Annexe 1 Documentation du Programme

Documentation du programme

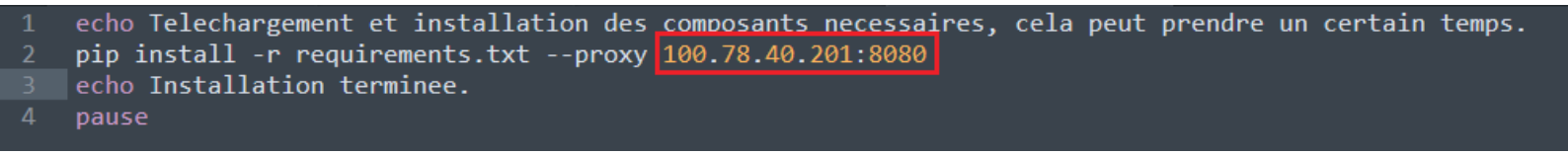
Documentation du programme d'automatisation des entrées de données dans la base territoire. Vous trouverez ici des informations sur tout ce que vous devez savoir pour utiliser le programme, ainsi que son fonctionnement et la description des erreurs que vous pourriez rencontrer.

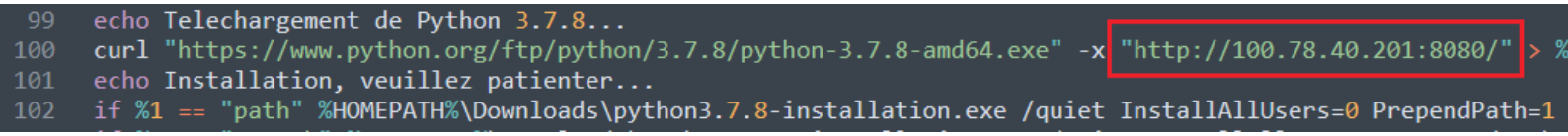
Table des matières

1) Installation

Étape 1 :
Décompressez l'archive zip et dirigez-vous dans le répertoire install. Double-cliquez en premier sur installation_python.bat et patientez. S'il y a une erreur pendant l'installation : droits d'administrateur requis ou autres, rendez-vous à l'étape 2. Sinon, vous pouvez passer à l'étape 3

Pour les personnes installant ce programme hors de la DDT12, et si jamais le proxy de la DDT12 venait à changer, il faudra éditer les 2 scripts d'installation qui se terminent par .bat dans l'archive zip pour modifier ces deux portions encadrées en rouge et y renseigner le proxy :





Étape 2 :
En cas d'erreurs, demandez à votre administrateur système d'installer **Python 3.7.8** pour vous. Vous devrez lui indiquer les choses suivantes : Ouvrez un terminal (Windows + r → cmd → Entrée) et tapez la commande `py --version`. Si vous voyez un message «Can't find a default version of Python », demandez à installer en ajoutant Python au Path, c'est une case à cocher.
S'il y a un message tel que « Python X.X.X », X.X.X étant un numéro de version, il ne faut pas cocher la case pour ajouter Python au PATH. Vous devrez, après l'installation, double-cliquer sur le script fin.bat.

Étape 3 :
Le programme d'installation se ferme, vous pouvez maintenant lancer le programme d'installation des librairies et patienter : double-cliquez sur installation_librairies.bat et attendez la fin.

Une fois l'installation terminée, vous pouvez supprimer le dossier install si vous le voulez, car ils ne sont plus utiles pour le programme.

2) Utilisation

Le programme propose deux façons d'exécution : avec interface graphique et sans. L'interface graphique est sympa pour les yeux et pour éditer les fichiers de configuration mais si jamais elle venait à échouer, le programme non graphique fonctionnera toujours. Autre plus du programme en ligne de commande : il permet de voir les messages d'erreurs que j'aurais pu oublier donc si certaines tables ne se mettent plus à jour, vous pouvez toujours lancer cette version pour essayer de trouver la cause.

Au lancement du programme, que ce soit avec interface graphique ou non, une sauvegarde automatique de la base de données est effectuée. Si vous voulez changer les paramètres de connexion, il faut aller voir dans les fichiers Projet/tout/main_graphique.py et Projet/tout/main_commande.py :

```
f'"C:\\Program Files\\PostgreSQL\\9.5\\bin\\pg_dump.exe" -p 5432 -U admin_bdd -h 10.12.1.5 --blobs --encoding UTF8 --schema "territoire" -F c -f {fichier} base_territoire'
```

cette commande exécute l'application pg_dump.exe avec les options suivantes :

-p port du serveur
-U nom de l'utilisateur qui fait la sauvegarde
-h adresse du serveur
--schema le nom du schéma que l'on veut sauvegarder
-f le chemin de destination de la sauvegarde (fait automatiquement)
et tout à la fin le nom de la base d'où vient le schéma

Le résultat de cette commande est un fichier .backup qui permet de restaurer la base. Il est sauvegardé dans Projet/backups/ et à un nom dans ce genre: Sauvegarde_date_heure.backup

S'il y a plus de 10 fichiers dans le répertoire backups, quel que soit le fichier, le plus vieux d'entre eux sera supprimé au lancement du programme.

2.A) Téléchargement et localisation des fichiers

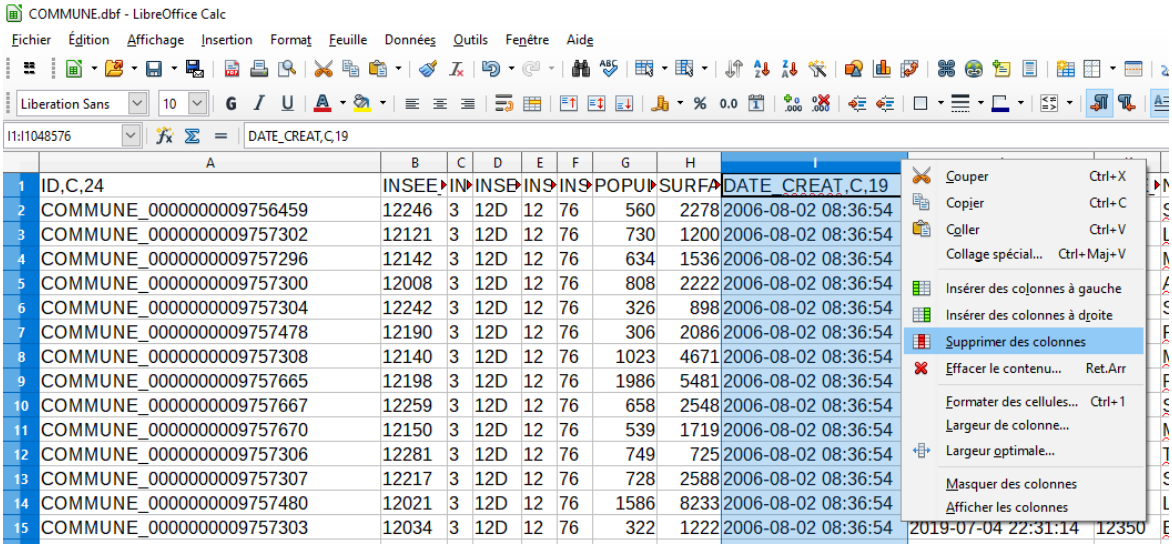
Ranger les fichiers dans les bons répertoires est optionnel, car on peut renseigner les chemins des fichiers et des répertoires dans les fichiers de configuration, mais je trouve plus simple de faire ainsi, comme ça, on n'a pas besoin de tout configurer à chaque utilisation :

Programme	Chemin d'enregistrement	Fichier à prendre
comp_epci	Projet\tout\fichiers\table-comp_epci\	Aspic > Aveyron > Intercommunalité > extraire les données > échelon géographique > Aveyron > Compétences exercées par les groupements
pnr	Extraire le fichier N_ENP_PNR_S_000.dbf du .zip	Télécharger la carte et couches SIG de la métropole au format shape :

