Jardin LIBRE, documentation technique

Partie I

Logo Arles-Gnu-Linux.png
Logo_Verrerie.png

Découverte du projet et du Raspberry Pi

cc-by-sa-icon-white-svg.png

Qu'est ce qu'un Raspberry PI ?

Le **Raspberry Pi** est un ordinateur monocarte de petite taille et à faible coût, développé par la Fondation Raspberry Pi. Il est conçu pour promouvoir l'enseignement de l'informatique raspberry-1.jpg et des compétences en programmation. Le Raspberry Pi est équipé d'un processeur ARM, de mémoire RAM, et de divers ports pour la connectivité, tels que HDMI, USB, et GPIO (General Purpose Input/Output). Il fonctionne avec plusieurs systèmes d'exploitation, le plus courant étant **Raspberry Pi OS**, une distribution GNU/Linux. Grâce à sa polyvalence et son coût abordable, le Raspberry Pi est largement utilisé dans des projets éducatifs, des applications IoT (Internet des objets), des projets de bricolage électronique, et même comme centre multimédia ou serveur léger.

raspberry-pi-5.png

Le **Raspberry Pi 5** est un nano-ordinateur puissant, économe en énergie et totalement silencieux. Plus rapide que ses prédécesseurs, il est idéal pour faire tourner en continu notre **station de jardin libre**. Grâce à ses ports **GPIO**, il peut piloter capteurs et relais (température, humidité, pression, arrosage…), tout en hébergeant une page web locale et un wiki, même hors connexion Internet. Compact, robuste et compatible avec l’écosystème libre, c’est la pierre angulaire de notre démarche numérique écoresponsable.

Le projet

Au cours de ces ateliers, nous explorerons les bases de l'électronique et de la programmation pour créer une station de **jardin LIBRE fonctionnelle**. Que vous soyez débutant ou que vous ayez déjà des connaissances, ces ateliers sont conçus pour vous offrir une expérience enrichissante et pratique.

Ce que vous apprendrez :

- Électronique de base : Découvrez les composants électroniques et apprenez à les utiliser pour construire des projets interactifs.
- Programmation : Initiez-vous à la programmation avec Python pour contrôler et lire les données de capteurs comme la **DHT22** et la **BMP280**.
- Développement web : Créez une interface utilisateur avec **HTML5** et **CSS** pour afficher les données de votre station météo en temps réel.

Installation de Raspberry Pi OS

raspberrypios-1.png

Le Raspberry Pi OS est un système d'exploitation libre basé sur Debian, spécialement conçu pour les ordinateurs Raspberry Pi. Il offre une interface graphique conviviale et une large gamme d'outils pour la programmation et l'apprentissage, le rendant idéal pour les projets éducatifs et les applications de bricolage.

1. Téléchargement de l'outil Raspberry Pi imager : **https://www.raspberrypi.com/software/**
2. Choisissez Raspberry Pi OS lite (sans interface graphique).
3. Dans la section général, vous pouvez configurer :
    - Le nom de la machine (hostname).
    - Votre nom d'utilisateur et mot de passe.
    - La configuration de votre wifi.
    - Les locales (clavier et langue).
4. Dans la section services, vous pouvez activer le SSH.

raspberrypiimager-1.png raspberrypiwifi-1.png

raspberrypissh-1.png

SSH (Secure Shell) est un protocole sécurisé permettant de se connecter à distance à un ordinateur (comme un Raspberry Pi) via le terminal. Il chiffre les échanges et permet de contrôler, configurer ou transférer des fichiers entre machines, sans avoir besoin d’un écran ou clavier branché localement.

Branchement de la sonde DHT22

gpio.png

- Connecter la broche **VCC** de la sonde à une broche **3.3V** du Raspberry Pi (**broche 1**)
- Connecter la broche **GND** de la sonde à une broche **GND** du Raspberry Pi (**broche 6**)
- Connecter la broche **DATA** de la sonde à une broche **GPIO** du Raspberry Pi (**broche 7 - GPIO 4**)

gpio-board.png

connexion-dht22.png

Partie II

Récupération des données avec python en mode interactif

Qu’est-ce que Python ?

python.jpeg

- Python est un langage de programmation populaire, connu pour sa simplicité et sa lisibilité.
- Il est largement utilisé dans divers domaines, comme le développement web, l'analyse de données, et l'intelligence artificielle.

Pourquoi Python ?

- Facile à apprendre et à utiliser, surtout pour les débutants.
- Grande communauté de soutien et de nombreuses ressources disponibles en ligne.
- Utilisé dans de nombreux projets éducatifs et scientifiques.

Découverte du terminal

Un terminal est une interface où l'on peut taper des commandes pour interagir avec l'ordinateur.

Nos premières commandes Python

Dans le terminal, tapez python3 et appuyez sur Entrée. Cela lancera l'interpréteur Python en mode interactif.

Commande print :

La commande print affiche du texte à l'écran.

```
print (“Python, c'est cool !”)
```

Constantes et variables

- Une **constante** est un emplacement de stockage nommé pour des données qui ne changent pas au cours de l'exécution d'un programme.
- Une **variable** est un emplacement de stockage nommé pour des données qui peuvent changer au cours de l'exécution d'un programme.

```
prenom = “Olivier”
```

Installation de la bibliothèque adafruit-circuitpython-dht sur le Raspberry Pi

Les bibliothèques permettent d'étendre les fonctionnalités de Python.

Pour éviter les conflits entre bibliothèques, nous allons créer un environnement virtuel isolé pour gérer les dépendances de notre projet.

Pour créer un environnement virtuel, nous devons installer le paquet python3-venv :

```
sudo apt update
sudo apt install python3-venv
```

install-python3-venv-on-ubuntu.png

Pour créer un environnement virtuel (par exemple pour le projet jardin LIBRE), taper la commande suivante :

```
python3 -m venv jardin
```

Tapez la commande suivante pour installer pip, le gestionnaire de paquets Python :

```
sudo apt update
sudo apt install python3-pip
```

Pour pouvoir installer la bibliothèque, commençons pas entrer dans notre environnement virtuel :

```
source jardin/bin/activate
```

Tapez les commandes suivantes pour installer la bibliothèque adafruit-circuitpython-dht :

```
pip3 install adafruit-circuitpython-dht
pip3 install lgpio
pip3 install gpiod
```

lgpio : la bibliothèque GPIO de nouvelle génération

Elle permet un **accès bas niveau rapide et fiable** aux broches GPIO.

Elle est utilisée en coulisse par certaines bibliothèques Adafruit pour gérer les **signaux numériques très précis** du capteur DHT.

Pour quitter un environnement virtuel, taper la commande suivante :

```
deactivate
```

Lecture et affichage des données

Démarrer le mode interactif de Python :

```
python3
```

Importer la bibliothèque adafruit-circuitpython-dht afin d'interagir avec la sonde DHT22 :

```
import adafruit_dht
```

Importer la bibliothèque board :

La bibliothèque board sert à **nommer les broches physiques du microcontrôleur ou du Raspberry Pi de façon claire et universelle**.

```
import board
```

Définition de notre capteur (ici, pour le GPIO4) :

```
dhtDevice = adafruit_dht.DHT22(board.D4)
```

Lire les données :

```
humidity = dhtDevice.humidity
temperature = dhtDevice.temperature
```

humidity et **temperature** sont deux variables, elles servent à stocker les valeurs.

Afficher les données :

```
humidity
temperature
```

dht22-pi5.png

Partie III

Création d'un script python

Pourquoi construire un script ?

