Concours Commun INP (CCINP), France

Session 2022, filière TSI

Informatique

Durée : 3 heures

Ce que traite le sujet

- Parties I et II : géolocalisation et données exif
- Parties III et IV : module Folium
- Partie V : recherche de photos dans un dossier
- Partie VI : une fonction mystère
- Partie VII : interrogation de bases de données SQL

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/ccinp/tsi/2022/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

CONCOURS COMMUN INP

ÉPREUVE SPÉCIFIQUE - FILIÈRE TSI

INFORMATIQUE

Durée : 3 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- *Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.*
- *Ne pas utiliser de correcteur.*
- *Écrire le mot FIN à la fin de votre composition.*

Les calculatrices sont interdites.

Le sujet est composé de sept parties et de trois annexes (voir fin du sujet).
Chaque Partie peut être traitée de manière indépendante.
Les fonctions Python créées dans une question (même non traitée) pourront être utilisées.

Représentation de données géolocalisées sur une carte numérique

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. On ajoutera des fonctionnalités à ce script au fur et à mesure de l'avancement du sujet, et la dernière partie permettra de travailler sur des informations concernant ces fichiers de photos en interrogeant une base de données. Dans tout le sujet, on supposera les modules Python importés.

Partie I - Géolocalisation de photos

Q1. Qu'affiche le script ci-dessous ?
Vous expliquerez votre résultat.

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

Lors des questions **Q2**, **Q3** et **Q4**, 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.

Q2. Écrire une fonction nommée `placeDuPoint` qui prend en entrée une chaîne de caractères et qui retourne l'indice du point dans cette chaîne caractères.
Exemple : dans `'devoir.ipynb'` le point a pour indice 6.

Q3. Écrire une fonction nommée `coupeExtension` qui prend en entrée une chaîne de caractères et qui retourne la chaîne de caractères précédent le caractère `'.'`.
Exemple : la fonction appliquée à `'devoir.ipynb'` retourne `'devoir'`.

Q4. Écrire une fonction nommée `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`.
Exemple : la fonction appliquée à `'photo.jpeg'` retourne `'photo.html'`.

Partie II - Lecture de données `exif`

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

Q5. Écrire une fonction `convertTodecimale` qui prend en entrée un tuple (degré, minutes, secondes) et qui retourne le flottant correspondant.

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

Q6. Écrire une fonction `imageToGpsPoint` qui prend en entrée une variable `my_image` et qui retourne une variable de type Point.

Q7. 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.

Partie III - 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.

Q8. 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`.

Q9. Écrire le script de la fonction nommée `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` au coordonnées du `Point` et le `popup`. Cette fonction retourne la carte ainsi modifiée.

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
```

Q10. Créer une fonction `transfertListeTocarte`, qui prendra en paramètres 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 de plus répondre aux exigences suivantes :
- chaque `popup` aura pour `nom` le nom ajouté ;
- la fonction retournera la carte créée.

Q11. 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.jpeg','Bellagio.jpeg','Palazzo.png']
```

III.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`.

Q12. Écrire une fonction `testRectangle` qui a deux paramètres d'entrée de type `Point` nommés `point1`, `point2` et qui retourne le booléen :
- Vrai si `point1` et `point2` forment un rectangle non aplati, dont `point1` sera le coin inférieur gauche et `point2` le coin supérieur droit ;
- Faux sinon.

Q13. 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`.

Q14. 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` appartient au rectangle formé par `inf`, `sup` et faux sinon.
On précise que le bord du rectangle est exclu.

III.2 - Ajout de markers à une carte numérique

On reprend la notion de rectangle présentée dans la **Q14**.
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')
```

Q15. 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.

Partie IV - Ajout d'une ligne entre chaque `marker`

On dispose d'une liste de noms de photos que l'on veut placer sur une carte numérique, les `markers` seront des noms de photos.
Chaque `marker` sera relié à un autre `marker` par une ligne dessinée sur la carte.
Chaque ligne dessinée est faite suivant l'ordre d'apparition des noms dans la liste.
Pour faire une ligne qui relie un `marker` à un autre `marker` dans l'ordre d'une liste, on procède comme ci-dessous.
Attention, dans l'exemple, les éléments de la liste sont des `tuples` de coordonnées GPS.