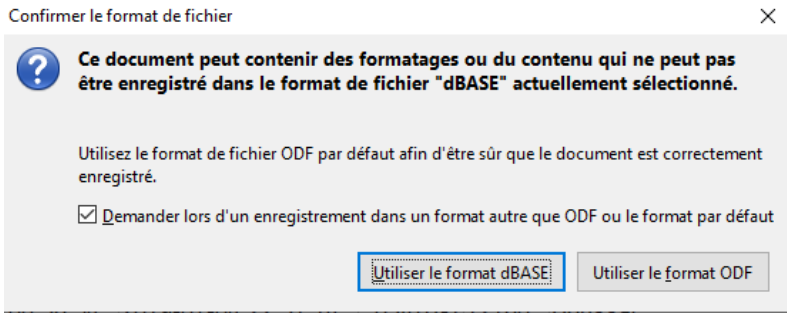
	et le mettre dans le répertoire Projet\tout\fichiers\table-pnr\	https://inpn.mnhn.fr/telechargement/cartes-et-information-geographique/ep/pnr
epci	Projet\tout\fichiers\table-epci\Aveyron\fich\	Aspic > Aveyron > Intercommunalité > extraire les données > échelon géographique > Aveyron > - Liste des groupements - Coordonnées des groupements - Président - Principales caractéristiques socio-démographiques - Compétences exercées par les groupements
	Projet\tout\fichiers\table-epci\Aveyron\special\	Aspic > Aveyron > Commune > extraire les données > Aveyron > Liste des communes
	Projet\tout\fichiers\table-epci\Lot\fich\	Aspic > Lot > Intercommunalité > extraire les données > échelon géographique > Lot > - Liste des groupements - Coordonnées des groupements - Président - Principales caractéristiques socio-démographiques - Compétences exercées par les groupements
	Projet\tout\fichiers\table-epci\Lot\special\	Aspic > Lot > Commune > extraire les données > Lot > - Liste des communes
nvllscomm	Projet\tout\fichiers\table-nvllscomm	Aspic > Aveyron > Commune > extraire les données > Aveyron > - Principales caractéristiques socio-démographiques - Coordonnées des communes - Maire - Liste de communes Aveyron
		Ce site est bloqué par le proxy, donc il n'est pas nécessaire de renseigner ce fichier, surtout que la donnée fournie peut être obtenue différemment. https://www.data.gouv.fr/fr/datasets/communes-du-departement-de-laveyron/ Télécharger, dézipper et prendre le fichier .dbf Attention : il y a une modification à faire, regardez à la fin du tableau (*)
		R:\BDCARTO\ADMINISTRATIF\N_COMMUNE_BDC_012.dbf

	Il n'y a pas à le ranger dans un dossier, il faut juste le renseigner dans la configuration.(voir 2.B ou 3.E)	Q:\AMENAGEMENT_URBANISME\N_ZONAGES_AMENAGEMENT\n_loi_littoral_s_012.dbf

(*) Ouvrez le fichier et supprimez les colonnes avec une date (En fait il suffit juste de supprimer les colonnes qui ont une date bizarre, par exemple avec un -1 dedans):



Ensuite enregistrez le fichier au format dBASE quand il vous le demandera :



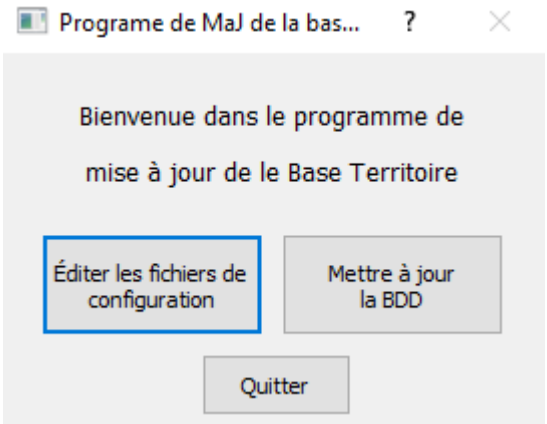
À la fin, vous devriez avoir une arborescence comme celle-ci :

```
fichiers
├──table-comp_epci
│       Compétences exercées par les groupements Aveyron.xls
│
├──table-epci
│   ├──Aveyron
│   │   ├──fich
│   │   │       CompétencesexercéesparlesgroupementsAveyron.xls
│   │   │       CoordonnéesdesgroupementsPrésidentAveyron.xls
│   │   │       Liste des groupements Aveyron.xls
│   │   │       Principales caractéristiques socio-démographiques Aveyron.xls
│   │   │
│   │   └──special
│   │           Liste des communes Aveyron.xls
│   │
│   └──Lot
│       ├──fich
│       │       Compétences exercées par les groupements Lot.xls
│       │       Coordonnées des groupements - Président Lot.xls
│       │       Liste des groupements Lot.xls
│       │       Principales caractéristiques socio-démographiques Lot.xls
│       │
│       └──special
│               Liste des communes Lot.xls
│
├──table-nvllscomm
│       Coordonnées des communes - Maire Aveyron.xls
│       Liste des communes Aveyron.xls
│       N_COMMUNE_BDC_012.DBF
│       Principales caractéristiques socio-démographiques Aveyron.xls
│
└──table-pnr
        N_ENP_PNR_S_000.dbf
```

Pour vérifier que vous avez bien la même arborescence, double cliquez sur arborescence.bat et vous pourrez comparer.

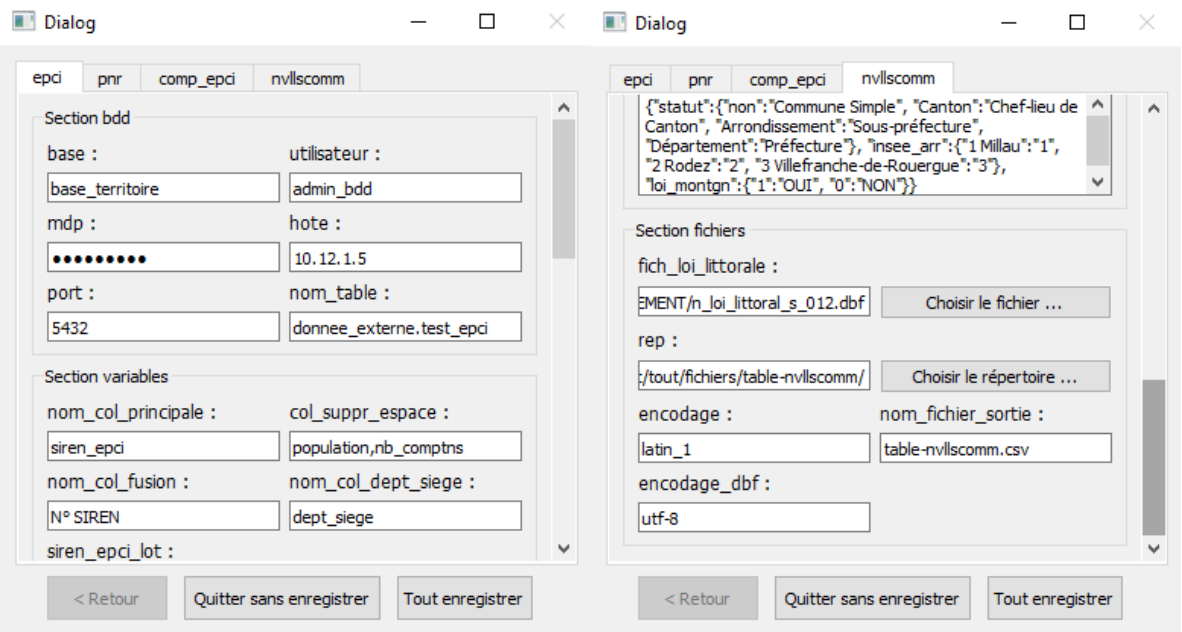
2.B) Interface graphique

Pour lancer, rien de plus simple : double-clic sur graphique.bat et voilà. Vous pouvez ensuite choisir entre éditer les fichiers de configuration et mettre à jour :