Lors du chapitre précédent, nous avons découvert comment lire les données du capteur DHT22 en mode interactif dans le terminal Python. Ce mode est idéal pour expérimenter, tester et comprendre le fonctionnement de la sonde. Mais il a ses limites : chaque mesure doit être lancée manuellement, les calculs doivent être refaits à la main, et les résultats ne sont pas sauvegardés.

Avec un script **Python autonome**, on passe à l'étape supérieure : notre station de jardin devient **automatisée** et **réutilisable**. Le code tourne en boucle, effectue les relevés à intervalle régulier, calcule automatiquement le point de rosée et l'humidex, puis affiche les résultats joliment formatés. C'est une première vraie brique vers une **station pour le jardin libre et autonome**, que l'on peut faire évoluer ensuite (enregistrement des données, affichage web, alertes, etc.). On quitte l'expérimentation manuelle pour poser les bases d'un **service automatisé, éthique, et maîtrisé de bout en bout**.

Bref : on libère notre station.

Création du script

Créer un script et éditer le avec la commande suivante :

```
nano station.py
```

Importation

Commençons par importer la **bibliothèque adafruit-circuitpython-dht**, qui permet de lire les données des capteurs DHT (température et humidité) :

```
import adafruit_dht
```

Importons aussi la **bibliothèque board** qui sert à indiquer l'emplacement de la sonde :

```
import board
```

Imports supplémentaires :

- **time** pour gérer les temporisations entre les mesures.
- **math** pour les calculs scientifiques.
- **datetime** pour afficher la date et l'heure des relevés.

```
import time
import math
from datetime import datetime
```

Différences entre import et from ... import ...

Quand on écrit import math, on importe toute la bibliothèque, et on accède à ses fonctions avec le préfixe math.nom_fonction (ex : math.log()).

Quand on écrit from datetime import datetime, on importe directement **une fonction ou une classe précise**, ce qui permet de l'utiliser sans préfixe (ex : datetime.now() au lieu de datetime.datetime.now()).

La première forme est plus explicite et lisible dans les grands scripts, la seconde est plus concise quand on utilise souvent la même fonction.

Déclaration du capteur

```
dhtDevice = adafruit_dht.DHT22(board.D4)
```

Cette ligne permet de créer **un objet Python qui représente le capteur DHT22** connecté à la broche **GPIO 4** du Raspberry Pi.

À noter : Ici, le capteur apparaît sous une forme de variable et non pas de constante car c'est un objet dont l'état peut évoluer (ex. : erreurs, fermeture avec .exit())

Les fonctions

Dans le script, on trouve deux blocs qui commencent par def. Ce sont des **fonctions**, c'est-à-dire des morceaux de code **réutilisables** qui effectuent une tâche bien précise :

```
def calculer_point_de_rosee(temperature, humidity):


...
```

```
def calculer_humidex(temperature, point_de_rosee):


---
```

Une fonction, c'est comme une **boîte à outils** : on lui donne des **données en entrée** (ici, la température et l'humidité pour le point de rosée), et elle nous renvoie un **résultat calculé** (le point de rosée ou l'humidex). On peut **réutiliser** cette fonction autant de fois qu'on veut, sans devoir réécrire le calcul à chaque fois.

Les fonctions permettent de structurer le code, de le rendre plus clair, plus facile à maintenir et à faire évoluer. C'est un pas de plus vers un code propre, compréhensible.

```
def calculer_point_de_rosee(temperature, humidity):
    # Formule pour calculer le point de rosée
    alpha = 17.27
    beta = 237.7
    gamma = (alpha * temperature) / (beta + temperature) + math.log(humidity / 100.0)
    point_de_rosee = (beta * gamma) / (alpha - gamma)
    return point_de_rosee
```

Fonction pour calculer le **point de rosée**, une mesure liée à la condensation de l'humidité (formule d'**August-Roche-Magnus**).

Le mot-clé **return** sert à **renvoyer le résultat** d'une fonction. C'est ce que la fonction renvoie à celui qui l'a appelée. Sans **return**, la fonction ferait les calculs, mais ne transmettrait rien de ses résultats !

```
def calculer_humidex(temperature, point_de_rosee):
    # Formule pour calculer l'humidex
    humidex = temperature + (5/9) * (6.11 * math.exp(5417.7530 * ((1/273.16) - (1/273.15 +
point_de_rosee))) - 10)
    return humidex
```

Fonction pour calculer l'**humidex**, un indice qui combine température et point de rosée pour donner une idée de la **température ressentie**.

Boucle infinie

Dans notre script, nous utilisons une **boucle infinie** grâce à la ligne :

```
while True:
```

Cela signifie que le bloc de code qui suit va s'exécuter en **boucle, sans fin**, tant qu'on n'interrompt pas manuellement le programme. Cette technique est parfaite pour une **station météo autonome** : elle va relever les données, les afficher, attendre un peu… puis recommencer, encore et encore. C'est ce qui permet de transformer notre Raspberry Pi en véritable service météo libre et auto-hébergé, toujours actif tant que la machine est allumée.

Les variables

```
humidity = dhtDevice.humidity
temperature = dhtDevice.temperature
```

on utilise **deux variables**, humidity et temperature. Une variable, c'est comme une **boîte** dans laquelle on peut **stocker une valeur** pour la réutiliser plus tard, contrairement aux constantes, ici la **valeur sera amenée à bouger**. La fonction interroge le capteur et renvoie deux valeurs : l'humidité et la température mesurées. On les **dépose directement dans deux variables**, pour pouvoir les afficher et/ou faire des calculs.

En nommant les variables de façon claire, on rend le code plus compréhensible et plus facile à relire.

Validité de la lecture

Une **condition** permets de vérifier si **la lecture de la sonde est valable**. Si la lecture est correcte, le programme continue normalement, sinon, un message d'erreur s'affiche.

```
if humidity is not None and temperature is not None:
    # Suite du programme
else:
    # Message en cas d'erreur
```

Date et heure

Dans notre station météo, on veut savoir quand chaque mesure a été prise. Pour ça, on utilise le module datetime, et plus précisément les lignes suivantes :

```
now = datetime.now()
date_heure = now.strftime("%d-%m-%Y %H:%M:%S")
```

- datetime.now() récupère **la date et l'heure actuelles** du système.
- strftime() permet de **formater** cette date dans un style plus lisible : ici, jour-mois-année heure:minute:seconde.

Calcul du point de rosée et de l'humidex

Après avoir lu la température et l'humidité, on utilise les fonctions **définies plus haut** pour effectuer deux calculs :

```
point_de_rosee = calculer_point_de_rosee(temperature, humidity)
humidex = calculer_humidex(temperature, point_de_rosee)
```

Ces lignes montrent comment on **réutilise nos fonctions** calculer_point_de_rosee() et calculer_humidex() en leur passant des **variables en paramètres**. Les résultats obtenus sont ensuite stockés dans **deux nouvelles variables**, point_de_rosee et humidex.

Cela montre la force des fonctions : une fois écrites, on peut les appeler facilement autant de fois qu'on veut, ce qui rend notre code modulaire, réutilisable et lisible

Couleurs ANSI

Nous pouvons ajouter en haut de notre script une série de **constantes** avec les couleurs :

On parle de constante car ces valeurs **ne changent pas pendant l'exécution du script**. On les écrit en **majuscules** pour le signaler clairement (c'est une convention en Python).

