SAE 105 : Traitement de données – CAHIER Des CHARGES

Objectifs

Compétences professionnelles

En tant que futur professionnel R&T, vous serez régulièrement amené à **traiter des données provenant du système d'information de votre entreprise** (extraction d'information comptable par exemple), pour **les présenter de façon pertinente et synthétique à votre hiérarchie ou à vos collaborateurs**. Ces traitements pouvant être **récurrents** (mensualisation de bilan par exemple), ils gagnent à être **automatisés** par des programmes/scripts informatiques pour les gérer de façon efficace.

La SAE 105 vous projette dans cette **situation professionnelle**. Elle vous propose d'exploiter des **données issues de fichiers csv**, de les traiter et de les présenter pour vos **collaborateurs universitaires** (services administratifs, financiers, collègues enseignants, collègues étudiants, …).

Contexte de réalisation

La **SAE 105** se déroulera selon le principe suivant :

- Vous êtes en charge, à titre **individuel**, d'un projet de développement informatique.
- Ce projet vous est confié par votre **hiérarchie** (l'enseignant mandataire en charge de votre groupe de TD).
- Votre hiérarchie **impose un cahier des charges fonctionnel et technique** (décrits par la suite), **qui devra être respecté avec des livrables fournis dans les temps** (**dépôt sur Moodle à la fin de chaque séance de TP**) ouverture du dépôt 2h après le début de la séance de TP et fermeture du dépôt 10 minutes après la fin du TP. Tout travail non rendu à temps sera sanctionné.

Organisation du temps de travail

Le projet devra être réalisé dans des **plages de travail réservées, obligatoires** et inscrites dans votre emploi du temps (charge homme/mois) :

- une partie des séances de travail seront **encadrées** (2 TPs) par des collaborateurs techniques (enseignants : Jean-Pierre et moi) que vous pourrez solliciter en cas de questions
- une autre partie des séances seront à faire **en autonomie** (7 TPs non encadrés) pour avancer votre travail et à l'issue desquelles un certain nombre de fonctionnalités devront être livrées (dépôt sur Moodle du programme développé en Python).

7 TPs de 3H non encadrés (sem48 à sem3)
2 TPs de 3h encadrés sem49 et sem1 pour répondre à vos questions
Rendre un petit rapport de 5 pages qui explique le projet et les fonctions qui sont utilisées. Pour chaque fonctions et procédures il faudra donner le nom des paramètres d'entrée et de sortie ainsi que leurs types. Il faudra également décrire brièvement ce que fait ce code. Ces informations seront à mettre dans le code en commentaires.

Pour l'**évaluation** en fin de la **semaine 3** :

- **2h de CTRL** (**14 pts**) dans lequel on vous donne un fichier à traiter et on vous demande de coder différentes fonctions que vous avez écrites lors de cette SAE.
- Approximativement : 1 pt pour l'assiduité aux séances non encadrées et dépôt sur Moodle, 0.5 pt pour le respect des consignes, 1.5 pts pour le code commenté et mini-rapport, 0.5 pt par étape qui fonctionne

1 projet par étudiant, donc à faire tout seul, pour vous habituer à travailler en autonomie.

Le fichier **villes_france.csv** est disponible sur Moodle. L'extension csv = comma separated values, soit données séparées par une virgule.

C'est ce fichier qu'il faudra traiter. Il se trouve également : https://sql.sh/736-base-donnees-villes-francaises

Vous pourrez extraire une partie du fichier pour faire des tests de certaines fonctionnalités avant de les tester sur l'ensemble du fichier.