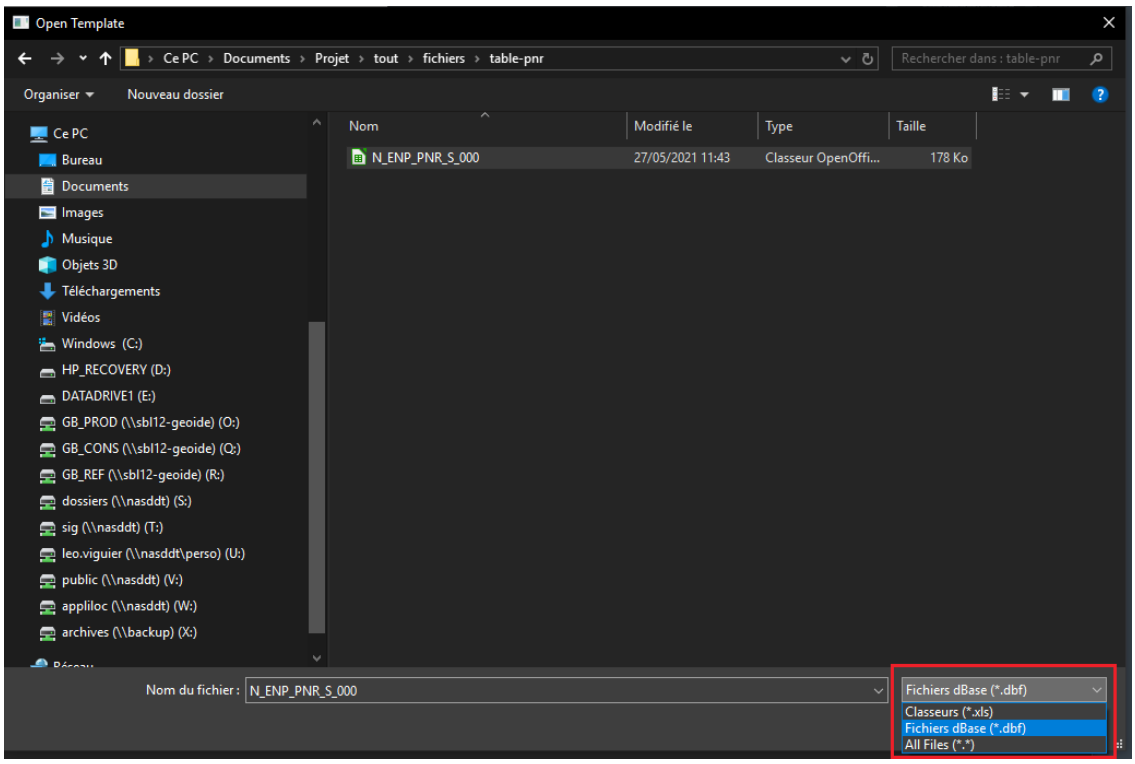


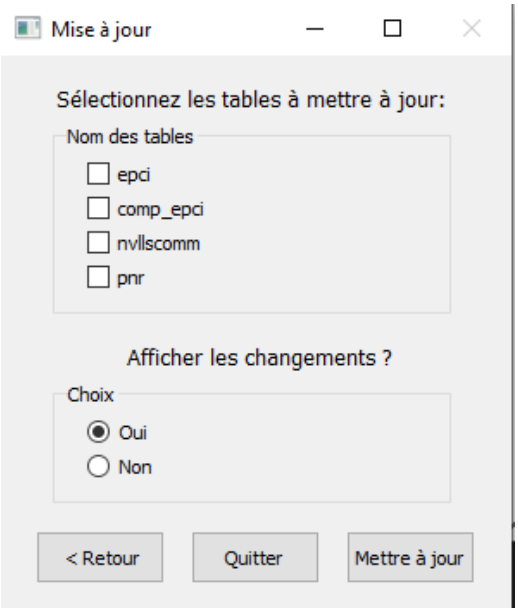
Vous remarquerez que le bouton de la croix en haut à gauche est grisé, c'est normal, si on veut quitter, il faudra cliquer sur les boutons "Quitter". Le bouton en point d'interrogation ne fait rien non plus.

L’édition des fichiers de configuration nous amène sur cette fenêtre. A savoir qu’il n’est pas nécessaire d’éditer les fichiers de configuration à chaque mise à jour tant que vous avez rangé les fichiers au bon endroit. Pour sauvegarder les modifications, il faut cliquer sur Tout enregistrer, cela enregistrera le contenu de toutes les cellules, y compris celles des autres onglets, donc faites attention à ce que vous écrivez. Le bouton “Retour” sera ensuite dégrisé et vous pourrez revenir à la fenêtre précédente. À savoir qu’au moment où vous cliquez sur tout enregistrer, le bouton “Quitter sans enregistrer” passera à “Quitter” donc vous ne pourrez pas annuler vos modifications après avoir cliqué dessus.



En cliquant sur “Choisir le fichier…” soyez sûr de sélectionner le bon type de fichier :



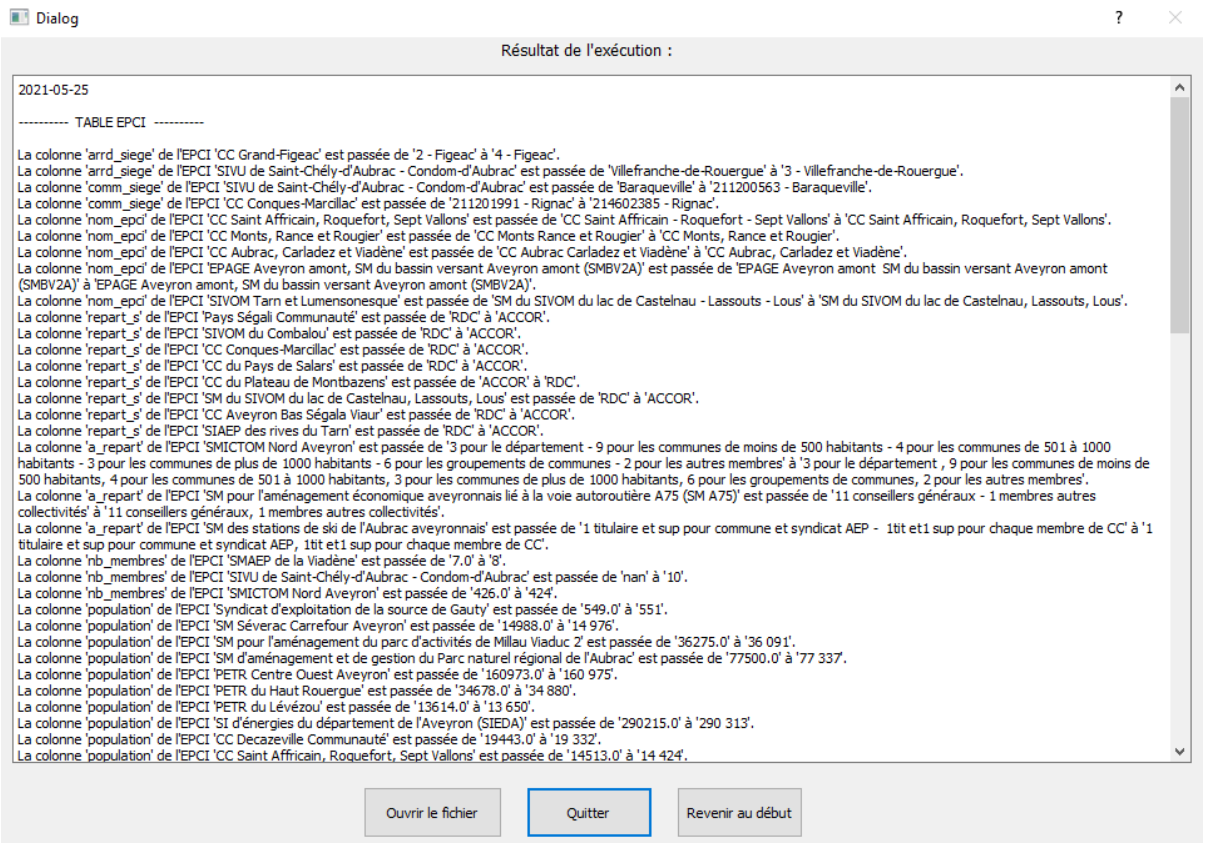


La fenêtre mise à jour permet de sélectionner les tables à mettre à jour. Une fois fait, vous devez choisir entre afficher les changements lors de la mise à jour ou non.

Cette option est utile en cas d'erreurs, comme a on les voit mieux passer.

Vous pouvez maintenant cliquer sur mettre à jour. Si jamais le titre de la fenêtre indique "Ne répond pas", il faut attendre, c'est normal. Il est aussi possible que Windows vous demande si vous voulez terminer le programme ou attendre que cela réponde, cliquez sur attendre. Normalement, la mise à jour ne devrait pas prendre plus d'une minute, deux à la limite donc si cela ne répond pas c'est qu'il y a peut-être un problème et qu'il faut aller voir dans le fichier de résultat Projet/tout/body ou regarder dans le terminal. Les erreurs que j'aurais peut-être oublié de documenter seront affichées dans le terminal.

