

Data logger 2

Un modèle de documentation **minimal** pour tous les types de projets. **Toutes** les catégories ci-dessous doivent être renseignées, même de façon succincte.

IMPORTANT : Merci de sélectionner le / les tags adéquats dans le menu de droite, et de ne pas créer de nouveau tag.

Les **fichiers sources** doivent idéalement être joints à cette page grâce à l'icône trombone du menu de droite.

Des hésitations sur comment bien documenter et utiliser l'interface ? Consultez le tutoriel

"Comment documenter"

Informations

- Caroline Sreng
- Adresse mail : caroline.vann_sreng@sorbonne-universite.fr
- Service civique au Fablab 2024/2025
- 01/10/2024- 2025

Contexte

Il s'agit d'un projet transversal à l'interface des espaces prototypage et biologie/chimie, pour lequel les langages informatiques C++ et Python, l'électronique et la chimie seront mobilisés.

Objectifs

Un objectif serait de pouvoir confectionner au Fablab un boîtier électronique permettant d'afficher les informations sur la prise de mesures de température, de pH, et d'autres valeurs physiques relevées lors de travaux pratiques effectués dans le cadre d'enseignement secondaire/supérieure. Les valeurs des mesures seraient affichées sur un ordinateur par une communication par le port USB pour une première version et par une application via un réseau internet, pour une version améliorée. Dans un second temps, à partir d'un ordinateur, des seuils et des alertes seraient mises en place lorsque des valeurs souhaitées seraient atteintes. De la même manière que précédemment, une première version vise à instaurer une communication par le port USB.



Ajouter au moins une image de votre projet

Matériel

Boîte contenant notre boîtier

- à voir ultérieurement (partie non traitée encore)

Electronique

- un m5Core2
- des fils électriques
- un câble USB-C vers port m5Core2
- un breadboard
- un ordinateur (logiciel Arduino IDE, Python, VSCode)
- une sonde de température DS18B20, (une sonde de pH de type SEN0161 par DFROBOT)
- des résistances

Machines utilisées

à voir ultérieurement (partie non traitée encore)

Construction

(Fichiers, photos, code, explications, paramètres d'usinage, photos, captures d'écran...)

Étape 1

Une première étape est de se familiariser avec l'électronique et les langages informatiques C++ et Python.

Étape 2

Étape 3

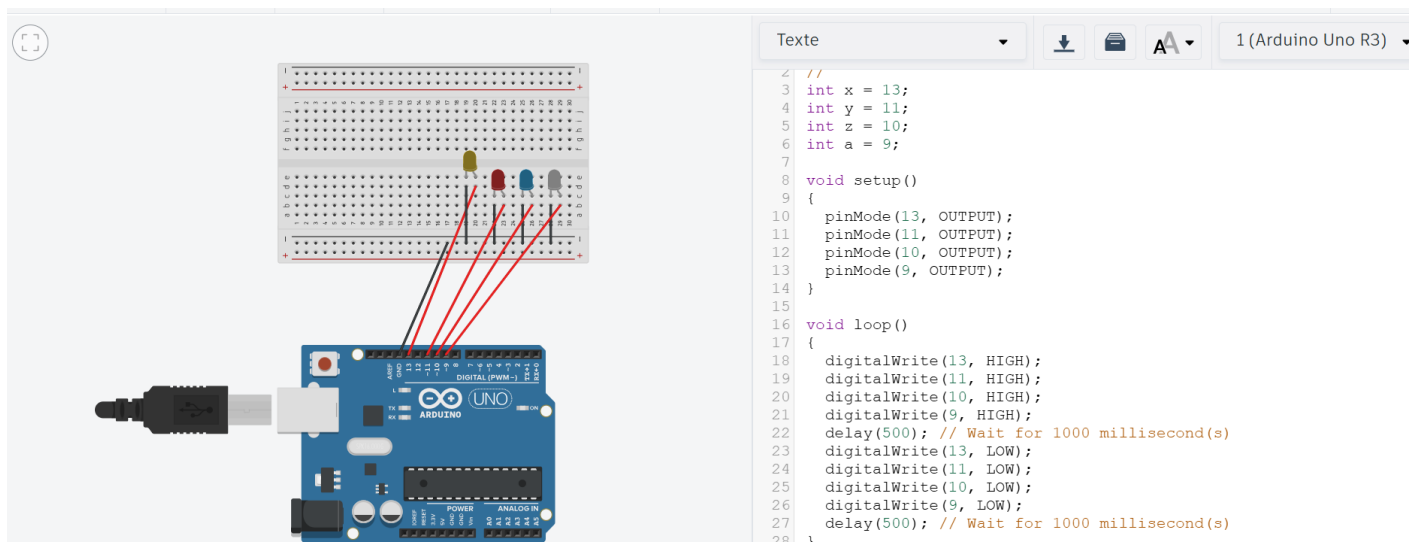
Journal de bord

Avancée du projet à chaque étape, difficultés rencontrées, modifications et adaptations (facultatif pour les petits projets)