Le fichier à traiter comporte **27** champs (colonnes) et **36700** lignes :

1 ligne = 1 ville

"1","01","ozan","OZAN","ozan","Ozan","O250","OSN","01190","284","01284","2","26","6","618","469","500","93","6.6","4.91667","46.3833","2866","51546","+45456","462330","170","205"

- **Département :** numéro du département.
- **Slug :** identifiant unique en minuscule, sans accent et sans espace. Peut servir pour faire les URLs d'un site web.
- **Nom :** nom en majuscule et sans accent.
- **Nom simple :** nom en minuscule, sans accent et avec les tirets remplacés par des espaces. Peut être utilisé pour faire une recherche lorsqu'on ne sait pas si le nom de ville possède un tiret ou des espaces (ex : "Saint-Étienne" possède un tiret comme séparateur, tandis que "Le Havre" possède un espace comme séparateur)
- **Nom reel :** nom correct avec les accents
- **Nom soundex :** soundex du nom de la ville (permet de trouver des villes qui se prononcent presque pareil) **[ajouté le 31/10/2013]**
- **Nom metaphone :** metaphone du nom de la ville (même utilité que le soundex) **[ajouté le 31/10/2013]**
- **Code postal :** code postal de la ville. Si la ville possède plusieurs codes postaux, ils sont tous listés et séparés par un tiret **[ajouté le 31/10/2013]**
- **Numéro de commune :** numéro de la commune dans le département. Combiné avec le numéro de département, il permet de créer le code INSEE sous 5 caractères.
- **Code commune (ou code INSEE) :** identifiant unique sous 5 caractères
- **Arrondissement :** arrondissement de la ville
- **Canton :** canton de la ville
- **Population en 2010 :** nombre d'habitants lors du recensement de 2010
- **Population en 1999 :** nombre d'habitants lors du recensement de 1999
- **Population en 2012 (approximatif) :** valeur exprimée en centaine
- **Densité en 2010 :** nombre d'habitants au kilomètre carré arrondie à l'entier. Calculé à partir du nombre d'habitant et de la surface de la ville **[corrigé le 02/07/2014]**
- **Surface / superficie :** surface de la ville en kilomètre carrée **[corrigé le 02/07/2014]**
- **Longitude/latitude en degré :** géolocalisation du centre de la ville. Permet de localiser la ville sur une carte (exemple : carte Google Maps) **[ajouté le 31/10/2013, corrigé le 07/11/2013]**
- **Longitude/latitude en GRD :** géolocalisation exprimée en GRD
- **Longitude/latitude en DMS (Degré Minute Seconde) :** géolocalisation exprimée en Degré Minute Seconde

- **Altitude minimale/maximale :** hauteur minimum et maximum de la ville par rapport au niveau de l'eau

De ce fichier on va extraire les **12 informations** suivantes sur lesquelles nous allons travailler :
Qui sont :
Numéro du département : 01
Nom de la ville : OZAN
Code postal : 01190
nbre d'habitants en 2010 : 618
nbre d'habitants en 1999 : 469
nbre d'habitants en 2012 : 500
densité : 93
surface : 6.6 km^2
longitude : 4.91667
latitude : 46.3833
altitude min : 170 m
altitude max : 205 m

TRAVAIL A REALISER :

Il faut impérativement respecter les notations données

Pour réaliser cette SAE, n'utiliser que les notions vues en Cours et TP, donc pas de dictionnaire

On vous donne **l'ossature complète du programme** à réaliser, qui au lancement **extrait les villes du fichier** et les stocke dans **une liste** puis qui affiche **un menu** : **taper 1** pour afficher le **nombre de villes** …, **taper 2** pour calculer des **Statistiques des villes** d'un département, **taper 3 pour calculer** la **distance Euclidienne et Géodésique** entre 2 villes, **taper 4** pour trouver **le plus court chemin entre 2 villes**, taper …. (menu à **respecter** et à **compléter** en fonction de l'avancement)

```
================ MENU ==================
taper 1: Nombre de villes en fonction de l'indicatif téléphonique
taper 2 : Extraire des Statistiques des Villes d'un département
taper 3: Distance Euclidienne et Géodésique entre 2 villes
taper 4: Plus court chemin entre 2 villes
F: pour finir
votre choix:
```

