Etudiants : Pierre DIVOUX – Tristan SAVIGNE
Enseignants responsables : M. N.LAGA et M. P.E-MASSE

PROJET DE PROGRAMMATION INFORMATIQUE – PYTHON

- **GIT / GITHUB**

Nous avons tout d'abord pris en main github et git afin de mettre en relation nos travaux. Tristan a donc créé un projet sur son compte github et nous avons tous les deux créé un dossier du projet sur nos ordinateurs respectifs. Par la suite, à l'aide de l'invite de commande windows nous avons fait les manipulations nécessaires pour lier ce dossier à git. Ensuite, nous avons lié le dossier (sur nos ordinateurs) avec le projet github, grâce au lien de partage.

Afin d'utiliser cet outil nous devions faire un « clique droit » dans notre dossier et appuyer sur « git bash here ».Ensuite nous pouvions utiliser les commandes de sauvegarde, pour conserver nos travaux sur la plateforme gitHub :

- git add + nom du fichier
- git status
- git commit + un commentaire
- git push

Puis, pour actualiser les fichiers sur notre ordinateur nous faisions :

- git pull

Nous pouvons même créer des fichiers :

- Excel, PowerPoint, Word,…
- Etc

- **MAITRISE DE PYTHON :**

Tout d'abord, nous avons dû apprendre à ouvrir un fichier csv et à en extraire des données. Pour cela nous avons utilisé le module pandas.

Pour éviter d'avoir à rouvrir le dossier dans chaque programme, nous avons créé une variable globale, que nous avons nommé matrice, et qui nous permet d'avoir accès facilement au fichier csv n'importe où. Dans la même optique, afin de traiter les « stat date » et « end date », nous avons créé des variables globales : une plus petite que toutes les dates du fichier (default_sd) et une autre égale à la plus grande date du fichier (default_ed). De cette façon on a pu donner des valeurs par défaut à start date et en date, dans le cas où l'utilisateur n'en rentre pas.

De plus, pour prendre en compte la « start date » et la « end date », nous avons décidé de trier le fichier csv par ordre chronologique. Nous nous étions lancés dans la réalisation d'un programme python triant le fichier, mais il s'est avéré que sa complexité était beaucoup trop importante et que le tri du fichier aurait duré de nombreuses heures. Ainsi nous avons préféré trier le fichier directement sur Excel, qui propose une fonction nativement présente.

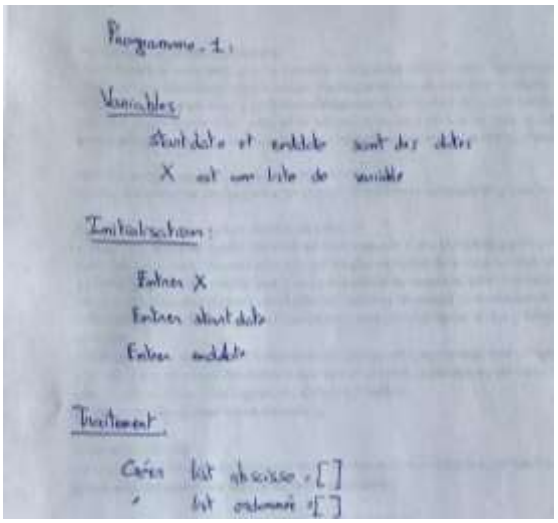
Une fois le fichier csv ouvert, nous l'avons manipulé comme une liste de listes, et ainsi avons extrait ses éléments avec la méthode : liste[i][j].

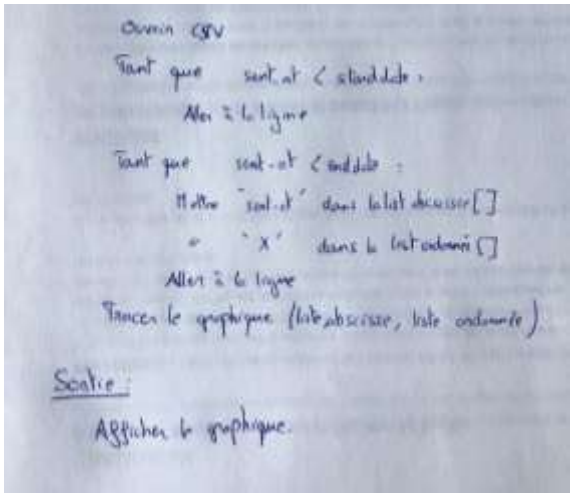
Une fois ces techniques acquises, il nous a fallu comparer les différentes dates du fichier. Cependant, que ce soit pour des chiffres ou des dates, python donnait comme type par défaut str (chaîne de caractères) à tous les éléments du fichier. Ainsi, lorsque nous voulions comparer ou utiliser des nombres du tableau, nous utilisions la méthode float(), par exemple : float(matrice[4][i]) pour retirer le nombre de la quatrième colonne, à la ligne i. Pour les dates, nous avons utilisé la méthode dt.strptime(matrice[7][i], '%Y-%m-%d %H:%M:%S%z') afin de transformer les chaînes de caractères en dates, en respectant bien le modèle utilisé dans le fichier. En effet, %Y correspond à l'année en 4 chiffres, %m au mois en 2 chiffres, %d au jour en deux chiffres, %H à l'heure en 2 chiffres et 24 heures, %M aux minutes en deux chiffres, %S aux secondes en deux chiffres et %z au fuseau horaire comme écrit dans le fichier.

