# SNT - Localisation, cartographie et mobilité TP 3

## Cartographie, itinéraires : présentation

- Utiliser un algorithme pour calculer un itinéraire en ligne et par programmation.
- Parcourir un graphe.
- Partager un itinéraire au format GPX

---

Comme vous avez pu le constater quand vous avez travaillé sur Open Street Map, il est possible de définir les voies de communication (principalement les routes). La base de données OSM contient donc les routes.

En utilisant ces données, il est possible de développer des outils capables de calculer des itinéraires routiers ou piétons (comme le propose les logiciels "GPS" : Waze, ViaMichelin, Mappy, Google maps...)

Vous renseignez :

- votre lieu de départ,
- votre lieu d'arrivée,

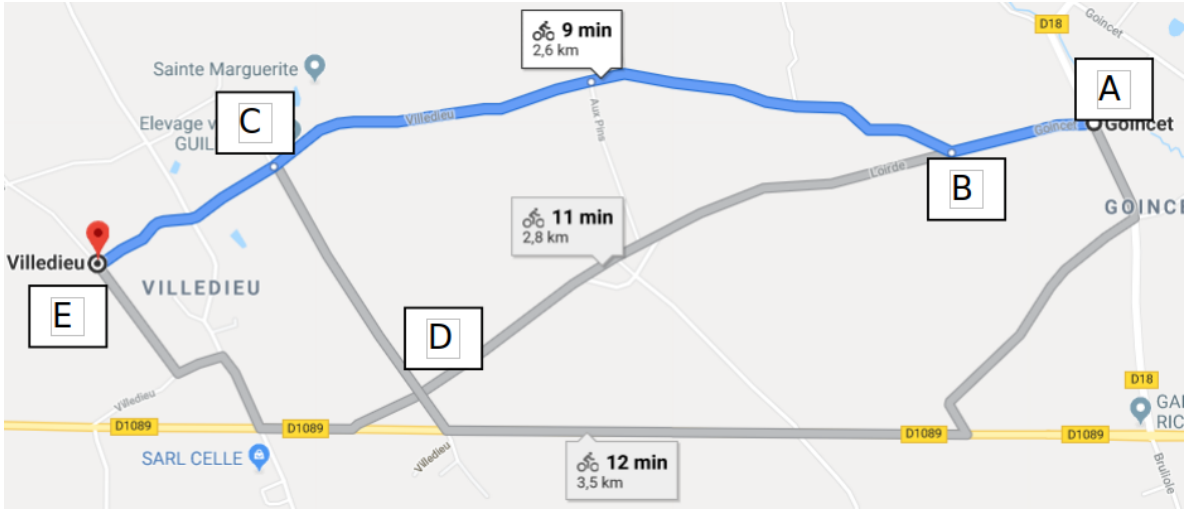
Puis le logiciel calcule votre itinéraire.

Ce calcul d'itinéraire repose sur des algorithmes relativement complexes, par exemple **l'algorithme** de **Dijkstra** qui permet d'obtenir le plus court chemin entre deux points.

### Activité 1 : algorithme de Dijkstra et parcours de graphe

Sans entrer dans les détails, l'algorithme de Dijkstra travaille sur des graphes (chaque ville est un sommet du graphe et chaque route est une arête du graphe)

Il existe bien sur de nombreux sites et applications pour trouver le meilleur itinéraire qui peut changer en cours de route en fonction de l'état de la circulation.