Pour les ***statistiques*** liées aux villes d'un département on sera redirigé sur un **sous-menu** lié aux différentes fonctionnalités à réaliser (décrites ci-dessous).

```
================ SOUS MENU : STATISTIQUES ==================
taper 1: Lister les 5 Villes ayant le plus/le moins d'habitants
taper 2: Afficher les 10 Villes en fonction de la DENSITE sur une carte
taper 3: Lister les 10 Villes ayant le plus fort/faible taux d'accroissement
taper 4: HISTOGRAMME des villes par habitants
taper 5: Lister les 5 Villes ayant la différence d'altitude max/min
taper 6: Afficher les 10 Villes en fonction de l'ALTITUDE sur une carte
Q: pour Quitter le sous-menu
votre choix:
```

Bien sûr il faudra faire les étapes 3 (respectivement 5) avant les étapes 4 (respectivement 6).

On donne **la fonction lire_fichier_csv(fichier)** qui extrait toutes **les lignes** du fichier csv et les **retourne** dans la liste nommée **uneListe**. Cette liste sera composée d'une chaîne de caractères (str = string).

Une ligne c'est :

"1","01","ozan","OZAN","ozan","Ozan","O250","OSN","01190","284","01284","2","26","6","618","469","500","93","6.6","4.91667","46.3833","2866","51546","+45456","462330","170","205"

A partir de la liste précédente **uneListe**, on vous donne **la fonction extract_infos_villes(uneListe)** qui retourne la liste **listeInfo** des **12 informations** retenues pour chaque ville. Chaque élément de la liste est au bon format ou type pour pouvoir l'utiliser par la suite.

Info = ["01","OZAN","01190", 618, 469, 500, 93, 6.6, 4.91667, 46.3833, 170, 205]

Dans un premier temps, relisez bien les lignes de codes qui permettent d'extraites les 12 informations importantes du fichier csv.

On vous demande de réaliser les étapes ci-dessous

- **Etape 1 :**
  A partir de la liste précédente (**listeInfo**) et en fonction de **l'indicatif téléphonique** (**indTel**) donc une région de France ("**01**" = île de France, "**02**" = Nord-Ouest, "**03**" = Nord-Est, "**04**" = Sud-Est, "**05**" = Sud-Ouest) associé à la liste des départements (**listeDept**) donné ci-dessous :
  - **1.1** Ecrire la procédure **appelNombre_Villes_Indicatif(indTel, listeInfo)** qui permet d'afficher le **nombre de villes de la région** associée à un **indicatif téléphonique**. Pour ce faire, **il faudra extraire les villes pour chaque département** associé à la région (**voir 1.2**).

    01 : [75,77,78,91,92,93,94,95] ➔ Pas attribué : on le laisse de côté

    02 : [14,18,22,27,28,29,35,36,37,41,44,45,49,50,53,56,61,72,76,85,974,976]

    03 : [02,08,10,21,25,39,51,52,54,55,57,58,59,60,62,67,68,70,71,80,88,89,90]

    04 : [01,03,04,05,06,07,11,13,15,2A,2B,26,30,34,38,42,43,48,63,66,69,73,74,83,84]

    05 [09,12,16,17,19,23,24,31,32,33,40,46,47,64,65,79,81,82,86,87,971,972,973,975,977,978]

    source : https://fr.wikipedia.org/wiki/Liste_des_indicatifs_téléphoniques_en_France

  - **1.2** Donc, écrire la fonction **extract_villes_depart_indicatif(listeDept, listeInfo)** qui extrait l'ensemble des villes des **xx** départements en fonction de **l'indicatif téléphonique**. La fonction devra retourner le **nombre de villes** que l'on affichera dans la console. Elle devra également **sauvegarder dans un fichier texte** (voir ci-dessous pour le nom du fichier, et le formatage attendu) la liste des villes. Par exemple le **02** comporte les **22 départements**.
    Voir le fichier **DOC2_Attribution_DEPT_2023_2024.pdf** : on aura 1 groupe de TP par indicatif téléphonique (03 = TP-A1, 04 = TP-A2, 05 = TP-B1, 02 = TP-B2). La fonction doit retourner un **fichier texte**, contenant le nom des villes pour chaque département.