L'intérêt ? C'est plus lisible, plus facile à modifier si on change de capteur ou de broche, et ça évite de répéter ces valeurs partout dans le code. On améliore donc la clarté et la maintenabilité du programme.

```
RED = "\033[91m"
GREEN = "\033[92m"
YELLOW = "\033[93m"
BLUE = "\033[94m"
MAGENTA = "\033[95m"
RESET = "\033[0m"
```

Affichage des données

Nous utilisons les **couleurs** pour rendre l'affichage **plus clair et agréable à lire dans le terminal**.

```
print(f"{BLUE}Date et heure:{RESET} {date_heure}")
print(f"{GREEN}Température:{RESET} {temperature:.1f}°C")
print(f"{YELLOW}Humidité:{RESET} {humidity:.1f}%")
print(f"{RED}Point de rosée:{RESET} {point_de_rosee:.1f}°C")
print(f"{MAGENTA}Humidex:{RESET} {humidex:.1f}")
print("----")
```

Les variables numériques sont affichés avec **une seule décimale** grâce à la syntaxe {temperature:.1f}. Ce format permet d'avoir des valeurs **précises mais lisibles**, sans afficher une longue série de chiffres inutiles. Chaque ligne correspond à une donnée spécifique.

Erreur de lecture de la sonde

En cas d'échec de la lecture, un message d'erreur est affiché. C'est important pour diagnostiquer les soucis matériels ou de câblage.

```
else:
    print("Échec de la lecture du capteur")
```

Pause entre les lectures

Mettons une pause de **20 secondes** entre deux lectures, pour éviter de surcharger le capteur et ralentir le flux d'informations.

```
time.sleep(20)
```

Script complet

```
#Importation
import adafruit_dht
import adafruit_bmp280
import board
import busio
import time
import math
from datetime import datetime

#Déclaration du capteur
dhtDevice = adafruit_dht.DHT22(board.D4)

i2c = busio.I2C(board.SCL, board.SDA)

bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c, address=0x76)

#Couleurs
RED = "\033[91m"
GREEN = "\033[92m"
YELLOW = "\033[93m"
```

```
BLUE = "\033[94m"
MAGENTA = "\033[95m"
RESET = "\033[0m"

#Calcul du point de rosée
def calculer_point_de_rosee(temperature, humidity):
    # Formule pour calculer le point de rosée
    alpha = 17.27
    beta = 237.7
    gamma = (alpha * temperature) / (beta + temperature) + math.log(humidity / 100.0)
    point_de_rosee = (beta * gamma) / (alpha - gamma)
    return point_de_rosee

#Calcul de l'humidex
def calculer_humidex(temperature, point_de_rosee):
    # Formule pour calculer l'humidex
    humidex = temperature + (5/9) * (6.11 * math.exp(5417.7530 * ((1/273.16) - (1/273.15 +
point_de_rosee))) - 10)
    return humidex

#Boucle et affichage
while True:
    humidity = dhtDevice.humidity
    temperature = dhtDevice.temperature
    pression = bmp280.pressure
    if humidity is not None and temperature is not None:
        now = datetime.now()
        date_heure = now.strftime("%d-%m-%Y %H:%M:%S")
        point_de_rosee = calculer_point_de_rosee(temperature, humidity)
        humidex = calculer_humidex(temperature, point_de_rosee)
        print(f"{BLUE}Date et heure:{RESET} {date_heure}")
        print(f"{GREEN}Température:{RESET} {temperature:.1f}°C")
        print(f"{YELLOW}Humidité:{RESET} {humidity:.1f}%")
        print(f"{RED}Point de rosée:{RESET} {point_de_rosee:.1f}°C")
        print(f"{MAGENTA}Humidex:{RESET} {humidex:.1f}")
        print("----")

    else:
        print("Échec de la lecture du capteur")
```

```
#Pause de 20 secondes
time.sleep(20)
```

Partie IV

Utilisation de la fonction round()

Nous allons modifier notre script python pour introduire l'utilisation de la fonction Python round() pour **arrondir les valeurs numériques** (température, humidité, point de rosée, humidex) à **une seule décimale**.

Avant

Les variables temperature ou humidity pouvaient contenir des valeurs longues comme 23.67893452, ce qui :

- encombrait la sortie dans le terminal ou sur la page web,
- n'était pas lisible pour l'utilisateur final,
- et ne servait à rien dans un contexte grand public où une précision de 0,1 est largement suffisante.

Pour avoir un affichage correct, nous utilisons le code suivant : print(f"{temperature:.1f}"). Celui-ci arrondi bien notre résultat à un chiffre après la virgule, mais il n'arrondi que l'affichage.

Après

Grâce à round(variable, 1), on obtient des valeurs comme 23.7, ce qui :

- améliore la **clarté visuelle** des résultats,
- simplifie l'envoi des données vers une interface web ou une base de données,
- et réduit les erreurs d'interprétation dans les calculs suivants.

round() arrondit réellement la valeur stockée, contrairement à l'ancienne méthode. Cela permet donc une meilleure réutilisation des données : les fonctions ou interfaces web utilisent des valeurs déjà simplifiées.

Changement dans le code

```
print(f"{GREEN}Température :{RESET} {round(temperature, 1)}°C")
print(f"{YELLOW}Humidité :{RESET} {round(humidity, 1)}%")
print(f"{RED}Point de rosée :{RESET} {round(point_de_rosee, 1)}°C")
print(f"{MAGENTA}Indice humidex :{RESET} {round(humidex, 1)}")
```

Ce changement, bien que minime à première vue, marque une **étape importante vers la structuration professionnelle** du script et prépare le terrain pour la future séparation des modules (capteur, API, interface web).

Partie V

Logique métier

Nous sommes prêt à structurer **plus proprement notre application** en séparant :

- la **partie capteurs** (lecture, calculs...) -> **Logique métier**
- et l'**application web** (routes, affichage)

Création d'un fichier capteur.py

Dans une démarche de développement propre, modulaire et réutilisable, on **sépare la logique métier** (la lecture des données et les calculs) de **l'interface utilisateur** (site web, affichage).

Le fichier capteur.py **ne contient plus de boucle** while True ni de traitement direct. Il est composé **exclusivement de fonctions**, que l'on pourra appeler **depuis une autre application**, ici app.py (notre serveur Flask).

Ce script agit comme **une boîte à outils**, il regroupe :

- la lecture du capteur DHT22,
- le calcul du point de rosée,
- le calcul de l'indice humidex,
- la récupération de la date et heure actuelles.

Code

```
#importations
import adafruit_dht
import board
import math
from datetime import datetime



def lire_donnees_capteur():
    try:
```

```
        dhtDevice = adafruit_dht.DHT22(board.D4)
        humidity = dhtDevice.humidity
        temperature = dhtDevice.temperature
        dhtDevice.exit()
        if humidity is not None and temperature is not None:
            return round(humidity, 1), round(temperature, 1)
        else:
            return None, None

    except RuntimeError as error:
        print("Erreur de lecture :", error)
        return None, None

    except Exception as error:
        dhtDevice.exit()
        raise error

def calculer_point_de_rosee(temperature, humidity):
    #Formule pour calculer le point de rosée
    alpha = 17.27
    beta = 237.7
    gamma = (alpha * temperature) / (beta + temperature) + math.log(humidity / 100)
    point_de_rosee = (beta * gamma) / (alpha - gamma)
    return round(point_de_rosee, 1)

def calculer_humidex(temperature, point_de_rosee):
    #Formule pour calculer l'indice humidex
    humidex = temperature + (5/9) * (6.11 * math.exp(5417.7530 * ((1/273.16) - (1/273.15 +
point_de_rosee))) - 10)
    return round(humidex, 1)

def recuperer_date_heure():
    return datetime.now().strftime("%d-%m-%Y %H:%M:%S")
```

Commentaires sur la fonction lire_donnees_capteur() :

- Cette fonction permet de lire la température et l’humidité à partir du capteur **DHT22** connecté au **GPIO4** du Raspberry Pi.