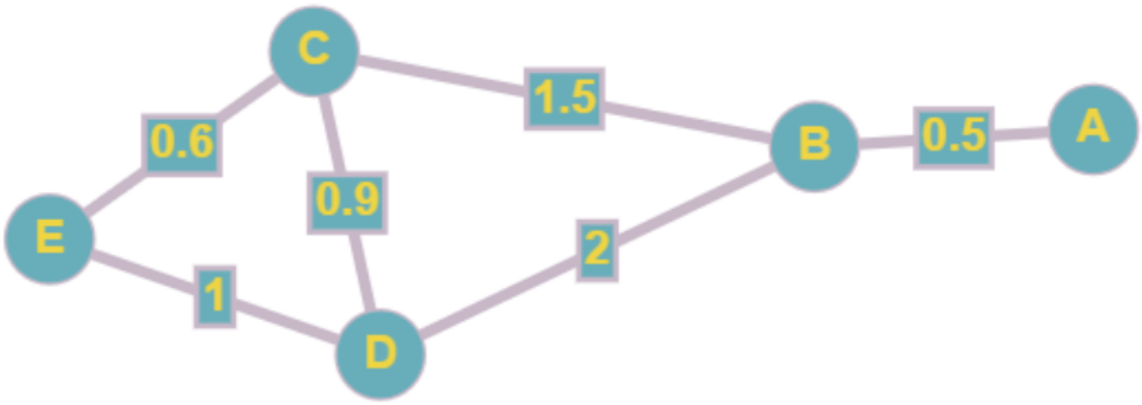

L'application nous propose plusieurs solutions, dont une qui est la plus rapide et ici une qui est la plus courte.

**1.1 Donner les différents chemins possibles ( en utilisant les lettres)**

```
Pour aller de A à E :
- A-> B-> C-> E
- A-> B-> C -> D-> E
- A-> B-> D -> C-> E
- A-> B-> D -> E

- A-> D-> E
- A-> D-> C-> E
```

Comment trouver le plus court des chemins possibles ?

Il est possible de modéliser un réseau routier à l'aide d'un graphe. Par exemple :




Chaque embranchement ou changement de direction est modélisé par un sommet, et une arête correspond à une voix de circulation. Le nombre correspond à la distance ( en km par exemple) qui sépare deux sommets.

**1.2 Donner alors les longueurs des différentes chemins pour aller de A à E**

| Chemins | Distance parcourue |
|---|---|
| ABCE | 2.6 |
| ABCDE | 3.9 |
| ABDCE | 4 |
| ABDE | 3.5 |

**1.3 Utiliser l'algorithme de Dijkstra en ligne**

> - Rendez-vous sur : https://graphonline.ru/en/
> - Faire **graph/import from file** pour charger le fichier : **graph_trajet.graphml**

On retouve le même graphe que précédemment !

Cette application permet de trouver le plus court chemin entre deux points à sélectionner :

> - Faire Algorithm/Find shorted path using Dijkstra algo.
> - Noter le chemin trouvé

Conclusion :

```
Comparer avec les calcul du tableau 1.2 :
Chemin le plus court : <ABCD> avec une valeur de 2.6 km
```

**1.4 Visionnez cette vidéo pour en savoir plus sur l'algorithme de Dijkstra.**

> https://www.youtube.com/watch?v=MybdP4kice4

**1.5 Utiliser l'algorithme de D pour remplir le tableau suivant et vérifier le plus court chemin**

| A | B | C | D | E |
|---|---|---|---|---|
| 0A | | | | |
| 0A | 0.5B | | | |
| | 0.5B | 2B | 2.5B | |
| | | 2B | 2.9C | 2.6C |
| | | | | 2.6C |

```
Conclusion :
On retrouve le Chemin le plus court <ABCD> avec une valeur de 2.6 km
```

