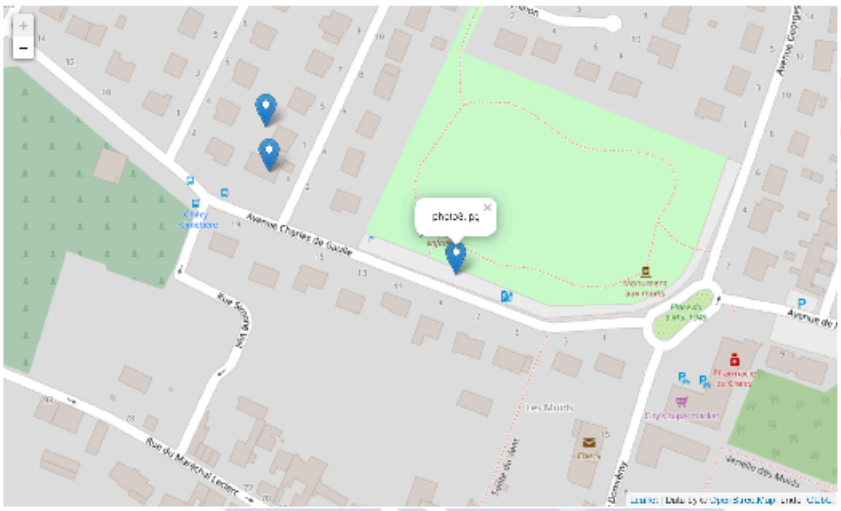

FIGURE 1 – Affichage obtenu à l'exécution du code

Partie IV – Ajout d'une ligne entre chaque marker

```
p1 = (39.900908, -73.040335)
p2 = (40.768571, -73.861603)
p3 = (41.011522, -73.960004)
centre = (40.70000, -73.70000)
coordinates = [p1, p2, p3]
m = folium.Map(location = centre, zoom_start = 20)
aline = folium.PolyLine(locations = coordinates, weight = 2, color = 'blue')
# Ajout d'une ligne polygoniale p1-p2-p3 de couleur bleue largeur 2
aline.add_to(m)
m.save('exemple.html')
```

Dans l'exemple ci-dessus fourni dans le sujet, les éléments de la liste `coordinates` sont des listes et non des tuples comme indiqué.

Question 16 Créer la fonction appelée `listeTomapLine` dont les paramètres d'entrée sont :

- une liste de noms de photos au format chaîne de caractères, nommée `listePhotos` ;
- `inf` et `sup` de type `Point` ;
- `titre` qui sera une chaîne de caractères (sans extension) représentant le nom de la carte ;

```
def listeTomapLine(listePhotos, inf, sup, titre):
    c = listeTomap(listePhotos, inf, sup, titre)
    if not c:
        return False
    coordinates = []
    for l in listePhotos:
        im = openImage(l)
        if intRectangle(imageToGpsPoint(im), inf, sup):
            coordinates.append(imageToGpsPoint(im))
    aline = folium.PolyLine(locations = coordinates, weight = 1, color = 'red')
    aline.add_to(c)
    c.save(titre + '.html')
    return c
```

Ensuite l'exécution du code suivant permet d'obtenir une carte comme celle proposée en figure ci-dessous.

```
listeTomapLine(listePhotos, (47.889, 2.019), (47.90, 2.021) , 'manouvellecarteline')
```

