|  |
| --- |
|  |

**Rapport Projet Programmation Python IVP1**

**Lien Github: https://github.com/nathangigou/projetpython.git**

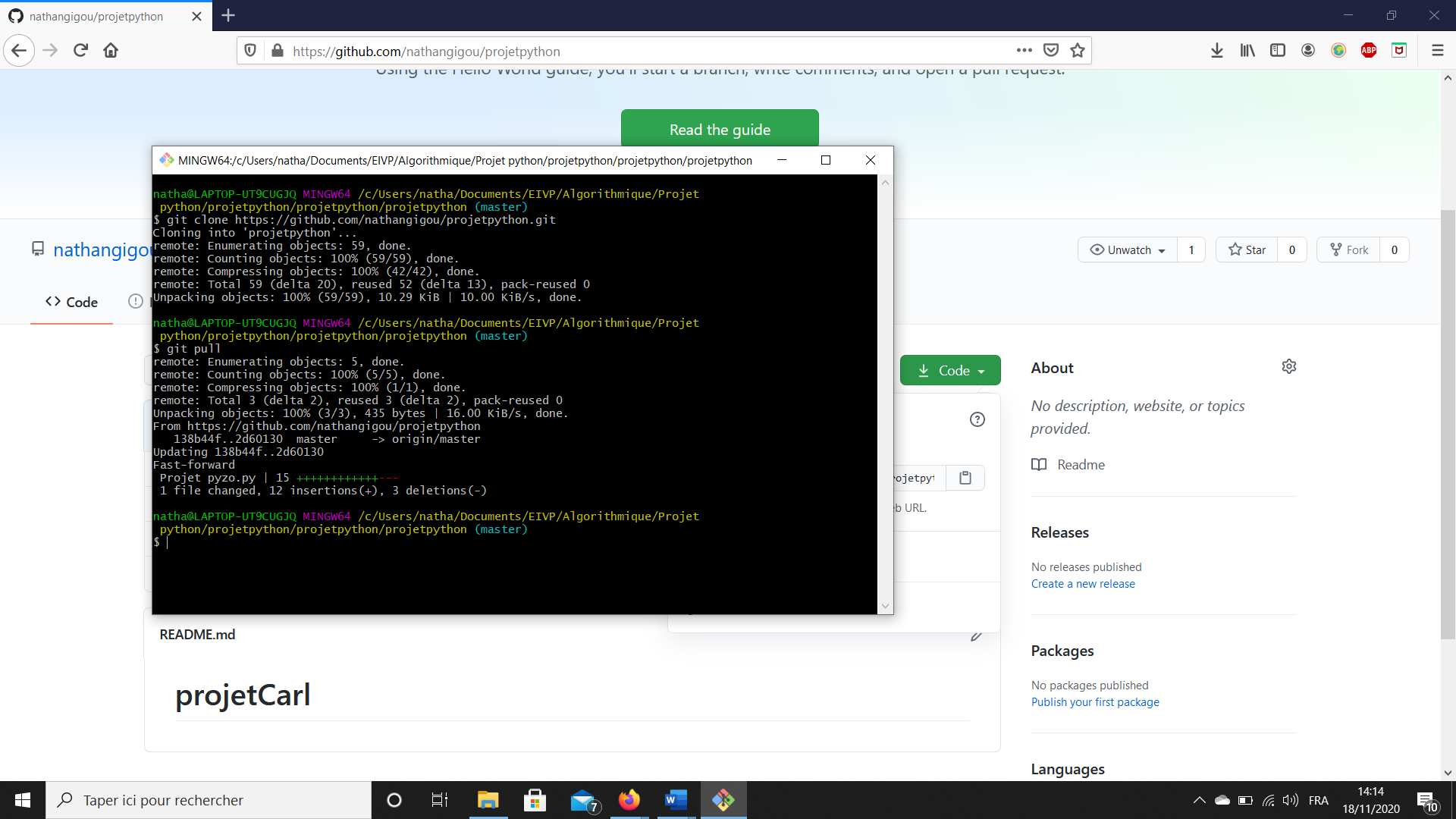
1. **Github/Git**

Après plusieurs jours, nous avons compris comment appréhender l’interface de partage qu’est Github et nous sommes familiarisé avec les commandes de base de Git. En effet le plus complexe était de bien initialiser la liaison entre notre dossier de travail personnel et Github. De plus, une mise en main efficace de ce logiciel demandait une bonne compréhension des différentes branches de modifications.

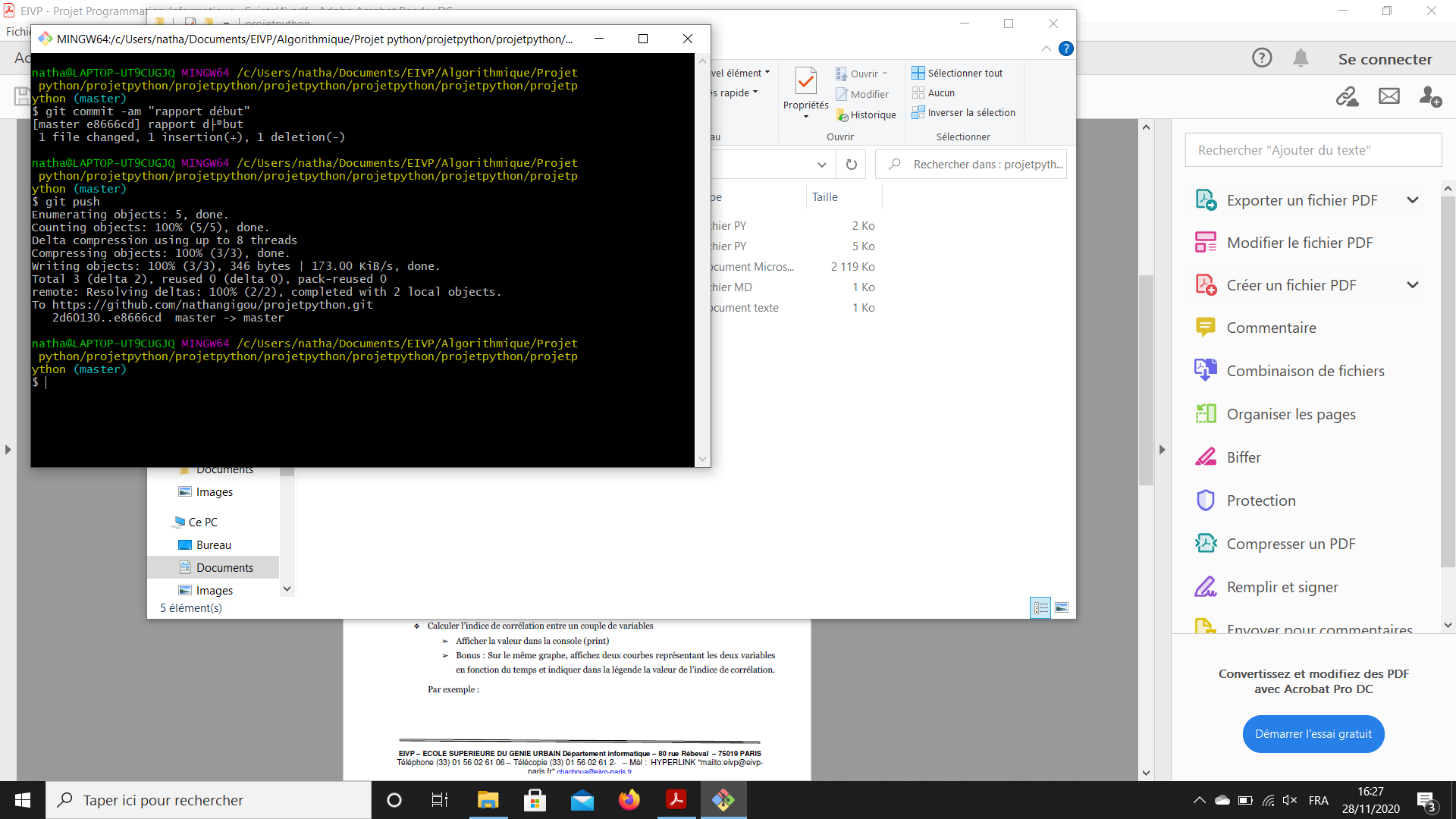
Nous avons donc créé un répertoire sur Github accessible à tous les deux. Même si la prise en main n’est pas tout de suite intuitive, le résultat est à la hauteur car il simplifie grandement l’interaction d’écriture du programme et garantit non seulement un accès, mais aussi une sauvegarde et un historique.

En effet, nous utilisions l’un la console Git Bash et l’autre l’invite de commande :

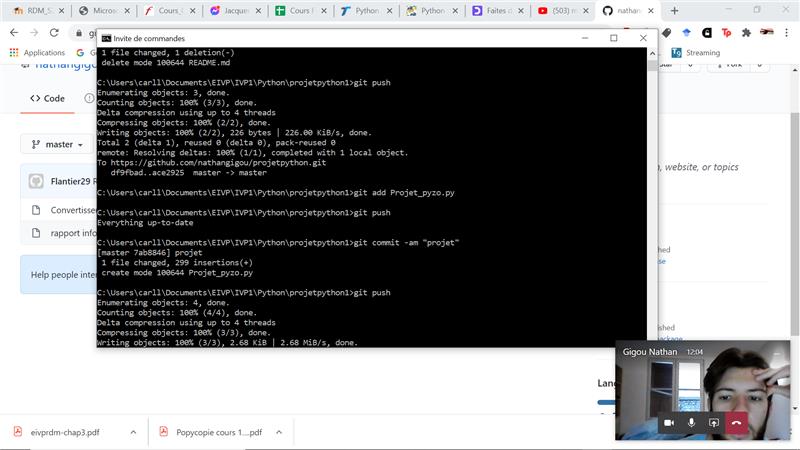
Pour récupérer les modifications apportées par le binôme nous utilisons(*git clone)/git pull*



Pour envoyer nos propres modifications au serveur, nous utilisons *git commit -am/git push*



Enfin pour ajouter un document nous utilisons *git add*

**

Github nous permet ensuite de visualiser rapidement les modifications et les commentaires associés.

L’inconvénient est qu’on ne peut pas travailler simultanément sur les fichiers.

1. **Compréhension du sujet**

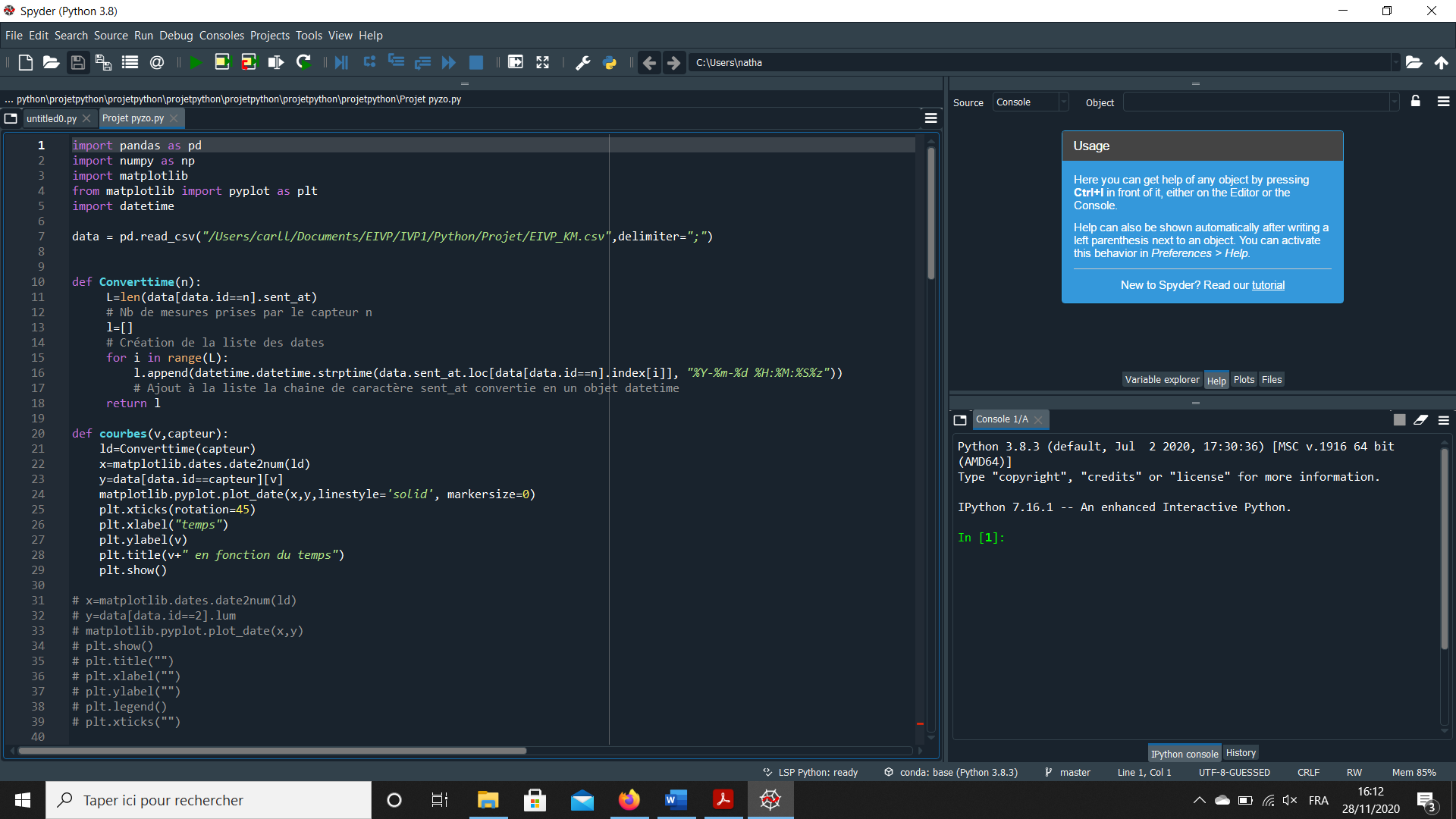
Le but du projet est d’appréhender la gestion/ le traitement de données, ici à partir d’un fichier CSV.