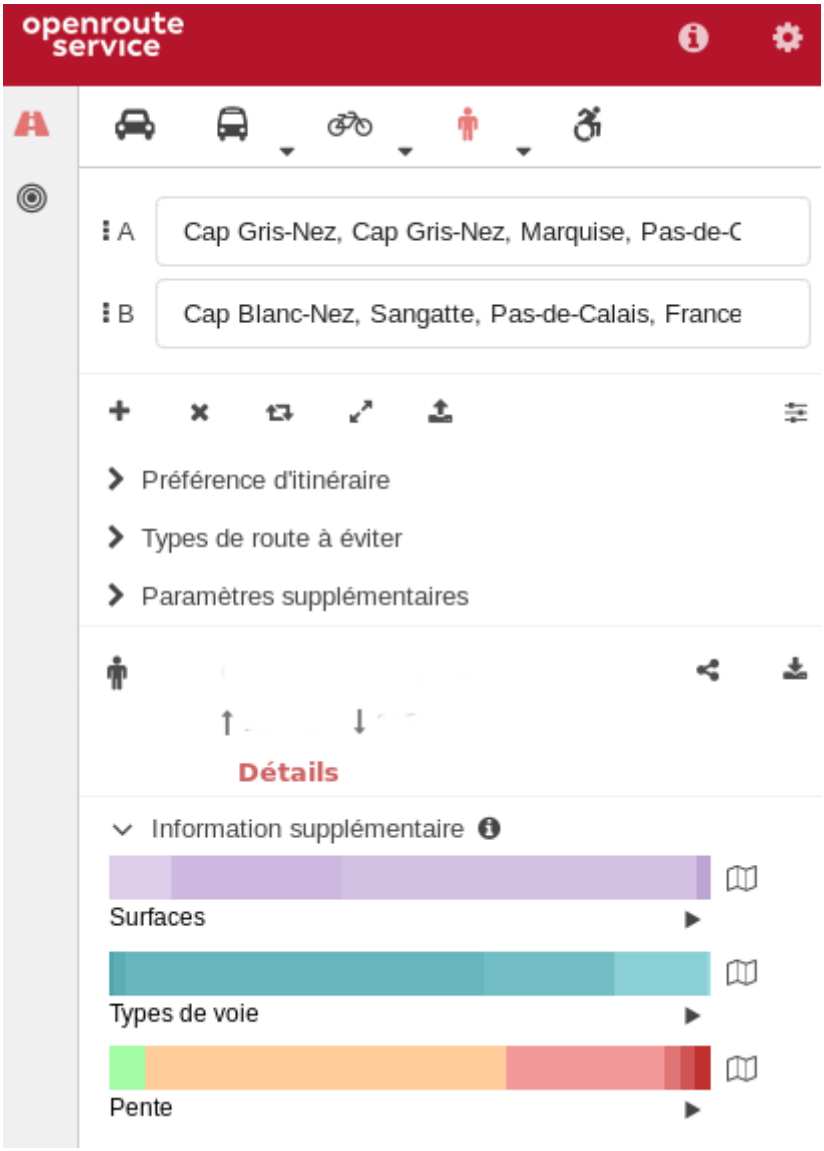
Pour aller plus loin sur l'algorithme de Dijkstra : Algorithme de Dijkstra

## Activité 2 : Itinéraire calculé à partir d'une application Web

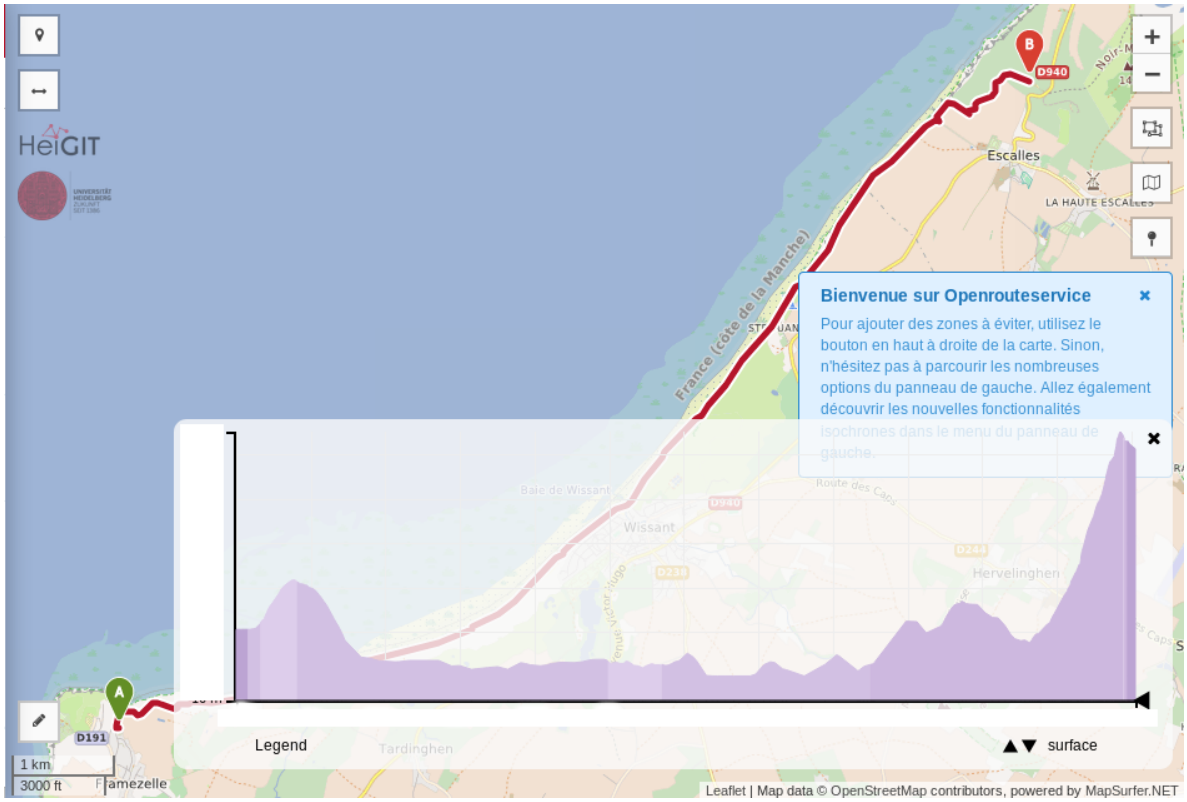
Prenons l'air et direction les caps !

