DST Informatique
Durée : 2 heures

Consignes
Écrire 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.

Une annexe sur les méthodes d'utilisation d'un fichier se trouve à la fin du sujet.

Sujet
Une montre destinée à la pratique du trail enregistre des données en utilisant un système de navigation par satellite (GPS) qui permet de déterminer l'heure et la position de l'athlète. De plus, cette montre détermine l'altitude en utilisant les différences de pression atmosphérique.

Par exemple, à la fin d'une course, l'ensemble des données enregistrées par la montre peut être la liste suivante où chaque caractère '␣' correspond à une espace.

```
["2023:08:16␣08:31:54␣N42.91343␣E00.13707␣1048",
"2023:08:16␣08:32:02␣N42.91399␣E00.13708␣1050",
"2023:08:16␣08:32:09␣N42.91421␣E00.13596␣1051",
"2023:08:16␣08:32:17␣N42.91475␣E00.13561␣1054",...]
```

Un élément de cette liste est appelé une *trame*. La première trame correspond au moment où l'athlète active sa montre et la dernière est enregistrée quand la montre est désactivée. Une trame est une chaîne de caractères qui comporte toujours 44 caractères. La première trame de l'exemple précédent, à savoir `"2023:08:16␣08:31:54␣N42.91343␣E00.13707␣1048"`, est constituée de 5 sous-chaînes séparées par des espaces :

- la sous-chaîne de 10 caractères `"2023:08:16"` correspond à la date ;
- la sous-chaîne de 8 caractères `"08:31:54"` correspond à l'heure ;
- la sous-chaîne de 9 caractères `"N42.91343"` correspond à la latitude ;
- la sous-chaîne de 9 caractères `"E00.13707"` correspond à la longitude ;
- la sous-chaîne de 4 caractères `"1048"` correspond à l'altitude en mètres entiers.

Dans tout l'exercice, on suppose que les données sont collectées à une même date.

I Gestion de l'heure

On souhaite pouvoir vérifier qu'une chaîne de caractères est au format des heures dans les trames (comme la sous-chaîne `"08:31:54"` de la première trame donnée en exemple).

Q1. Écrire une fonction `format` qui prend en argument une chaîne de caractères `heure` et qui renvoie un booléen : elle renvoie `True` si `heure` comporte 8 caractères et si le caractère `":"` est situé aux bons emplacements et elle renvoie `False` sinon.
Par exemple :

- `format("04:12:55")` et `format("23:07:56")` valent `True` ;
- `format("12:2:55")` et `format("12;10:25")` valent `False`.

Solution

```
def format (heure: str) -> bool :
  return (len(heure) == 8) and (heure[2] == ':') and (heure[5] ==
    ':')
```

On souhaite maintenant pouvoir vérifier qu'une chaîne de caractères au bon format [1] est bien une heure valide.
Par exemple :

- `"04:12:55"` et `"23:07:56"` sont des heures valides ;
- `"18:62:55"` et `"24:00:00"` ne sont pas des heures valides.

Q2. Écrire le code de la fonction `heure_valide`. On rappelle que `int("25")` renvoie l'entier `25`.

Solution

```
def heure_valide (heure: str) -> bool :
  if not format(heure) :
    return False
  else :
    h = int(heure[0] + heure[1])
    m = int(heure[3] + heure[4])
    s = int(heure[6] + heure[7])
    h_valide = (0 <= h and h <= 23)
    m_valide = (0 <= m and m <= 59)
    s_valide = (0 <= s and s <= 59)
    return h_valide and m_valide and s_valide
```

Q3. Écrire une fonction `extrait_temps_ecoule` qui prend en argument une chaîne de caractères `heure` représentant une heure valide et qui renvoie un entier correspondant au nombre de secondes depuis minuit. Par exemple, `extrait_temps_ecoule("00:02:12")` vaut l'entier `132`.

Solution

```
def extrait_temps_ecoule (heure: str) -> int :
  assert(heure_valide(heure))
  h = int(heure[0] + heure[1])
  m = int(heure[3] + heure[4])
  s = int(heure[6] + heure[7])
  return h*3600 + m*60 + s
```

Il arrive que certaines trames soient interverties ou dupliquées lors du transfert de données, on veut pouvoir vérifier que ce n'est pas le cas dans les données reçues.