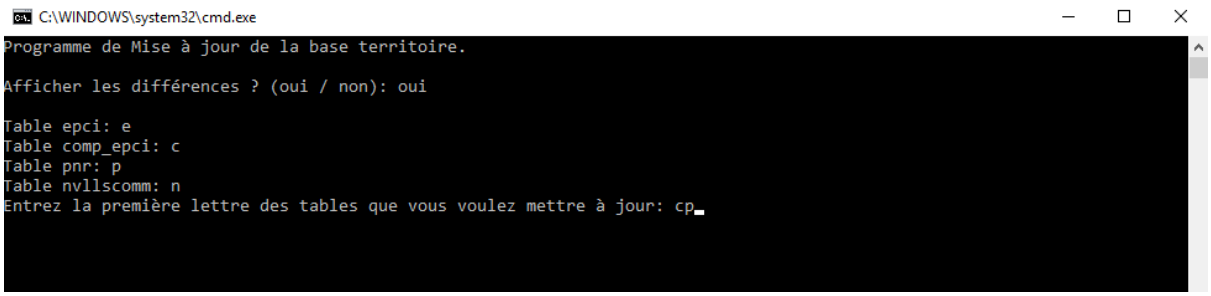
Si l'exécution se finit bien, vous aurez droit à cette fenêtre indiquant toutes les modifications effectuées :



Le bouton Ouvrir le fichier vous ouvre le fichier contenant ce résultat.

2.C) Ligne de commande

Double-cliquez sur le fichier main_commande.bat et vous aurez des questions qui vous seront posées. Dans l'exemple ci-dessous, j'ai choisi d'afficher les différences et de mettre à jour des tables comp_epci et pnr. Une fois les questions répondues, appuyez sur entrée et le programme s'exécutera.



3) Configuration

Le programme c'est en réalité pas un seul fichier mais est constitué de plusieurs fichiers .py ainsi que des fichiers de configuration .ini qui leur sont associés. C'est dans ces fichiers que vous pourrez éditer les variables afin de permettre la bonne exécution du programme si jamais des noms, ceux des colonnes par exemple, changent.

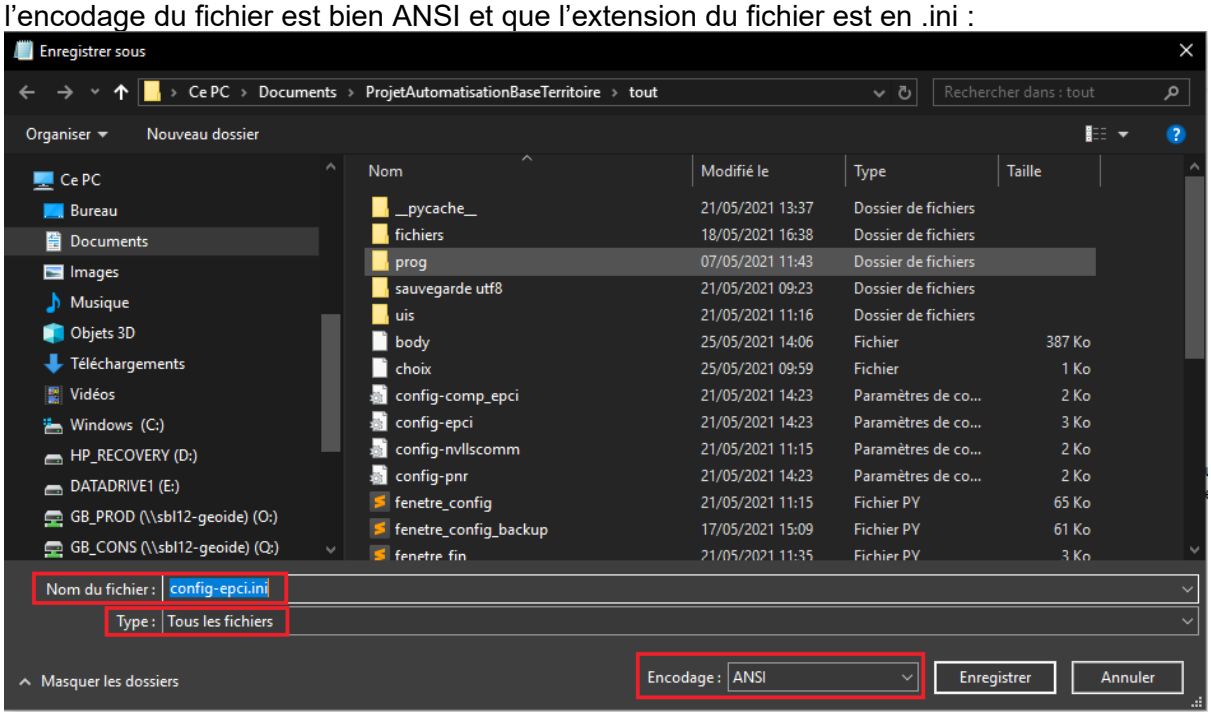
La structure d'un fichier .ini n'est pas compliquée à comprendre, en voici un exemple :
[bdd]
base = Nom de la base
utilisateur = Nom d'utilisateur

[variables]
nom_col_siren_bdd = siren_epci
nom_col_nom_epci_bdd = nom_epci

un fichier .ini est divisé en sections, et sont identifiées par des crochets. Nous avons donc dans cet exemple 2 sections qui sont : bdd et variables. Dans ces sections, les variables sont identifiées par nom_de_la_variable = contenu_de_la_variable. Les commentaires ne peuvent être écrits que sur toute la ligne et sont précédés par un point-virgule. Voilà tout ce qu'il faut savoir sur ces fichiers. À savoir, ces fichiers n'étant constitués que de texte, vous pouvez les ouvrir dans n'importe quel logiciel de texte tel que le bloc-note ou notepad++.

Avant toute modification, je vous suggère fortement de faire une copie du fichier que vous êtes sur le point de modifier.

Lors de l'enregistrement, si vous faites des modifications avec le bloc-notes, vérifiez que l'encodage du fichier est bien ANSI et que l'extension du fichier est en .ini :



3.A) Variables communes à tous les fichiers

Ces variables ont la même signification dans tous les fichiers de configuration.

Nom de la Section	Nom de la variable	Explication
bdd	base	Le nom de la base sur laquelle vous allez travailler, où les tables sont stockées.
	utilisateur	Le nom d'utilisateur avec lequel vous voulez vous connecter. Pensez à prendre un utilisateur avec des droits d'écriture dans la BDD sinon, il sera impossible d'entrer les données.
	mdp	Le mot de passe de cet utilisateur.
	hote	L'adresse IP du serveur de BDD.
	port	Le port du serveur de BDD, c'est souvent le port par défaut, à savoir 5432.
	nom_table	Le nom de la table qui va être mise à jour
variables	nom_col_principale	Le nom de la colonne qui sert à ordonner les lignes, il y a de grandes chances que c'est aussi le nom de la colonne contenant la clé primaire dans la table traitée.
	nom_col_fusion	Le nom de la colonne principale (correspondant à celle au-dessus) telle qu'elle apparaît dans les fichiers téléchargés. Elle sert à fusionner les fichiers en fonction d'un nom.
	dico_correspondance	Le dictionnaire qui fait le lien entre le nom des colonnes tels qu'ils apparaissent dans le(s) fichier(s) et les noms dans la table de la BDD. La valeur de cette variable est un dictionnaire en python, et est écrit de la forme : {"Nom de la Colonne":"nom_col", Population" :"population"}. Si vous devez modifier quelque chose, veillez à ce que l'écriture soit bien respectée, autrement, vous aurez une erreur.
	valeurs_par_defaut	Certaines colonnes stockent la même chose alors on crée ici un dictionnaire qui informe sur ces valeurs. Le format est le suivant : {"nom_col":"valeur"} Vous ne devez surtout pas renseigner des colonnes vides.
	col_suppr_espace	Certaines colonnes ont des chiffres qui sont notés avec un espace entre les milliers : 12 345 par exemple. Il faudra noter ici le nom de ces colonnes telles qu'elles apparaissent dans la bdd en les séparant par une **virgule seulement** : pas par une ~~virgule et espace~~. L'ordre n'est pas important. Par

		défaut, je vous conseille vivement d'y renseigner toutes les colonnes qui ont le type numeric ou integer dans la BDD, cela permettra d'éviter des erreurs dans le futur. Laisser vide sinon.
fichiers	rep	Le chemin vers le répertoire où sont stockés les fichiers. Plus le chemin est précis, mieux cela sera.
	encodage	L'encodage utilisé par les fichiers. C'est aussi l'encodage utilisé pour écrire le fichier .csv à la sortie.
	nom_fichier_sortie	Le nom utilisé pour sauvegarder la table créée à la fin du programme.

3.B) Fichier config-comp_epci.ini

Tout est détaillé à la section *3.A)*.