- Elle renvoie ces deux valeurs sous forme de nombres arrondis à une décimale.
- Si la lecture échoue, elle retourne None, None.

```
try:
```

- On démarre un bloc qui va tenter de réaliser la lecture
- Si une erreur survient, Python basculera dans le bloc except.

```
dhtDevice = adafruit_dht.DHT22(board.D4)
```

- Cette ligne crée un objet capteur, ici un DHT22, branché sur le GPIO4 (représenté par board.D4).
- □□ □□ C’est une étape indispensable pour communiquer avec le capteur. L'objet est créé directement dans la fonction pour éviter tous risques de blocage du capteur.

```
humidity = dhtDevice.humidity temperature = dhtDevice.temperature
```

Ces lignes interrogent le capteur pour récupérer :

- humidity → le taux d’humidité.
- temperature → la température.

```
dhtDevice.exit()
```

- Cette commande est **très importante** : elle **libère les ressources GPIO** utilisées par le capteur.
- Cela évite les erreurs fréquentes sur Raspberry Pi telles que : “Lost access to message queue”.

```
if humidity is not None and temperature is not None:
```

- On vérifie que le capteur a bien répondu avec des données valides.
- Les capteurs DHT peuvent parfois échouer à donner une valeur.

```
return round(humidity, 1), round(temperature, 1)
```

Si les valeurs sont valides :

- On les arrondit à une décimale pour un affichage plus lisible.
- Puis on les renvoie sous la forme de deux nombres.

```
else: return None, None
```

Si le capteur n’a pas répondu correctement, la fonction renvoie None pour les deux mesures.

```
except RuntimeError as error:
```

- Gestion des erreurs courantes dues aux capteurs DHT (perte temporaire de lecture).
- On affiche l’erreur mais le programme continue de fonctionner.

```
except Exception as error:
```

- Gestion des erreurs plus graves (problème matériel, GPIO bloqué, etc.).
- On ferme proprement le capteur avec dhtDevice.exit() avant de relancer l’erreur (raise error) pour éventuellement arrêter le programme.

Partie VI

Interface utilisateur

La bibliothèque flask

Flask est un micro-framework web en Python, simple et léger. Il permet de créer rapidement un petit site web ou une API sans dépendances complexes.

Dans notre cas, Flask permet de :

- transformer votre Raspberry Pi en serveur web local ;
- créer une interface web pour afficher les données météo en temps réel ;
- séparer le traitement (dans capteur.py) de l'affichage (dans app.py).

Installation de flask

```
pip3 install flask
```

Importation de flask

Importons flask au début de notre code :

```
from flask import Flask, render_template_string
```

- Flask : la classe qui permet de **créer l'application web**.
- render_template_string : permet de **générer une page HTML directement dans le script** (sans créer de fichier .html pour le moment).

Importation des données de l'application métier

Dans notre fichier app.py, nous avons besoin d'accéder aux données produites par nos capteurs (température, humidité, etc.) et aux fonctions de calcul (point de rosée, humidex...).

Ces fonctions ne sont **pas écrites directement dans app.py**, mais dans un autre fichier appelé capteur.py. Ce fichier est ce qu'on appelle notre **application métier** : il contient toute la **logique de calcul et de lecture**.

```
from capteur import (
    lire_donnees_capteur,
    calculer_point_de_rosee,
    calculer_humidex,
    recuperer_date_heure
)
```

- from capteur : on indique qu'on veut importer **depuis le fichier capteur.py**.
- import (...) : on précise **quelles fonctions on veut utiliser** dans ce fichier.

C'est un peu comme si on disait :

« Va chercher dans la boîte capteur.py ces outils bien précis, et rends-les disponibles ici. »

Création de l'application Flask

```
# Création de l'application Flask
app = Flask(__name__)
```

Cette ligne est essentielle. Elle signifie :

- On crée une instance de notre application Flask.
- name est une variable spéciale en Python qui contient le nom du fichier. Flask s'en sert pour retrouver le chemin vers les fichiers associés (comme les templates HTML ou les fichiers CSS).

Autrement dit :

« Je démarre une application web avec ce fichier comme point d'entrée. »

Définir une route dans Flask

```
@app.route('/')
def index():
```

@app.route('/') : Indique que la fonction juste en dessous va être exécutée quand un utilisateur accède à l'adresse « / », c'est-à-dire la racine du site, la page d'accueil.

def index(): C'est une fonction Python classique, appelée ici index. Elle contiendra le code qui définit ce que l'on affiche ou retourne quand on visite cette page.

Récupération des données

```
humidity, temperature = lire_donnees_capteur()
if humidity is not None and temperature is not None:
    point_de_rosee = calculer_point_de_rosee(temperature, humidity)
    humidex = calculer_humidex(temperature, point_de_rosee)
    date_heure = recuperer_date_heure()
```

nous mettons en œuvre **la logique métier définie dans notre fichier capteur.py** :

- **lire_donnees_capteur()** : interroge le capteur DHT22 et renvoie les valeurs d'humidité et de température.
- **calculer_point_de_rosee()** : calcule le point à partir duquel l'humidité de l'air commence à se condenser.
- **calculer_humidex()** : estime la température ressentie selon l'humidité ambiante.
- **recuperer_date_heure()** : renvoie l'heure exacte de la mesure.

Génération simple d'une page HTML avec une f-string

```
html = f"""
<h1>Données météo locales</h1>
<ul>
  <li>Date et heure : {date_heure}</li>
  <li>Température : {temperature} °C</li>
  <li>Humidité : {humidity} %</li>
  <li>Point de rosée : {point_de_rosee} °C</li>
  <li>Humidex : {humidex}</li>
  </ul>
"""
```

Nous créons **une chaîne de caractères contenant du HTML**, dans laquelle nous insérons directement les **valeurs des variables Python** (température, humidité, etc.).

- La **balise h1** sert à afficher titre de niveau 1.
- La **balise ul** sert à afficher une liste à puce, les **balises li** correspondent aux éléments de la liste.
- La **balise p** sert à afficher un paragraphe.

Pourquoi le f devant la chaîne ?

Le f signifie que c'est une **f-string** (ou formatted string literal en anglais).

Cela permet d'écrire **des variables à l'intérieur de la chaîne** entre des accolades {}.

C’est une **méthode claire et moderne pour mélanger texte et variables** dans une même ligne.

Et si la lecture échoue ?

```
else:
    html = "<p>Erreur de lecture du capteur.</p>"
```

Ce code gère le cas où le capteur ne renvoie pas de valeurs valides. Dans ce cas, on prépare un message simple à afficher sur la page web.

Affichage de la page web et démarrage de l'application

```
return render_template_string(html)
```

Ce code indique à Flask d'**afficher le HTML généré comme contenu de la page web**.

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Ce code permet de lancer l'application :

- **host='0.0.0.0'** : rend l'application accessible sur le réseau.
- **port=5000** : port utilisé pour acceder au site.