Les fichiers texte auront les noms suivants :

INDICATIF 02 : **NO02.txt (**Nord-Ouest 02) : 22 départements
INDICATIF 03 : **NE03.txt** (Nord-Est 03) : 23 départements
INDICATIF 04 : **SE04.txt** (Sud-Est 04) : 25 départements
INDICATIF 05 : **SO05.txt** (Sud-Ouest 05) : 27 départements

Par exemple, les 5 premières lignes des fichiers doivent être sous la forme :

NO02.txt	NE03.txt	SE04.txt	SO05.txt
1 SAINT-PIERRE-CANIVET (14)	1 GERCY (02)	1 OZAN (01)	1 BEDEILHAC-ET-AYNAT (09)
2 GRANGUES (14)	2 VILLEMONTOIRE (02)	2 CORMORANCHE-SUR-SAONE (01)	2 CONTRAZY (09)
3 VIEUX-BOURG (14)	3 BRANCOURT-LE-GRAND (02)	3 PLAGNE (01)	3 SUC-ET-SENTENAC (09)
4 ENGLESQUEVILLE-LA-PERCEE (14)	4 LICY-CLIGNON (02)	4 TOSSIAT (01)	4 VENTENAC (09)
5 RANCHY (14)	5 PROISY (02)	5 POUILLAT (01)	5 THOUARS-SUR-ARIZE (09)

Pour chaque groupe de TP, nous avons réparti **1 département** par étudiant (voir fichier Doc2_Attribution_DEPT_2023_2024.pdf)

- **1.3** En fonction du département qui vous a été attribué, écrire une **fonction** nommée extract_villes_NumDepart(*numDept*, listeInfo) qui extrait de la listeInfo toutes les villes du numéro de département passé en paramètre, et retourne le **nombre de villes** de ce département et la **liste** (listeInfoDept**) des villes** de ce département qui contient les **12 informations** précédemment extraites. Cette fonction **sauvegarde** ces villes dans le fichier nommé villes_n°Dept.txt (*exemple villes_74.txt*).
  ['74', 'ANNECY', '74000', 50379, 50324, 50100, 3690 , 13.65, 6.11667, 45.9, 418, 926]
  ….

  Remarque : On peut extraire le département de **listeInfo**, ou utiliser listeInfoDept dans tout ce qui suit. Donc, on peut ne pas mettre *numDept*, le département, en paramètre d'entrée de la fonction

- Etape 2 :

- Faire **3 procédures** qui sauvegardent dans **les résultats dans des fichiers** :
  Pour trier la liste des villes on utilisera l'algorithme **tri bulle** étudié dans le **TP7**.
  - **a)** Ecrire la procédure **MinMax5Villes_Habitants(***numDept***,** listeInfo**)** ou **MinMax5Villes_Habitants(**listeInfoDept**)** qui permet de sauvegarder le nom des **5 villes** qui ont **le plus (respectivement le moins) d'habitants** en **2010**. Chaque fichier contient les **12 informations** des 5 villes retenues. Nom des fichiers Top5Villes_n°Dept.txt **(nomfich1)**, (respectivement Min5Villes_n°Dept.txt **(nomfich2)**)

par exemple : pour le département 40 : fichier Min5Villes_40.txt

[40, 'BAUDIGNAN', '40310', 42, 36, 0, 1.0, 23.3, 0.05, 44.0833, 104, 161]
[40, 'ARX', '40310', 67, 57, 100, 2.0, 24.18, 0.066667, 44.1167, 90, 159]
[40, 'MONTEGUT', '40190', 71, 81, 100, 14.0, 4.82, -0.2, 43.8667, 63, 110]
[40, 'LUSSAGNET', '40270', 81, 82, 100, 9.0, 8.43, -0.233333, 43.7667, 87, 141]
[40, 'LAURET', '40320', 82, 74, 100, 11.0, 7.34, -0.333333, 43.5667, 117, 237]