3.C) Fichier config-epci.ini

Nom de la Section	Nom de la variable	Explication
	colonnes_a_formater	C'est un dictionnaire de dictionnaires (n'ayez pas peur) qui sert à formater le nom de certaines cellules des fichiers en quelque chose de plus simple à lire. Dans ce dictionnaire, on retrouve comme clé le nom de la colonne contenant des valeurs à formater et comme valeur, un dictionnaire contenant comme clé le nom avant formatage puis comme valeur le nom après formatage. Petit résumé : {"nom_colonne": {"valeur_avant": "valeur_apres", "valeur_avant2": "valeur_apres2"}}
	siren_epci_lot	Un chiffre qui correspond au siren de l'EPCI du Lot qui apparaît dans la BDD
	nom_col_dept_siege	Le nom de la colonne contenant le nom du département du siège de l'EPCI
fichiers	rep_aveyron et rep_lot	Les fichiers venant d'ASPIC Aveyron et d'ASPIC Lot doivent aller dans leurs répertoires spécifiques. Il y a donc deux répertoires pour les deux départements. C'est un chemin qu'il faut indiquer ici. Évitez de mettre le backslash \ car python ne

		l'aime pas trop, à moins de le doubler. Un simple slash / suffit.
	rep_special_aveyron et rep_special_lot	Il y a un fichier dont il faut extraire une donnée qui ne peut pas être fusionné avec les autres donc on le met dans un répertoire spécial dont le chemin est indiqué ici. Évitez de mettre le backslash \ car python ne l'aime pas trop, à moins de le doubler. Un simple slash / suffit.

3.D) Fichier config-pnr.ini

Nom de la Section	Nom de la variable	Explication
variables	pnr_a_garder	Le fichier à télécharger contient les informations nécessaires de tous les PNR de France. Or, nous sommes uniquement intéressés par les PNR d'Aveyron. Il faut donc indiquer leurs id_mnhn en les séparant par une virgule.

3.E) Fichier config-nvllscomm.ini

Nom de la Section	Nom de la variable	Explication
variables	autre_nom_col_ci	Certains fichiers ont le nom de la colonne du code INSEE (ma clé principale pour ce programme) qui diffère des autres fichiers. Vous devez donc renseigner le nom de colonnes contenant le code INSEE, en les séparant par une virgule uniquement.
	colonnes_a_formater	C'est un dictionnaire de dictionnaires (n'ayez pas peur) qui sert à formater le nom de certaines cellules des fichiers en quelque chose de plus simple à lire. Dans ce dictionnaire, on retrouve comme clé le nom de la colonne contenant des valeurs à formater et comme valeur, un dictionnaire contenant comme clé le nom avant formatage puis comme valeur le nom après formatage. Petit résumé :

		{“nom_colonne”: {“valeur_avant”: “valeur_apres”, “valeur_avant2”: “valeur_apres2”}}
fichiers	fich_loi_littorale	le chemin vers le fichier listant les communes sujettes à la loi littorale. Mettez des simples / pour accéder au fichier.
	encodage_dbf et encodage	Les fichiers .xls ont un encodage différent des fichiers dbf. Encodage correspond aux fichiers xls et encodage_dbf aux fichiers dbf.

4) Fonctionnement

Tous les fichiers prennent majoritairement des fonctions du fichier programme_toolbox.py. Cela me permettait de programmer plus vite mais aussi de me forcer à rendre le programme le plus flexible possible, afin qu'il s'adapte facilement à des nouveaux fichiers. Voici la description de ses fonctions :

chargement_table_existante : Se connecte à la BDD et charge la table existante dans une variable. Renseigne aussi des variables utiles.

date_ou_pas : recherche une année comme 2018 ou 1975 dans un nom passé en argument et le supprime. Cette fonction est utile car dans un certain nombre de colonnes, il y a des dates et ce serait bête de devoir renseigner cette nouvelle date dans un fichier de configuration tous les ans.

nettoyage : remplace les noms des colonnes par celui renvoyé par la fonction date_ou_pas. Le fait pour mes fichiers mais aussi pour le dictionnaire de correspondance.

fusion_fichier : prend la liste de fichiers passée en argument et les fusionne en fonction de l'indicateur passé en argument.

eciture_fichier_csv : prend la DataFrame (tableau) passé en argument et l'écrit sous un fichier csv avec le nom passé en argument

tri_colonnes : tri les colonnes du fichier passé en argument pour en sortir les colonnes qui nous sont inutiles. Cette fonction remplace aussi le nom des colonnes par celui tel qu'il apparaît dans la BDD.

supprime_doublon : certains fichiers contiennent les mêmes colonnes et donc les mêmes informations. Avant la fusion, il faut donc se débarrasser de celles-ci et n'en laisser qu'une de chaque. C'est ce que fait cette fonction.

formattage_valeurs : Certaines valeurs de la table existante ont un formatage pour rendre les informations plus lisibles. Dans cette fonction, on s'occupe de faire ceci sur le fichier passé en argument.

affiche_differences : cette fonction compare les valeurs de la table créée avec celles de la table précédente et affiche les différences.

ajout_colonne_manuel : ajoute une colonne à un fichier. Fonction nécessaire quand une valeur ne peut pas être obtenue depuis un fichier et quand elle vaut la même chose partout. Vous me direz, "mais pourquoi ne pas avoir mis cette colonne et ses valeurs dans dico_val_par_defaut ?" C'est parce que cette fonction est appelée pour renseigner le département de l'EPCI. Or, cette fameuse EPCI du Lot est située dans le Lot mais toutes les autres sont en Aveyron. Cela signifie que ce n'est plus une valeur par défaut.

enleve_espace_chiffre : enlève les espaces dans les chiffres comme celui-ci : 12 345 et les transforme en de vrais chiffres entier que l'ordinateur comprend en tant que chiffre : 12345

bdd_nouvelle_table : copie les caractéristiques de la table existante pour en créer une nouvelle sous le nom passé en argument. Faites bien attention aux noms que vous

fournissez : au début de la fonction, la table nom_nouvelle_table est supprimée.
Cette fonction ne fonctionne plus, mais je la laisse au cas où.

disparue_apparue : Cherche si des éléments (EPCI ou commune ou PNR) ont disparu ou sont apparus. C'est utile pour la fonction suivante.

construit_querys : Le but est simple : créer les requêtes SQL pour mettre à jour la table.
1re étape : regarder dans toutes les cellules de la nouvelle table et si une cellule n'est pas vide, cela sauvegarde le nom de la colonne et le contenu de la cellule dans une liste.
2e étape : construire le milieu de la requête UPDATE avec le nom de la colonne et sa valeur
3e étape : construire la requête UPDATE en mettant un WHERE qui permet de cibler la bonne ligne à mettre à jour.
4e étape : Regarder s'il y a des éléments disparus (EPCI ou communes ou PNR) et faire des requêtes pour ces disparitions. La même chose est faite pour les apparitions. Enfin, ces requêtes sont sauvegardées dans une liste qui sera envoyée à la fonction suivante.

bdd_reecriture_table: prend les requêtes SQL générées par la fonction **construit_querys** et les exécute. Si cela échoue, affiche l'erreur et stoppe l'exécution du programme pour cette table, et passe à la suivante.

5) Erreurs

Les erreurs apparaîtront sous la forme ERREUR […]: Message… correspondant au code d'erreur. Toutes les erreurs stoppent la mise à jour d'une table mais pas les autres, en fonction de l'erreur. Vous pouvez aussi tomber sur des ATTENTION: ils indiquent que quelque chose s'est mal exécuté, mais, il n'est peut-être pas nécessaire d'arrêter le programme pour autant, car ce n'est peut-être pas quelque chose d'indispensable.