01/10/2024 --mi-novembre 2024

J'ai commencé à me familiariser avec l'électronique/les microcontrôleurs Arduino en lisant :

[arduino-premiers-pas-en-informatique-embarquee.pdf](#) . J'ai à peu près tout lu dans les grandes lignes exceptée la partie sur les écrans LCD. J'ai aussi utilisé Tinkercad pour tester certains branchements de manière virtuelle et aussi IRL une breadboard, un microcontrôleur Arduino Uno et des LED.

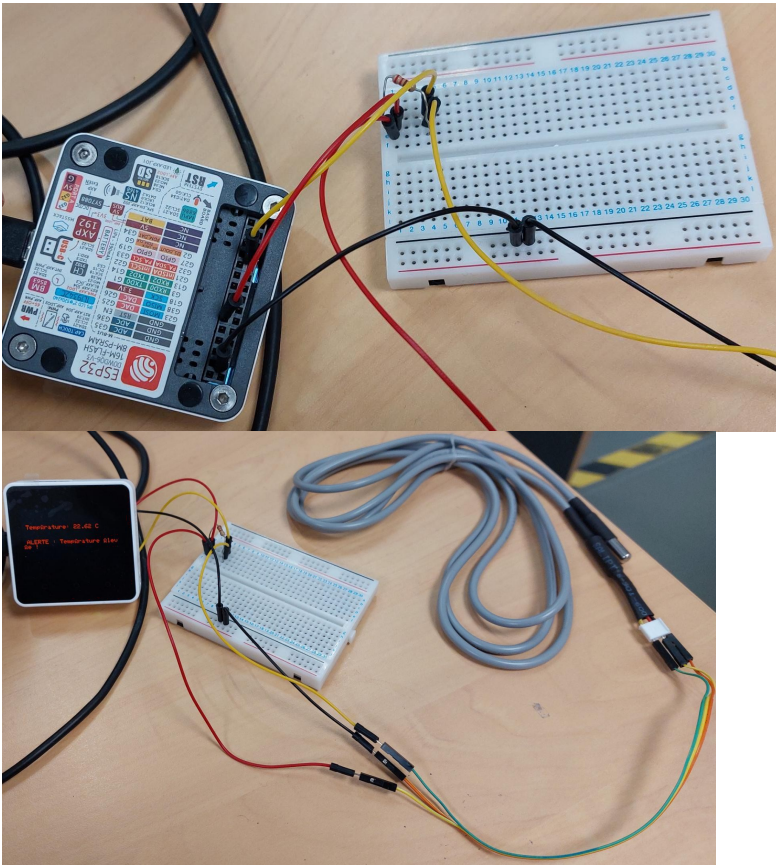




mi-novembre 2024 - 6/01/2025

Etant donné qu'il serait pertinent de créer une communication par l'utilisation d'une application entre un microcontrôleur et un ordinateur, je me suis tournée vers un M5Core2 qui possède de nombreuses caractéristiques comme un écran, la possibilité de configurer entre autre une alarme, les connexions bluetooth et WI-FI. De plus, pour la création d'une application, il serait préférable d'utiliser un autre langage informatique que le C++, je me suis tournée vers le langage Python. J'ai donc commencé à me familiariser avec le langage en lisant le livre [Apprenez à programmer en Python par Vincent Le Goff aux Editions Eyrolles \(4ème édition\)](#). En parallèle, j'ai regardé des vidéos sur youtube en faisant des exercices de cas. J'ai aussi fait des branchements entre un M5core2 et un capteur de température, en récupérant les données de température sur le terminal de série de l'Arduino IDE et mettant des seuils depuis ce terminal de série.

