Corrigé Concours Blanc 2026 Informatique TSI

Partie I - Géolocalisation des photos

Q.1

Cette commande affiche le booléen `False` puisque il est demand ? d'imprimer le r ?sultat du test `x>y` (donc un booléen) qui compare les dates des variables `x` et `y` (et que `x` est antérieur à `y`).

Q.2

```
def placeDuPoint(chaine) :
  for i in range(len(chaine)):
    if chaine[i] == '.' : return i
```

Q.3

```
def coupeExtension(chaine) :
  point = placeDuPoint(chaine)
  return chaine[:point]
```

Q.4

```
def jpgTohtml(chaine) :
  return coupeExtension(chaine)+'.html'
```

Partie II - Lecture de données `exif`

Q.5

```
def convertTodecimale((d,m,s)) :
  return d+m/60+s/3600
```

Q.6

On commence par extraire de l'image la latitude en degrés, minutes, secondes, que l'on convertit en flottant. Ensuite, on extrait la direction nord ou sud, et on signe négativement le flottant si la direction est sud. On fait de même pour la latitude avant de renvoyer le tuple correspondant aux deux flottants signés :

```
def imageToGpsPoint(my_image) :
  lat_deg = my_image.gps_latitude
  lat_decimal = convertTodecimale(lat_deg)
  if my_image.gps_latitude_ref == 'S' : lat_decimal = -lat_decimal
  long_deg = my_image.gps_longitude
  long_decimal = convertTodecimale(long_deg)
  if my_image.gps_longitude_ref == 'W' : long_decimal = -long_decimal
  return (lat_decimal,long_decimal)
```

Q.7

On parcourt la liste des photos, et pour chaque photo, on extrait l'heure de prise de vue (d'abord en extrayant la date, puis ne retenant que la partie de la chaine de caractéres qui s'y rapporte). Si cette heure de prise vue est inférieure à 12, la photo a été prise avant midi, elle est donc incluse dans la liste des photos à renvoyer :

```
def listeAvantMidi(liste_photos) :
  liste_photo_matin = []
  for photo in liste_photos :
    my_image = openImage(photo)
    date = my_image.datetime
    heure = int(date[11:13])
    if heure < 12 : liste_photo_matin.append(photo)
  return liste_photo_matin
```

Partie III : Module Folium

Q.8

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

Q.9

```
def tranfertTocarte(carte,my_image,popupPass?) :
  location = imageToGpsPoint(my_image)
  folium.Marker(location,popup=popupPass?).add_to(carte)
  return carte
```

Q.10

```
def tranfertListeTocarte(centre,listePhotos) :
  carte = carteVide(centre)
  for photo in listePhotos :
    my_image = openImage(photo)
    transfertTocarte(carte,my_image,photo)
  return carte
```

(L'énoncé n'est pas clair sur le popup qui doit être placé, on pourra préférer pour faciliter la lecture de la carte remplacer l'argument `photo` de `transfertTocarte` par `coupeExtension(photo)` afin d'enlever les .jpeg ou .png)

Q.11

Si on veut utiliser explicitement les `markers` comme demandé par l'énoncé, on peut utiliser la séqunce d'instructions suivantes :

```
photo_1 = listePhotos[0]
centre = imageToGpsPoint(photo_1)
carte = carteVide(centre)
for photo in listePhotos :
  my_image = openImage(photo)
  location = imageToGpsPoint(my_image)
  folium.Marker(location,popup=photo).add_to(carte)
carte.save('Las␣Vegas.html')
```

ou bien en utilisant les fonctions définies précédemment :

```
photo_1 = listePhotos[0]
centre = imageToGpsPoint(photo_1)
carte = tranfertListeTocarte(centre,listePhotos)
carte.save('Las␣Vegas.html')
```

III.1 : Coordonnées GPS

Q.12

Pour que le rectangle soit non aplati et que le point `inf` soit bien l'inéfrieur gauche, il faut que sa latitude et sa longitude soient plus petites que celles de `sup`. On peut donc écrire :

```
def testRectangle(point1, point2) :
  (lat1, long1) = point1
  (lat2,long2) = point2
  return lat1 < lat2 & long1 < long2
```

Q.13

```
def milieu(point1, point2) :
  (lat1, long1) = point1
  (lat2,long2) = point2
  return ((lat1 + lat2)/2,(long1+long2)/2)
```

Q.14

```
def intRectangle(point, inf, sup) :
  (lat, long) = point
  (latinf,longinf) = inf
  (latsup,longsup) = sup
  return latinf < lat < latsup & longinf<long < longsup
```

III.2 : Ajout de markers à une carte numérique

Q.15

```
def listeTomap(listePhotos, inf, sup,titre) :
  centre = milieu(inf,sup)
  carte = carteVide(centre)
  vide = True
  for photo in listePhotos :
    my_image = open.Image(photo)
    location = imageToGpsPoint(my_image)
    if intRectangle(location,inf,sup) :
      vide = False
      transfertTocarte(carte,my_image)
  if vide : return False
  else :
    carte.save(titre+'.html')
    return carte
```