Code	Type	Explication(s) et Solution(s)
01	BDD	Il est probable que les noms de tables que vous avez renseignés n'existent pas dans la BDD que vous avez renseignée. Rendez-vous dans le fichier de configuration pour corriger cette erreur.
02	Table	1) La table est vide donc on ne peut pas ordonner en fonction de rien → Vérifier le nom de la table dans le fichier de configuration 2) Le nom de la colonne (nom_col_principale) utilisé pour ordonner la table n'existe pas → Modifier nom_col_principale dans le bon fichier de configuration
03	Variable	Le dictionnaire que vous avez renseigné pour dico_correspondance ou valeurs_par_defaut ou colonnes_a_formater n'est pas bon. Vérifiez l'écriture de celui-ci et pensez à ce que les clés et valeurs de type str() prennent des " et non pas des ' (voir 3.A)
04	BDD	1) Le nom des tables n'est pas bon et elles n'existent pas → vérifiez ce que vous avez écrit dans le fichier de configuration 2) L'utilisateur que vous utilisez pour vous connecter à la BDD n'a pas les droits suffisants pour créer / supprimer des tables → vérifiez ce que vous avez écrit dans le fichier de configuration
05	BDD	1) Les valeurs n'ont pas le bon type et ne correspondent pas avec le type demandé par la table. → Vérifiez ce que vous avez entré dans les variables dico_correspondance, valeurs_par_defaut et colonnes_a_formater (si vous avez ces 2 derniers). Vérifiez aussi si vous avez écrit le bon chemin vers le répertoire contenant les fichiers et avec le bon encodage. 2) Les droits de l'utilisateur ne sont pas bons. Vérifiez ce que vous avez écrit dans le fichier de configuration 3) Des chiffres auparavant sans espace entre les milliers ont peut-être des espaces maintenant. Il faudra rajouter des colonnes dans la variable col_supp_espace
06	BDD	nom_nouvelle_table et nom_ancienne_table ont le même nom. Vérifiez ce que vous avez écrit en faisant attention au nom, car la table nom_nouvelle_table sera écrasée si elle existe.
07	BDD	Connexion impossible à la Base de Données. Vérifiez les paramètres entrés dans le fichier de configuration et regardez le

		message d'erreur pour plus d'informations.
08	Variable	1) Aucun fichier du Lot n'a été chargé → vérifiez ce que vous avez écrit à la variable rep_lot 2) Les fichiers du Lot n'ont pas de colonne nommée par la variable nom_col_fusion 3) Les fichiers ont été ouverts avec le mauvais encodage → vérifiez la variable encodage
09	Variable et table	1) Aucun fichier n'a pu être chargé → vérifiez les variables rep_aveyron et rep_lot 2) La nom de colonne utilisé pour la fusion n'existe pas dans les fichiers chargés → vérifiez la variable nom_col_fusion
10	Variable	Vous avez probablement donné en argument à la fonction supprime_doublon un seul fichier ou quelque chose sur lequel il n'est pas possible d'itérer (les dictionnaires n'étant pas pris en compte par cette fonction).
11	Variable	Vous avez probablement mal renseigné les variables de votre dictionnaire colonnes_a_formater. Vérifiez ce que vous avez écrit dans la variable colonnes_a_formater
12	Variable	(Pour les développeurs) Vous n'avez pas passé une DataFrame de pandas. Je les utilise dans tout le programme car c'est plus simple de travailler avec.
13	Variable	Vérifiez le nom du fichier que vous avez inscrits dans la variable nom_fichier du fichier de configuration
14	Variable	(Pour les développeurs) Vous n'avez pas passé une DataFrame de pandas
15	Fichier	Les chemins de répertoires que vous avez indiqués ne contiennent rien / n'existent pas / ne sont pas accessibles en lecture. Vérifiez ce que vous avez écrit dans le fichier de configuration.
16	Variable	Le nom de certaines colonnes que vous avez renseigné dans le fichier de configuration ne convient pas, il n'existe pas dans la BDD. Vérifiez ce que vous avez écrit.
17	Fichier	Impossible de télécharger le fichier. Vérifiez votre connexion internet ou L'URL ou le lien du proxy dans le fichier programme_toolbox.py
18	Fichier	Impossible de charger un fichier de configuration. Vérifiez que vous ne l'avez pas supprimé, que le nom n'ait pas changé ou alors que l'encodage du fichier n'ait pas changé (voir 3) Configuration)
19	Fichier	Vous n'avez probablement pas supprimé les colonnes avec une date buggée dans le fichier COMMUNE.dbf (voir 2.A))

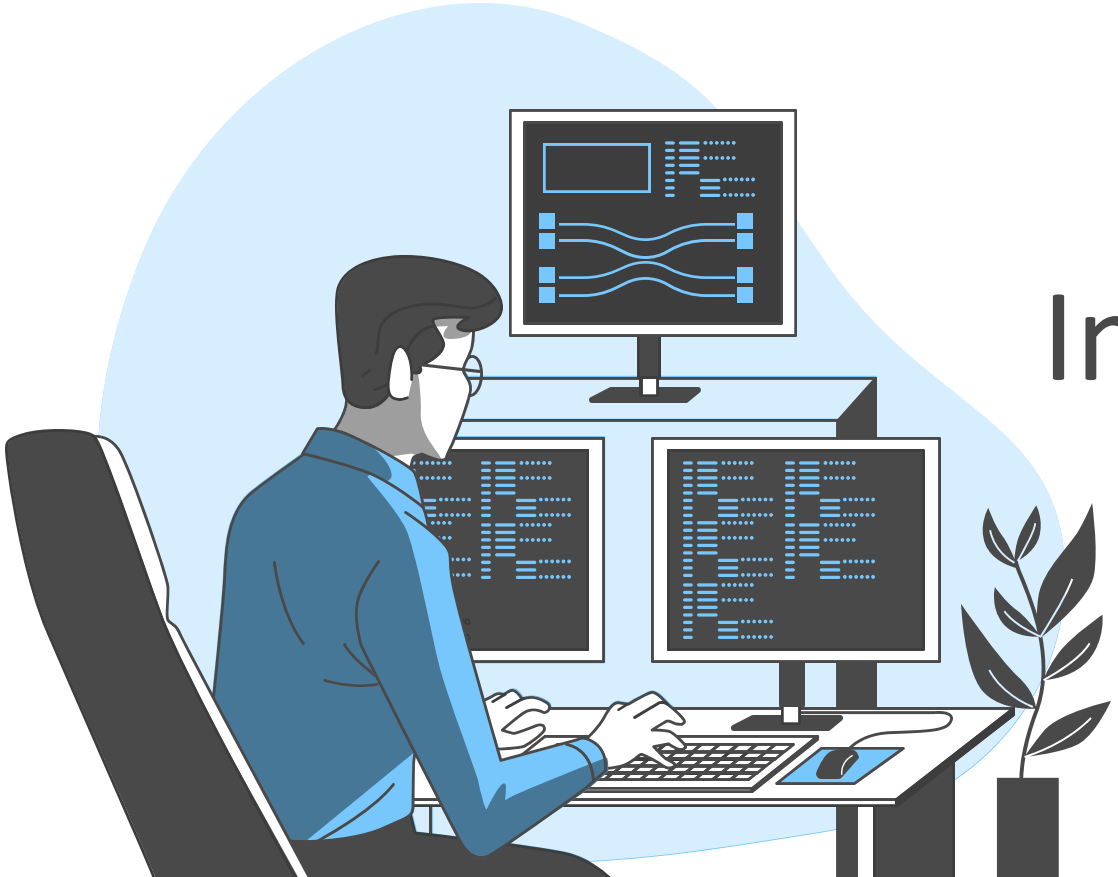
6) Modification du Programme

Les seuls fichiers que vous serez amenés à modifier sont les fichiers de la forme programme-…py ou programme_toolbox.py. Je vous déconseille fortement de toucher aux fichiers fenêtre-…py, car on est très rapidement perdu si on ne s'y connaît pas. De plus, il y a la version du programme en ligne de commande si jamais l'interface graphique ne marche plus. Je vous déconseille encore plus de modifier les interfaces graphiques via l'application "designer" (tapez designer dans le menu démarrer) car j'ai fait beaucoup de modifications dans les fichiers fenetre-…py et quand on convertit le résultat de l'application designer en fichier python, cela ne garde pas ces modifications. Je vous laisse quand même les fichiers permettant de faire ces modifications dans le dossier uis.

Quoi qu'il en soit, si vous souhaitez quand même faire des modifications, pensez avant toutes choses à faire une copie du fichier.

Si vous voulez mettre plusieurs fichiers dans un répertoire (celui de pnr par exemple) car vous avez des nouvelles infos à mettre, vous pouvez le faire mais pensez à renseigner le nom des colonnes à prendre dans la configuration. Par contre, évitez de mélanger les types de fichier, par exemple, le programme PNR ne prend en compte que des fichiers .DBF donc vous devrez peut-être les convertir.