1. c'est-dire une chaîne constituée de deux chiffres, un caractère `":"`, deux chiffres, un caractère `":"` et enfin deux chiffres

Q4. Écrire une fonction `verification_ordre` qui prend en arguments deux chaînes de caractères `h1` et `h2` représentant des heures valides et qui renvoie un booléen. Ce booléen vaut `True` si `h2` est strictement postérieure à `h1` et `False` sinon.

Solution

```
def verification_ordre (h1: str, h2: str) -> bool :
  t1 = extrait_temps_ecoule(h1)
  t2 = extrait_temps_ecoule(h2)
  return t1 < t2
```

Q5. Écrire une fonction `duree` qui prend en arguments deux chaînes de caractères `h1` et `h2` représentant des heures valides et qui renvoie un entier. Cet entier représente la durée, en secondes, entre les instants `h1` et `h2`. Par exemple, `duree("15:48:12","15:50:18")` vaut l'entier `126`.

Solution

```
def duree(h1: str, h2: str) -> int :
  t1 = extrait_temps_ecoule(h1)
  t2 = extrait_temps_ecoule(h2)
  return t2 - t1
```

II Validité et extraction des données d'une trame

On souhaite vérifier qu'au cours du transfert d'une trame, aucune des cinq sous-chaînes (date, heure, latitude, longitude et altitude) n'a été oubliée ni dégradée. On dit qu'une trame est *valide* si elle comporte bien ces cinq éléments, séparés par des espaces, et que chacun d'eux est lui-même valide.

On suppose disposer dans la suite des fonctions suivantes :

- `date_valide` qui prend en argument une chaîne de caractères `date` et qui renvoie `True` si `date` est de longueur 10, au bon format, et représente une date valide et renvoie `False` sinon ;
- `lat_valide` qui prend en argument une chaîne de caractères `lat` et qui renvoie `True` si `lat` est de longueur 9, au bon format, et représente une latitude valide et renvoie `False` sinon ;
- `long_valide` qui prend en argument une chaîne de caractères `long` et qui renvoie `True` si `long` est de longueur 9, au bon format, et représente une longitude valide et renvoie `False` sinon ;
- `alt_valide` qui prend en argument une chaîne de caractères `alt` et qui renvoie `True` si `alt` est de longueur 4, au bon format, et représente une altitude valide et renvoie `False` sinon.

Q6. Écrire une fonction `trame_valide` qui prend en argument une chaîne de caractères `trame` et qui renvoie `True` si `trame` représente bien une trame valide et `False` sinon.

Solution

```
def trame_valide (trame: str) -> bool :
  if (len(trame) != 44) or (trame[10] != " ") or (trame[19] != "
   ") or\
    (trame[29] !=" ") or (trame[39] != " ") :
    return False
```

```
  else :
    date = trame[0:10]
    heure = trame[11:19]
    lati = trame[20:29]
    longi = trame[30:39]
    alti = trame[40:44]
    return heure_valide(heure) and date_valide(date) and\
    long_valide(longi) and lat_valide(lati) and alt_valide(alti)
```

On suppose disposer dans la suite des fonctions suivantes :

- `extrait_lat` qui prend en argument une chaîne de 9 caractères `lat` qui est une latitude valide et qui renvoie un flottant représentant la latitude ;
- `extrait_long` qui prend en argument une chaîne de 9 caractères `long` qui est une longitude valide, et qui renvoie un flottant représentant la longitude ;
- `extrait_alt` qui prend en argument une chaîne de 4 caractères `alt` qui est une altitude valide, et qui renvoie un entier représentant l'altitude.

Q7. Écrire une fonction `extrait_trame` qui prend en argument une chaîne de caractères `trame` représentant une trame valide. et qui renvoie les informations de cette trame sous la forme d'un tuple `(t, lati, longi, alt)` où :

- `t` est de type `int` et représente le temps écoulé depuis minuit en secondes ;
- `lati` est de type `float` et représente la latitude ;
- `longi` est de type `float` et représente la longitude ;
- `alt` est de type `int` et représente l'altitude en mètres.