On va donc modéliser des représentations graphiques, des points/valeurs clés et évaluer les liens entre les sous-ensembles de données.

En effet, notre tableau initial comprend des relevés de 6 différents capteurs, des paramètres lumière-co2-bruit-humidité-température, relevés recueillis à différents horaires quasi-périodiquement.

1. **Ecriture du programme**

La première étape est tout simplement de lire le fichier CSV grâce à Python et pour mieux appréhender le format horaire de l’heure des relevés, nous séparons le fuseau horaire, le mois, le jour, l’heure, la minute et la seconde grâce au module datetime. Cela nous facilite l’utilisation de cette donnée.

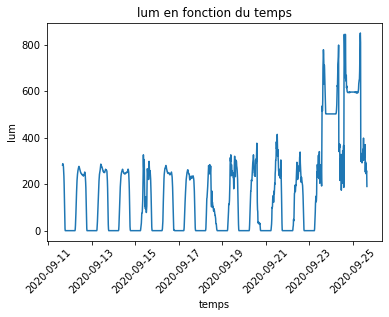


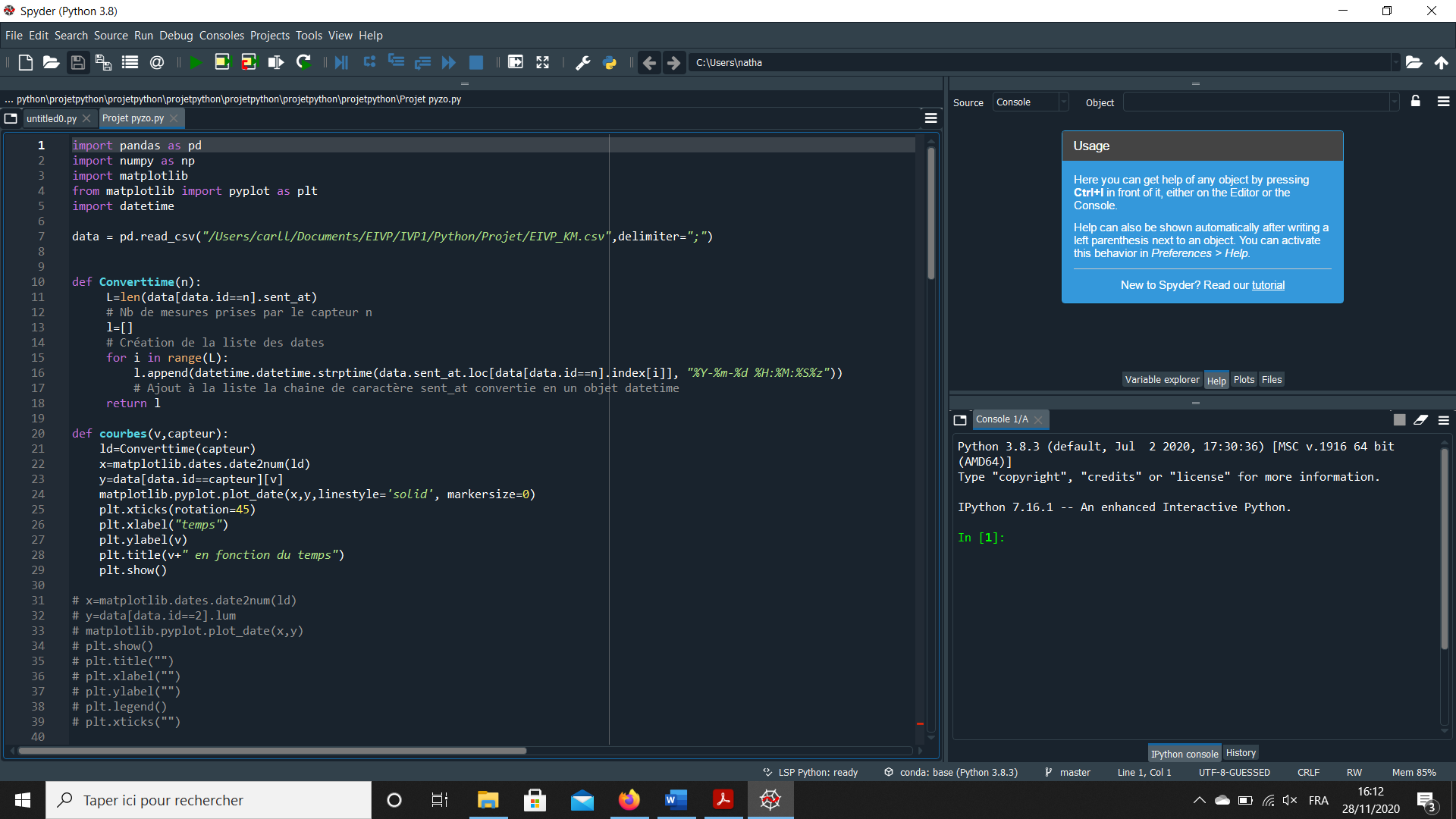
*La complexité est θ(L), car on ne fait qu’un parcours.*

* Affichage des courbes d’un paramètre et capteur particulier en fonction du temps

Il s’agit ici de sélectionner les plages de données qui nous intéressent et d’être capable de sélectionner un capteur et un paramètre à partir du fichier brut.

Par exemple, on veut afficher la luminosité du capteur 1 :





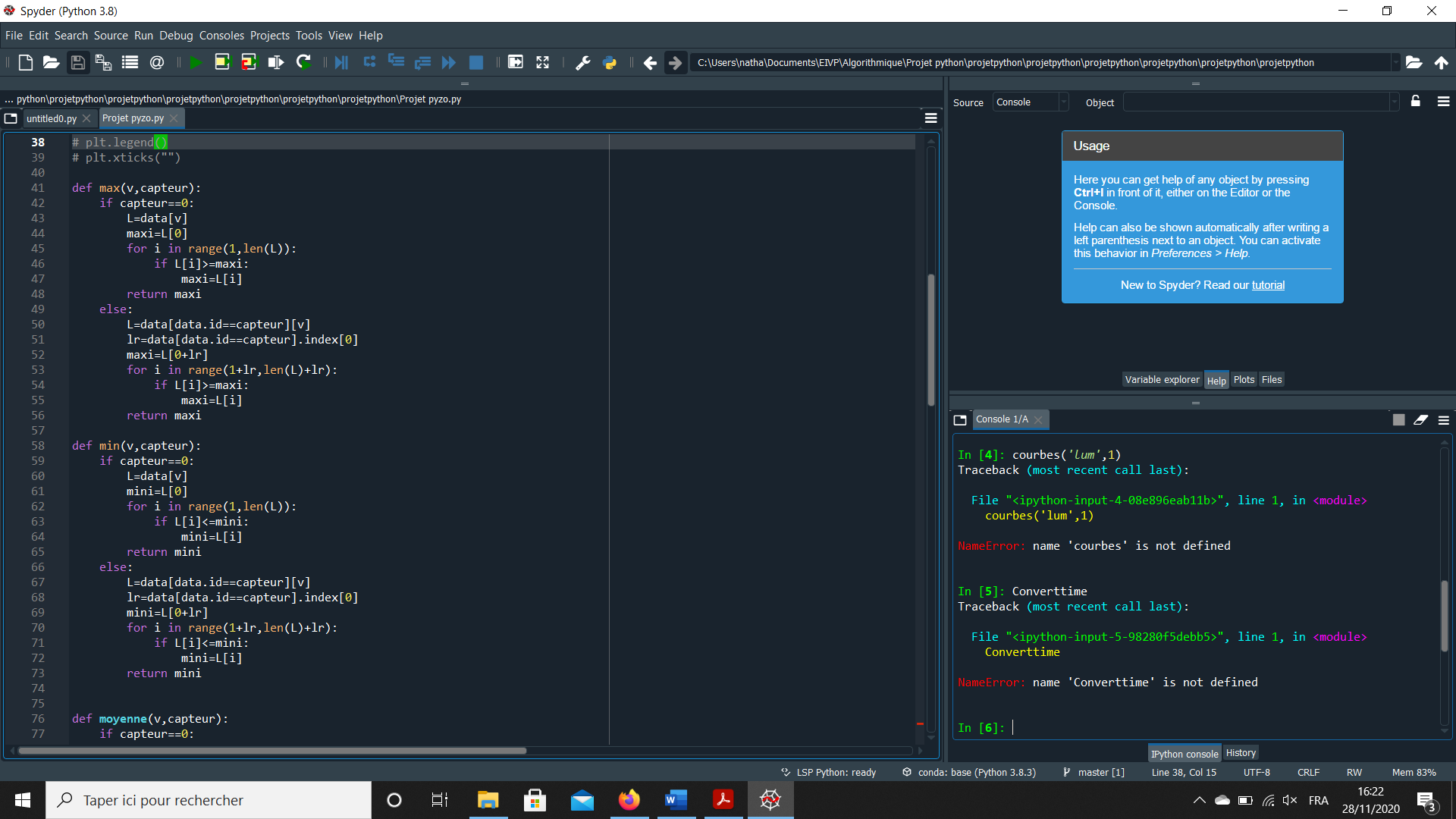
*La complexité de cette fonction est θ(L) qui découle de l’utilisation de Converttime.*

On affilie à l’axe des abscisses la donnée temporelle convertie préalablement, l’axe des ordonnées dépend des deux paramètres de la fonction à savoir le numéro du capteur et v l’entité qu’on veut afficher (luminosité, température…).

On utilise ensuite la fonction plot pour afficher, le reste des commandes est de la mise en forme.

* Affichage des valeurs statistiques (min, max…) sur la courbe

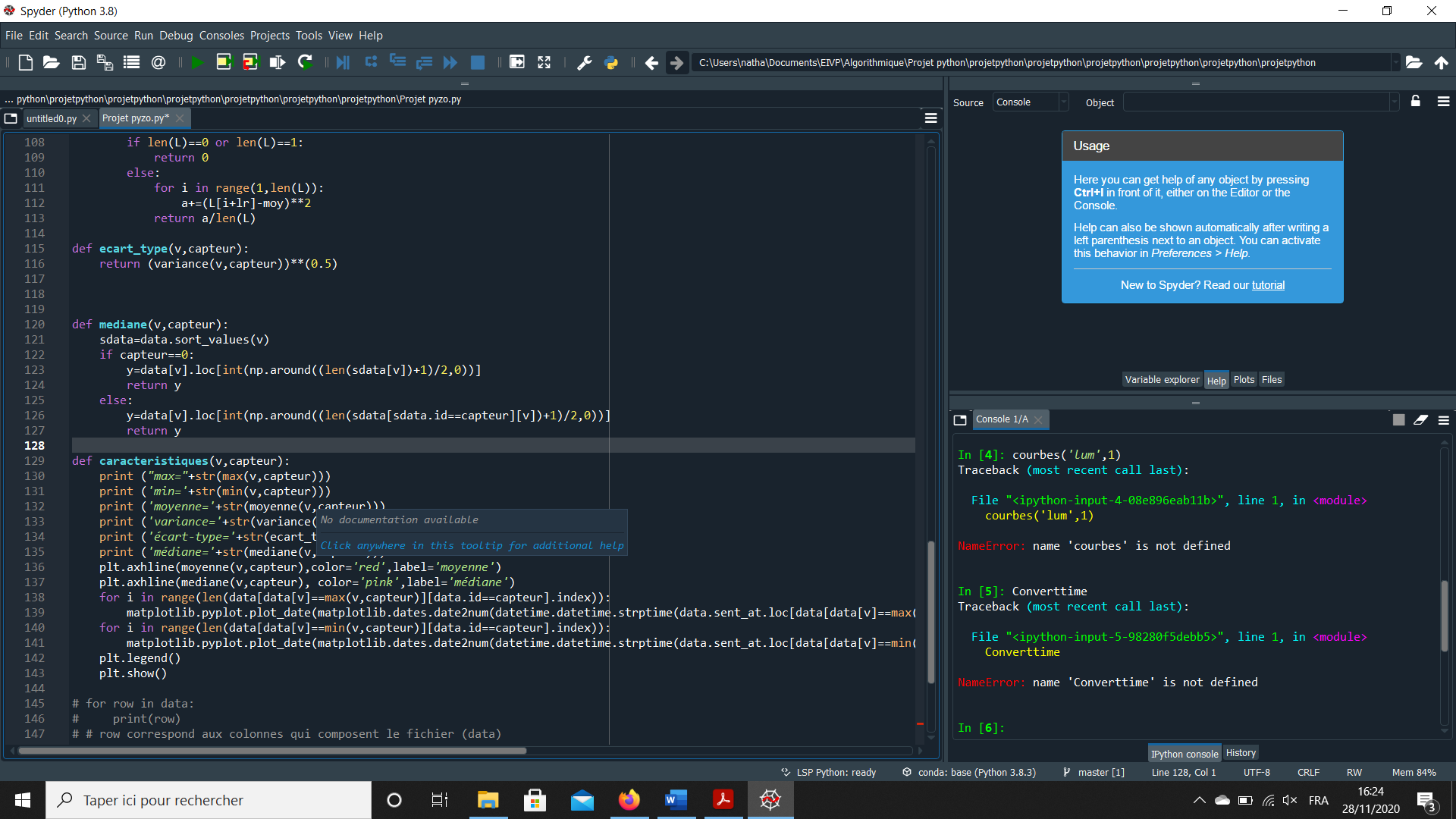
D’abord on écrit chaque sous fonctions :