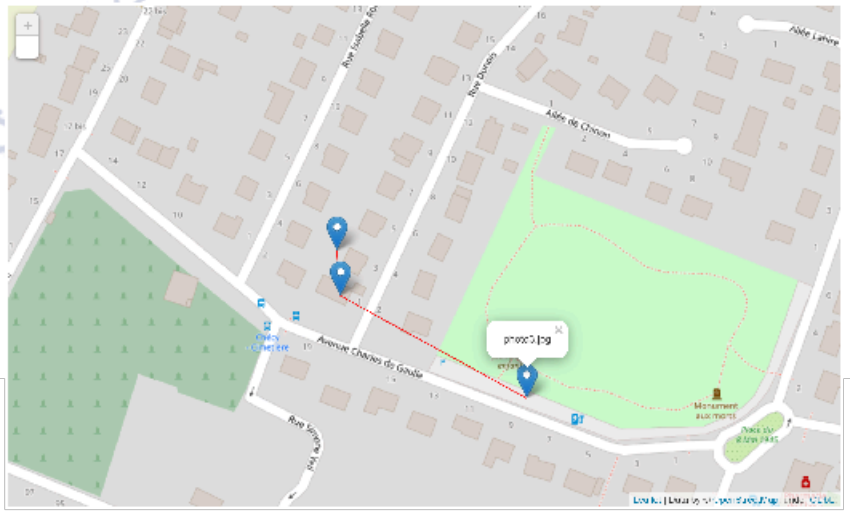

Figure 2 – Affichage obtenu à l'exécution du code

Question 17 Écrire le script qui permet de dessiner un rectangle bleu (épaisseur 1) sur la carte précédente dont la diagonale est `inf - sup`. Le nom de sauvegarde de cette nouvelle carte est `monRectangle`.

```
inf, sup = (47.8938, 2.019), (47.895, 2.021)
inf2, sup2 =  (sup[0], inf[1]), (inf[0], sup[1])
coordinates = [inf, inf2, sup, sup2, inf]
c = listeTomapLine(listePhotos, inf, sup, 'monRectangle')
aline = folium.PolyLine(locations = coordinates, weight = 1, color = 'blue')
aline.add_to(c)
c.save('monRectangle.html')
```

Ensuite l'exécution du code suivant permet d'obtenir une carte comme celle proposée en figure ci-dessous.

```
c
```

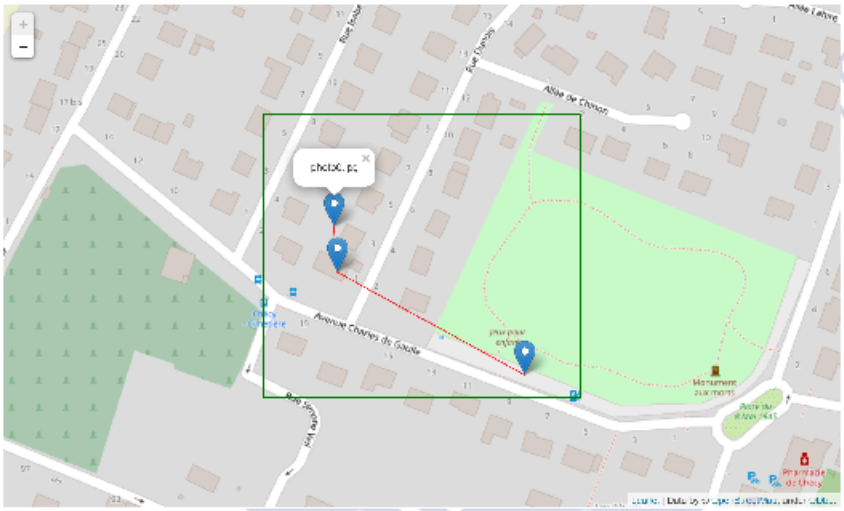

FIGURE 3 – Affichage obtenu à l'exécution du code

Partie V – Recherches de photos dans un dossier

Question 18 Écrire le code Python qui permet de lister toutes les photos de la liste `donnees` ci-dessous.

La question n'est pas très explicite. Doit-on lister seulement les photos de la liste ou prendre en compte aussi les photos contenues dans les dossiers de la liste ?

```
donnees = ['photo.jpeg', 'dir', 'dir2', 'photo2.jpeg', 'photo1211.png', 'dir4110']
#'dir1', 'dir2', ..., 'dir4110' sont des dossiers
liste_photos = []
for d in donnees:
    if '.' in d:
        liste_photos.append(d)
    else: # si on veut lister aussi les photos des sous-dossiers
        liste_photos += os.listdir(d)
        # contrairement à ce que dit le sujet, la méthode append va ajouter la liste et non les éléments
        # de la liste
print(liste_photos)
```

Question 19 Que contient `liste` après l'exécution du code Python ci-dessous :

```
#mot est une variable qui contient une chaîne de caractères
#qui est le nom d'un dossier ou d'une photo
monDir = 'rep' #le dossier initial
if '.' not in mot:
    monDir = monDir + '\\' + mot
    liste = os.listdir(monDir)
```

Après exécution du code ci-dessus, la liste `liste` contient :

- la liste des éléments (dossiers et fichiers) du dossier initial si `mot` est un nom de fichier (contenant un '.') ;
- la liste des éléments (dossiers et fichiers) du sous-dossier `mot` si `mot` est un nom de dossier.