Solution

```
def extrait_trame(trame: str) -> tuple :
  h = extrait_temps_ecoule(trame[11:19])
  lati = extrait_lat(trame[20:29])
  longi = extrait_long(trame[30:39])
  alti = extrait_alt(trame[40:44])
  return (h, lati, longi, alti)
```

III Analyse des performances de l'athlète

Dans toute la suite, le parcours réalisé par l'athlète est représenté par une liste dont chaque élément est un tuple de la forme `(h, lat, long, alt)` qui, comme expliqué à la question précédente, représente une trame. On appellera *parcours* de telles listes. Par exemple, le parcours correspondant à la liste de trames donnée en exemple en début de sujet (page 1) est la liste suivante.

```
[(30714, 42.91343, 0.13707, 1048),
 (30722, 42.91399, 0.13708, 1050),
 (30729, 42.91421, 0.13596, 1051),
 (30737, 42.91475, 0.13561, 1054),
 ...]
```

Q8. Écrire une fonction `alt_max` qui prend en argument un parcours non vide `p` (représenté par une liste comme décrit ci-avant) et qui renvoie l'altitude maximale, en mètres, atteinte par l'athlète durant le parcours `p`. *On n'utilisera pas la fonction Python max et on supposera qu'il y a au moins une altitude positive dans le parcours.*

Solution

```
def alt_max(p: Parcours) -> int :
  res = 0
  for point in p :
    (_, _, _, alti) = point
```

```
    if alti > res :
      res = alti
  return res
```

Le dénivelé positif réalisé lors d'un parcours est le total des montées réalisées lors de ce parcours. Par exemple, si les altitudes successives d'un parcours sont 1048, 1051, 1054, 1049 et 1053, le dénivelé positif de ce parcours vaut 10 mètres.

Q9. Écrire une fonction `denivele_positif` qui prend en argument un parcours `p` et qui renvoie, sous la forme d'un entier, le total des montées, en mètres, du parcours `p`.

Solution

```
def denivele_positif(p: Parcours) -> int :
  tot_montee = 0
  for i in range(len(p)-1) :
    if p[i+1][3] > p[i][3] :
      tot_montee += p[i+1][3]-p[i][3]
  return tot_montee
```

Dans la suite, on dispose d'une fonction `distance(lati1, longi1, lati2, longi2)` qui renvoie la distance sur le globe terrestre, en mètres, entre des points de coordonnées géographiques respectives `(lat1,long1)` et `(lat2,long2)`.

Q10. Écrire la fonction `vitesse_moyenne` qui prend en argument un parcours `p` et qui renvoie la vitesse moyenne en mètres par seconde de l'athlète durant le parcours `p`. On suppose que le parcours `p` contient au moins deux éléments correspondant à des instants distincts.

Solution

```
def vitesse_moyenne(p: Parcours) -> float :
  dist = 0
  for i in range(len(p)-1) :
    (_, lati1, longi1, _) = p[i]
    (_, lati2, longi2, _) = p[i+1]
    dist += distance(lati1, longi1, lati2, longi2)
  duree = p[len(p)-1][0] - p[0][0]
  assert(duree > 0) #optionnel
  return dist/duree
```

On considère que l'athlète est en pause entre 2 instants `t1` et `t2` consécutifs si la distance entre ses positions aux instants `t1` et `t2` est inférieure ou égale à 3 mètres.

Q11. Écrire la fonction `en_pause` qui prend en argument quatre flottants `lati1`, `longi1`, `lati2` et `longi2` et qui renvoie `True` si l'athlète est en pause entre les points de coordonnées géographiques `(lati1,longi1)` et `(lati2,longi2)` et `False` sinon.

Solution

```
def en_pause(lati1: float, longi1: float, lati2: float, longi2:
    float) -> bool :
  return distance(lati1, longi1, lati2, longi2) <= 3
```

Q12. Écrire une fonction `temps_pause` qui prend en argument un parcours `p` et qui renvoie, sous la forme d'un nombre entier de secondes, la durée totale des pauses de l'athlète durant le parcours `p`.