Code

```
#importations
from flask import Flask, render_template_string
from capteur import (
    lire_donnees_capteur,
    calculer_point_de_rosee,
    calculer_humidex,
    recuperer_date_heure
)

# Définition de l'application Flask
app = Flask(__name__)

@app.route('/')
def index():
    humidity, temperature = lire_donnees_capteur()
```

```
    if humidity is not None and temperature is not None:
        point_de_rosee = calculer_point_de_rosee(temperature, humidity)
        humidex = calculer_humidex(temperature, point_de_rosee)
        date_heure = recuperer_date_heure()

        html = f"""
        <h1>Données météo locales</h1>
        <ul>
            <li>Date et heure : {date_heure}</li>
            <li>Température : {temperature} °C</li>
            <li>Humidité : {humidity} %</li>
            <li>Point de rosée : {point_de_rosee} °C</li>
            <li>Humidex : {humidex}</li>
        </ul>
        """

    else:
        html = "<p>Erreur de lecture du capteur.</p>"

    return render_template_string(html)
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

capture-station.png

Partie VII

Présentation de la sonde BMP280

Le **BMP280** est un petit capteur environnemental, capable de mesurer :

- La **pression atmosphérique** (en hPa)
- La **température de l'air** (en °C)

C'est un capteur léger, peu gourmand en énergie et parfaitement adapté aux stations météo autonomes.

Câblage du BMP280 sur le Raspberry Pi

Le BMP280 communique avec le Raspberry Pi via le **protocole I2C**, qui utilise **2 fils de données**, plus l'alimentation et la masse.

Broche BMP280	Broche Raspberry PI	Rôle
VCC	Broche 17 (3.3V)	Alimentation
GND	Broche 9 (GND)	Masse
SCL	Broche 5 (GPIO3)	Horloge (Clock)
SDA	Broche 3 (GPIO2)	Données (Data)

Activation du protocole I2C sur le Raspberry Pi

Le protocole I2C est désactivé par défaut sur Raspberry Pi OS. Il faut l'activer manuellement.

Ouvrir la configuration du Raspberry Pi :

```
sudo raspi-config
```

raspi-config.png

Aller dans le menu :

```
3 Interface Options
I2C → Enable
```

interface-option.png

Redémarrer le Raspberry Pi :

```
sudo reboot
```

Vérifier si le BMP280 est détecté

Installer les **outils I2C** :

```
sudo apt install i2c-tools
```

Scanner le bus I2C pour vérifier que le capteur est bien détecté :

```
i2cdetect -y 1
```

Exemple de retour :

scan-i2c.png

Ici, on voit bien 76, ce qui indique que le BMP280 est bien détecté.

Test de la sonde en mode interactif

Dans votre **environnement virtuel Python**, commencer par installer la **bibliothèque pour la sonde BMP280** :

```
 pip3 install adafruit-circuitpython-bmp280
```

Test en mode interactif :

```
python3

import board
import busio
import adafruit_bmp280

i2c = busio.I2C(board.SCL, board.SDA)
bm280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c, address=0x76)
```

```
print(bme280.pressure)
print(bme280.temperature)
```

Si le scan de votre bus i2c à donné 77 comme résultat, remplacer address=0x76 par address=0x77

Partie VIII

Ajout de la sonde BMP280 aux scripts

Nous allons modifier nos scripts existants pour tester la sonde bmp280 aussi bien dans le **terminal** que sur le **serveur web (flask)**.

Modification du script station.py

```
#Importation
import adafruit_dht
import adafruit_bmp280
import board
import busio
import time
import math
from datetime import datetime

#Déclaration du capteur
dhtDevice = adafruit_dht.DHT22(board.D4)

i2c = busio.I2C(board.SCL, board.SDA)

bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c, address=0x76)

#Couleurs
RED = "\033[91m"
GREEN = "\033[92m"
YELLOW = "\033[93m"
BLUE = "\033[94m"
MAGENTA = "\033[95m"
CYAN = "\033[96m"
```

```
RESET = "\033[0m"

#Calcul du point de rosée
def calculer_point_de_rosee(temperature, humidity):
    # Formule pour calculer le point de rosée
    alpha = 17.27
    beta = 237.7
    gamma = (alpha * temperature) / (beta + temperature) + math.log(humidity / 100.0)
    point_de_rosee = (beta * gamma) / (alpha - gamma)
    return point_de_rosee

#Calcul de l'humidex
def calculer_humidex(temperature, point_de_rosee):
    # Formule pour calculer l'humidex
    humidex = temperature + (5/9) * (6.11 * math.exp(5417.7530 * ((1/273.16) - (1/273.15 +
point_de_rosee))) - 10)
    return humidex

#Boucle et affichage
while True:
    humidity = dhtDevice.humidity
    temperature = dhtDevice.temperature
    pression = bmp280.pressure
    if humidity is not None and temperature is not None:
        now = datetime.now()
        date_heure = now.strftime("%d-%m-%Y %H:%M:%S")
        point_de_rosee = calculer_point_de_rosee(temperature, humidity)
        humidex = calculer_humidex(temperature, point_de_rosee)
        print(f"{BLUE}Date et heure:{RESET} {date_heure}")
        print(f"{GREEN}Température:{RESET} {round(temperature, 1)}°C")
        print(f"{YELLOW}Humidité:{RESET} {round(humidity, 1)}%")
        print(f"{RED}Point de rosée:{RESET} {round(point_de_rosee, 1)}°C")
        print(f"{MAGENTA}Humidex:{RESET} {round(humidex, 1)}")
        print(f"{CYAN}Pression atmosphérique:{RESET} {round(pression, 2)} hPa")
        print("----")
    else:
        print("Échec de la lecture du capteur")

    #Pause de 20 secondes
```

```
    time.sleep(20)
```

Modification du script capteur.py

```
#importations
import adafruit_dht
import adafruit_bmp280
import board
import busio #Bibliothèque permettant d'ouvrir une communication sur un bus matériel
import math
from datetime import datetime

#Initialisation du bus I2C
i2c = busio.I2C(board.SCL, board.SDA)

#Initialisation de la sonde
bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c, address=0x76)

def lire_donnees_capteur():
    try:
        dhtDevice = adafruit_dht.DHT22(board.D4)
        humidity = dhtDevice.humidity
        temperature = dhtDevice.temperature
        dhtDevice.exit()
        if humidity is not None and temperature is not None:
            return round(humidity, 1), round(temperature, 1)
        else:
            return None, None

    except RuntimeError as error:
        print("Erreur de lecture :", error)
        return None, None

    except Exception as error:
        dhtDevice.exit() #Libérer le GPIO même en cas de crash
        raise error #Le programme crashe, car c'est une erreur critique

def calculer_point_de_rosee(temperature, humidity):
    #Formule pour calculer le point de rosée
```

```
    alpha = 17.27
    beta = 237.7
    gamma = (alpha * temperature) / (beta + temperature) + math.log(humidity / 100)
    point_de_rosee = (beta * gamma) / (alpha - gamma)
    return round(point_de_rosee, 1)

def calculer_humidex(temperature, point_de_rosee):
    #Formule pour calculer l'indice humidex
    humidex = temperature + (5/9) * (6.11 * math.exp(5417.7530 * ((1/273.16) - (1/273.15 +
point_de_rosee))) - 10)
    return round(humidex, 1)

def lire_pression():
    try:
        pression = bmp280.pressure
        return round(pression, 1)
    except Exception as error:
        print("Erreur de lecture BMP280 :", error)
        return None

def recuperer_date_heure():
    return datetime.now().strftime("%d-%m-%Y %H:%M:%S")
```

Modification du script app.py

```
#importations
from flask import Flask, render_template_string
from capteur import (
    lire_donnees_capteur,
    calculer_point_de_rosee,
    calculer_humidex,
    recuperer_date_heure,
    lire_pression
)

# Définition de l'application Flask
app = Flask(__name__)

@app.route('/')
```

```
def index():
    humidity, temperature = lire_donnees_capteur()
    if humidity is not None and temperature is not None:
        point_de_rosee = calculer_point_de_rosee(temperature, humidity)
        humidex = calculer_humidex(temperature, point_de_rosee)
        date_heure = recuperer_date_heure()
        pression = lire_pression()

        html = f"""
        <h1>Données météo locales</h1>
        <ul>
            <li>Date et heure : {date_heure}</li>
            <li>Température : {temperature} °C</li>
            <li>Humidité : {humidity} %</li>
            <li>Point de rosée : {point_de_rosee} °C</li>
            <li>Humidex : {humidex}</li>
            <li>Pression atmosphérique : {pression} hPa</li>
        </ul>
        """
    else:
        html = "<p>Erreur de lecture du capteur.</p>"

    return render_template_string(html)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Partie IX

Mise en couleur dans le terminal avec Rich

Pourquoi améliorer l'affichage dans un terminal ?