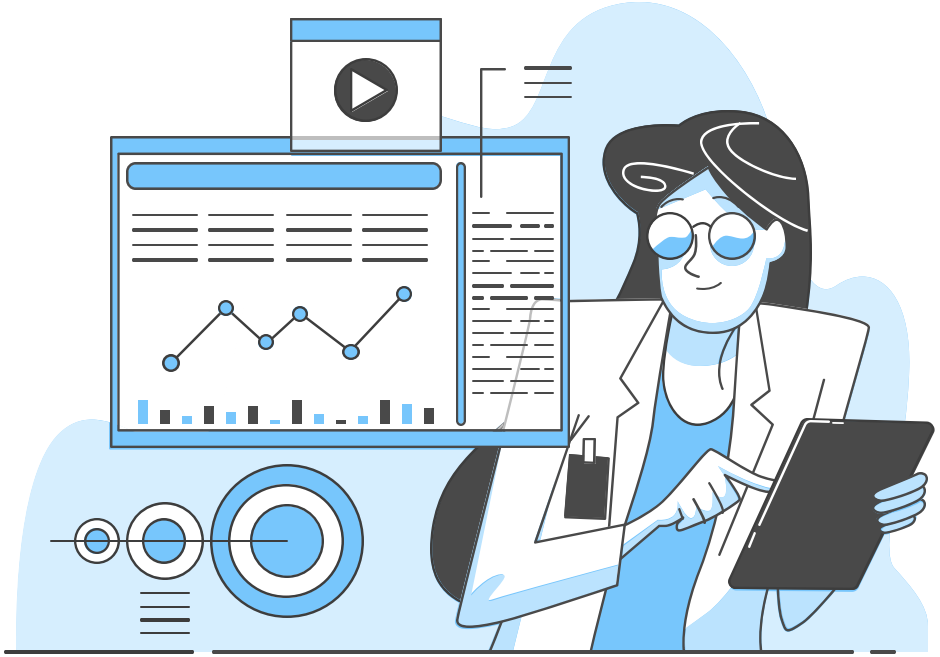
Annexe 2 Présentation de Python



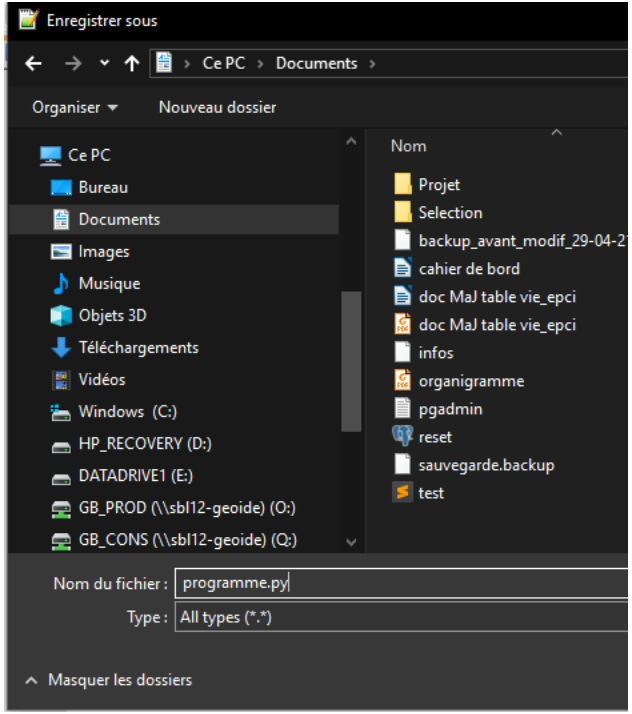
Introduction à Python

Sommaire

1. **Introduction**
   Variables, types, opérations, if else ...
2. **Itérables**
   Listes, boucles for & while
3. **Fonctions et Classes**
   & portée des variables
4. **Choses à savoir**
   Fichiers, dictionnaires, exceptions, raccourcis



Lancement



Ouvrez un terminal (windows > cmd > entrée) et tapez py puis entrée

Sinon exécutez le script suivant :

```
cd /
for /R %f in (python.exe) do
@IF EXIST %f @echo "%f"
```

Et utilisez un des chemins

Ouvrez notepad ++, créez un nouveau fichier et enregistrez avec l'extension .py

Variables et print()

Une variable stocke une information et un type.
Pour afficher son contenu, on utilise print()
Exemple :

```
1  variable = "Valeur"
2  autre_variable = 0
3  print(variable)
4  # affiche "Valeur"
5  print(autre_variable)
6  # affiche 0
```

Les variables peuvent uniquement contenir des lettre majuscules et minuscules, de chiffres et l'underscore _.

L'écriture se fait une variable par ligne.

Certains noms sont réservés par Python.

Types des variables

Type	string	int	float	list	bool
Exemple	"test" "'Aujourd'hui'" '12zç_$aer'	0 -1 894684684	0.1 -1.3333333 10e8	[18, 9, 5] ["a", "b", "c"] ["a", [1,2], 3]	True False
Création	a = "test" a = str()	a = 123 a = int()	a = 0.1 a = float()	a = [1,"a","3"] a = list()	a = True a = bool()
Entrée Sortie	a = str(8) a vaut "8"	a = int("8") a = int(8.99) a vaut 8	a = float(1) a = float("1") a vaut 1.0	a = list("ab") a vaut ["a", "b"]	a = bool(1) a = bool("True") a vaut True
Signification	Chaine de caractères	Chiffre entier	Chiffre décimal	Liste d'éléments	Booléen, que True et False possible

Opérations (presque) mathématiques

Opération math	Addition	Soustraction	Division	Multiplication	Puissance
En Python	+	-	/	*	**
Equivalent math	+	-	÷	×	^
Types supportant l'opération	int, float, str, list	int, float	int, float	Int, float, str	Int, float
Fonctionne **Fonctionne pas** **L'ordre est important**	a = 2 + 2.3 a = "ab" + "c" a = [1,2] + ["3", 4] a = "ab" + 3 a = [1,2] + "e"	a = 2 - 1.3	a = 1 / 3.5	a = 2.5 * 6.3 a = "ab" * 3 a = "ab" * 1.5 a = "ab" * "cd"	a = 3 ** 2 a = 2.5 ** 3.2

Attention : Pour certaines opération, il y a des particularités entre le type des éléments de l'opération.

if, elif et else

Permet d'effectuer des tests sur le contenu des variables
Fonctionne de la manière suivante:

```
4  if condition:
5      # si la condition1 est vérifiée,
6      # le programme rentre ici
7      print('ok')
8
9  elif condition2:
10     # Sinon, si la condition2 est vérifiée,
11     # le programme rentre ici
12     print('autre chose')
13
14 else:
15     # si aucune des conditions au-dessus
16     # n'est vérifiée, le programme entre ici
17     print("fait quelque chose")
18
```

L'indentation (les espaces entre la marge et le code) est obligatoire en Python.
C'est soit une tabulation, soit 4 espaces

DOIT commencer par un if, le elif et else sont optionnels

Opérateur logiques et de comparaison

Les opérateur logiques sont utiles pour les if – else. Ici, True = 1 et False = 0

and	1	0
1	1	0
0	0	0

Et logique

or	1	0
1	1	1
0	1	0

Ou logique

not	1	0
résultat	0	1

Négation

==	"a"	"b"
"a"	1	0
"b"	0	1

Égalité

!=	"a"	"b"
"a"	0	1
"b"	1	0

Différence

Exemple :

```
21  print("a" == "b" or "b" == "b")
22  # affiche True car on a
23  # False or True qui donne True
24  print( not (10 != 11 and 2.0 != 2))
25  # 10 != 11 donne True
26  # 2.0 != 2 donne False
27  # True and False donne False
28  # not False donne True
29  # Le résultat est donc True
```

Opérateur logiques et de comparaison (suite)

>	3
5	True

Ces tableaux se lisent 5 > 3 donne True

>	5	3
5	False	True
3	False	False

Supérieur

>=	5	3
5	True	True
3	False	True

Supérieur ou égal

<	5	3
5	False	False
3	True	False

Inférieur

<=	5	3
5	True	False
3	True	True

Inférieur ou égal

Itérables et boucles for

Les itérables sont tous les types de variable que l'on peut accéder un par un avec une boucle.
Les list, str en sont deux exemples. Les int et float ne fonctionnent pas.

```
37  # la structure d'une boucle for est la suivante :
38  for valeur in iterable:
39      # faire quelque chose avec valeur
40  # la variable valeur va prendre un apres l'autre
41  # les éléments contenus dans iterable.
```

Exemple :

```
46  for valeur in [1, 2, 3]:
47      print(valeur)
48  # va afficher 1, puis 2 puis 3
49
50  for lettre in "bonjour":
51      print(lettre)
52  # va afficher "b", puis "o", puis "n" ...
```

Boucles while

Les boucles while (tant que en français) ne s'arrêtent pas tant que leur condition est égale à True :

```
65  while condition:
66      # faire quelque chose
67      if condition1:
68          condition = False
69          # on va sortir de la boucle à la fin de ce tour
70
71
72  # boucle infinie
73  while True:
74      print("J'adore apprendre Python")
```

Index d'un itérable et élément range()