*Complexités : θ(taille de l’échantillon du capteur)*

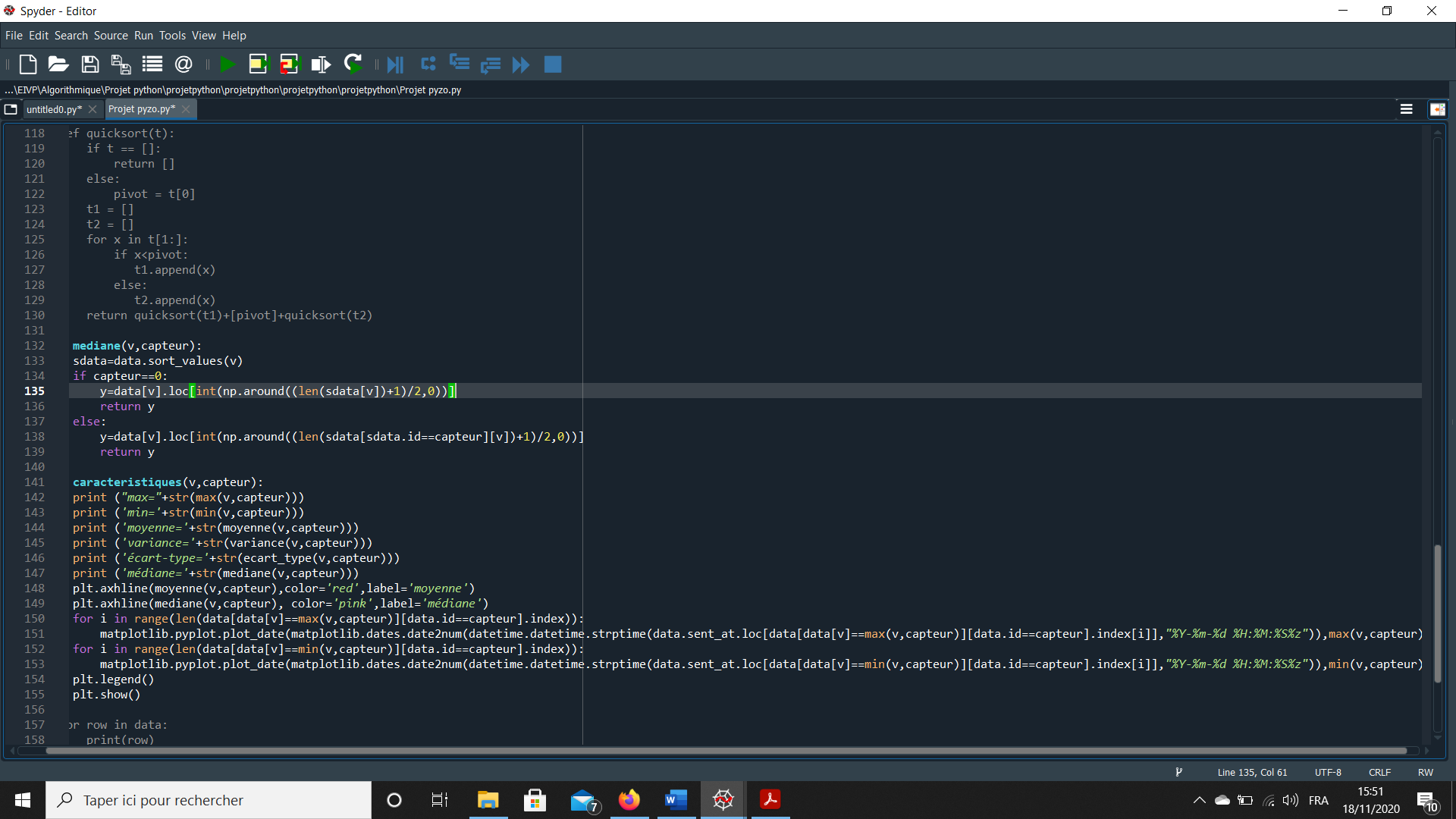
*A cause de l’unique parcours à chaque fois*





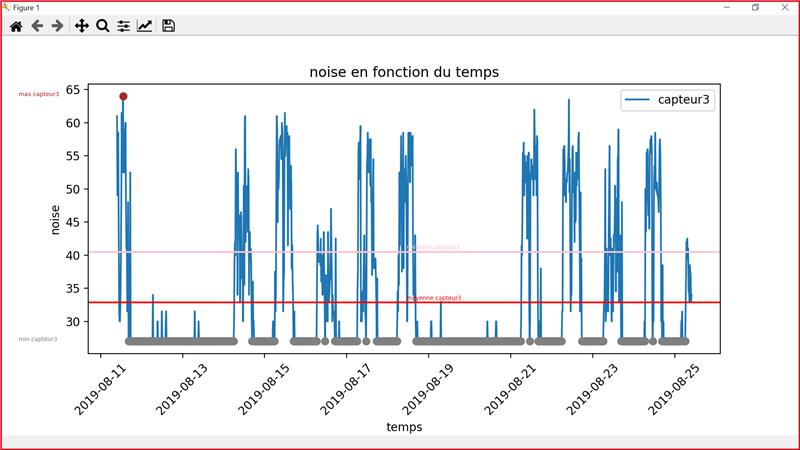
*Complexités : θ(1)*

Puis la fonction d’affichage qui reprend les fonctions précédentes :



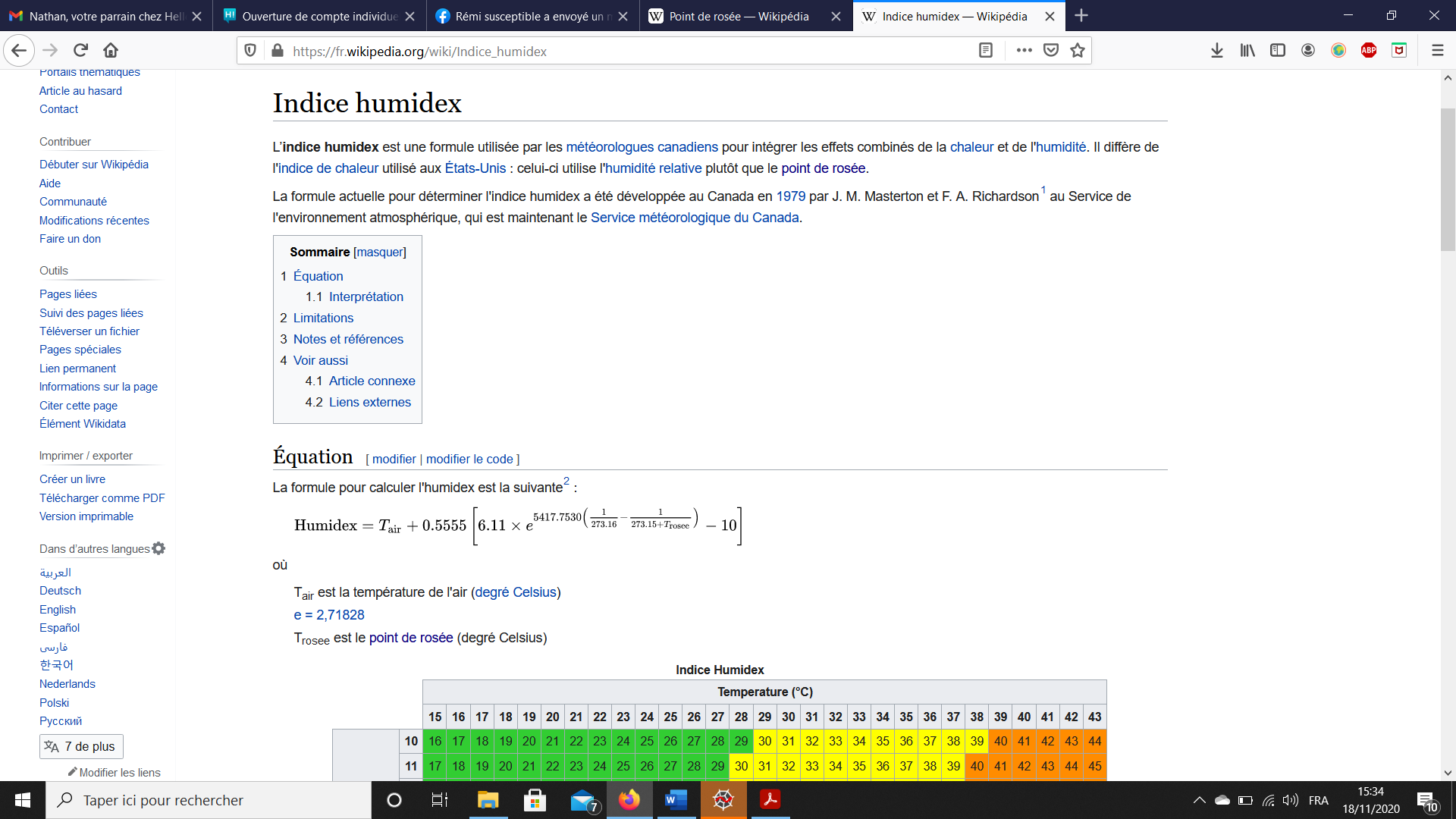
*La complexité est ici de θ(2\*taille des échantillons de chaque capteur par variable) à cause des deux boucles.*

Les deux boucles permettent d’afficher tous les points max et min, si les valeurs extremum sont atteintes plusieurs fois puisqu’on parcours toutes les données au lieu de s’arrêter à la première valeur max et min.