```
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=20)
aline=folium.PolyLine(locations=coordinates,weight=2,color='blue')
# Ajout d'une ligne polygonale p1-p2-p3 de couleur bleue largeur 2
aline.add_to(m)
```

Q16. Créer la fonction appelée `listeTomapLine` dont les paramètres d'entrée sont :

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

Cette fonction crée une carte dont le nom est `titre`, et y positionne uniquement le nom des photos si elles sont dans le rectangle délimité par `inf` et `sup`, et relie chaque élément consécutif ajouté à la carte par une ligne rouge de largeur 1.
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.

Q17. 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`.

On donne la liste ordonnée des points de ce rectangle à utiliser dans cette question :

```
coordinates=[inf,inf2,sup,sup2,inf]
```

Partie V - 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 ou des dossiers de dossiers de photos, etc.
Le module `os` permet de gérer ces dossiers et nous allons utiliser en particulier la fonction `os.listdir` qui prend en paramètres une chaîne de caractères (le nom d'un dossier) et qui retourne la liste de noms de fichiers avec leur extension ou de dossiers.
Par exemple : l'appel de la fonction `os.listdir('Photos')` retourne la liste des noms des fichiers et des sous-dossiers du dossier appelé ici `Photos`.

Exemple :

```
import os
liste=os.listdir('Photos') # os.listdir ne peut prendre en entrée qu'un repertoire
print(liste)
>>>['Photos2019','Photos2018','photo1.jpg','photo2.jpg','photo3.jpg']
liste2=os.listdir('dossier\dossier2')
print(liste2)
>>>['dossier 3 bis','dossier 3 ter','dossier3','photo2_1.png','photo2_2.jpg']
```

On précise que si un dossier nommé `'Photos2019'` est un sous-dossier du dossier nommé `'Photos'` pour accéder aux éléments de `'Photos2019'` il faut donner le chemin `'Photos\Photos2019'`. Ainsi, si `'Photos2019'` contient le dossier `'Photos2019Janvier'`, alors pour accéder aux éléments de `'Photos2019Janvier'`, il faut écrire : `'Photos\Photos2019\Photos2019Janvier`.
On rappelle que le nom d'un fichier contient une seule et unique fois le caractère `'.'` et que le nom d'un dossier ne comporte pas ce caractère. Dans les **Q17**, **Q18** et **Q19**, les chaînes de caractères utilisées ne peuvent être que des noms de dossiers ou des noms de fichiers.

Q18. Écrire le code Python qui permet de lister toutes les photos de la liste `donnees` ci-dessous. Avec cette liste on n'a qu'une vue partielle des `données`, mais on sait que chaque dossier de cette liste ne contient que des photos et rien d'autre.

```
donnees=['photo.jpeg','dir1','dir2','photo2.jpeg',...,'photo1211.png','dir4110']
#'dir1','dir2',...,'dir4110' sont des dossiers
```

Conseil : pour ajouter un élément, ou une liste à une liste, se référer à **l'annexe 1**.

Q19. 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.listedir(monDir)
```

Q20. 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. Rappelons que ce dossier peut contenir des photos mais aussi des dossiers de photos ou des dossiers de dossiers, etc.

Partie VI - Une fonction mystère et son application

On donne ci-dessous la fonction `mysteryMachine` :

```
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
```

Q21. **1.** Quel type de données attend cette fonction (réponse détaillée attendue) ?
2. Que fait la boucle while numérotée # Question 2.
3. Justifiez le test numéroté # Question 3.
4. Quel algorithme de tri est en en jeu dans cette fonction ?
5. Quelle est la complexité de cet algorithme ?
6. Que fait cette fonction ?

Q22. Dans le script suivant, `inf` et `sup` sont les deux `Point` de la diagonale d'un rectangle non aplati et `inf2,sup2` sont les deux sommets manquants du rectangle.