Voici une photo du branchement (alimentation 3.3V, GPIO 27 et ground) :



Le code **C++** saisi sur Arduino IDE:

```
#include <M5Core2.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Pin du capteur
#define ONE_WIRE_BUS 27

// Créer une instance OneWire
OneWire oneWire(ONE_WIRE_BUS);

// Passer l'instance OneWire à DallasTemperature
DallasTemperature sensors(&oneWire);

// Déclarer la constante seuil par défaut à 20°C
float seuil = 20;
```

```
void setup() {  
  // Initialiser M5Core2  
  M5.begin();  
  M5.Axp.SetSpkEnable(true); //activer le haut-parleur du m5core2  
  // Initialise la communication série  
  Serial.begin(115200);  
  Serial.println("Programme démarré. Tapez un nouveau seuil ou 'exit' pour quitter.");  
  Serial.print("Seuil actuel : ");  
  Serial.println(seuil);  
  
  // Initialiser le capteur  
  sensors.begin();  
  
  // Initialiser l'affichage  
  M5.Lcd.setTextSize(2);  
  M5.Lcd.setCursor(0, 0);  
  M5.Lcd.print("Température : ");  
}  
  
void loop() {  
  // Demander les températures  
  sensors.requestTemperatures();  
  
  // Lire la température en degrés Celsius  
  float temperatureC = sensors.getTempCByIndex(0);  
  
  // Afficher la température sur l'écran  
  M5.Lcd.setCursor(0, 30);  
  M5.Lcd.fillRect(0, 30, 240, 40, BLACK); // Effacer la ligne précédente  
  M5.Lcd.print(temperatureC);  
  M5.Lcd.print(" °C");  
  
  // Afficher la température sur le terminal série  
  Serial.print("Température actuelle : ");
```

```
Serial.print(temperatureC);  
Serial.println(" °C");
```

```
// Vérifie si des données sont disponibles sur le port série
```

```
if (Serial.available() > 0) {
```

```
    String input = Serial.readStringUntil('\n'); // Lit l'entrée jusqu'à un retour à la ligne
```

```
    // Permet à l'utilisateur de quitter le programme
```

```
    if (input.equalsIgnoreCase("exit")) {
```

```
        Serial.println("Programme terminé.");
```

```
        while (true); // Arrête le programme
```

```
    } else {
```

```
        // Convertir l'entrée en float et mettre à jour le seuil
```

```
        float newSeuil = input.toFloat();
```

```
        if (newSeuil != 0 || input.equals("0")) { // Vérifie si la conversion est valide
```

```
            seuil = newSeuil;
```

```
            Serial.print("Nouveau seuil défini : ");
```

```
            Serial.println(seuil);
```

```
        } else {
```

```
            Serial.println("Entrée invalide. Veuillez entrer un nombre valide.");
```

```
        }
```

```
    }
```

```
}
```

```
// Vérifier si la température dépasse le seuil
```

```
if (temperatureC > seuil) {
```

```
    M5.Lcd.setCursor(0, 80);
```

```
    M5.Lcd.setTextColor(RED);
```

```
    M5.Lcd.fillRect(0, 80, 240, 40, BLACK); // Effacer la ligne précédente
```

```
    M5.Lcd.println("ALERTE : Température élevée !");
```

```
    Serial.println("ALERTE : Température élevée !");
```

```
    M5.Axp.SetVibration(true); // Open the vibration.
```

```
    delay(1000);
```

```
    M5.Axp.SetVibration(false); // Open the vibration.
```

```
    delay(1000);
```

```
} else {
```

```
    M5.Lcd.setCursor(0, 80);
```

```
M5.Lcd.setTextColor(WHITE);
M5.Lcd.fillRect(0, 80, 240, 40, BLACK); // Effacer la ligne précédente
M5.Lcd.println("Température OK.");
Serial.println("Température OK.");
}

// Attendre 2s avant la prochaine lecture
delay(2000);
}
```

J'ai ensuite compilé et téléchargé le script vers le M5Core2.

6/01/2025-27/02/2025

Après m'être familiarisé avec le langage Python, j'ai commencé à me pencher sur la communication des données via le port série. Il existe une bibliothèque du langage de programmation Python appelée Pyserial qui permet ceci. J'ai réussi à faire afficher dans le terminal de l'éditeur de code VScode les valeurs de températures mesurées toutes les 2 secondes et également à saisir des valeurs de seuil.