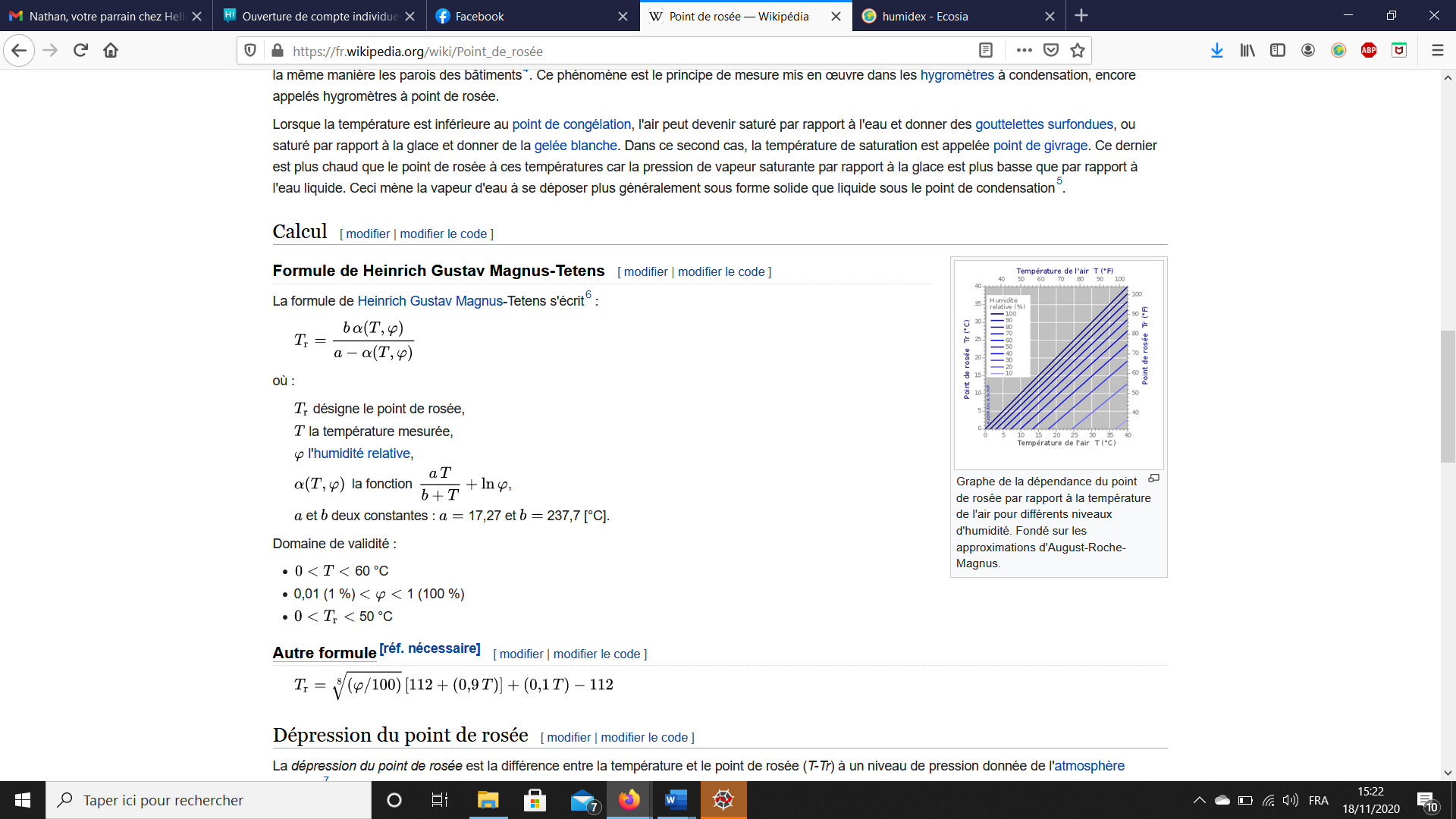


* Humidex

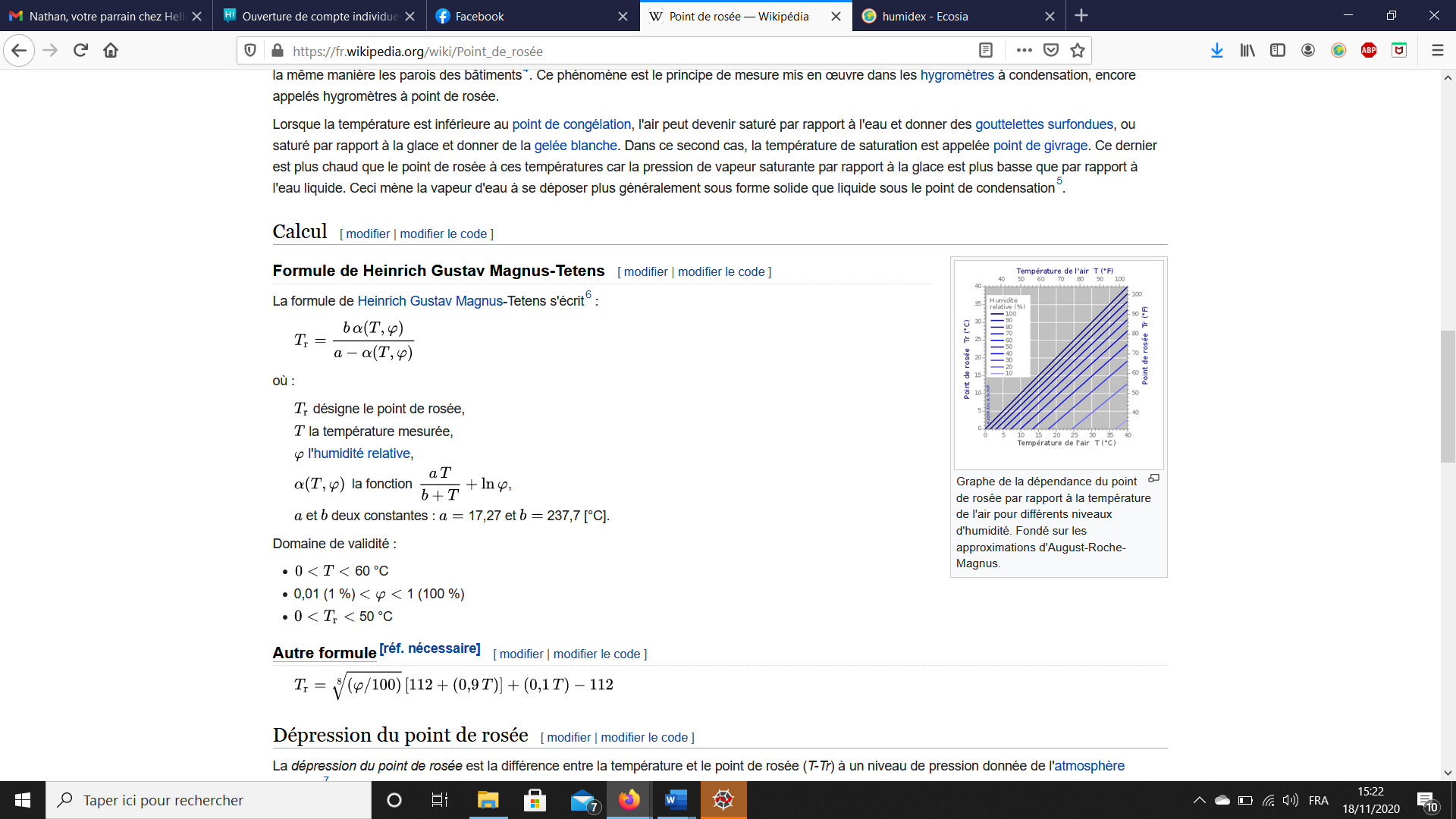
Après de rapides recherches nous avons trouvé la formule de l’humidex (indice permettant d’intégrer les effets combinés de l’humidité et la chaleur)



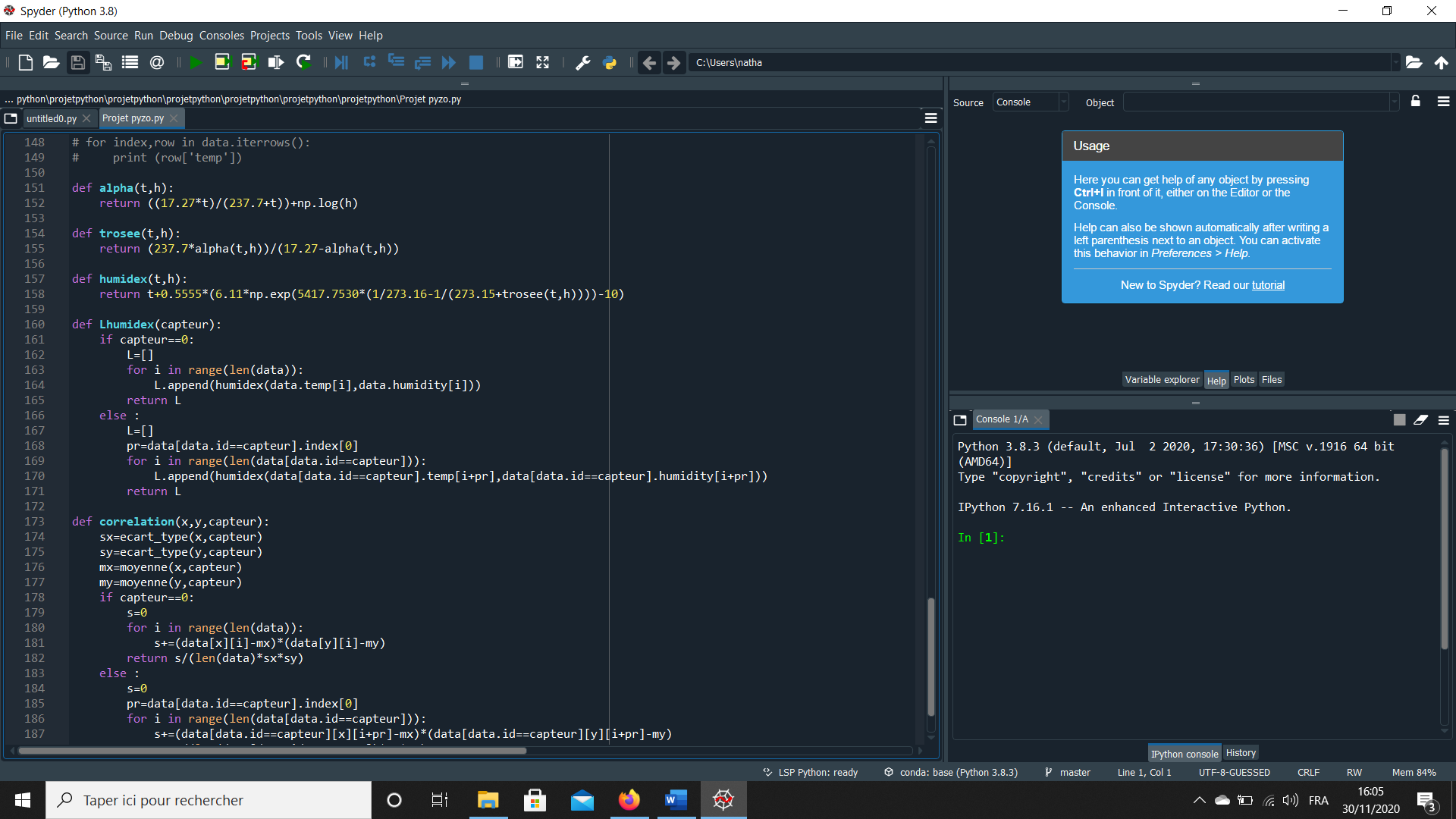
Nous avons besoin du point de rosée dont la formule apparaît ci-dessous :



Sachant que cette formule demande une dernière formule alpha :

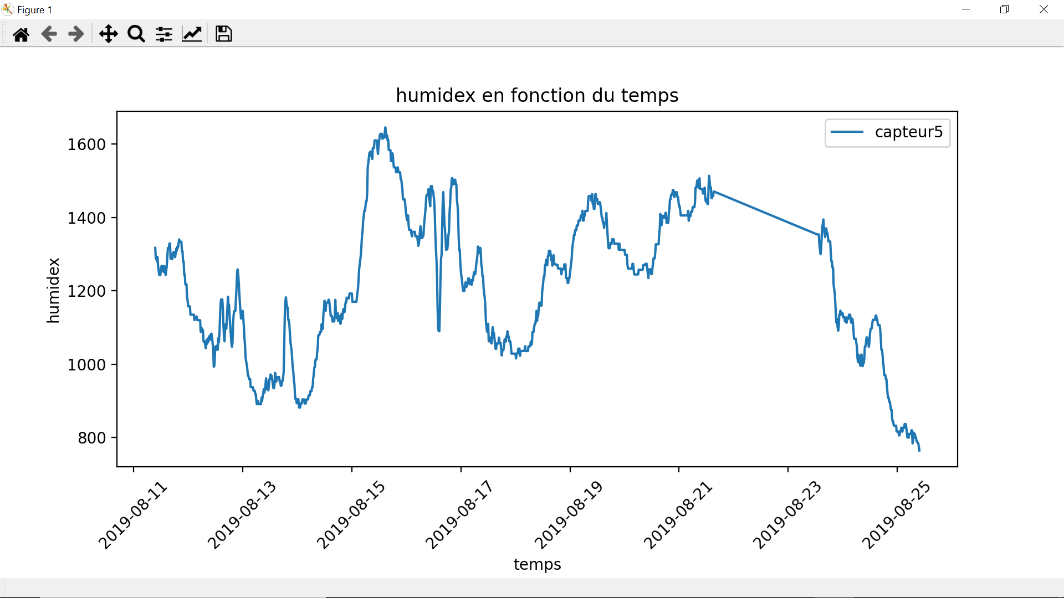
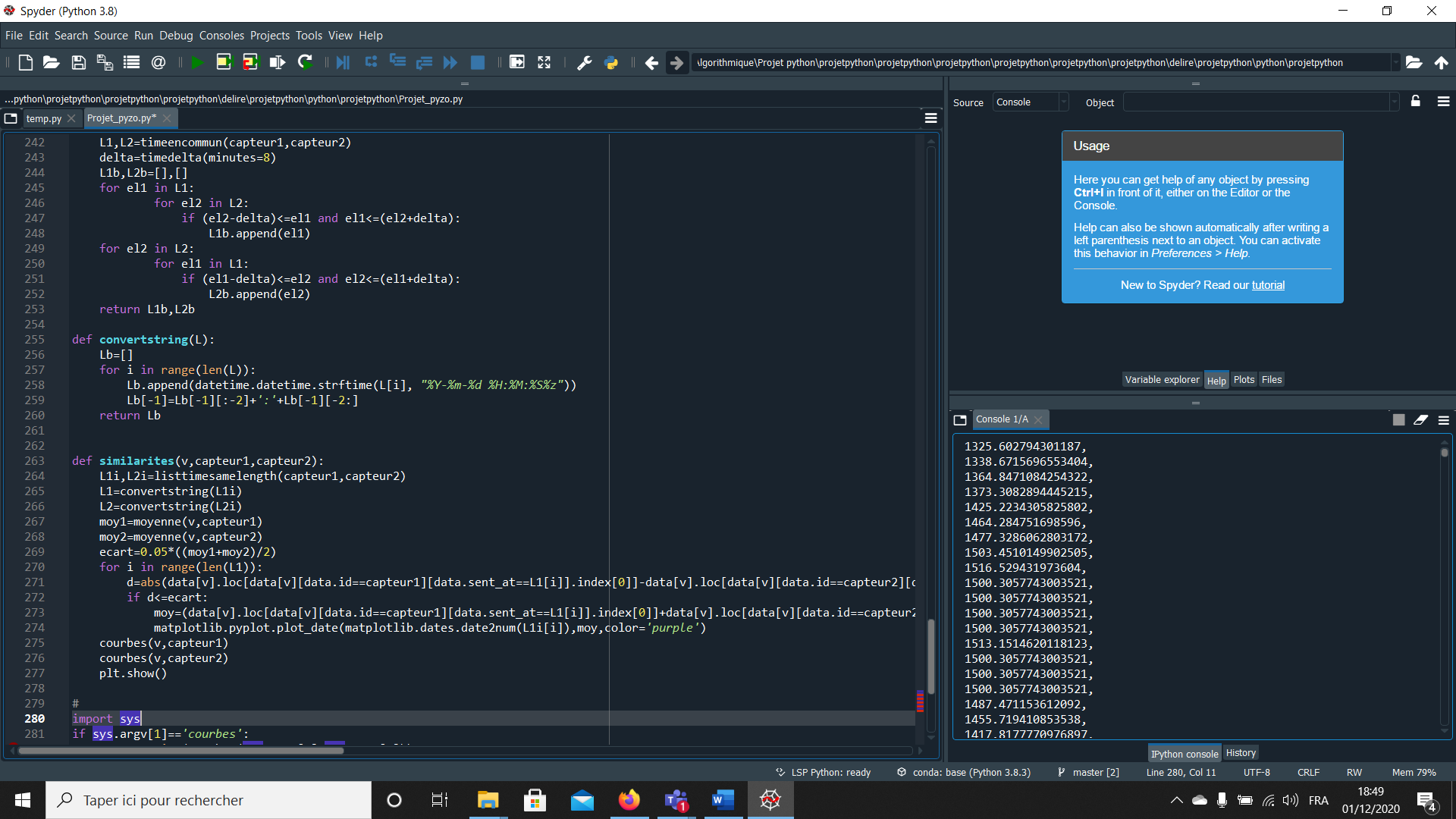