Lorsque notre station de jardin renvoie des données météo (température, humidité, pression...), il est utile d'avoir un affichage **lisible** et **coloré** pour mieux distinguer les différentes mesures. Jusqu'ici, nous utilisions des **codes ANSI** pour ajouter un peu de couleur dans le terminal.

Mais pour un rendu plus **propre**, **lisible**, **personnalisable** et **moderne**, la bibliothèque rich est une excellente alternative.

Couleurs ANSI vs Rich

◫◫ ANSI (codes couleurs bruts)	◫◫ Rich (interface haut niveau)
Utilise des codes comme `\033[91m`	Utilise des noms lisibles : `“red”`, `“green”`, `“cyan”`...
Peu de contrôle sur le style	Affichage centré, aligné, stylisé facilement
Difficile à maintenir et à lire	Code clair, moderne, accessible
8 couleurs de base	Plus de 140 couleurs disponibles
Pas d'affichage enrichi	Icônes, tableaux, mise en page possible

Liste de couleurs Rich utiles

Voici quelques couleurs bien contrastées à utiliser dans un terminal (parmi les 140 disponibles) :

- red
- green
- blue
- yellow

- magenta
- cyan
- white
- black
- orange1
- violet
- deep_sky_blue1
- spring_green2
- dark_orange3
- turquoise2
- light_salmon1ark_orange3
- turquoise2
- light_salmon1
- chartreuse2
- sky_blue1
- gold3
- plum4
- aquamarine1
- medium_violet_red
- khaki1
- grey50

⚠ Certaines couleurs ne sont visibles correctement que sur les terminaux qui supportent le mode 256 couleurs.

Installation de la bibliothèque Rich

```
pip install rich
```

Initialisation de Rich

```
from rich.console import Console
from rich.text import Text
```

- **Console** est l'outil de base de la bibliothèque **Rich**. C’est lui qui gère l’affichage dans le terminal.
- **Text** permet de **styliser dynamiquement** des portions de texte dans une même ligne (**gras**, *italique*, souligné...).

Création de l'objet console :

```
console = Console()
```

Cette ligne permet de créer un objet console qui va gérer l'affichage dans le terminal. C’est avec cet objet qu’on peut utiliser toutes les fonctions de rich.

Affichage des données

```
# Importation des bibliothèques permettant de gérer couleurs et mise en page
from rich.console import Console
from rich.text import Text

# Déclaration de l'objet console pour gérer les couleurs et mise en page
console = Console()

console.print(f"[bold cyan]Date et heure :[/bold cyan] {date_heure}")
console.print(f"[bold red]Température :[/bold red] {round(temperature, 1)}°C")
console.print(f"[bold blue]Humidité :[/bold blue] {round(humidity, 1)}%")
console.print(f"[bold magenta]Point de rosée :[/bold magenta] {round(point_de_rosee, 1)}°C")
console.print(f"[bold yellow]Humidex :[/bold yellow] {round(humidex, 1)}")
console.print(f"[bold green]Pression atmosphérique :[/bold green] {round(pression, 2)} hPa")
```

- console.print(...) : méthode principale pour **afficher du texte enrichi** (couleur, gras, emoji...).
- [bold red]Température :[/bold red] : indique que le mot “**Température** :” sera **affiché en rouge gras**.

couleur-rich.png

Partie X

Icônes Unicode dans les scripts Python

Pourquoi les utiliser ?

Les caractères Unicode (émojis et pictogrammes) permettent :

- d'améliorer la **lisibilité** de vos affichages terminal ;
- de rendre les données plus **intuitives** ;
- de créer des **interfaces plus conviviales**, même en mode texte.

Comment afficher un caractère Unicode en python ?

Chaque icône a un **code Unicode** qu'on peut insérer en Python comme ceci :

```
print("\U0001F4C5")  # Affiche □□(calendrier)
```

⚠ Il faut **8 caractères hexadécimaux** après \U (majuscule). Complétez avec des zéros si besoin.

Écrire un caractère Unicode avec son clavier (Ubuntu) ?

Sur le **système Ubuntu** (et plus généralement sous tout système Linux basé sur X11), vous pouvez **taper un caractère Unicode** grâce à une combinaison de touches très simple :

- Appuyer sur **Ctrl + Shift + u** → un u souligné apparaît.
- Taper ensuite le code hexadécimal Unicode (par exemple 1f4c5 pour □□).
- Appuyer sur Entrée ou Espace → le caractère est inséré.

Liste d'icônes par thème

?? Météo & Environnement :

Icône	Signification	Code Unicode
□□	Thermomètre / température	1f321
□□	Goutte d’eau / humidité	1f4a7
❄	Flocon de neige / point de rosée	2746
□□	Flamme / chaleur / humidex	1f525
□□	Vent / pression atmosphérique	1f32c
□□	Jeune pousse / jardin / sol	1f331
□□	Tournesol / jardin	1f33b

? Temps & Organisation :

Icône	Signification	Code Unicode
□□	Calendrier	1f4c5
□□	Horloge / heure	1f552
□□	Calendrier à feuillets	1f5d3

? Capteurs & Science :

Icône	Signification	Code Unicode
□□	Expérience / capteur	1f9ea
□□	ADN / science	1f9ec
⚗	Alambic / mesure scientifique	2697

?? Informatique et électronique :

Icône	Signification	Code Unicode
□□	Ordinateur	1f5a5
□□	Sauvegarde / stockage	1f4be

Icône	Signification	Code Unicode
□□	Électricité / câblage	1f50c
□□	Boîte à outils / montage	1f9f0
□□	Électronique / aimant	1f9f2

?? Outils & Fabrication :

Icône	Signification	Code Unicode
□□	Outils / montage	1f6e0
□□	Clé à molette	1f527
□□	Marteau	1f528
□□	Tournevis	1fa9b

? Apprentissage & Partage :

Icône	Signification	Code Unicode
□□	Apprentissage / formation	1f393
□□□□	Animateur / médiateur	1f9d1 ou 200d ou 1f3eb
□□	Savoirs / wiki / doc	1f4da
□□	Note / documentation	1f4dd

? Final & Événement :

Icône	Signification	Code Unicode
□□	Fête / restitution	1f389
□□	Diplôme / récompense	1f3c6
□□	Partenariat / inclusion	1f91d

Vous pouvez retrouver l'ensemble des émojis en symbole ici :

https://www.unicode.org/emoji/charts/full-emoji-list.html

⚠ Le chargement peut être très très long !