> Rendez-vous sur : https://maps.openrouteservice.org/

Tracer un itinéraire à pied du **cap gris nez** au **cap blanc nez**




> Utiliser la carte pour trouver l'altitude la plus élevée sur votre parcours




**2.1 Définir un itinéraire et ses caractéristiques, trajet des 2 caps**

```
Altitude maximale sur le parcours : ...
Durée du trajet : ...
Distance parcourue : ...
Cumul de dénivellé >0 ...
Cumul de dénivellé <0 ...
```

On peut aussi enregistrer le trajet sous le **format GPX** par exemple (voir plus loin)

Pour approfondir : https://wiki.openstreetmap.org/wiki/FR:Calcul_d%27itin%C3%A9raires

## Activité 3 : Itinéraire calculé sous python avec le module pyroutelib3

le fichier **pyroutelib3.py** doit être dans le même dossier que le fichier itineraire1.py qui contient le contenu suivant :

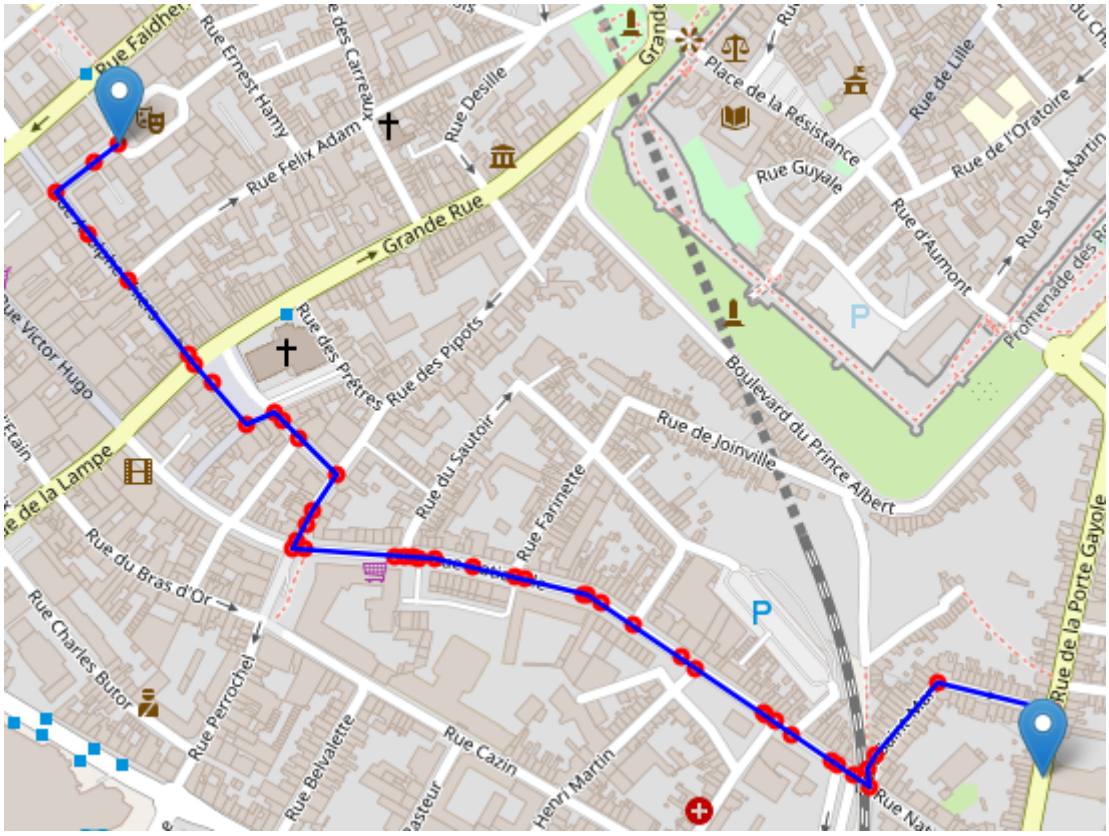
### 3.1 Les étapes d'un programme de calcul d'itinéraires