Solution

```
def temps_pause(p: Parcours) -> int :
  tps_p = 0
  for i in range(0,len(p)-1) :
    (t1, lt1, lg1, _) = p[i]
    (t2, lt2, lg2, _) = p[i+1]
    if en_pause(lt1, lg1, lt2, lg2) :
      tps_p += (t2 - t1)
  return tps_p
```

Q13. Écrire la fonction `nombre_pauses` qui prend en argument un parcours `p` qui renvoie le nombre de pauses effectuées durant le parcours `p`.

Solution

```
def nombre_pauses(p: Parcours) -> int :
  nb_p = 0
  deja_en_pause = False
  for i in range(0,len(p)-1) :
    (t1, lt1, lg1, _) = p[i]
    (t2, lt2, lg2, _) = p[i+1]
    if en_pause(lt1, lg1, lt2, lg2) :
      if not deja_en_pause :
        nb_p = nb_p + 1 #on compte la pause quand elle commence
        deja_en_pause = True
    else :
      deja_en_pause = False
  return nb_p
```

IV Gestion des performances des participants au trail

Le but de cette partie est de gérer ou d'utiliser les données des athlètes participant à une course organisée sur un circuit.

Pour participer à un trail, les athlètes doivent s'inscrire via un site internet et renseigner leur nom, leur prénom, leur sexe et leur année de naissance.

Au franchissement de la ligne d'arrivée, le temps réalisé par le coureur est relevé et les données de chaque finisher sont collectées dans un fichier texte.

Chaque ligne du fichier correspond à un coureur et les données sont dans l'ordre, le numéro de dossard, le nom, le prénom, le sexe, l'année de naissance, le temps.
On donne ici un extrait du fichier `trail.txt`.

```
1,Germain,Lucille,F,1998,01:38:25
2,Robert,Luc,M,1994,01:39:12
3,Durand,Albert,M,1990,00:58:16
4,Gelin,Lina,F,1991,01:24:52
```

Pour pouvoir manipuler les données du fichier `trail.txt`, celles-ci seront enregistrées dans une liste dont les éléments sont des listes de chaînes de caractères.

```
liste_trail = [["1", "Germain", "Lucille", "F", "1998", "01:38:25"],
 ["2", "Robert", "Luc", "M", "1994", "01:39:12"],
 ["3", "Durand", "Albert", "M", "1990", "00:58:16"],
 ["4", "Gelin", "Lina", "F", "1991", "01:24:52"]]
```

Q14. Écrire la fonction `extraire_liste` qui a pour paramètre un nom de fichier `fichier` de type `str` correspondant au nom de fichier et qui renvoie une telle liste.

Solution

```
def extraire_liste(fichier):
    f = open(fichier)
    liste = f.readlines()
    f.close()
    for k in range(len(liste)):
        liste[k] = liste[k].strip().split(",")
    return liste
```

L'instruction
`liste_trail_ordonnee = sorted(liste_trail, key = lambda a:a[5])`
permet d'ordonner cette liste du coureur le plus rapide au coureur le plus lent.
`liste_trail_ordonnee` a pour contenu :

```
[["3", "Durand", "Albert", "M", "1990", "00:58:16"],
 ["4", "Gelin", "Lina", "F", "1991", "01:24:52"],
 ["1", "Germain", "Lucille", "F", "1998", "01:38:25"],
 ["2", "Robert", "Luc", "M", "1994", "01:39:12"]]
```

Dans la suite de l'énoncé, le paramètre `lto` représentera la liste des finishers d'un trail ordonnée dans l'ordre d'arrivée des coureurs et aura la même structure que `liste_trail_ordonnee`.

Q15. Écrire une fonction `liste_par_sexe` qui a pour paramètres la liste ordonnée des finishers `lto` et une chaîne de caractères `sexe`, et qui renvoie la liste ordonnée des finishers de sexe `sexe`.