Ce qui se traduit au niveau du code par :



*La complexité est de θ(taille des échantillons pour chaque capteur)*

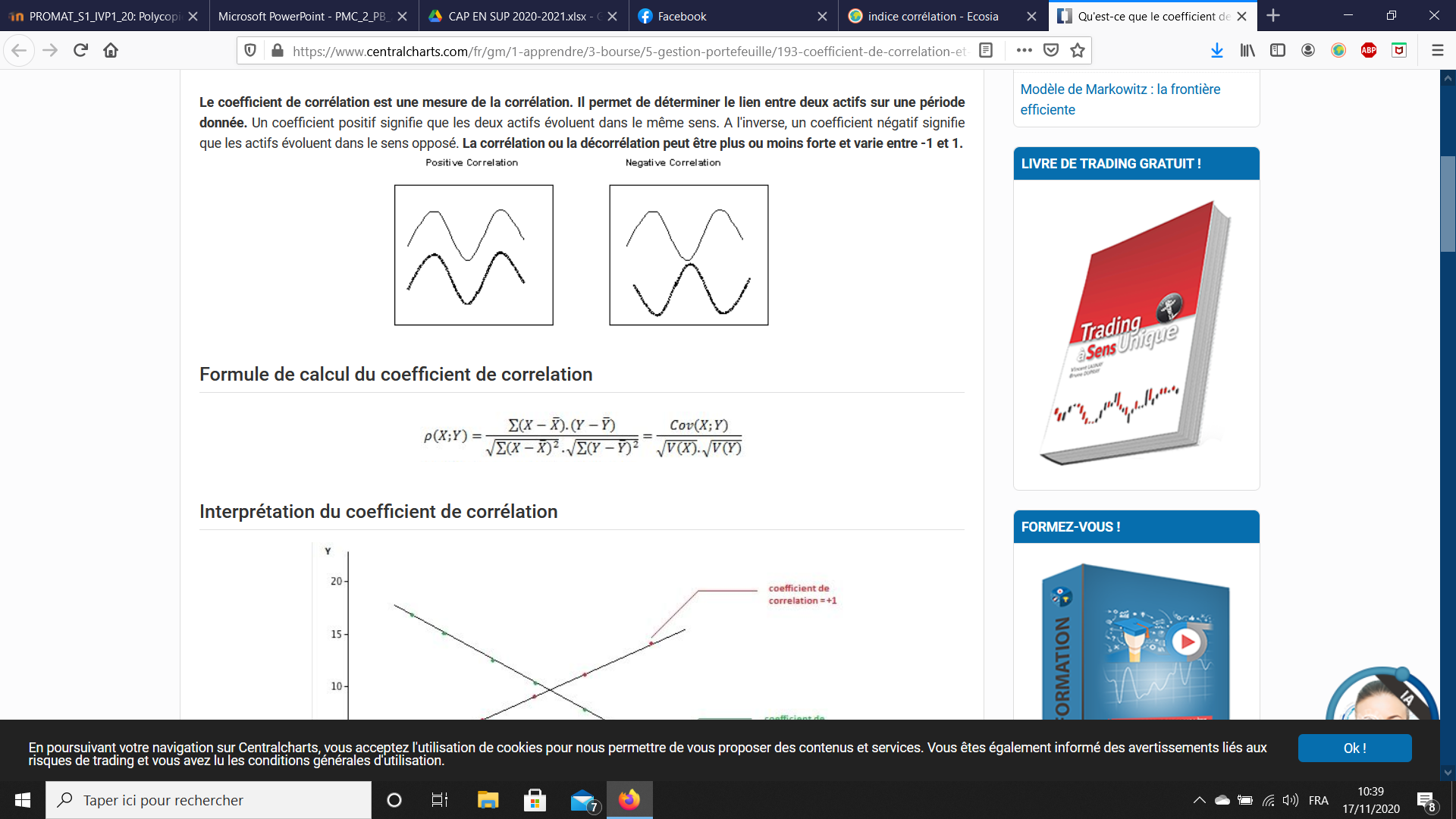
Notre fonction, grâce à la boucle retourne tous les humidex du capteur choisi.