Code

```
console.print(f"\U0001f4c5      [bold cyan]Date et heure :[/bold cyan] {date_heure}")
console.print(f"\U0001f321      [bold red]Température :[/bold red] {round(temperature, 1)}°C")
console.print(f"\U0001f4a7      [bold blue]Humidité :[/bold blue] {round(humidity, 1)}%")
console.print(f"\U00002746      [bold magenta]Point de rosée :[/bold magenta]
{round(point_de_rosee, 1)}°C")
console.print(f"\U0001f525      [bold yellow]Humidex :[/bold yellow] {round(humidex, 1)}")
console.print(f"\U0001f32c      [bold green]Pression atmosphérique :[/bold green]
{round(pression, 2)} hPa")
```

affichage_emojis.png

Partie XI

Comprendre la breadboard (plaque de prototypage rapide)

Qu'est-ce qu'une breadboard ?

Une **breadboard**, ou **plaque d’essai sans soudure**, est un outil essentiel pour tester et assembler des circuits électroniques **sans avoir à souder**. Elle permet de réaliser des montages rapidement, de les modifier facilement et de bien comprendre comment circulent les signaux et l’énergie dans un circuit.

Elle est composée :

- de **colonnes verticales** de chaque côté (repérées souvent par + et -), on y connecte généralement le + (**3,3V ou 5V**) d’un côté et le **GND** (**masse**) de l’autre,
- de **lignes horizontales** centrales, connectées 5 par 5, où l’on insère les **composants et capteurs** (DHT22, BMP280, etc.).

Pourquoi utiliser une breadboard pour notre station de jardin ?

breadboard-400-punti.jpeg

- □□ **D’alimenter plusieurs capteurs en parallèle** avec une seule broche 3.3V ou GND : chaque ligne de la breadboard partage l’électricité à tous les composants branchés dessus.
- □□ **De partager le protocole I2C (SDA/SCL)** : le bus I2C permet à plusieurs capteurs de communiquer sur les mêmes broches, tant qu’ils ont une adresse différente (comme le BMP280 sur 0x76 ou 0x77).
- □□ **De prototyper, tester et modifier facilement** votre montage sans soudure.
- □□ **De déporter les capteurs** pour qu’ils soient mieux placés dans le jardin, tout en gardant le Raspberry Pi au sec dans un boîtier.

Plutôt que de brancher chaque composant directement sur les broches du Raspberry Pi (ce qui deviendrait vite ingérable), **on déporte les connexions sur la breadboard**, qui agit comme **un**

répartiteur.

Montage des capteurs sur la breadboard

La breadboard jour le rôle de hub de connexion entre le Raspberry Pi et les sondes :

breadboard.jpg

- **3.3V** (alimentation) du Raspberry Pi est envoyé sur **la ligne d'alimentation rouge** de la breadboard.
- **GND** (masse) du Raspberry Pi est envoyé sur **la ligne noire (ou bleue)** de la breadboard.
- Les **GPIO** (pour la DHT22) et **SDA/SCL** (pour la BMP280) sont envoyés sur des **rangées centrales de la breadboard**.

DHT22 :

- VCC (alimentation) → ligne rouge (3.3V)
- GND → ligne bleue (GND)
- Data → GPIO4 (broche 7 du Pi) via une rangée de la breadboard

BMP280 (I2C) :

- VCC → ligne rouge (3.3V)
- GND → ligne bleue
- SDA → broche 3 du Pi (GPIO2) via une rangée de la breadboard
- SCL → broche 5 du Pi (GPIO3) via une rangée de la breadboard

Tous ces fils passent par la **breadboard**, qui facilite leur organisation et future extension (ajout de relais, convertisseur, etc.).

Partie XII

À quoi sert un convertisseur analogique-numérique (ADC) ?

Le Raspberry Pi, contrairement à un Arduino, **n'a pas d'entrée analogique**. Il ne peut lire que **des valeurs numériques** (0 ou 1, HIGH ou LOW). Or, certains capteurs, comme **la sonde d'humidité du sol Gravity SEN0193**, peuvent envoyer **une tension variable** représentant un niveau d'humidité, et non un simple "sec" ou "humide".

C'est là qu'intervient **le convertisseur analogique-numérique (ADC)**, comme le **MCP3008**.

Pourquoi préférer une lecture analogique pour la sonde SEN0193 ?

mcp3008-ip-convertisseur-analogique-numerique.jpgLa sonde Gravity SEN0193, selon le modèle peut fonctionner en deux modes :

- **Numérique** : elle renvoie simplement 1 (sol sec) ou 0 (sol humide). C'est tout.
- **Analogique** : elle renvoie une **valeur continue** entre 0 et 3.3V, que le MCP3008 convertit en une valeur entre **0 et 1023**.

Avantages de l'analogique :

- Permet un **suivi plus fin et progressif de l'humidité** (utile pour déclencher l'arrosage à un seuil précis).
- Donne la **possibilité de créer des graphiques, des seuils personnalisés**, des alertes...
- Ouvre la voie à une gestion **intelligente et économe de l'eau**.

Présentation du MCP3008

Le **MCP3008** est une petite puce qui permet de convertir jusqu'à **8 signaux analogiques** en valeurs numériques que le Raspberry Pi peut comprendre, via le protocole SPI.

Il se connecte au Raspberry Pi **via la breadboard** pour simplifier les branchements.

Branchement du MCP3008

mcp3008.jpegLa **puce MCP3008** doit être placée sur le breadboard **à cheval sur la “tranchée centrale”**, de manière à ce que **chaque broche soit positionnée sur une rangée indépendante**, ce qui permet un câblage propre et organisé.

MCP3008	Fonction	Branchement sur la breadboard
16 (VDD)	Alimentation	Jumper vers la ligne rouge (3.3V)
15 (VREF)	Référence	Jumper vers la ligne rouge (3,3V)
14 (AGND)	Masse analogique	Jumper vers la ligne noire (GND)
13 (CLK)	Horloge SPI	GPIO11 du Raspberry Pi
12 (DOUT)	Données vers le Raspberry Pi	GPIO9 du Raspberry Pi
11 (DIN)	Données du Raspberry Pi	GPIO10 du Raspberry Pi
10 (CS)	Chip select	GPIO5 du Raspberry Pi
9 (DGND)	Masse numérique	Jumper vers la ligne noire (GND)
1 (CH0)	Canal analogique 0	Sortie A0 de la sonde

Broche	Nom	Rôle / Explication
16	VDD	Tension d’alimentation **du circuit numérique** (généralement **3,3V ou 5V**).
15	VREF	Tension de **référence** pour la conversion analogique. **On la relie à VDD**. La précision des mesures dépend de cette valeur.
14	AGND	**Masse (GND)** pour la partie **analogique** (capteurs). À relier au GND du circuit.
13	CLK	**Horloge SPI** : le Raspberry Pi envoie un signal ici pour cadencer les échanges de données.
12	DOUT	**Data OUT** : les **données numériques sortent** du MCP3008 vers le Raspberry Pi.

Broche	Nom	Rôle / Explication
11	DIN	**Data IN** : le Raspberry Pi **envoie des commandes vers le MCP3008** (ex : “lis le canal 0”).
10	CS	**Chip Select** (ou CE = Chip Enable). Sert à dire “je parle maintenant à ce composant SPI”.
9	DGND	**Masse (GND)** pour la partie **numérique** (Raspberry Pi). À relier au GND.
1 à 8	CH0 à CH7	**Canaux analogiques d’entrée** (pour capteurs). Le MCP3008 peut lire **jusqu’à 8 capteurs analogiques**. CH0 est le plus utilisé.

- □□ VDD + VREF = alimentent la puce et définissent la précision.
- □□ AGND + DGND = masses nécessaires pour les parties analogiques et numériques.
- □□ CLK, DOUT, DIN, CS = communication SPI avec le Raspberry Pi.
- □□ CH0 à CH7 = brancher ici les capteurs analogiques, comme l’humidité du sol SEN0193.