Question 20 Écrire une fonction `listerFichiers` qui prend en entrée une chaîne de caractères (on suppose que c'est un nom de dossier) et qui retourne la liste des fichiers dans ce dossier.

La question est difficile, nous proposons une solution récursive.

```
# Récursif
def listerFichiers(s):
    def listerFichiersRec(s):
        L = os.listdir(s)
        for l in L:
            if '.' in l:
                liste.append(l) # si on a un fichier, on l'ajoute à la liste
            else:
                w = s + '\\' + l # on change le nom de la variable sinon on concatène
                # le nom de tous les dossiers
                listerFichiersRec(w) # appel récursif de la fonction pour parcourir en profondeur
                # les dossiers
        return liste
    liste = [] # liste définie de manière globale pour être complétée au fur et à mesure
    listerFichiersRec(s) # appel de la fonction récursive
    return liste
```

Partie VI – Une fonction mystère et son application

```python
def mysteryMachine(liste):
    x = 1
    while (x < len(liste)):
        nom = liste[x]
        i = x
        while (openImage(nom).datetime < openImage(liste[i - 1]).datetime): # Question 2
            liste[i] = liste[i - 1]
            i = i - 1
            if (i == 0): # Question 3
                break
        liste[i] = nom
        x += 1
    return liste
```

Question 21

1. Quel type de données attend cette fonction ?
   La fonction attend une liste de noms de photos sous forme de chaînes de caractères.
2. Que fait la boucle while numérotée # Question 2
   La boucle `while` est exécutée tant que la photo actuellement traitée a une date antérieure à celle de la photo qui la précède dans la liste.
3. Justifier le test numéroté # Question 3
   Ce test permet d'arrêter la boucle si on arrive à l'indice 0. Sinon la commande `liste[i-1]` renverra au dernier élément (numéro -1) ce qui n'est pas souhaité.
4. Quel algorithme de tri est en jeu dans cette fonction ?
   Il s'agit d'un algorithme de tri par insertion.
5. Quelle est la complexité de cet algorithme ?
   Soit n le nombre d'élément dans la liste.
   Meilleurs des cas : la liste est déjà triée donc $C = 4 \times n = \mathcal{O}(n)$.
   Pire des cas :
   - x = 1 : $C_1 = 4 + 2$
   - x = 2 : $C_2 = 4 + 2 \times 2$
   - ...
   - x = n-1 : $C_{n-1} = 4 + 2 \times (n-1)$

   Donc $C = \sum_{x=1}^{n-1} C_x = \sum (4 + 2 \times x) = (n-1)\frac{8 + 2 \times (n-1)}{2} = \mathcal{O}(n^2)$
6. Que fait cette fonction ?
   Cette fonction trie les photos de la plus ancienne à la plus récente.

Question 22 Dans le script suivant `inf` et `sup` sont les deux points de la diagonale d'un rectangle non aplati et `inf2`, `sup2` sont les deux sommets manquants. Décrire de manière détaillée ce que fait le script.

```
nom = input("Nom dossier ?")
liste = listerFichiers(nom)
liste = mysteryMachine(liste)
titre = jpgTohtml(liste[0])
carte = listeTomap(liste, inf, sup, titre)
coordinates = [inf, inf2, sup, sup2, inf]
aline = folium.PolyLine(locations = coordinates, weight = 1, color = 'blue')
aline.add_to(carte)
carte.save(titre)
```

Le programme demande de renseigner un nom de dossier. Ensuite on obtient la liste de tous les fichiers de ce dossier et de ses sous-dossiers. Cette liste est ensuite triée chronologiquement. On définit pour titre de la carte

le nom de la première photo en remplaçant l'extension par `.html`. On crée une carte portant ce titre dans laquelle on met les markers des photos prises dans le rectangle défini par la diagonale `inf-sup`. On trace ce rectangle en bleu, épaisseur 1. Enfin on enregistre la carte.
En exécutant le script on obtient une carte comme celle de la figure ci-dessous.