Voici le code à saisir en langage **Python** dans un éditeur de code comme VScode:

```
import serial
import threading

# Configuration du port série
PORT = 'COM9' # Remplacez par le port série de votre M5Core2, port 9 pour moi
BAUD_RATE = 115200 # Taux de transmission utilisé par le M5Core2.

def lire_temperature(ser):
    """
    Fonction qui lit les données reçues sur le port série en continu
    et les affiche dans le terminal.
    """
    while True:
        try:
            # Vérifie s'il y a des données disponibles sur le port série.
            if ser.in_waiting > 0:
```



```

        # Lit une ligne complète, la décode et supprime les espaces/sauts de ligne inutiles.
        data = ser.readline().decode('utf-8').strip()

        # Affiche les données reçues (par exemple : température mesurée).
        print(f"[M5Core2] {data}")

except Exception as e:
    # En cas de problème lors de la lecture, affiche un message d'erreur.
    print(f"Erreur lors de la lecture du port série : {e}")
    break

```

```
def envoyer_seuil(ser):
```

```
    """
```

```
    Fonction qui permet d'envoyer un seuil de température à partir du terminal.
```

```
    L'utilisateur entre un seuil qui est envoyé au M5Core2.
```

```
    """
```

```
while True:
```

```
    try:
```

```
        # Demande à l'utilisateur d'entrer un seuil ou de taper "exit" pour quitter.
```

```
        user_input = input("Entrez un seuil de température (ou 'exit' pour quitter) : ")
```

```
        if user_input.lower() == 'exit':
```

```
            # Si l'utilisateur tape 'exit', on arrête la boucle.
```

```
            print("Fin de l'envoi de commandes.")
```

```
            break
```

```
        # Envoie la valeur entrée au M5Core2 via le port série.
```

```
        ser.write((user_input + '\n').encode('utf-8'))
```

```
except Exception as e:
```

```
    # En cas de problème lors de l'envoi, affiche un message d'erreur.
```

```
    print(f"Erreur lors de l'envoi du seuil : {e}")
```

```
    break
```

```
def main():
```

```
    """
```

```
    Fonction principale qui initialise la connexion série, lance les threads
```

```
    pour la lecture et gère l'envoi des seuils.
```

```
    """
```

```
try:
```

```
    # Ouvre la connexion série avec le port et le baud rate configurés.
```

```

ser = serial.Serial(PORT, BAUD_RATE, timeout=1)
print("Connexion au M5Core2 établie.\n")

# Démarre un thread séparé pour lire les températures en permanence.
thread_lecture = threading.Thread(target=lire_temperature, args=(ser,))
thread_lecture.daemon = True # Assure que le thread s'arrête lorsque le programme principal termine.
thread_lecture.start()

# Lance la fonction d'envoi des seuils dans le thread principal.
envoyer_seuil(ser)

except serial.SerialException as e:
    # Si le port série ne peut pas être ouvert, affiche un message d'erreur.
    print(f"Erreur de connexion série : {e}")
except Exception as e:
    # Capture d'autres erreurs potentielles et affichage.
    print(f"Erreur : {e}")
finally:
    # Ferme la connexion série proprement à la fin du programme.
    if 'ser' in locals() and ser.is_open:
        ser.close()
        print("Connexion série fermée.")

# Point d'entrée du programme.
if __name__ == "__main__":
    main()

```

Pour récupérer les données, il faut téléverser le script ci-dessus dans l'éditeur de code. **Veillez à faire attention à ne pas laisser le terminal Arduino IDE ouvert sinon on ne pourra avoir accès au port sur VSCode.** Si une erreur de type "Erreur de connexion série : could not open port 'COM9': Permission Error(13, 'Accès refusé.', None, 5)" s'affiche, fermez le terminal sur Arduino IDE, débranchez le port USB et rebranchez-le.

28/01/2025-30/01/2025

A présent, afin d'utiliser les données pour des analyses, je vais essayer de récupérer ces mesures de températures (mesurées à une fréquence de 2 secondes ici) en fonction du temps. Pour cela, il faudrait enregistrer dans un fichier CSV ces données et faire une copie des log du M5Core2 sur une carte micro SD où les données seront récupérées. On pourra récupérer ces données sur un ordinateur via cette carte micro SD.