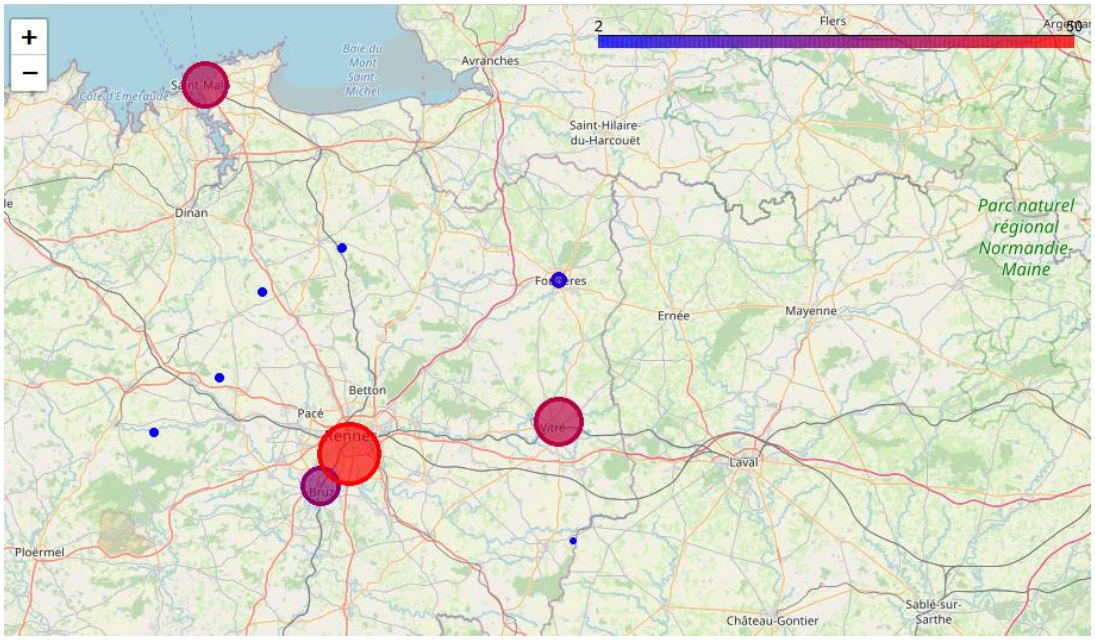
- **b)** Ecrire la procédure **mapTenVilles(nomfich1, nomfich2)** qui permet d'**afficher** ces **10 villes** sur OpenStreetMap, avec des couleurs différentes en fonction des densités : *un cercle centré sur la ville de rayon proportionnel à la **densité** avec une couleur plus ou moins intense en fonction de la **population***. **nomfich1** et **nomfich2** étant les 2 noms de fichiers de la partie **a)**.

  Remarque : pour afficher une ville sur OpenStreetMap voir : la bibliothèque **folium** et **branca** :
  import folium, branca

  Pour installer une bibliothèque, aller dans **File** puis **Settings**, puis **Project** …, puis **Python interpreter**. Une fenêtre s'ouvre, et au-dessus de **package**, cliquer sur + pour ajouter la bibliothèque. Une autre fenêtre s'ouvre dans laquelle vous mettez **folium**, puis en bas de la fenêtre, cliquez sur **Install Package**.

