Devoir d'informatique

Consignes

- Le devoir se fera sur copie double uniquement.
- Le numéro de chaque exercice et de chaque question devra être indiqué sur votre copie.
- Les indentations devront correctement figurer sur votre copie. Vous pourrez par exemple tracer une barre verticale.
- Pour chaque fonction vous donnerez au plus une ligne de commentaire permettant de spécifier votre fonction.

1 Randonnée

1.1 Analyse du profil global

Question 1
Donner l'instruction permettant de connaître la taille du tableau `lat`.

Correction

```
>>> len(lat)
```

Le module `math` fournit la fonction `radians` qui convertit des degrés en radians.

Question 2 *Implémenter la fonction* `conversion(L:list)-> list` *permettant de convertir chacune des valeurs d'un tableau de flottants en radians. Cette fonction agira sans effet de bord.*

Correction

```
def conversion(L:list) -> list :
    tab = []
    for el in L :
        tab.append(m.radians(el))
    return tab
```

Question 3 *Donner les instructions permettant de créer les tableaux* `latr` *et* `longr`*, résultats de la conversion en radians des tableaux* `lat` *et* `long`*.*

Correction

```
>>> latr = conversion(lat)
>>> longr = conversion(long)
```

Question 4 *Implémenter la fonction* `plus_haut(L:list) -> int` *permettant de déterminer l'altitude la plus haute atteinte lors de la randonnée (la fonction* `max` *sera ici interdite).*

Correction

```
def plus_haut(L:list) -> float :
    maxi = L[0]
    for el in L :
        if el>maxi :
            maxi = el
    return maxi
```

Question 5 *Implémenter la fonction* `plus_haut_indice(L:list) -> float` *permettant de déterminer l'indice de l'altitude la plus haute atteinte lors de la randonnée.*

Correction

```
def plus_haut_indice(L:list) -> float :
    m = 0
    for i in range(len(L)):
        if L[i]>L[m] :
            m = i
    return m
```

Question 6 *En utilisant la fonction précédente, implémenter la fonction* `coords_plus_haut(alt:list, long:list, lat:list)-> list` *permettant de renvoyer la liste* `[latitude, longitude]` *des coordonnées du point le plus haut de la randonnée.*

Correction

```
def coords_plus_haut(alt:list, long:list, lat:list)-> list :
    m = plus_haut_indice(alt)
    return[lat[m],long[m]]
```

Question 7 *Implémenter la fonction* `deniveles(alt:list) -> list` *qui calcule les dénivelés cumulés positif et négatif (en mètres) de la randonnée, sous forme d'une liste de deux flottants. Le dénivelé positif est la somme des variations d'altitude positives sur le chemin, et inversement pour le dénivelé négatif.*

Correction

```
def deniveles(alt:list) -> list:
    pos,neg = 0,0
    for i in range(1,len(alt)) :
        delta = alt[i]-alt[i-1]
        if delta > 0:
            pos = pos + delta
        else :
            neg = neg + delta
    return [pos,neg]
```

Question 8 *Implémenter la fonction* `distance_totale(alt:list, long:list, lat:list) -> float` *renvoyant la distance parcourue (en mètres) au cours de la randonnée.*

Correction

```
def distance_totale(alt:list, long:list, lat:list)-> float :
    D = 0
    for i in range(1,len(alt)):
        c1 = [alt[i-1],lat[i-1],long[i-1]]
        c2 = [alt[i],lat[i],long[i]]
        D = D + distance(c1,c2)
```

```
    return D
```

1.2 Découpage du profil

Question 9 *Implémenter la fonction* `moyenne(alt:list) -> float` *permettant de calculer la moyenne des altitudes mesurées par le GPS.*

Correction

```
def moyenne(alt:list):
    somme = 0
    for a in alt :
        somme = somme + a
    return somme/len(alt)
```

Question 10 *Implémenter la fonction* `indice_premier_PNM(alt:list) -> int` *renvoyant, s'il existe, l'indice* `i` *du premier élément de la liste tel que cet élément soit inférieur à la moyenne et l'élément suivant soit supérieur à la moyenne. Cette fonction devra renvoyer* `-1` *si aucun élément vérifiant cette condition n'existe.*

Correction

```
def indice_premier_PNM(alt:list):
    m = moyenne(alt)
    indice = -1
    for i in range(len(alt)-1):
        if alt[i]<m and alt[i+1]>m:
            return i
    return indice
```

Question 11 *Implémenter la fonction* `indices_PNM(alt:list) -> list` *retournant la liste des indices de tous les PNM.*

Correction

```
def indices_PNM(alt:list):
    m = moyenne(alt)
    les_PNM = []
    for i in range(len(alt)-1):
        if alt[i]<m and alt[i+1]>m:
            les_PNM.append(i)
    return les_PNM
```

Question 12 *Dans le but de séparer les différents profils, nous allons chercher les indices des altitudes minimales entre deux PNM successifs. Implémenter la fonction* `liste_alt_mini(alt:list) -> list` *qui répond à ce besoin.*

Correction

```
def liste_alt_min(alt:list):
    les_PNM = indices_PNM(alt)
    les_min = []
    for i in range(len(les_PNM)-1):
        mini = les_PNM[i]
        for j in range(les_PNM[i],les_PNM[i+1]):
            if alt[j]<alt[mini]:
                mini = j
        les_min.append(mini)
    return les_min
```

Question 13 *Implémenter la fonction* `creer_montagnes(alt) -> list` *renvoyant une liste constituée de la liste des montagnes élémentaires.*

Correction

```
def creer_montagnes(alt):
    pam = liste_alt_min(alt)
    montagnes = []
    mont = []
    for i in range(0,pam[0]):
        mont.append(alt[i])
    montagnes.append(mont)
    for i in range(len(pam)-1):
        mont = []
        for j in range(pam[i],pam[i+1]):
            mont.append(alt[j])
        montagnes.append(mont)
    mont = []
    for i in range(pam[-1],len(alt)):
        mont.append(alt[i])
    montagnes.append(mont)
    return montagnes
```