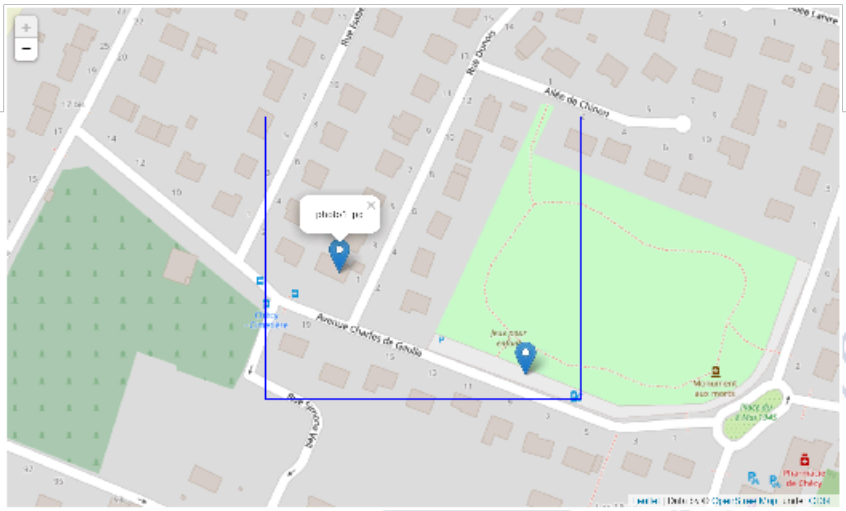

FIGURE 4 – Carte obtenue à l'exécution du code

Partie VII – Interrogation de Bases de données SQL

Question 23 Écrire la requête SQL qui permet d'avoir le nom, la latitude et la longitude des photos faites le 2020/06/16

```
SELECT Nom, Latitude, Longitude
FROM Photos
WHERE Date = 2020/06/16 ;
```

Question 24 Écrire la requête SQL qui permet d'avoir le nom des photos faites le 2020/06/16 et entre les latitude 35935 et 355940.

```
SELECT Nom
FROM Photos
WHERE Date = 2020/06/16 AND Latitude BETWEEN 35935 AND 355940 ;
```

Question 25 Écrire la requête SQL qui permet de compter le nombre de photos situées dans le dossier C :\Images\Las Vegas\Bellagio.

```
SELECT COUNT(*)
FROM Dir
WHERE Dossier = "C:\Images\Las Vegas\Bellagio" ;
```

Question 26 Écrire la requête SQL qui permet de trouver le nom des photos prises le 2020/06/17 et situées dans le dossier C :\Images\Las Vegas\Bellagio.

```
SELECT Nom
FROM Photos JOIN Dir ON Nom = NomPhoto
WHERE Date = 2020/06/17 AND Dossier = "C:\Images\Las Vegas\Bellagio" ;
```

Question 27 En utilisant les fonctions définies précédemment, écrire le script Python qui permet de savoir si, pour le dossier C :\Images\Las Vegas\Bellagio, de la table Dir, toutes les photos qui y sont enregistrées, dont la date de prise de vue est comprise entre le 2020/06/15 et le 2020/06/21, sont dans la base Photos. Le script devra afficher la liste des photos ordonnées par date qui ne sont pas dans la base Photos.

```
import pysqlite3

def requete_to_liste(sql):
    # fonction qui renvoie la liste des sorties de la requête sql
    return liste

requete = "SELECT Nom, Date FROM Photos WHERE Date BETWEEN 2020/06/15 AND 2020/06/21 ORDER BY Date;"
# liste des photos du Bellagio prises entre les 2 dates, triées par date :
listePhotosBase = requete_to_liste(requete)
# c'est une liste de listes car on a 2 informations par ligne

listePhotos = listerFichiers('C:\Images\Las Vegas\Bellagio') # liste des photos dans le dossier

listePhotos = mysteryMachine(listePhotos) # liste des photos triées par date

listeDiff = []
i = 0
# On ne regarde pas les photos avant la date initiale:
while openImage(listePhotos[i]).datetime < "2020:06:15":
    i += 1
j = 0
# on regarde les photos jusqu'à la date finale:
while openImage(listePhotos[i]).datetime <= "2020:06:21":
    # si les photos sont dans les 2 dossiers, on ne fait rien:
    if listePhotos[i] == listePhotosBase[j][0]:
        i += 1
        j += 1
    else:
        listeDiff.append(listePhotos[i]) # si la photo n'est pas dans la base on l'ajoute au résultat
        i += 1

print(listeDiff) # On affiche la liste, les photos sont déjà triées par date
```