  …..
  Voir un exemple sur **moodle**, le scrip test_geoloc.py crée un fichier **mapIUT.html**, que l'on peut ouvrir avec un navigateur.
  Pour les cercles avec la densité, voir la bibliothèque **branca**.
  Voir un exemple sur **moodle** : Utilisation de FOLIUM et **BRANCA** (Affiche villes sur OpenStreetMap)

  Voir ci-dessous un exemple avec le département de l'**Ile-et-Vilaine (35)**



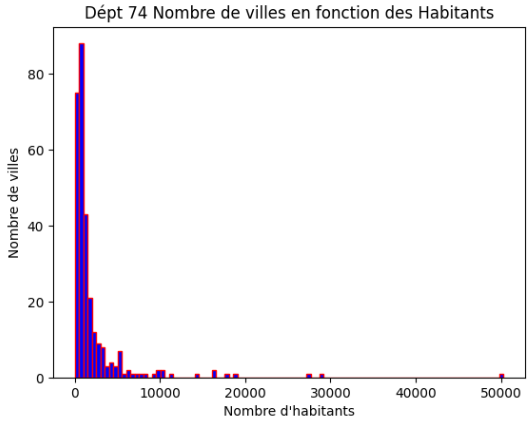
- **c)** Ecrire la procédure **MinMax10Accroissement(**listeInfoDept**)** qui permet de sauvegarder dans un fichier les **10 villes** qui ont eu **le plus fort accroissement** (respectivement la plus forte baisse) de sa population entre **1999** et **2012**. Les fichiers nommés TopAcc10Villes_n°Dept.txt (respectivement TopBaisse10Villes_n°Dept.txt) contiendront, le numéro du département, le nom de la ville et la valeur de l'accroissement.

  Par exemple le contenu du fichier TopAcc10Villes_35.txt est :
  35,MORDELLES,1198

35,PACE,1210
35,VITRE,1376
35,CHANTEPIE,1405
35,BETTON,1645
35,BAIN-DE-BRETAGNE,1684
35,LE RHEU,1965
35,BRUZ,2419
35,SAINT-JACQUES-DE-LA-LANDE,2617
35,JANZE,2739

- Etape 3 :

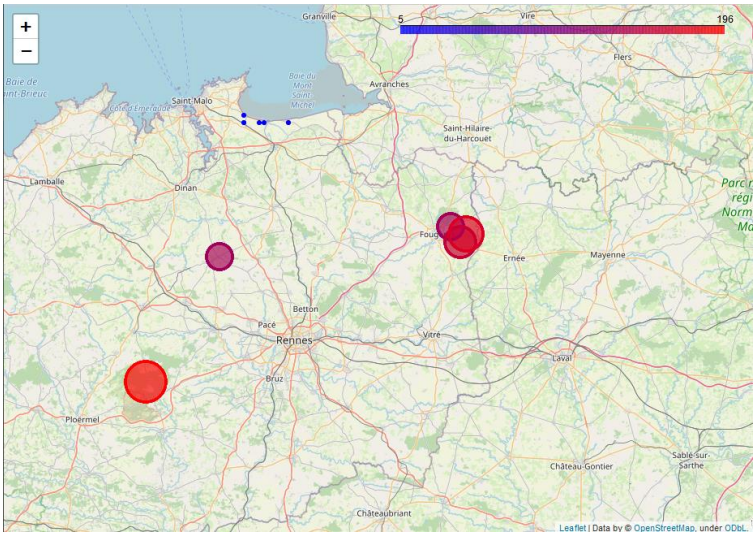
- Pour **le département** que vous devez traiter, écrire la procédure **traceHistoVilles(**listeInfoDept**)** qui permet de tracer l'**histogramme du nombre de villes** en fonction du nombre d'habitants en **2010**, de calculer la **moyenne** du nombre d'habitants, et **l'écart-type** (*dispersion autour de la moyenne*). Vous répartirez les nombres d'habitants en **100 classes** de **même amplitude**. Utiliser la bibliothèque matplotlib.pyplot pour faire ce tracé. Voir l'exemple ci-dessous :



- Pour ce même département faire la procédure **MinMax5Alt_Dept(**listeInfoDept**)** qui permet de sauvegarder le nom des **5 villes** qui ont **la plus forte (respectivement la plus faible) différence d'altitude**. Chaque fichier de 5 villes contient les **5 informations** suivantes : le département, le nom de la ville, la latitude, la longitude, et la différence d'altitude. Nom des fichiers Top5Alt_n°Dept.txt **(nomfich3)**, (respectivement Min5Alt_n°Dept.txt **(nomfich4)**).