J'ai à nouveau modifié le code en **C++** afin d'enregistrer les données sur la carte micro SD en affichant la date et l'heure.

```
#include <M5Core2.h>
#include <OneWire.h>      // Bibliothèque pour le capteur de température
#include <DallasTemperature.h> // Bibliothèque pour la gestion du capteur Dallas
#include <SD.h>           // Bibliothèque pour la lecture/écriture sur carte SD
#include <SPI.h>          // Bibliothèque pour la communication SPI

// Définition des broches
#define ONE_WIRE_BUS 27 // GPIO 27 pour le capteur
#define SD_CS_PIN 4     // GPIO 4 pour la carte SD

// Initialisation du capteur de température
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

// Seuil de température par défaut
float seuil = 30.0;

void setup() {
    M5.begin();
    M5.Axp.SetSpkEnable(true); // Activer le haut-parleur du M5Core2

    // Initialisation de la communication série
    Serial.begin(115200);
    Serial.println("Programme démarré. Tapez un nouveau seuil ou 'exit' pour quitter.");
    Serial.print("Seuil actuel : ");
    Serial.println(seuil);

    // Vérifier la présence de la carte SD
    if (!SD.begin(SD_CS_PIN)) {
        Serial.println("Échec de montage de la carte SD !");
        M5.Lcd.println("Carte SD absente !");
        return;
    }
}
```

```

    } else {
        Serial.println("Carte SD détectée avec succès.");
    }

    // Initialisation du capteur de température
    sensors.begin();

    // Affichage initial sur l'écran
    M5.Lcd.setTextSize(2);
    M5.Lcd.setCursor(0, 0);
    M5.Lcd.print("Température : ");
}

void loop() {
    sensors.requestTemperatures(); // Demander la température actuelle
    float temperatureC = sensors.getTempCByIndex(0); // Lire la température

    // Récupération de la date et heure actuelles
    RTC_TimeTypeDef TimeStruct;
    RTC_DateTypeDef DateStruct;
    M5.Rtc.GetTime(&TimeStruct);
    M5.Rtc.GetDate(&DateStruct);
    String dateTime = String(DateStruct.Date) + "/" + String(DateStruct.Month) + "/" + String(2000 +
DateStruct.Year) + " " +
        String(TimeStruct.Hours) + ":" + String(TimeStruct.Minutes) + ":" + String(TimeStruct.Seconds);

    // Ouverture du fichier sur la carte SD et enregistrement des données
    File dataFile = SD.open("/temp_data.txt", FILE_APPEND);
    if (dataFile) {
        dataFile.print(dateTime);
        dataFile.print(" - Température : ");
        dataFile.print(temperatureC);
        dataFile.println(" °C");
        dataFile.close();
        Serial.println("Donnée enregistrée sur la carte SD !");
    } else {
        Serial.println("Erreur lors de l'ouverture du fichier pour l'enregistrement !");
    }

    // Mise à jour de l'affichage sur l'écran

```

```

M5.Lcd.setCursor(0, 30);
M5.Lcd.fillRect(0, 30, 240, 40, BLACK); // Effacer la ligne précédente
M5.Lcd.print(temperatureC);
M5.Lcd.print(" °C");

// Affichage sur le terminal série
Serial.print("Température actuelle : ");
Serial.print(temperatureC);
Serial.println(" °C");

// Vérification d'une entrée utilisateur sur le port série
if (Serial.available() > 0) {
    String input = Serial.readStringUntil('\n'); // Lire l'entrée jusqu'au retour à la ligne

    if (input.equalsIgnoreCase("exit")) {
        Serial.println("Programme terminé.");
        while (true);
    } else {
        float newSeuil = input.toFloat();
        if (newSeuil != 0 || input.equals("0")) {
            seuil = newSeuil;
            Serial.print("Nouveau seuil défini : ");
            Serial.println(seuil);
        } else {
            Serial.println("Entrée invalide. Veuillez entrer un nombre valide.");
        }
    }
}

// Vérification du seuil de température et alerte
if (temperatureC > seuil) {
    M5.Lcd.setCursor(0, 80);
    M5.Lcd.setTextColor(RED);
    M5.Lcd.fillRect(0, 80, 240, 40, BLACK);
    M5.Lcd.println("ALERTE : Température élevée !");
    Serial.println("ALERTE : Température élevée !");
    M5.Axp.SetVibration(true);
    delay(1000);
    M5.Axp.SetVibration(false);
} else {

```

```

M5.Lcd.setCursor(0, 80);
M5.Lcd.setTextColor(WHITE);
M5.Lcd.fillRect(0, 80, 240, 40, BLACK);
M5.Lcd.println("Température OK.");
Serial.println("Température OK.");
}

delay(2000); // Pause de 2 secondes avant la prochaine mesure
}

```

Attention, il faudrait formater la date et l'heure. Lorsque l'on ouvre le fichier txt, on observe qu'elles sont incorrectes. De plus, lorsque l'on regarde les temps auxquels les mesures sont prises, il y a des intervalles de 3s et non de 2s comme d'après notre code. Probablement que les mesures enregistrées dans le fichier TXT sur la carte SD ne correspondent pas à celles enregistrées dans le fichier CSV sur l'ordinateur.

