http://www.mesmaths.com/spip.php?article421

2-parcours

- SNT - 6-Cartographie -

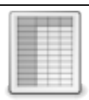
Publication date: mercredi 22 mai 2019

Nous allons ajouter progressivement sur une carte un parcours avec divers éléments.

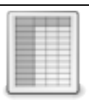
trace GPS

Les compteurs de vélo (type Garmin) ou montres de ce type **enregistrent les coordonnées GPS**.

Voici deux fichiers **.gpx** prêts à l'emploi, déjà convertis correctement avec une extension **.csv** :



parcours1



parcours2

Vous pouvez travailler à partir de vos propres données ; ci-dessous une aide pour le faire :

aide pour utiliser ses propres données

Une fois le compteur (ou la montre) connecté à l'ordinateur, on télécharge le fichier retraçant un parcours ; sur un compteur Garmin, le fichier est en .fit

On doit le convertir en .gpx

Ce site https://www.gpsies.com/convert.do?language=fr&client=summit le fait ; après s'être inscrit, on peut simplement faire la conversion en déposant le fichier .fit ; un fichier .gpx est rendu ; on l'enregistre ensuite avec l'extension .csv.

Petite manipulation à faire : il faut supprimer toutes les espaces (qui sont comptabilisées en python et qui poseront problème).

Pour cela, faire **Ctrl H**, puis taper une espace dans la partie 'rechercher', rien dans la partie 'remplacer' et demander à remplacer partout.

Une vidéo pour illustrer.

import

On va **importer** dans un programme Python **un fichier .csv** contenant de nombreuses informations, notamment les coordonnées GPS (latitude, longitude) de nombreux points, leur altitude et l'heure à laquelle on se trouvait à cet endroit.

En voici le code commenté :

```
##importation des bibliothèques
 import os
 import folium

 ##fonction permettant de stocker les données qui nous concernent dans des listes
 def lire_fich(file):
 f = open(file, mode = "r")#ouverture du fichier en mode lecture ("r" pour read)
 lc, la, lt = [], [], [] #initalisation des listes lc (liste coordonnées) / la (liste altitudes) / lt (liste
temps)
 for ligne in f:
 if ligne[:5] == "
 l = ligne.split('"')#on trie les données séparées par " par .split('"')
 lc.append( (float(l[1]), float(l[3])) )#on ne conserve que les valeurs voulues
 elif ligne[:3] == "
 l = ligne.split("ele")#on trie les données séparées par ele par .split('ele')
 la.append( float(l[1][1:-2]) )#on ne conserve que les valeurs voulues
 elif ligne[:3] == "
 l = ligne.split(":")#on trie les données séparées par : par .split(':')
 lt.append( int(l[0][-2:]+l[1]+l[2][:2]) )#on ne conserve que les valeurs voulues
 f.close()#on ferme le fichier
 return lc, la, lt #on retourne trois listes ; la première est une "liste de listes" car chaque valeur est un
couple de coordonées.

 ##programme principal
 data=lire_fich("tour3.csv")#on décode le fichier gps -> data est un tuple (~une liste) dont le premier terme
est le couple de coordonnées, le second l'altitude et le troisième donne le temps
 c= folium.Map(location=[data[0][0][0], data[0][0][1]],zoom_start=12)#intialisation de la carte aux premières
coordonnées gps, avec un zoom de 12
 L=len(data[0])#nombre de points dans le fichier

 #des marqueurs pour le départ et l'arrivée (on y indiquera l'altitude)
 folium.Marker(data[0][0], popup="DEPART altitude : "+str(data[1][0]), icon=folium.Icon(icon="bicycle",
prefix="fa", color="green")).add_to(c)
 folium.Marker(data[0][L-1], popup="ARRIVEE altitude : "+str(data[1][0]), icon=folium.Icon(icon="bicycle",
prefix="fa", color="red")).add_to(c)


 folium.PolyLine(data[0], color="red", weight=2.5, opacity=1).add_to(c)#on trace la route en une couleur
voulue

 c.save('monParcours3.html')#sauvegarde du fichier en extension .html
```

A faire : modifier quelques paramètres dans ce code (la couleur du parcours, la manière d'indiquer le départ et l'arrivée) et visualiser les effets en rafraîchissant la page (par F5).