Solution

```
def liste_par_sexe(lto, sexe):
    lst_sexe = []
    for coureur in lto:
```

```
        if coureur [3] == sexe:
            lst_sexe.append(coureur)
    return lst_sexe
```

Q16. Écrire une fonction `classement_par_sexe` qui a pour paramètres la liste ordonnée des finishers `lto` et une chaîne de caractères `sexe` et qui crée un fichier texte dont le nom est `classementM.txt` ou `classementF.txt` (suivant la valeur de `sexe`) donnant le classement des finishers homme ou femme. Cette fonction ne renvoie rien.
Par exemple, `classementM.txt` aura pour contenu :

```
3,Durand,Albert,M,1990,00:58:16
2,Robert,Luc,M,1994,01:39:12
```

Solution

```
def classement_par_sexe(lto, sexe):
    lst_sexe = liste_par_sexe(lto, sexe)
    f = open("classement"+sexe+".txt", "w")
    for coureur in lst_sexe:
        ligne = ""
        for k in range(len(coureur) -1):
            ligne += coureur[k]+","
        ligne += coureur[-1] +"\n"
        f.write(ligne)
    f.close()
```

Q17. Écrire une fonction `donnee_coureur_num` qui a pour paramètres une liste ordonnée de finishers `lto` et une chaîne de caractères `num` représentant le numéro de dossard. Cette fonction renvoie le tuple `(c, p)` où `c` la liste de chaîne de caractères représentant le finisher de dossard `num` et `p` est un entier correspondant à la place du coureur `c` dans la liste `lto`.
Par exemple, `donnee_coureur_num(liste_par_sexe(liste_trail, "M"), "2")` renvoie `(["2", "Robert", "Luc", "M", "1994", "01:39:12"], 2)`

Solution

```
def donnee_coureur_num(lto, num):
    k = 1
    for coureur in lto:
        if coureur[0] == num:
            return coureur, k
        k += 1
```

Q18. Écrire une fonction `resume_num` qui a pour paramètres la liste ordonnée des finishers `lto` et une chaîne de caractères `num` représentant le numéro de dossard. Cette fonction crée un fichier texte dont le nom a pour format `nom_prenom.txt` et qui enregistre des données correspondant au finisher de numéro de dossard `num` avec le classement général et le classement par sexe et d'autres informations comme dans l'exemple suivant.
Le fichier `Gelin_Lina.txt` sera exactement :

```
Gelin Lina
Place,Temps,Pl/Sexe
2,01:24:52,1
```

Solution

```
def resume_num(lto, num):
    coureur, plg = donnee_coureur_num(lto, num)
    liste_s = liste_par_sexe(lto, coureur[3])
    coureur, pls = donnee_coureur_num(liste_s, num)
    nom = coureur[1]
    prenom = coureur[2]
    f = open(nom+"_"+prenom+".txt", "w")
    f.write(nom + " "+prenom+"\n")
    f.write("Place,Temps,Pl/Sexe\n")
    f.write(str(plg)+","+coureur[5]+","+str(pls))
    f.close()
```

Annexe : Méthodes principales d'utilisation d'un fichier

La méthode `split` s'applique à une chaîne de caractères `S` : `S.split(c)`. Elle la sépare en une liste de chaînes de caractères. Par défaut, le délimiteur `c` est l'espace.

La méthode `strip` s'applique à une chaîne de caractères `S` : `S.strip()`. Elle renvoie la chaîne de caractères `S` dans laquelle les caractères d'espacement sont retirés en début et en fin de chaîne.

Écriture dans un fichier :

```
montxt=open('fichier.txt','w')
...
montxt.write('texte...')
...
montxt.close()
```

Lecture d'un fichier :

```
montxt=open('fichier.txt','r')
...
S=montxt.read() # ou
L=montxt.readlines()
...
montxt.close()
```

Toutes les méthodes suivantes s'appliquent à un objet de type fichier comme `montxt` : `L=montxt.readline()`, `montxt.write("Hello")`...

`read()`	lit le fichier en entier	renvoie une chaîne de caractères
`read(n)`	lit n caractères	renvoie une chaîne de longueur au plus n, vide si tout le fichier a été lu.
`readline()`	lit une ligne	renvoie une chaîne de caractère, vide si le fichier est fini
`readlines()`	lit toutes les lignes	renvoie la liste de toutes les lignes sous forme de chaînes de caractères.
`write(S)`	écrit la chaîne `S`	
`writelines(L)`	écrit les lignes de la liste `L`	
`close()`	enregistre le fichier	