Charger le fichier suivant **itineraire1.py** sous Thonny et exécutez le.

```
from pyroutelib3 import Router
import folium
import os

"""
    créer la carte (fichier HTML + openstreetmap + javascript)
    Attention réponse assez longue : parfois plusieurs minutes !

    https://github.com/MKuranowski/pyroutelib3/issues/3

"""

c = folium.Map(location=[50.723960,1.614347], zoom_start=15)

coor_depart= [ 50.72046, 1.61538]       # lycée Branly
coor_arrivee= [50.72475, 1.60548]      # théatre Monsigny

#coor_arrivee= [50.72712, 1.60874]      # bug

# marquer sur la carte les points de départ et d'arrivée
folium.Marker(coor_depart,popup="Départ").add_to(c)
folium.Marker(coor_arrivee,popup="Arrivée").add_to(c)

print("debut routage, veuillez patientez svp ...")
#router = Router("car","map.osm")
router = Router("foot")
depart = router.findNode(coor_depart[0], coor_depart[1])

print(depart)
arrivee = router.findNode(coor_arrivee[0], coor_arrivee[1])
print(arrivee)

routeLatLons=[coor_depart,coor_arrivee]
status, route = router.doRoute(depart, arrivee)
if status == 'success':
    print("route trouvée")
    routeLatLons = list(map(router.nodeLatLon, route))
else:
```

```
    print("route pas trouvée !")

print(routeLatLons)
for coord in routeLatLons:
    coord=list(coord)
    folium.CircleMarker(coord,radius = 3,fill=True, color='red' ).add_to(c)


folium.PolyLine(routeLatLons, color="blue", weight=2.5, opacity=1).add_to(c)

print("Le fichier HTML/CARTE est disponible")
fichier_carte = 'routage_branly-monsigny.html'
c.save(fichier_carte)

# ouvrir le fichier dans le navigateur
rep_cour = os.getcwd()
fichier = 'firefox file://'+rep_cour +'/'+ fichier_carte
os.popen(fichier);
```

On obtient :




Quelques explications sur ce programme :

- Nous commençons par importer la bibliothèque "pyroutelib3" avec la première ligne "from pyroutelib3 import Router"
- Dans notre cas, nous voulons parcourir le trajet à pied ("foot"), mais il est possible de choisir d'autres moyens de transport : car, cycle, foot, horse, tram, train
- Ensuite, on définit le point de départ et le point d'arrivée. Nous avons "router.findNode(latitude, longitude)", il suffit de renseigner la latitude et la longitude du lieu.
- La ligne "status, route = router.doRoute(depart, arrivee)" permet d'effectuer le calcul de l'itinéraire.