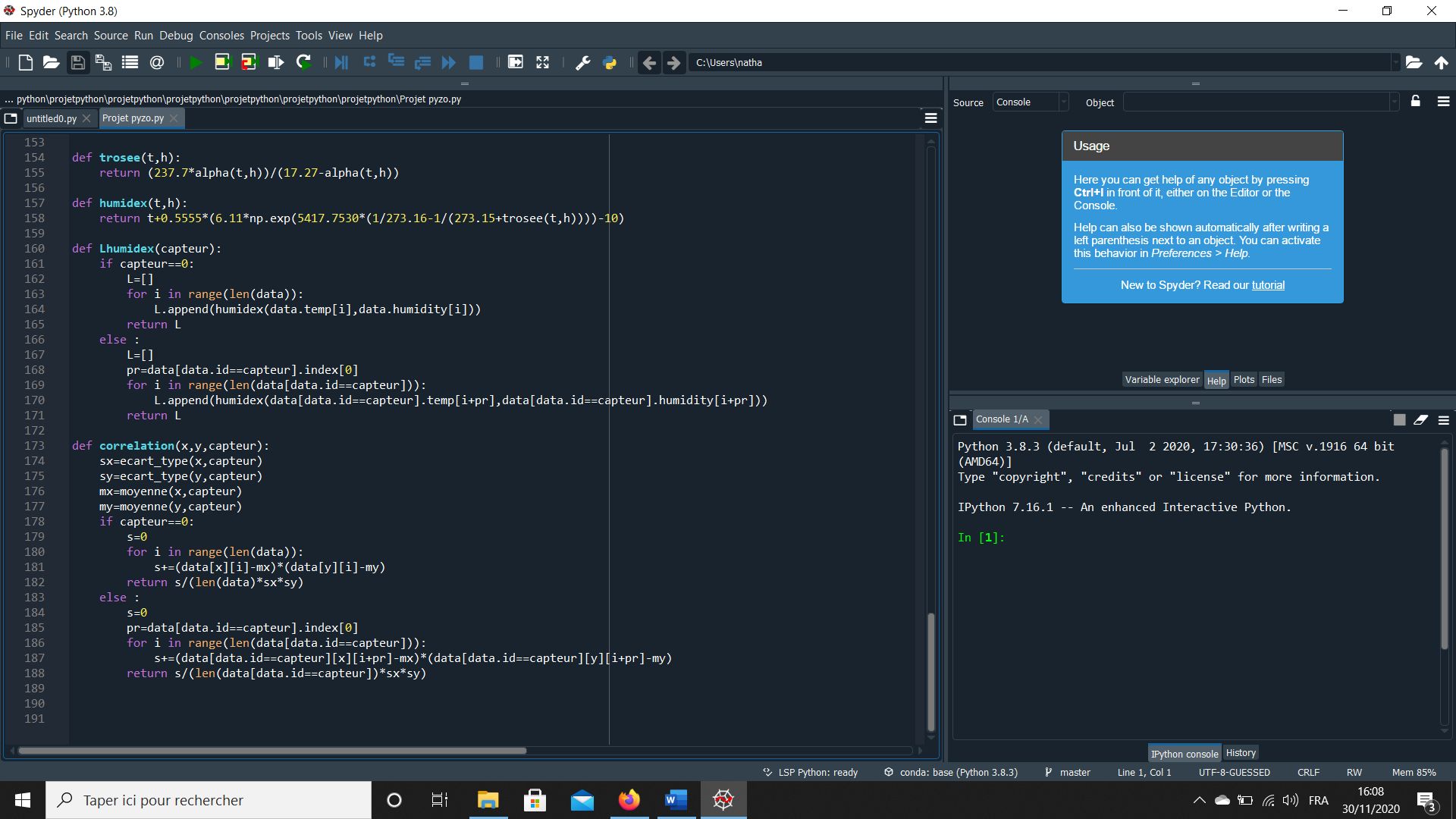


**...**

* Indice de corrélation

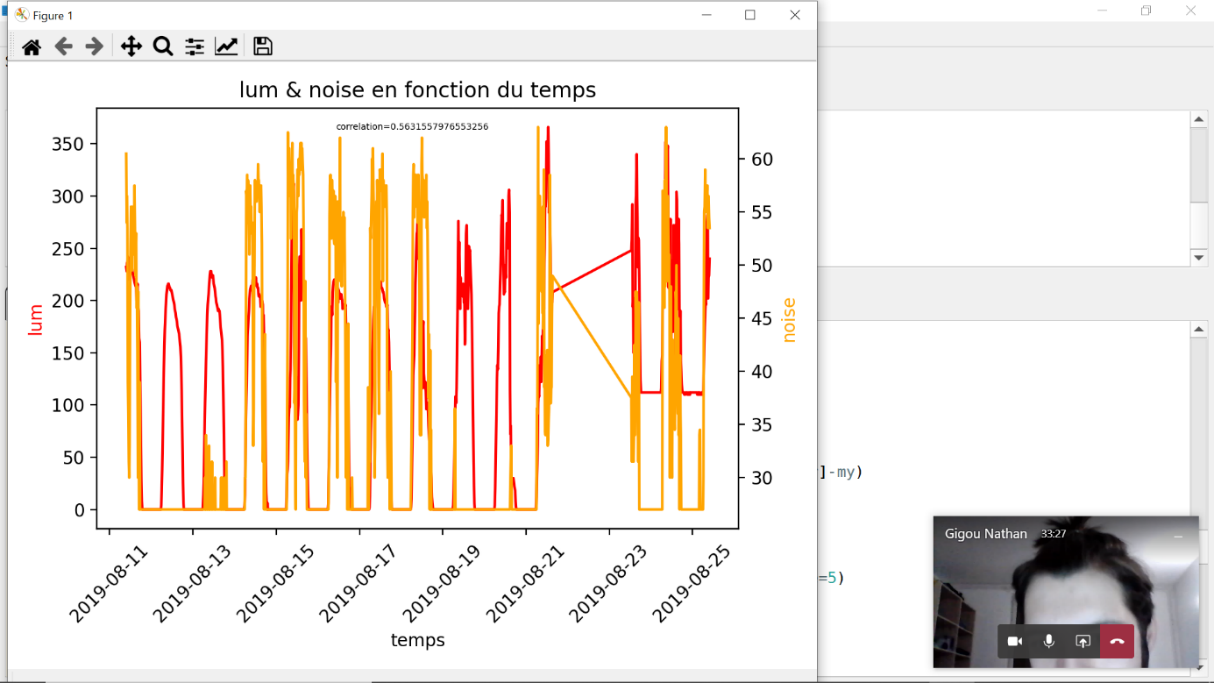
Ci-dessous la formule





*La complexité est de θ(taille des échantillons pour chaque capteur)*

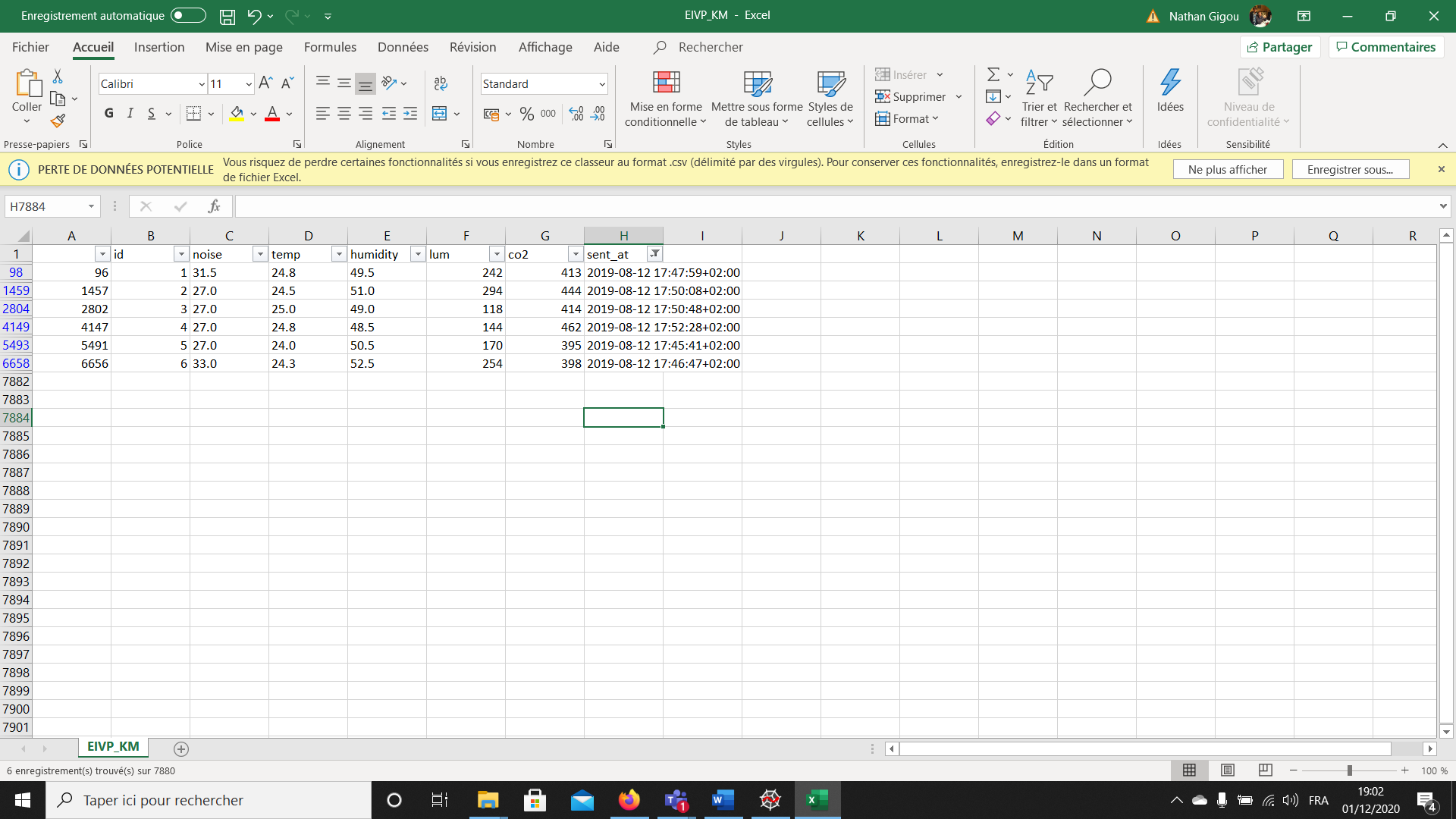
La somme est réalisée grâce à la boucle et on utilise les fonctions moyennes et écart-type déjà écrites.



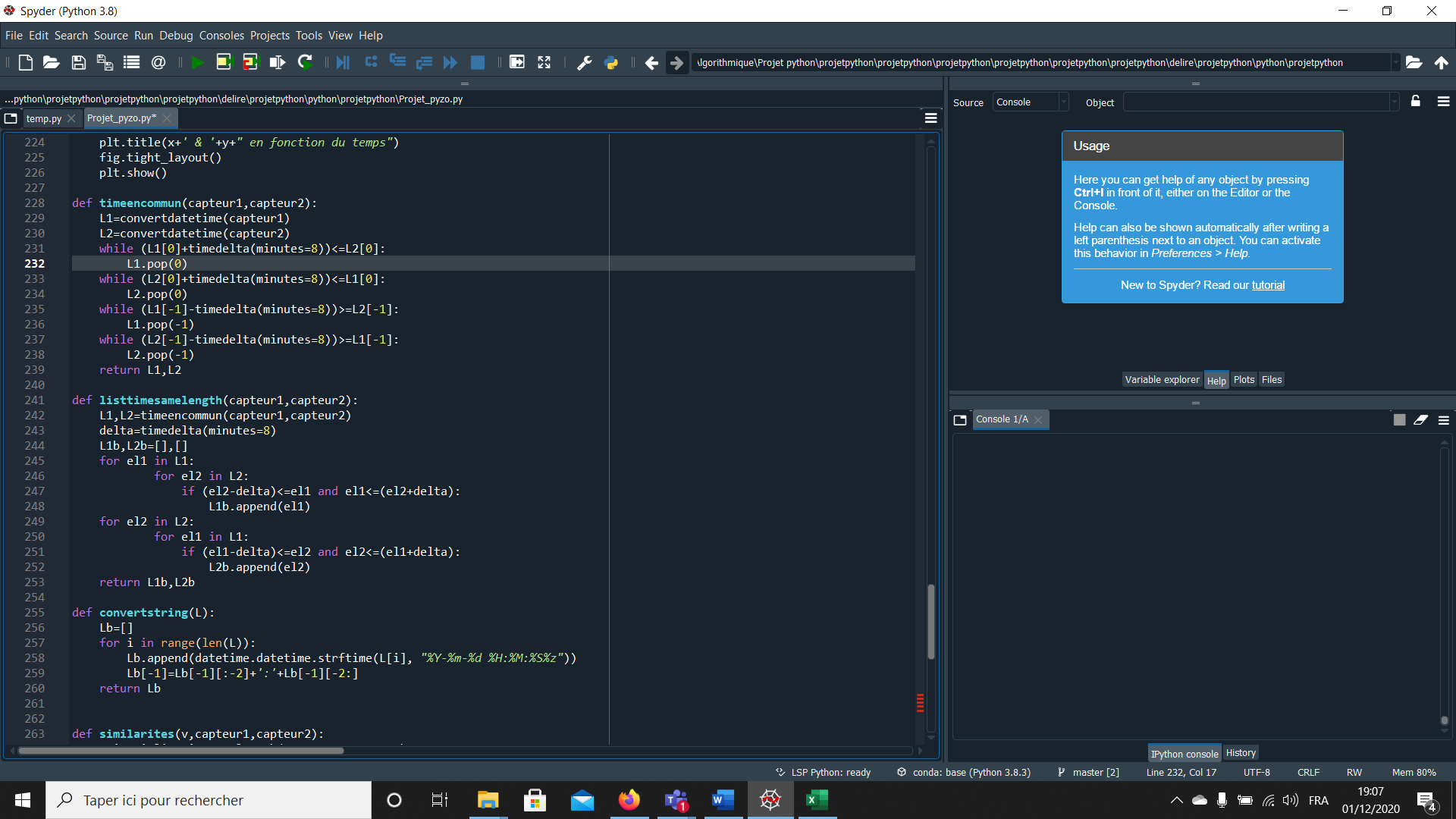
* Recherche des similarités

On souhaite être capable de savoir si les comportements des capteurs sont proches, s’ils sont posés à des endroits proches ou non, on va donc s’intéresser aux écarts points par points et admettre un pourcentage d’écart considéré comme indicateur de l’équivalence (- de 5% de la moyenne des moyennes de la variable des capteurs = similaires).

Seulement pour réaliser cette comparaison point par point, il faut des listes de relevés de même taille et dont les dates de prises de relevés sont coïncidentes. On remarque que les relevés sont effectués environ toutes les 15 minutes pour chaque capteurs. Les relevés sont légèrement décalés entre chaque capteur mais grâce à un intervalle de 8 minutes on peut retrouver les valeurs liés à une même plage horaire :



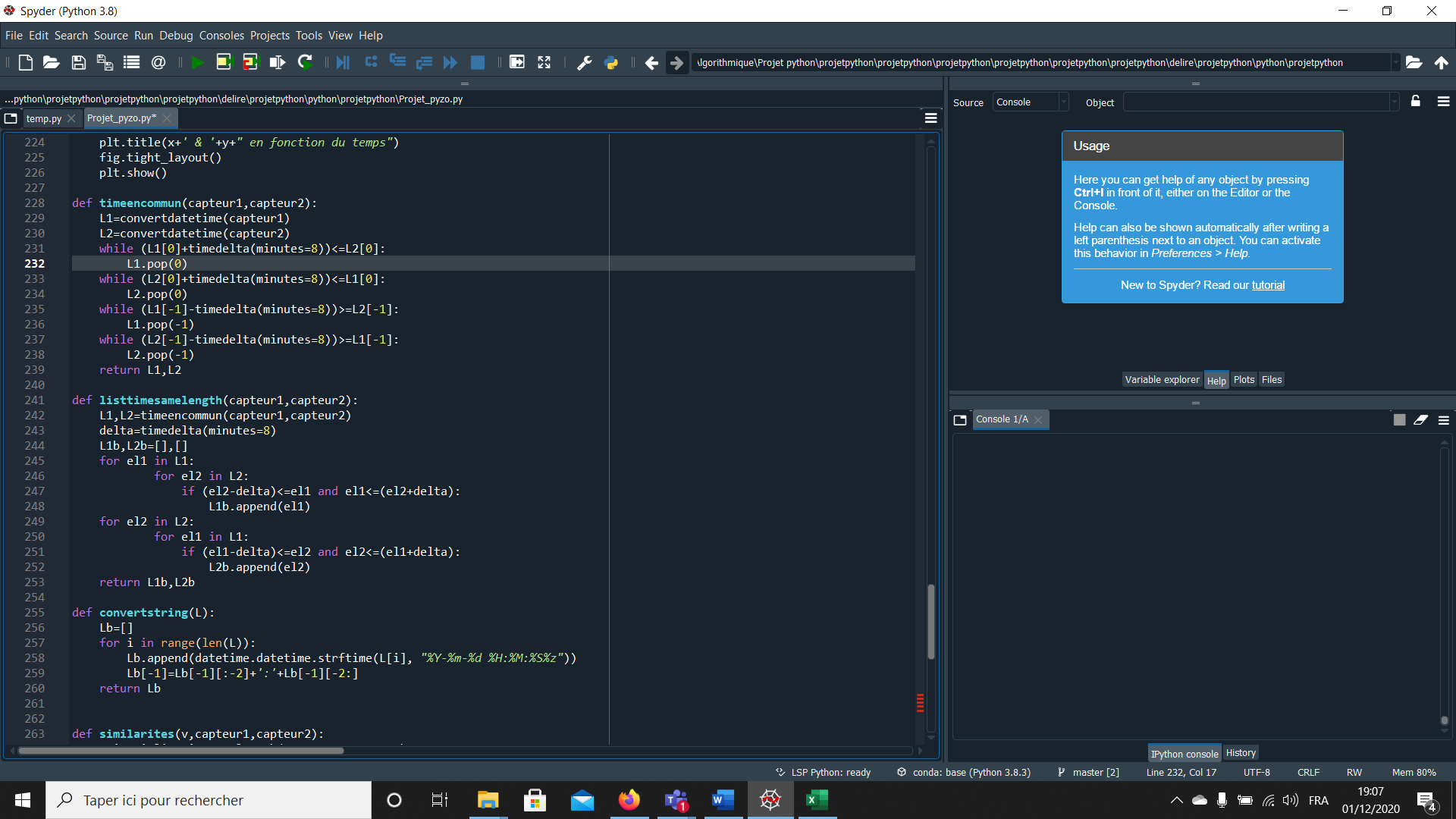
Le soucis est alors d’avoir des tableaux de même longueurs mais 2 problèmes viennent à jour :

* Certains capteurs commencent les relevés plus tôt ou finissent plus tard, il faut donc tronquer les listes pour qu’elles se coordonnent sur une même plage horaire commune

*Complexité : θ(taille des données capteur 1+ taille des données capteur 2)*

*A cause de converttime*

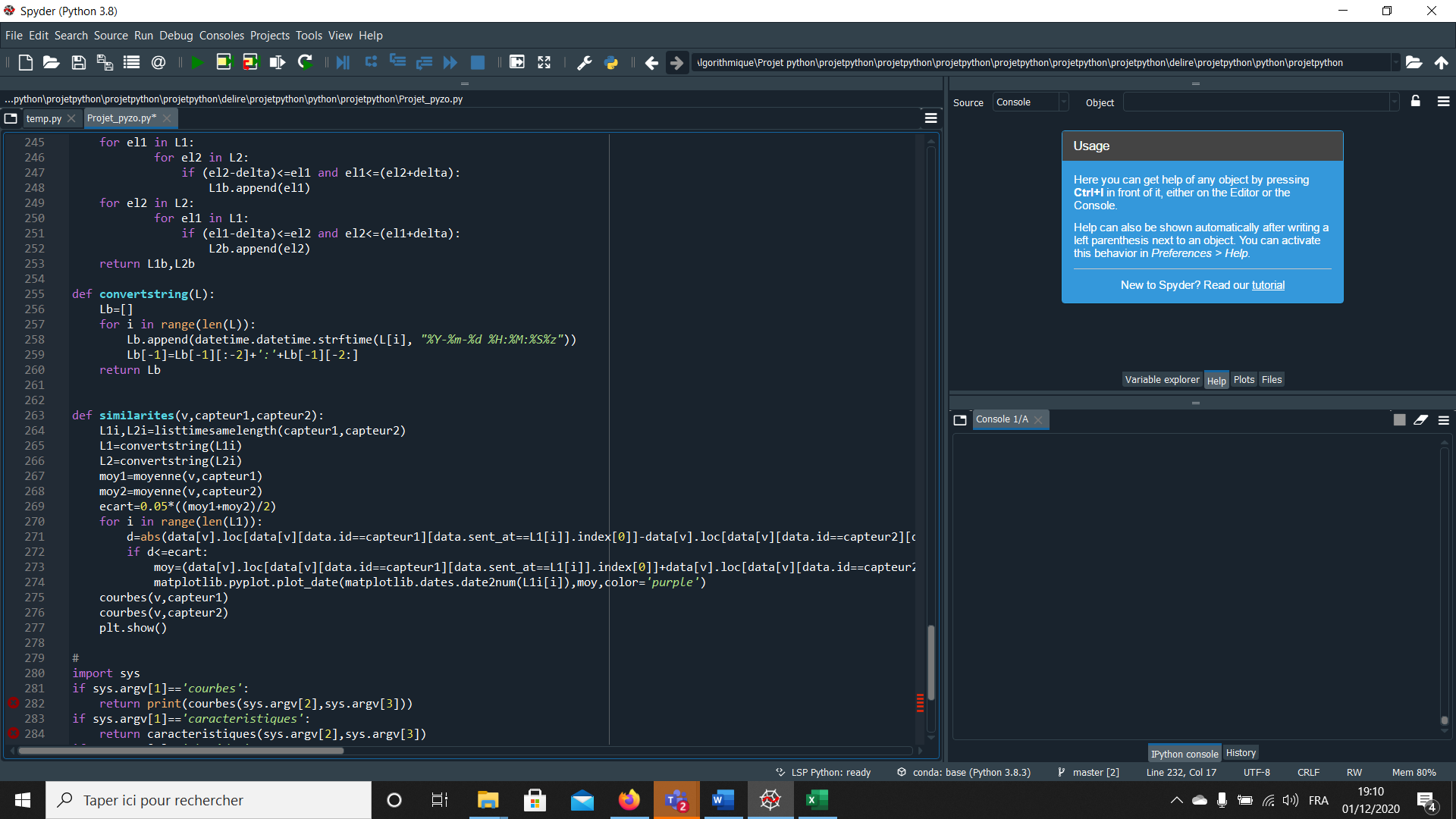
* Il peut y avoir des valeurs de relevés inexistantes ou des valeurs de relevés dans une intervalle très proche, bien inférieure aux 15min régulières : on introduit donc une intervalle de plus ou moins 8 minutes pour trouver les temps d’acquisition équivalents entre les capteurs.