  Contenu du fichier Top5Alt_35.txt
  35,LONGAULNAY,48.3,-1.93333,126
  35,LAIGNELET,48.3667,-1.15,127
  35,FLEURIGNE,48.3333,-1.11667,154
  35,LA CHAPELLE-JANSON,48.35,-1.1,170
  35,PAIMPONT,48.0167,-2.18333,196

- Ecrire la procédure **mapTenAlt(nomfich3, nomfich4)** qui permet d'**afficher** ces **10 villes** sur OpenStreetMap, avec des couleurs différentes en fonction de l'altitude : *un cercle centré sur la ville avec une couleur plus ou moins intense en fonction de l'**altitude***. **Nomfich3** et **nomfich4** étant les 2 noms de fichiers de la partie précédente.



Etapes 1 à 3 pré requises pour le contrôle TP de SAE

- **Etape 4 :**
  - A faire en fonction de votre groupe de TP
  - On demande d'écrire la **fonction dist_Euclidienne(ville1, ville2)** qui retourne la **distance euclidienne** entre les villes suivantes :
    - **a)** Paris – Marseille (TP-A1),
    - **b)** Lille – Bordeaux (TP-A2),
    - **c)** Strasbourg – Brest (TP-B1),
    - **d)** Calais – Toulouse (TP-B2),

  Remarque : on écrire la fonction **rechercheVille(ville1, listeInfo)** qui retourne les **12 informations** de la **ville1** passée en paramètre. Une fonction similaire sera utile lors de l'étape 5.

  - On demande d'écrire la **fonction dist_Geodesique(ville1, ville2)** qui retourne la **distance géodésique** entre ces 2 villes. Faites également le test avec 2 villes proches (*50 km*) de votre choix, mais appartenant au département qui vous a été attribué.

Remarque : pour le calcul voir Doc3 : calcul de la distance géodésique sur Moodle
ou le site https://en.wikipedia.org/wiki/Great-circle_distance

- **Etape 5 :**
  - On souhaite tracer sur **OpenStreetMap** la route qui permet d'aller de la **ville1** (départ) à la **ville2** (destination) en utilisant la **distance Géodésique**.

  - **A vous d'imaginer un algorithme qui réalise cette tâche**.

Si vous manquez d'imagination, vous pouvez suivre les instructions ci-dessous.

- Ecrire la fonction **ensembleVilles(centre, rayon, listeVilles)** qui retourne la liste **listeVillesTrouvees** des **300 villes** (maximum) appartenant au disque de **centre = villeX** et de **rayon = R** (pour les tests, on prendra R = 50 km).

- A partir de la liste des au plus **300 villes**, on ne conservera que la ville qui est la plus proche de la destination (**ville2**). Donc écrire la fonction **plusProche(listeVillesTrouvees, ville2)** qui retourne la ville qui est la plus proche de la **ville2**.

- Ecrire la fonction **isInDisque(uneVille, uneListe, rayon)** qui retourne un **booléen** qui vaut **True** si la ville destination est dans le disque de rayon voulu, et **False** dans le cas contraire.