- La variable "routeLatLons" contient la liste des coordonnées des points de cheminement (points qui constituent le chemin entre le point de départ et le point d'arrivée)

### 3.2 Créer votre carte avec le tracé d'un itinéraire de votre choix

Modifiez le programme **itineraire1.py** pour en faire **itineraire_perso.py** : pour calculer le trajet entre deux villes de votre choix avec le moyen de transport de votre choix.

```
# votre programme ici :
```

## Activité 4 : Partager un trajet ou un fichier de randonnée au format GPX

le format de fichier **GPX** (GPS eXchange Format) est un format de fichier permettant l'échange de coordonnées GPS. Ce format permet de décrire une collection de points utilisables sous forme de point de cheminement (*waypoint*), trace (*track*) ou itinéraire (*route*).

### 4.1 Programmer l'affichage d'un itinéraire

Charger le fichier suivant **affiche_randonnee_gpx.py** sous Thonny et exécutez le.

```
import gpxpy
import gpxpy.gpx
import folium
import os

gpx_file = open('Leubringhen-randoDuChef.gpx', 'r')

gpx = gpxpy.parse(gpx_file)
points = []
for track in gpx.tracks:
    for segment in track.segments:
        for point in segment.points:
            points.append([point.latitude, point.longitude])

# vérifier qu'il y a au moins 2 points enregistrés !
if len(points) > 1 :
    coord_depart=points[0]
    coor_arrivee=points[len(points)-1]

    #carte centrée sur le début de la randonnée
    c = folium.Map(location=coord_depart, zoom_start=13)

    for coord in points:
        #ajouter un cercle rouge à chaque noeud
        folium.CircleMarker(coord,radius = 2,fill=True, color='red' ).add_to(c)

    folium.Marker(coord_depart,popup="Départ du 2017-06-24 à
12:39:00").add_to(c)
    folium.Marker(coor_arrivee,popup="Fin de la randonnée à 17:48:00").add_to(c)
    #tracer le chemin en bleu
    folium.PolyLine(points, color="blue", weight=2.5, opacity=1).add_to(c)
    # enregistrer la carte
    fichier = 'carte_openmap.html'
    c.save(fichier)
```

```
    # ouvrir le fichier dans le navigateur
    rep_cour = os.getcwd()
    commande = 'firefox file:///'+rep_cour +'/'+ fichier
    os.popen(commande);
else :
    print("la randonnée ne contient pas de points enregistrés !")
```

Le fichier **affiche_randonnee_gpx-BN-GN_duree.py** peut aussi être utilisé (en modifiant : gpx_file = open('Leubringhen-randoDuChef.gpx', 'r'))

Il contient en plus l'affichage de la durée et de la distance du parcours, quand on click sur le marqueur "Arrivée" de la carte créée. (voir : Pour aller plus loin 4.2.2)

**4.2 Pour aller plus loin :**

#### 4.2.1 Centrer la carte - Algorithme.

Retour sur l'activité 2

> Rendez-vous sur : https://maps.openrouteservice.org/

> Enregistrer le trajet du **cap gris nez** au **cap blanc nez** au format GPX ( exporter la route) fichier : **DuCapGNauCapBN.gpx**
>
> Modifier le fichier **affiche_randonnee_gpx.py** pour afficher ce trajet.
>
> il suffit de modifier la ligne de code : **gpx_file = open('Leubringhen-randoDuChef.gpx', 'r')**
>
> par : **gpx_file = open('DuCapGNauCapBN.gpx', 'r')**

Examiner le code et dites pourquoi la carte est relativement bien centrée sur le trajet

```
...
#La carte centrée sur le début de la randonnée avec : coord_depart=points[0]
    c = folium.Map(location=coord_depart, zoom_start=13)
```

```
...
la carte est centrée sur les coordonnées de départ de la rando.
```

> Modifier ce code pour centrer sur le point milieu du trajet. Le calcul proposé doit pouvoir fonctionner pour un autre trajet !

```
#...
    coord_depart=points[0]
    coor_arrivee=points[len(points)-1]

    # moyenne des Latitudes longitudes pour les points de départs et d'arrivée
    # pour centrer la carte sur le milieu
    # Bien sûr, ne marche pas pour un trajet aller/retour
    latMoyen=(coord_depart[0] + coor_arrivee[0])/2
    longMoyen=(coord_depart[1] + coor_arrivee[1])/2

    #carte centrée sur le début de la randonnée
    c = folium.Map(location=[latMoyen,longMoyen], zoom_start=13)
```

#### 4.2.2 Calcul de la durée du trajet - Algorithme.

> Vous allez créer un algorithme de conversion de secondes en heures minutes.