Enfin, pour l'aspect de nos graphiques, nous avons appris à leur donner un titre avec plt.title() et à y placer des flèches (voir commande utilisée sur les programmes).

- **ALGORITHME**

Pour commercer le projet nous avons mis les programmes sous forme d'algorithmes simplifiés et non codés de type : Variables, Initialisation, Traitement, Sortie.





Puis, nous avons voulu prendre chaque question individuellement, pour répondre au problème posé. Le programme est donc divisé en 2 parties :
- Les quatre premiers programmes répondent à chaque question et le cinquième en fait la synthèse.
- Le reste des programmes permets de relever les différentes erreurs que nous avons remarqué.

Tout d'abord développons la **première partie** :

En préambule nous avons dû trier les dates du CSV dans l'ordre croissant.

- Le « programme_1 » :

Afin d'afficher une variable sur un graphique en fonction du temps, le programme se développe avec des listes (abscisse et ordonnées) qui sont remplis par les boucles « While » avec les variables temps et X. Puis grâce à « plt.show » il peut afficher ce qui est demandé.

- Le « programme_2 » :

Pour obtenir les données statistiques le programme nécessitait d'importer des modules depuis « anaconda ».

- Le « programme_3 » :

Calculer l'indice humidex par rapport au temps est possible toujours par la mise en place de liste pour les abscisses (temps) et ordonnées (humidex) géré par des boucles « while ».
Ensuite, la bouche « for » permet au programme d'obtenir toutes les valeurs d'humidex en fonction des variables d'humidité et de température (organisés à travers 2 listes).

- Le « programme_4 » :

Pour obtenir l'indice de corrélation entre deux couples de variables, nous cherchons tout d'abord à identifier les deux couples dans le CSV, d'où l'utilisation des deux première boucles « while ». Puis des boucles « for » ajoute à des listes toutes les valeurs de ses variables. Ensuite, une nouvelle boucle for permet de faire l'espérance x*y. Enfin, l'indice de corrélation est calculé. Le programme s'achève avec la mise en forme des deux graphiques avec leurs légendes.

- Le « MONSCRIPT » :

Cet algorithme de synthèse, permet d'exécuter tous les programmes énoncés précédemment grâce aux « actions » énoncées dans le sujet. Il fait correspondre chaque action à son programme.

Ensuite la **deuxième partie** :

- « erreur_lum() : »

On remarque dans les données que la lumière des bureaux s'allume aux alentours de 7h45 tous les matins et s'éteint vers 20h40 le soir. Cependant, il peut arriver qu'il y ait de la lumière détectée entre 20h40 et 7h45 sans pour autant que ce soit une erreur (sûrement de la lumière issue de l'extérieur) : captation de lumière faible (< 10) et non régulière. Considérons qu'il y a une erreur pour toute captation > 30 et pour toute captation de lumière sur 3 échantillons de suite entre 22h et 7h (si on prend en compte les gens qui peuvent arriver plus tôt ou repartir plus tard). Pour un souci de lisibilité, on ne montrera qu'une erreur par jour s'il y en a.

Pour ce faire, le programme délimites jours que nous avons (du 11 au 25 Aout). Puis pour trouver les erreurs qui se produisent la nuit, il met en place une boucle « while » délimitant un cadre de recherche. Afin de séparer le passage d'un jour un autre, l'algorithme développe deux boucles « while » distinctes mais similaires dans leur fonctionnement. Ils permettent de détecter la première erreur dans chaque journée à travers deux méthodes :
Si un pic est situé au-dessus de 30 (lux) ou si trois valeurs de suites ne sont pas égale à 0 indépendamment.
Ensuite, le programme affiche la courbe « lum » en fonction du temps. Enfin à l'aide d'une boucle « for » des flèches montants les erreurs apparaissent.
L'algorithme s'achève en affichant la courbe en fct du temps avec les flèches et un titre.

- « erreur_humidity() : »

Normalement, l'humidité relative et la température ont un comportement miroir, c'est a dire h augmente quand t décroit et réciproquement. Cependant, on observe dans les données qu'il y a une zone ou cela n'est pas vérifié. Le but est de faire un programme capable de déceler ces zones. On choisit de relever comme erreur tous les points tels que humlidity et temp ont des sens de variation identiques deux fois de suite.

Pour ce faire, nous avons d'abord comparé le sens de variations des courbes « humidity » et « temp » puis relevé les points où ce sens est identique pour les deux courbes (alors qu'il devrait être opposé). Ces points étant relativement nombreux, nous avons décidé de relever uniquement les points où le sens de variation est différent deux fois de suite. Nous avons programmé cela à l'aide d'une boucle « while » qui remplit la liste des erreurs.
Ensuite, le programme affiche la courbe « humidity » en fonction du temps. Enfin à l'aide d'une boucle « for » des flèches montants les erreurs apparaissent.
L'algorithme s'achève en affichant la courbe en fonction du temps avec les flèches et un titre.

- « erreur_co2() : »

D'après l'étude graphique, on constate que la quantité de co2 est anormalement élevé certains jours. D'après les données, on considère qu'au-dessus de 650 ptm, la quantité de co2 est anormale. Notre programme a donc pour objectif de relever et indiquer les pics anormaux.
Pour ce faire, on se sert d'une boucle « while » pour remplir la liste des erreurs.
Ensuite, le programme affiche la courbe « co2 » en fonction du temps. Enfin à l'aide d'une boucle « for » des flèches montants les erreurs apparaissent.
L'algorithme s'achève en affichant la courbe en fonction du temps avec les flèches et un titre.