- Ecrire la fonction **parcoursVilles(ville1, ville2, listeInfo, rayon)** qui permet de sauvegarder l'ensemble des villes traversées pour aller de la **ville1** à la **ville2** en utilisant **l'algorithme suivant** : on débute avec la **ville1** et on cherche au plus **300 villes** qui sont dans le disque de centre (**ville1**) et de rayon **50 km**. On obtient un ensemble de villes sous la forme d'une liste. A partir de cette liste, on ne conservera que la **villeX** qui est la plus proche de la destination **ville2**, au sens de la distance euclidienne. La **villeX** devient alors le nouveau centre et on procède de proche en proche jusqu'à arriver à la **ville2**. On s'arrête lorsque la **ville2** est isInDisque(…). L'ensemble des villes traversées seront sauvegardées dans un fichier nommé **parcours.txt**, et la liste créée servira à tracer le parcours sur OpenStreetMap.

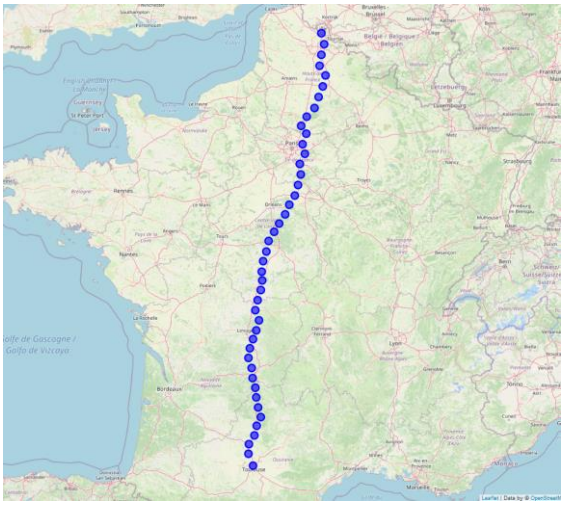
Remarque : il y a un compromis à faire, **plus le rayon est grand** et plus il faut augmenter le **nombre de villes** à rechercher dans le disque de centre **villeX** (fonction ensembleVilles(…)).

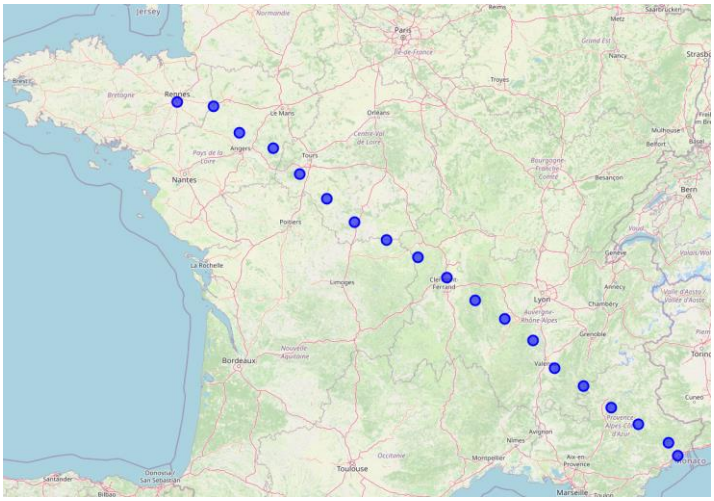
Si rayon = 100 km, il faut 800 villes
Si rayon = 50 km, il faut 300 villes
Si rayon = 20 km, il faut 100 villes

- Utiliser OpenStreetMap pour tracer le parcours et afficher les villes traversées avec un disque bleu, que vous sauvegarderez dans le fichier **map_parcours.html**.

Voir les résultats ci-dessous :

- Ville de départ (ville1) = Lille — Rennes
- Ville d'arrivée (ville2) = Toulouse — Nice
- Disque de recherche (rayon) = **20 km**. — **50 km**





- Etape 6 : Pour les plus rapides …

- Faire en sorte que **la map s'ouvre au centre du département** qui vous est attribué**.**
- **Optimiser le trajet** en recherchant le **chemin le plus court** (parmi les villes qui comptent **le plus d'habitants**) pour aller de **ville1** à **ville2**, dans un rayon de **100 km**.
- A compléter avec **des fonctions de votre choix** …