On peut ajouter le code suivant dans le programme précédent inspiré de :
**affiche_randonnee_gpx.py**

```
    # obtention de la durée en seconde avec le module gpx
    duree_en_seconde = gpx.get_duration()
    print("Durée du trajet : ",duree_en_seconde," secondes")

    # conversion en  heure minutes
    ?????
```

**Modifier ce code pour**

- calculer les heures
- calculer les minutes
- afficher les heures et minutes

Exemples:

- si duree_en_seconde=3600, on doit afficher 1 h
- si duree_en_seconde=3700, on doit afficher 1 h 1 m
- si duree_en_seconde=800, on doit afficher 13 m

On pourra analyser cette ligne de code :

```
int(duree_en_seconde/3600)
```

```
    #   une solution :
    # obtention de la durée en seconde avec le module gpx
    duree_en_seconde = gpx.get_duration()
    # conversion en  heure minutes
    h = int(duree_en_seconde/3600)
    if h > 0 :
        m = int((duree_en_seconde/3600 - h)*60)
        duree_en_heure_min = str(h)+' h '+ str(m) +' m'
    else :
        m = int(duree_en_seconde/60)
        duree_en_heure_min = str(m) +' m'
    print("Durée du trajet : ",duree_en_heure_min)
```

Le code complet pour afficher la durée du parcours ( et la distance parcourue) quand on click sur le marqueur "Arrivée" de la carte créée. fichier **affiche_randonnee_gpx-BN-GN_duree.py**

```
import gpxpy
import gpxpy.gpx
import folium
import os

gpx_file = open('DuCapGNauCapBN.gpx', 'r')

gpx = gpxpy.parse(gpx_file)
points = []
for track in gpx.tracks:
    for segment in track.segments:
        for point in segment.points:
```

```
            points.append([point.latitude, point.longitude])

length_2d=int(gpx.length_2d())
print("Longueur du trajet : ",length_2d," km")

# vérifier qu'il y a au moins 2 points enregistrés !
if len(points) > 1 :
    coord_depart=points[0]
    coor_arrivee=points[len(points)-1]

    # moyenne des Latitudes longitudes pour les points de départs et d'arrivée
    # pour centrer la carte sur le milieu
    # Bien sûr, ne marche pas pour un trajet aller retour
    latMoyen=(coord_depart[0] + coor_arrivee[0])/2
    longMoyen=(coord_depart[1] + coor_arrivee[1])/2

    #carte centrée sur le début de la randonnée
    c = folium.Map(location=[latMoyen,longMoyen], zoom_start=13)

    # obtention de la durée en seconde
    duree_en_seconde = gpx.get_duration()
    # conversion en  heure minutes
    h = int(duree_en_seconde/3600)
    if h > 0 :
        m = int((duree_en_seconde/3600 - h)*60)
        duree_en_heure_min = str(h)+' h '+ str(m) +' m'
    else :
        m = int(duree_en_seconde/60)
        duree_en_heure_min = '0 h '+ str(m) +' m'
    print("Durée du trajet : ",duree_en_heure_min)

    for coord in points:
        #ajouter un cercle rouge à chaque noeud
        folium.CircleMarker(coord,radius = 2,fill=True, color='red' ).add_to(c)

    folium.Marker(coord_depart,popup="Départ de la randonnée du 2017-06-24 à
12:39:00").add_to(c)
    folium.Marker(coor_arrivee,popup="Fin de la randonnée à 17:48:00 - Longeur :
"+ str(length_2d) +" km" +" Durée du trajet : " +
str(duree_en_heure_min)).add_to(c)
    #tracer le chemin en bleu
    folium.PolyLine(points, color="blue", weight=2.5, opacity=1).add_to(c)
    # enregistrer la carte
    fichier = 'carte_openmap.html'
    c.save(fichier)

    # ouvrir le fichier dans le navigateur
    rep_cour = os.getcwd()
    commande = 'firefox file://'+rep_cour +'/'+ fichier
    os.popen(commande);
else :
    print("la randonnée ne contient pas de points enregistrés !")
```