*Complexité : θ(taille des données capteur 1+ taille des données capteur 2)*

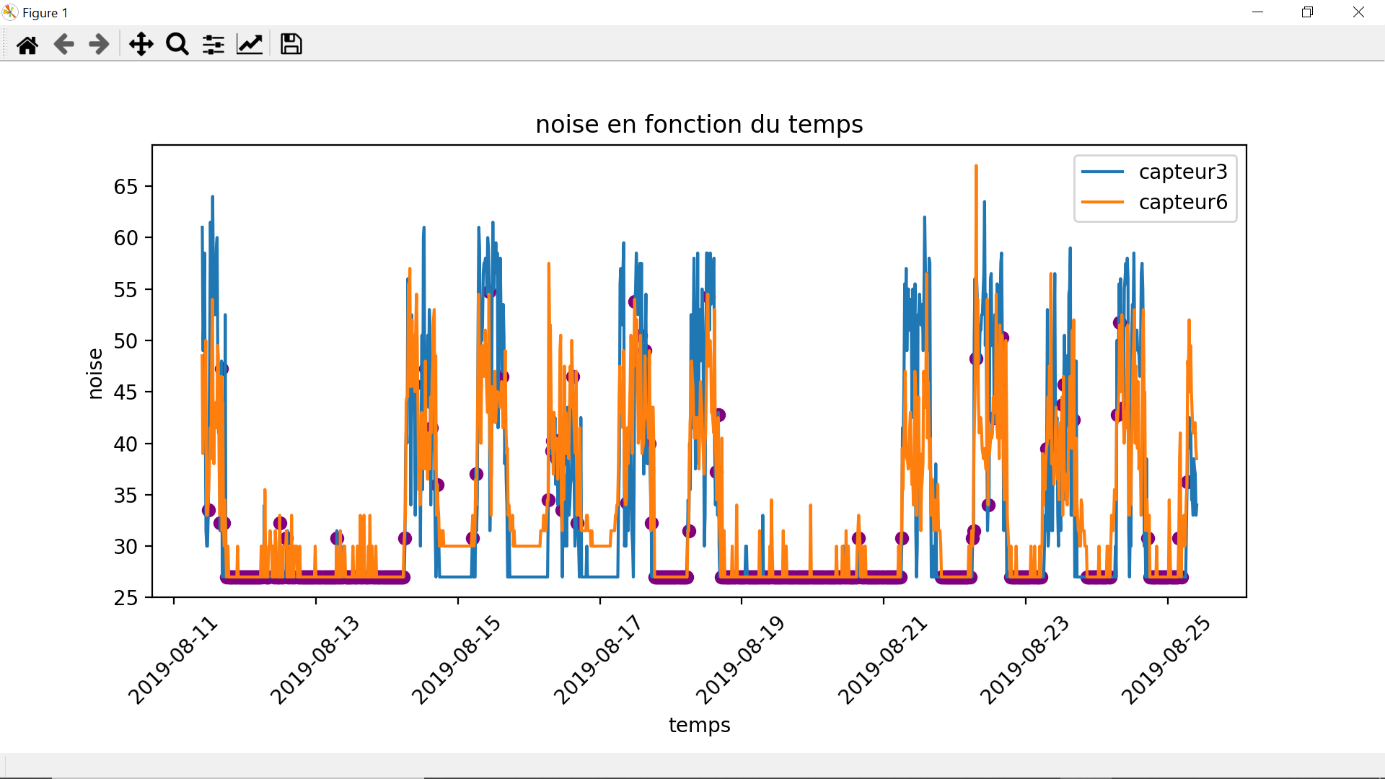
*A cause du parcours des deux listes à comparer*

Une fois ces soucis réglés, on peut grâce à une boucle parcourir les données d’un capteur point par point et comparer avec un autre capteur s’ils sont similaires.

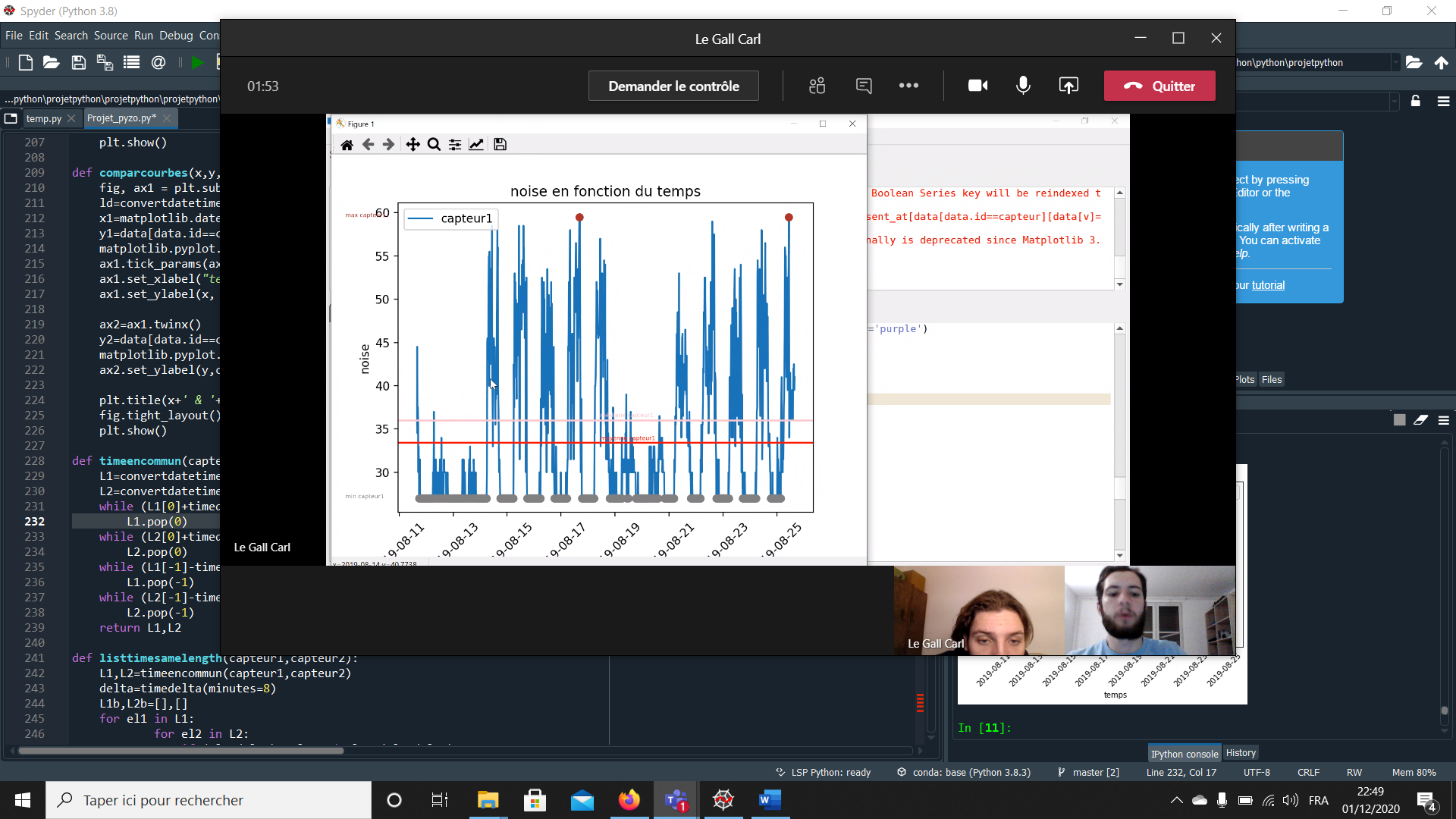


*Complexité : θ(2\*taille des données capteur 1+ taille des données capteur 2)*

*A cause du parcours de la liste à vérifier et des deux boucles de convertString, convertString étant une sous-fonction de complexité de la taille de l’échantillon et permettant de convertir le format des horaires en caractères.*

On représentera ensuite par un point de couleur les points similaires, ce qui permettra de discerner à quel endroit de la courbe les données sont similaires ou non. 

-Bonus horaires d’ouverture :



Nous observons que le bruit modélise bien la présence du lundi au vendredi, seulement il est compliqué de définir une valeur seuil. En effet, notre idée première était de définir un palier en décibels au-dessus duquel l’on considère les bureaux occupés. Or deux soucis se présentent.

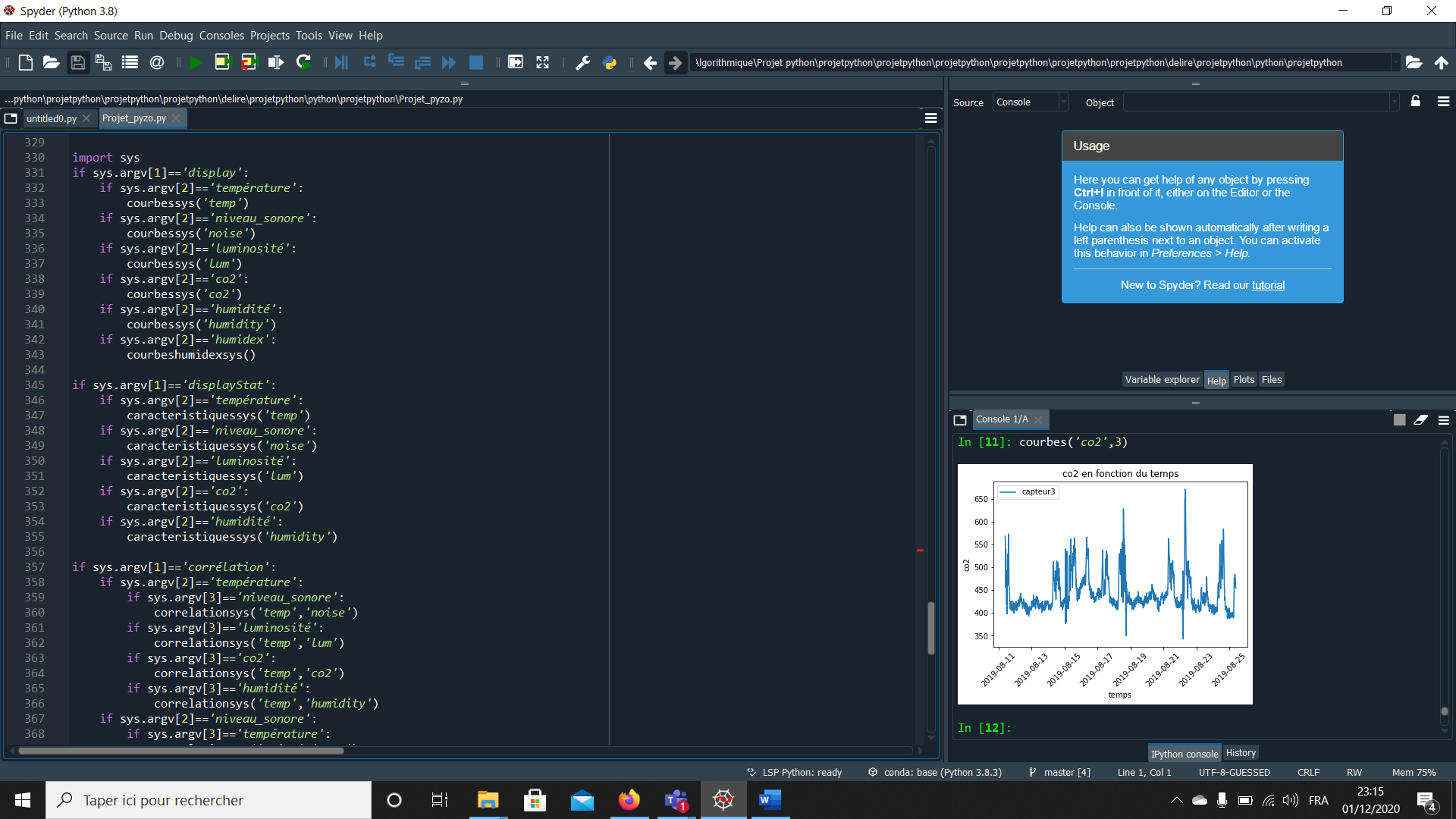
Le premier est qu’on doit donc choisir une valeur au-dessus de la valeur max recensée le week-end et cela implique un seuil relativement haut. Donc certaines heures des matin travaillés (qui sont peu bruyantes ou en tout cas moins que le max du week-end) se retrouvent être écarté de notre intervalle.