Partie IV : Ajout d'une ligne entre chaque marker

Q.16

```
def listeTomapLine(listePhotos, inf, sup,titre) :
  centre = milieu(inf,sup)
  carte = carteVide(centre)
  vide = True
  liste_positions = []
  for photo in listePhotos :
    my_image = open.Image(photo)
    location = imageToGpsPoint(my_image)
    if intRectangle(location,inf,sup) :
      vide = False
      transfertTocarte(carte,my_image)
      liste_positions.append(location)
  if vide : return False
  else :
    aline = folium.PolyLine(locations=liste_position,weight = 1,color='red')
    aline.add_to(carte)
    carte.save(titre+'.html')
    return carte
```

Q.17

```
carte = listeTomapLine(listePhotos, inf, sup,titre)
aline = folium.PolyLine(locations=coordinates,weight = 1,color='blue')
aline.add_to(carte)
carte.save('monRectangle.html')
```

Partie V : Recherche de photos dans un dossier

Q.18

```
liste_photos =[]
for nom in donnees :
  if '.' in nom : liste_photos.append(nom)
print(liste_photos)
```

Q.19

Le test permet de dire si l'élément `mot` est un dossier ou un fichier. Si c'est un dossier, on construit le chemin correspondant à ce sous-dossier du dossier initial, puis on renvoie la liste de tous les éléments de ce sous-dossier (fichiers et sous-sous-dossiers). La variable `liste` contient ainsi la liste des données du sous-dossier `mot` si c'est un sous-dossier, et n'est pas définie s'il s'agit d'un fichier.

Q.20

```
def listerFichiers(chaine) :
  donnees = os.listdir(chaine)
  liste_fichiers = []
  while donnees != [] :
    donnee = donnees.pop()
    if '.' in donnee : liste_fichier.append(donnee)
    else :
      monDir = chaine+'/'+donnee
      liste2 = os.listdir(monDir)
      for i in liste2 :
        nom = donnee+'/'+i
        donnees.append(nom)
```

La liste `donnees` contient l'ensemble des noms de fichiers ou dossiers à traiter, elle est initialisé à partir des donn ?es pr ?sentes dans le fichier `chaine`. Tant qu'il reste des données à traiter, on extrait la dernière : s'il s'agit d'un fichier on l'ajoute à la liste des fichiers, s'il s'agit d'un dossier, on extrait tout le contenu du sous dossier, que l'on ajoute aux données à traiter en gardant l'arborescence (sauf la racine `chaine`)

Partie VI : Une fonction mystère et son application

Q.21

1) Cette fonction attend une liste (pour pouvoir itérer sur les éléments `liste[x]`, ce doit être une liste ou une chaîne de caractère). Il faut de plus que chaque élément `nom` de liste soit un argument de openImage, il faut donc que l'argument de cette fonction soit une liste de noms de photos, donc une liste de chaines de caractères.

2) Le test d'entrée dans la boucle compare les dates de prise de vues des photos en position `x` dans la liste des photos et celle à une position `i` antérieure, et les inverse si tel est le cas, elle place donc l'image en position `x` initialement à la position parmi celles qui sont situées avant qui est la plus loin dans la liste et oùù la photo précédente est antèrieure.

3) Ce test permet d'arrêter de déplacer la photo `x` vers la gauche dans la liste dans le cas oùù elle est la première prise parmi les `x` premières.

4) Il s'agit d'un algorithme de tri à bulles puisque les éléments sont ordonnés par permutations successives d'éléments consécutifs.

5) La complexité de cet algorithme est quadratique en moyenne (linéaire dans le cas d'une liste déjà triée).

6) Cette fonction trie la liste de photos `liste` par ordre chronologique de prise de vue.

Q.22

Dans ce script, les opérations suivantes sont effectuées :

— on crée la liste des fichiers (donc de photos) pr ?sentes dans le dossier dont le nom est saisi par l'opérateur, puis on la trie par ordre chronologique de prise de vue.

— La variable `carte` est alors constituée de la carte oùù chaque photo de la liste est située sur la carte si elle a été prise dans le rectangle non aplati délimité par les deux Points `inf` et `sup`.

— On ajoute ensuite une bordure en bleu à cette carte avant de la sauver avec comme nom celui de la première photo (dont on remplace l'extension par html).

Commentaire : l'intérêt de trier les photos chronologiquement est ici nul, il aurait mieux valu utiliser la fonction `listeTomapLine` au lieu de `listeTomap` afin de visualiser en plus en rouge le trajet suivi par le ou la photographe.

Partie VII : Interrogation de Bases de données SQL

Q.23

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

Q.24

```
SELECT Nom FROM Photos WHERE Date = ”2020/06/16" AND 35935 < Latitude < 35940 ;
```

Q.25

```
SELECT COUNT(NomPhoto) FROM Dir HAVING Dossier = "C: Images  Las Vegas  Bellagio" ;
```

Q.26

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

Q.27

```
requete_dir = ('SELECT␣NomPhoto␣FROM␣Dir␣WHERE␣Dossier␣=␣"C:\Images\Las␣Vegas\Bellagio"␣;'
liste_photos_dir = requete_to_liste(requete_dir)
requete_ph = 'SELECT␣Nom␣FROM␣Photos␣WHERE␣''2020/06/15"␣<␣Date␣<␣''2020/06/21"␣;'
liste_photos_ph = requete_to_liste(requete_ph}
liste_photos_absentes = []
for i in liste_photos_dir :
  if i not in liste_photos_ph : liste_photo_absentes.append(i)
print(mysteryMachine(liste_photos_absentes))
```