J'ai modifié le code en **Python** afin d'enregistrer les données sur un fichier CSV.

```

#enregistrer les données sur un fichier csv
import serial
import threading
from datetime import datetime
import os

# Configuration du port série
PORT = 'COM9' # Remplacez par votre port série
BAUD_RATE = 115200 # Taux de transmission utilisé par le M5Core2
os.chdir('chemin où vous voulez enregistrer votre fichier')
DATA_CSV = "temperatures v2 CS.csv" # Nom du fichier pour enregistrer les données

def enregistrer_donnees(fichier, temperature):
    """
    Enregistre une température avec un horodatage dans un fichier avec extension `.csv`.
    """
    try:
        with open(fichier, mode='a') as file:
            # Récupère l'horodatage actuel
            horodatage = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            # Écrit les données sous forme de texte
            file.write(f"{horodatage} - Température : {temperature} °C\n")
    
```

except Exception as e:

```
print(f"Erreur lors de l'écriture dans le fichier : {e}")
```

def lire_temperature(ser):

```
    """
```

Lit les données reçues sur le port série et les affiche tout en les enregistrant.

```
    """
```

while True:

```
    try:
```

```
        # Vérifie s'il y a des données disponibles sur le port série
```

```
        if ser.in_waiting > 0:
```

```
            # Lit une ligne complète et supprime les espaces inutiles
```

```
            data = ser.readline().decode('utf-8').strip()
```

```
            print(f"[M5Core2] {data}") # Affiche les données reçues
```

```
            # Enregistre les données dans le fichier .usd
```

```
            enregistrer_donnees(DATA_CSV, data)
```

```
    except Exception as e:
```

```
        print(f"Erreur lors de la lecture du port série : {e}")
```

```
        break
```

def envoyer_seuil(ser):

```
    """
```

Permet à l'utilisateur d'envoyer un seuil de température via le terminal.

```
    """
```

while True:

```
    try:
```

```
        user_input = input("Entrez un seuil de température (ou 'exit' pour quitter) : ")
```

```
        if user_input.lower() == 'exit':
```

```
            print("Fin de l'envoi de commandes.")
```

```
            break
```

```
        ser.write((user_input + '\n').encode('utf-8'))
```

```
    except Exception as e:
```

```
        print(f"Erreur lors de l'envoi du seuil : {e}")
```

```
        break
```

def main():

```
    """
```

Programme principal qui gère la connexion série, la lecture et l'enregistrement des données.

```
    """
```

```
    try:
```

```

# Ouvre la connexion série
ser = serial.Serial(PORT, BAUD_RATE, timeout=1)
print("Connexion au M5Core2 établie.")
print(f"Les données seront enregistrées dans le fichier : {DATA_CSV}\n")

# Lance un thread pour lire les températures
thread_lecture = threading.Thread(target=lire_temperature, args=(ser,))
thread_lecture.daemon = True
thread_lecture.start()

# Permet d'envoyer des seuils de température
envoyer_seuil(ser)

except serial.SerialException as e:
    print(f"Erreur de connexion série : {e}")
except Exception as e:
    print(f"Erreur : {e}")
finally:
    # Ferme le port série proprement
    if 'ser' in locals() and ser.is_open:
        ser.close()
        print("Connexion série fermée.")

# Point d'entrée du script
if __name__ == "__main__":
    main()

```

31/01/2025-

Une autre étape est de pouvoir déclarer les ports sur lesquels seront branchés les différents capteurs et les différentes librairies à exploiter en fonction des capteurs utilisés.

Une autre étape encore serait de faire un graphique en utilisant les données à partir d'un script.

Revision #18

Created 27 January 2025 13:26:20 by Caroline

Updated 30 January 2025 16:57:59 by Caroline