Décrire de manière détaillée ce que fait ce 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)
```

Partie VII - Interrogation de Bases de données SQL

`SQLite` permet de créer et de gérer des bases de données. Le module Python `pysqlite3` permet d'interagir avec des bases de données `SQLite`.
Les tables utiles pour répondre aux questions sont en **annexe 3**. On remarquera que dans la table `Photos`, les latitudes et longitudes sont des entiers : une division par 10 000 permet d'avoir la latitude et la longitude réelle.
On notera que dans la table `Photos`, `Date` correspond à la date de création (année/mois/jour) de photos présentes dans le dossier (colonne nommée Dossier). On précise qu'à un nom de photo correspond une unique photo et que chaque dossier ne contient que des photos.

Q23. Écrire la requête SQL qui permet d'avoir le nom, la latitude et la longitude des photos faites le 2020/06/16.

Q24. Écrire la requête SQL qui permet d'avoir le nom des photos faites le 2020/06/16 et entre les latitudes 35935 et 35940.

Q25. Écrire la requête SQL qui permet de compter le nombre de photos situées dans le Dossier C :\Images\Las Vegas\Bellagio.

Q26. 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.

On donne le prototype de la fonction `requete_to_liste` écrite en Python qui attend en entrée une requête SQL au format chaîne de caractères : tout simplement une requête SQL usuelle "entourée de guillemets".

```
def requete_to_liste(sql):
    ...
    return liste
```

Cette fonction retourne une liste d'informations, par exemple une liste de noms de photos, qui répond à la requête.

Q27. En utilisant des 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.

ANNEXE 1
Rappels sur Python

Rappels sur les listes

```
#On suppose que maListe est une liste
len(maListe) # donne la longueur de maListe
maListe.append(objet) # ajoute l'objet à maListe
#il sera le dernier élément une fois ajouté
#soit autreListe une liste
maListe.append(autreListe)#ajoute tous les éléments de autreListe à maliste
#autreListe est non modifiée
```

Rappels sur les tuples

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

```
monTuple=(donnee1,donnee2)
monTuple[0]# permet d'accéder à donnee1
monTuple[1]# permet d'accéder à donnee2
#on peut aussi accéder aux données en faisant
val1,val2=monTuple
print(val1)
>>>donnee1
print(val2)
>>>donnee2
#Pour parcourir un tuple :
for x in monTuple:
    print(x)
>>>donnee1
>>>donnee2
```

Une fonction peut retourner 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 les chaînes de caractères

Parcours d'une chaîne :

```
mot="test"
for lettre in mot:
    print(lettre)
#Résultat console :
T
e
s
t
```

Rappels sur le Slicing :

```
print(mot[:3])
>>>Bon
print(mot[3:])
>>>jour
```

Rappel sur la fonction len :

```
print(len(mot))
>>>7
```

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:
        return Image(imageFile)
    imageFile.close()
#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 `(degré, 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 : négative si on est `'W'` ou `'S'` et positive 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.
```

ANNEXE 3
Tables SQL

Nom	Date	Latitude	Longitude	Id
Photo10	2020/06/11	35937	-1149686	1
Photo11	2020/06/22	35937	-1149685	2
Photo24	2020/06/17	35937	-1149686	3
Photo15	2020/06/17	35938	-1149686	4
Photo33	2020/06/16	35939	-1149688	5
⋮	⋮	⋮	⋮	⋮

Table Photos

Id	Dossier	NomPhoto
1	C :\Images\LasVegas\Bellagio	Photo10
2	C :\Images\LasVegas\Palazzo	Photo11
3	C :\Images\LasVegas\Flamingo	Photo24
⋮	⋮	⋮
10	C :\Images\LasVegas\Flamingo	Photo40
11	C :\Images\LasVegas\Palazzo	Photo30
12	C :\Images\LasVegas\Flamingo	Photo15
⋮	⋮	⋮

Table Dir

Table	Champ	Type
Photos	Nom	Chaîne de caractères
Photos	Date	Chaîne de caractères
Photos	Latitude	Entier
Photos	Longitude	Entier
Photos	Id	Entier
Dir	Id	Entier
Dir	NomPhoto	Chaîne de caractères
Dir	Dossier	Chaîne de caractères

Type des données par Table

FIN

IMPRIMERIE NATIONALE – 22 1185 – D'après documents fournis

Travailler les chapitres de ce sujet

Prepa Help suit le programme de TSI chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Bases de Python : expressions, types, instructions, fonctions
- Récursivité

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/ccinp/tsi