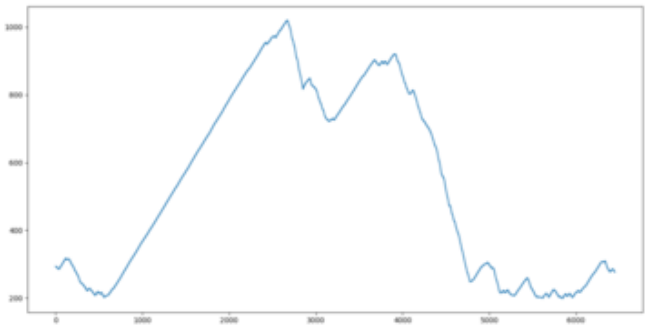
profil

On peut à présent exploiter les données stockées dans `data` et chercher à produire un **profil du parcours**.

En effet, `data[1]` est une liste qui contient toutes les altitudes.

On pourra dans un premier temps représenter ces altitudes en supposant que la distance parcourue entre deux marqueurs est la même.

On obtient une courbe du type :



réponse

```
###profil du parcours
 import matplotlib.pyplot as plt
 abscisse=[]
 for i in range(L):
 abscisse.append(i)
 plt.plot(abscisse, data[1])
 plt.show()
```

kilométrage

On peut chercher à donner un **cumulé du kilométrage** qui serait indiqué par exemple sur chaque 'popup'

On utilise pour cela la fonction suivante (pas évidente du tout !!)

code de la fonction distance

```
###distance entre deux points
 import math#pour utiliser la fonction racine carrée (sqrt)

 def distance(origin, destination):#fonction donnant la distance entre deux points connaissant leurs
coordonnées gps ; la variable "origin" est un couple de valeurs (long, lat) de même que la variable
"destination"
 lat1, lon1 = origin
 lat2, lon2 = destination
 radius = 6371 # km

 dlat = math.radians(lat2-lat1)
 dlon = math.radians(lon2-lon1)
 a = math.sin(dlat/2) * math.sin(dlat/2) + math.cos(math.radians(lat1)) \
 * math.cos(math.radians(lat2)) * math.sin(dlon/2) * math.sin(dlon/2)
 c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
 d = radius * c

 return d#distance donnée en km
```

Ainsi, `distance( data[0][0] , data[0][1])` donne la distance (en km) entre les deux premiers points relevés par le GPS.

A faire : stocker dans une liste `d_entre` les distances entres deux points GPS, puis dans une liste `d_cum` la distance parcourue depuis le départ.

réponse

```
##distance et distance cumulée
 d_entre=[]
 d=0
 d_cum=[]

 for i in range(L-1):
 d_entre.append(distance(data[0][i] , data[0][i+1]))
 d=d+distance(data[0][i] , data[0][i+1])
 d_cum.append(d)
```

des points sur le parcours

On peut également positionner **des points le long du parcours qui indiqueront par exemple l'altitude** du lieu.

Les points étant trop nombreux, on ne les placera pas tous.

Voici le code :

```
pas=100
 for i in range(0, L, pas):#on positionne des marqueurs tous les ...pas... points
 folium.Circle([data[0][i][0], data[0][i][1]], radius=2, popup="point n°"+str(i)+" :  "+str(data[1][i])+"
d'altitude").add_to(c)
```

A faire : modifier l'indication donnée par chaque point afin qu'elle renseigne sur (au choix) : la distance partielle parcourue, le dénivelé cumulé.

remarques

On peut à présent tracer le '**vrai profil**' du parcours, en construisant les points d'abscisse 'distance parcourue' et d'ordonnée 'altitude'

réponse

```
##vrai profil
 import matplotlib.pyplot as plt
 d_cum.insert(0, 0)#on ajoute à la liste d_cum 0 en première position
 plt.plot(d_cum, data[1])#pour rappel, data[1] est une liste contenant les altitudes des points
 plt.show()
```

Détermination du dénivelé cumulé (positif et négatif)

réponse

```
##dénivelé cumulé
 cum_pos=0
 cum_neg=0

 for i in range(L-1):
 delta=data[1][i+1]-data[1][i]
 if delta>0:
 cum_pos+=delta
 else:
 cum_neg-=delta
```