Le deuxième est qu’il y a des pics anormaux ou irréguliers qui en plein milieu des journées travaillées viennent atteindre un niveau très bas et donc fausser notre analyse via l’intervalle.

Par manque de temps, nous n’avons pas pu explorer d’autres hypothèses.

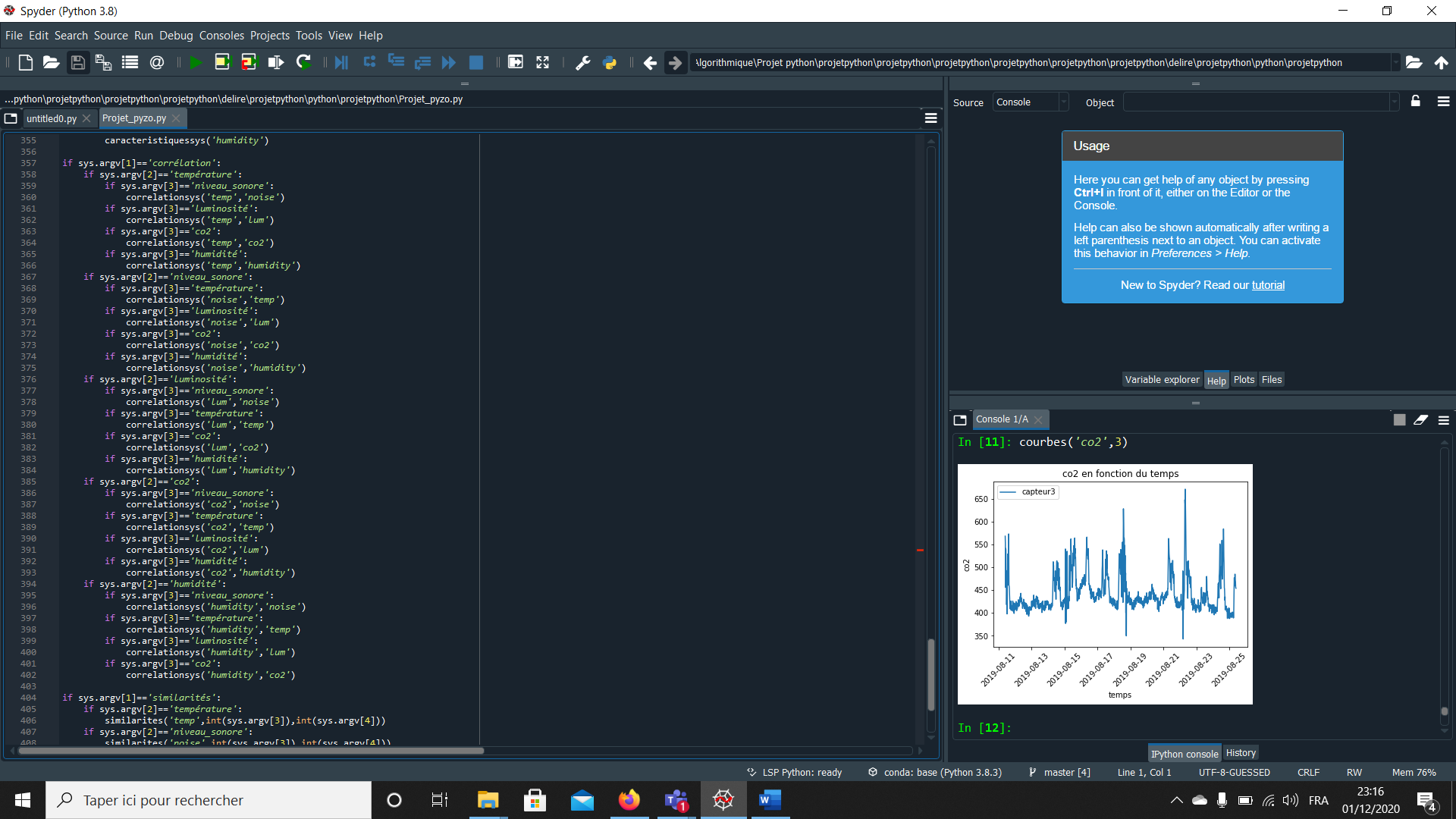
1. **Mise en forme de l’écriture sur console**

On va utiliser la banque sys et écrire une fonction globale reprenant nos fonctions déjà crées grâce à une imbrication de conditions if qui relie la demande écrit dans la commande à la fonction qui répond à son besoin. ( Les conditions bonus start-date et end-date sont en cours de création)

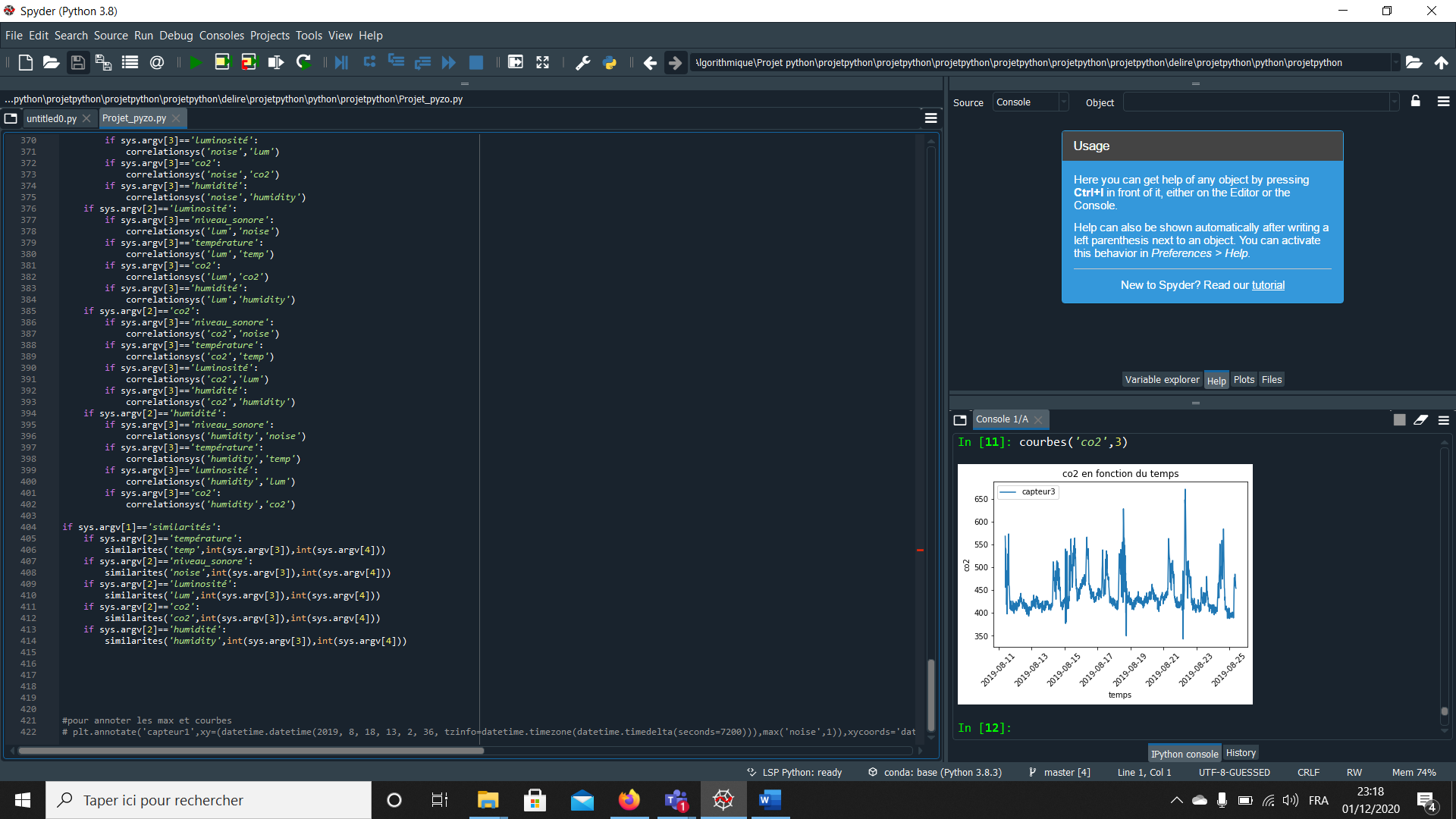


Pour « display », il faut écrire dans la commande les variables choisies selon les orthographes suivants « température, niveau\_sonore, luminosité, co2, humidité, humidex »

Pour « displayStat », il faut écrire dans la commande les variables choisies selon les orthographes suivants « température, niveau\_sonore, luminosité, co2, humidité »



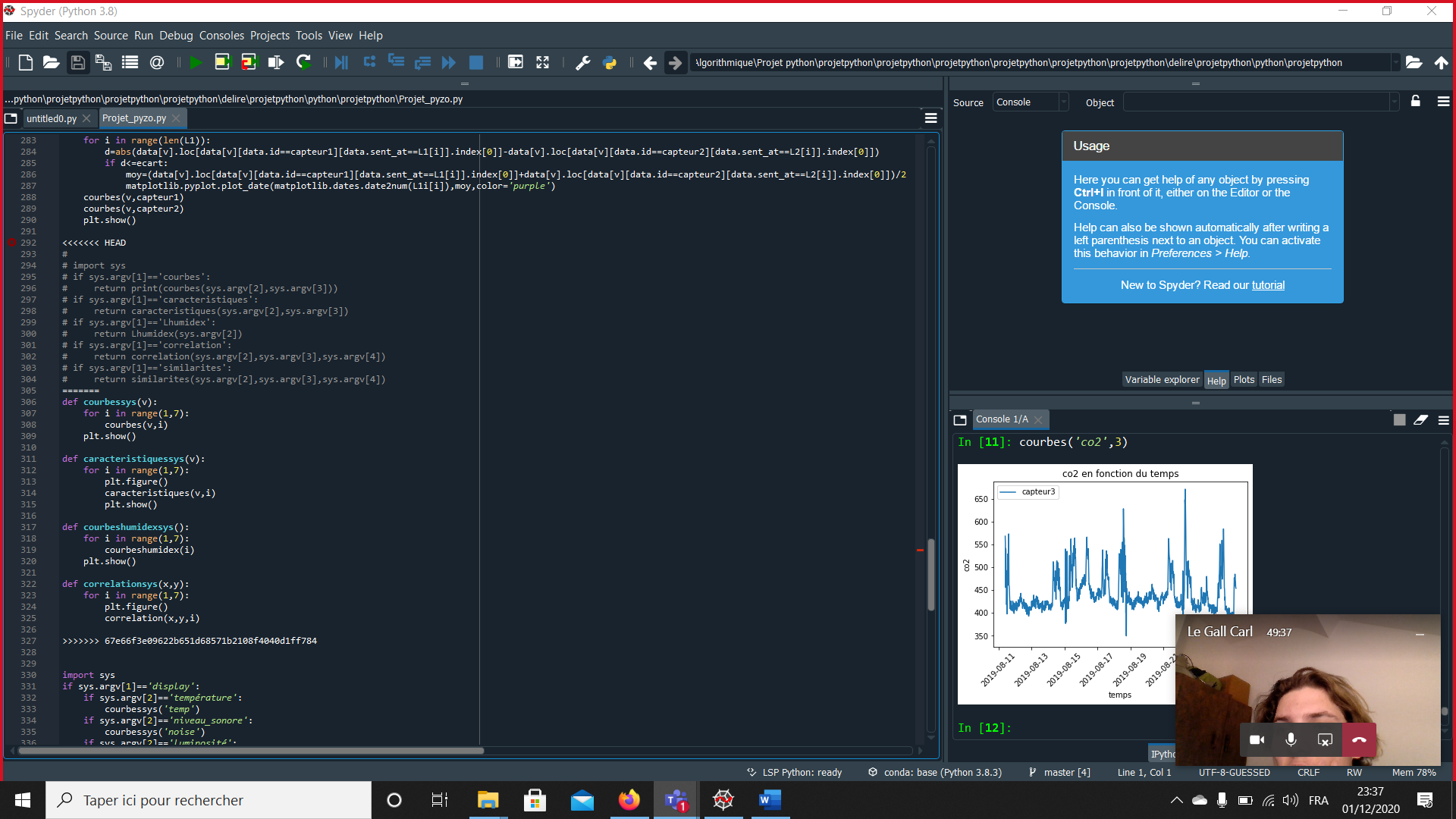
Pour « corrélation », il faut écrire dans la commande les variables choisies selon les orthographes suivants « température, niveau\_sonore, luminosité, co2, humidité » deux fois.



Pour « similarités », il faut écrire dans la commande les variables choisies selon les orthographes suivants « température, niveau\_sonore, luminosité, co2, humidité »

PUIS les numéros des 2 capteurs à comparer.

**Sys est initialisé grâce à :**



**Lien Github: https://github.com/nathangigou/projetpython.git**