Les itérables ont tous un index associé à chaque élément. Exemple :

```
77  var = "Bonjour"
78  # le B est à l'index 0
79  # le r est à l'indez 6
80  print(var[3]) # affiche j
81  print(var[6]) # affiche r
82  print(var[-1]) # affiche r
```

range(debut, fin, pas) construit un itérable constitué de tout les chiffres de debut à fin-1, en suivant le pas. Exemple :

```
86  for chiffre in range(2, 8, 1):
87      print(chiffre)
88      # va afficher 2, puis 3, ... jusqu'à 7
89
90  for chiffre in range(8):
91      print(chiffre)
92      # va afficher les chiffres de 0 à 7
93
94  for chiffre in range(0, 11, 2):
95      print(chiffre)
96      # va afficher 0, puis 2, puis 4, jusqu'à 10.
```

Détails sur le type str

Plusieurs moyens de le définir :

```
 99  chaine1 = "test"
100  chaine2 = 'test'
101  chaine3 = '''test'''
102  chaine4 = """test"""
```

Opérations et formattage du type str :

```
105  age = 19
106  prenom = "Léo"
107  phrase1 = "Je m'appelle " + prenom + ", j'ai " + str(age) + " ans."
108  phrase2 = "Je m'appelle {}, j'ai {} ans.".format(prenom, age)
109  phrase3 = f"Je m'appelle {prenom}, j'ai {age} ans."
110  # afficher ces 3 phrases donne :
111  # Je m'apelle Léo, j'ai 19 ans.
```

Détails sur le type str (suite)

Slicing (découpe en français) :

```
var1 = "Bonjour"
var2 = var1[0:2]
var3 = var1[0:len(var1):2]
var4 = var1[::-1]
print(var2)
# affiche 'Bon'
print(var3)
# affiche 'Bnor'
print(var4)
# affiche 'ruojnoB'
```

Attributs utiles du type str :

```
var1 = "Test des attributs d'un type str"
# variable.split("élément de séparation")
var2 = var1.split('e')
# variable.replace("élément à remplacer", "valeur de remplacement")
var3 = var1.replace("t", "W")
var4 = var1.lower() # met la chaine de caracteres en minuscules
var5 = var1.upper() # met la chaine de caracteres en majuscules
bool1 = var1.endswith(" str") # donne True si var1 fini par ' str' sinon False
bool2 = var1.startswith("test") # donne True si var1 commence par 'test' sinon False
print(var2) # affiche une liste: ['T', 'st d', "s attributs d'un typ", ' str']
print(var3) # affiche: TesW des aWWribuWs d'un Wype sWr
print(var4) # affiche: test des attributs d'un type str
print(var5) # affiche: TEST DES ATTRIBUTS D'UN TYPE STR
print(bool1) # affiche: True
print(bool2) # affiche: False
```

Fonctions

Les fonctions sont utiles pour ne pas surcharger un programme en codant des choses en double

```
147  resultat = 0
148  chiffre0 = 10
149  def addition(chiffre1, chiffre2, chiffre3=0):
150      resultat = chiffre0 + chiffre1 + chiffre2 + chiffre3
151      return resultat
152
153  test = addition(10, 5)
154  print(test)
155  # affiche 25
156  test = addition(10, 5, 6)
157  print(test)
158  # affiche 31
159  print(resultat)
160  # affiche 0
```

Une variable créée dans une fonction ne peux pas être accédée hors d'une fonction.
Le contraire est possible lui.

Classes

Les classes contiennent des fonction avec un interet majeur : augmenter la portée des variables grâce à *self*

```
160 class Maclasse():
161     def __init__(self):
162         self.var = 10
163 
164     def calcule(self, chiffre):
165         self.chiffre = chiffre
166         self.multiplication()
167         print("Le résultat de {} * {} est {}".format(self.var, chiffre, self.resultat))
168 
169     def multiplication(self):
170         self.resultat = self.chiffre * self.var
171 
172 cla = Maclasse()
173 print(cla.var)
174 # affiche 10
175 cla.calcule(12)
176 # affiche 'Le résultat de 10 * 12 est 120'
```

self n'est pas compté comme un argument par une fonction, plutôt comme un indicateur permettant d'accéder aux variables de la classe. Il ne faut pas le renseigner lors d'un appel à une fonction, il faut le faire précéder avec un . .

Fichiers

On peut facilement ouvrir des fichiers pour lire et écrire et Python se charge de fermer le fichier après utilisation.

```
183  # La syntaxe est la suivante:
184  # with open("chemin/vers/le/fichier.txt", "mode_d'ouverture") as nom_variable:
185  #     faire quelque chose avec le fichier.
186  # il y a 3 modes d'ouverture communs :
187  # w pour écrire (écrase le contenu du fichier à l'ouverture)
188  # r pour lire uniquement
189  # a pour ajouter du contenu à la fin du fichier sans écraser.
190  # r+ pour lire et ajouter du contenu à la fin du fichier, sans écraser
191  with open("texte.txt", "r") as fichier:
192      contenu = fichier.read()
193
194  with open("ecriture.txt", "a") as fichier:
195      fichier.write("Ajout d'une ligne.\n")
```

Librairies et import

Les librairies peuvent etre installées avec pip. Attention à la version de python que vous utilisez !

```
PS C:\Users\leo.viguier> pip install nom_librairie --proy 100.78.40.201:8080
```

```
198  from datetime import datetime as dt
199  print(dt.now())
200
201  import time
202  print(time.time())
```

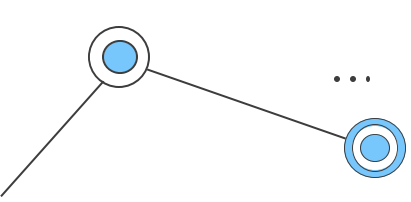
Gestion des erreurs

Les erreurs sont des appelées *Exception* en Python et peuvent etre gérées de la sorte :

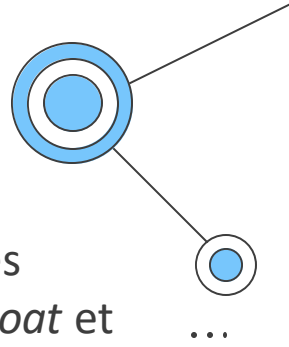
```
209  valeur = input("Entrez un chiffre : ")
210  try:
211      # essai d'un code
212      valeur = int(valeur)
213  except Exception as e:
214      # chose a faire en cas d'erreur
215      print(f"Vous n'avez pas entré un chiffre:\n{e}")
```

Les exception "remontent" les différents appels aux fonction jusqu'au programme principal, afin de le stopper.
Exécutez ca et regardez le résultat :

```
218  def fa():
219      val = int("a")
220
221  def fb():
222      fa()
223
224  def fc():
225      fb()
226
227  fc()
```



Dictionnaires



Les dictionnaires sont des itérables, indexés par des clés, et non ordonnés. Les valeurs peuvent prendre tous les types, mais les clés sont limitées à *str*, *int*, *float* et *bool*:

```
230 dico = {"clé": "valeur", 1.3333: [1, 3], "dico2": {1:"un", 2:"deux"}}
231 # itérer à travers le dictionnaire nous donne ses clés
232 # pour accéder à la valeur, on fait dictionnaire[clé] :
233 for cle in dico:
234     valeur = dico[cle]
235     phrase = f"La clé '{cle}' à pour valeur {valeur}."
236     print(phrase)
237
238 # on peut assigner des nouvelles valeurs et clés comme ceci:
239 dico[True] = "oui"
240 print(dico)
241 # affiche {'clé': 'valeur', 1.3333: [1, 3], 'dico2': {1: 'un', 2: 'deux'}, True: 'oui'}
```

Raccourcis utilisés dans le programme

Ces raccourcis ne sont pas apprendre, c'est juste pour vous aider à les comprendre lorsque vous les rencontrez dans le programme.

```
244  elements = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
245  chiffres_pairs = [i for i in elements if i % 2 == 0]
246  # i for i in elements, c'est pour iterer, comme une boucle for classique
247  # if i % 2 == 0 c'est la condition pour que i soit ajouter à chiffres_pairs
248  print(chiffres_pairs) # affiche [2, 4, 6, 8, 10]
249
250  path = "C:/Users/"
251  chemin = path if path.endswith("/") else path + "/"
252  # si path ne finit pas par /, on le rajoute
253  # var = valeur_condition_True if condition else valeur_condition_False
254
255  var = 0
256  var += 1 # correspond à var = var + 1
257  var -= 1 # correspond à var = var - 1
258  var *= 1 # correspond à var = var * 1
259  var /= 1 # correspond à var = var / 1
```