Le protocole SPI

C’est un **protocole de communication** utilisé pour faire dialoguer un microcontrôleur (comme le Raspberry Pi) avec des composants externes

C’est un **bus rapide**, synchrone (horloge partagée), **plein-duplex** (on peut envoyer et recevoir en même temps).

Le protocole SPI utilise 4 fils :

Nom du fil	Fonction	Correspondance sur le Raspberry Pi
`MOSI`	Master Out Slave In : le Pi envoie les données vers le capteur	GPIO10 (Pin 19)
`MISO`	Master In Slave Out : le Pi lit les données du capteur	GPIO9 (Pin 21)
`SCLK` ou `CLK`	Clock : le signal d’horloge synchronise les échanges	GPIO11 (Pin 23)
`CS` ou `CE`	Chip Select : active le capteur concerné	Par exemple GPIO5 (Pin 29)

Activer le protocole SPI

```
sudo raspi-config
```

interface-option.png

Aller dans l'interface d'administration du Raspberry Pi :

- Aller dans Interfaces
- Activer SPI
- Redémarrer

Installation de la bibliothèque

Pour pouvoir installer la bibliothèque, commençons pas entrer dans notre environnement virtuel :

```
source meteo/bin/activate
```

Installation de la bibliothèque :

```
pip3 install Adafruit_CircuitPython_MCP3xxx
```

Partie XIII

Présentation de la sonde d’humidité du sol Gravity SEN0193

Description

La **Gravity SEN0193** est une **sonde capacitive d’humidité du sol** développée par DFRobot. Elle mesure **l’humidité du sol** de façon **capacitive**, ce qui est **plus fiable** et **plus durable** que les anciennes sondes résistives (qui s’oxydent avec le temps).

Avantages

- □□ Capacitive : ne s’oxyde pas, dure plus longtemps que les modèles bas de gamme.
- □□ Compacte : facile à insérer dans un pot ou un bac.
- □□ Compatible Raspberry Pi via convertisseur ADC.

Branchement de la sonde

Voici le schéma logique de câblage pour la sonde sur le MCP3008 déjà alimenté et installé :

Broche de la sonde	À connecter sur…
`VCC`	ligne 3,3V (pin 1 du Raspberry Pi) via breadboard
`GND`	ligne GND (pin 6 du Raspberry Pi) via breadboard
`A0` (sortie)	une des entrées **CH0 à CH7** du MCP3008 (ex: **CH0**)

Test de la sonde en mode interactif

```
#Importation des bibliothèques nécessaires
import busio # Pour initialiser et utiliser le bus SPI (communication avec le MCP3008)
import digitalio  # Pour gérer les entrées/sorties numériques (notamment la broche CS du SPI)
```

```
import board  # Pour accéder aux broches physiques du Raspberry Pi via des noms symboliques

# Importation de la bibliothèque du convertisseur analogique/numérique MCP3008
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn

#Initialisation du bus SPI matériel (horloge, entrée et sortie de données)
spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)

#Définir la broche utilisée pour sélectionner le MCP3008 (CS = Chip Select)
# Ici on utilise la broche physique D5 (GPIO5), mais on peut en choisir une autre
cs = digitalio.DigitalInOut(board.D5)

#Création d’une instance du MCP3008 relié via SPI
mcp = MCP.MCP3008(spi, cs)

#Définir une entrée analogique sur le canal CH0 (où est branchée la sonde d’humidité Gravity)
capteur_1 = AnalogIn(mcp, MCP.P0)  # Capteur 1 branché sur CH0
# capteur_2 = AnalogIn(mcp, MCP.P1)  # Capteur 2 (à décommenter si besoin)
# capteur_3 = AnalogIn(mcp, MCP.P2)  # Capteur 3...

#Affichage des valeurs
capteur_1.value
capteur_1.voltage
```

Partie XIV

Ajout de la sonde d’humidité du sol aux scripts

Modification du script station.py

```
# ==========
# Bibliothèques pour les cpteurs DHT22 (température, humidité) et BMP280 (pression
atmosphérique)
# ==========
import adafruit_dht
import adafruit_bmp280

# ==========
# Bibliothèques pour le convertisseur analogique-numérique MCP3008
# qui permet de lire la valeur de la sonde d'humidité du sol (analogique - numérique)
# ==========
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn

# ==========
# Bibliothèque pour gérer la broche CS (chip select) utilisée avec le protocole SPI
# ==========
import digitalio

# ==========
# Bibliothèque pour gérer les bus de communication (I2C pour BMP280, SPI pour MCP3008)
# ==========
import busio

# ==========
# Bibliothèque qui simplifie l'accès auc broches GPIO (par exemple board.D4)
```

```
# ==========
import board

# ==========
# Bibliothèques pour affichage en couleur et mise en page dans le terminal
# ==========
from rich.console import Console
from rich.text import Text

# ==========
# # Bibliothèques Python standards
# ==========
import time # Gérer les pauses entre deux lectures
import math # Calculs mathématiques (point de rosée, humidex)
from datetime import datetime # Obtenir la date et l'heure actuelles

# ==========
# Déclaration des capteurs :
# ==========

# Capteur DHT22 branché sur la broche D4 (température et humidité de l'air)
dhtDevice = adafruit_dht.DHT22(board.D4)

# Capteur BMP280 branché en I2C (pression atmosphérique)
i2c = busio.I2C(board.SCL, board.SDA)
bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c, address=0x76)

# Convertisseur MCP3008 branché en SPI (permet de lire la sonde Gravity analogique)
spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)
cs = digitalio.DigitalInOut(board.D5)
mcp = MCP.MCP3008(spi, cs)

# ==========
# Fonctions de calcul météo
# ==========

def calculer_point_de_rosee(temperature, humidity):
    # Calcule le point de rosée à partir de la température et de l'humidité relative.
    # Le point de rosée indique à quelle température l'air devient saturé en humidité
(condensation)
```

```
    alpha = 17.27
    beta = 237.7
    gamma = (alpha * temperature) / (beta + temperature) + math.log(humidity / 100.0)
    point_de_rosee = (beta * gamma) / (alpha - gamma)
    return point_de_rosee

def calculer_humidex(temperature, point_de_rosee):
    # Calcule l'indice Humidex à partir de la température et du point de rosée.
    # L'humidex est un indicateur de confort thermique développé au Canada
    # Il combine la température et le point de rosée pour extimer la sensation de chaleur ressentie par le corps humain :
    # Un humidex de 30 indique une chaleur lourde
    # Au-delà de 40, la chaleur devient dangereuse pour la santé
    humidex = temperature + (5/9) * (6.11 * math.exp(5417.7530 * ((1/273.16) - (1/273.15 + point_de_rosee))) - 10)
    return humidex

def convertir_tension_en_pourcentage(voltage):
# Fonction permettant de convertir une tension mesurée par la sonde d'humidité du sol (0V - 3.3V) en un pourcentage d'humidité du sol (0% - 100%)
    valeur_normalisee = voltage / 3.3
    # 1. Normaliser la valeur : rapport entre la tension mesurée et la tension max (3.3V), donne un résultat entre 0 (0V) et 1 (3,3V)
    valeur_inversee = 1 - valeur_normalisee
    # 2. Inverser l'échelle car la sonde donne 0V = sol humide, 3.3V = sol sec, l'inversion donnera donc un résultat entre 1 (trés humide, 0V) et 0 (trés sec, 3.3V)
    pourcentage = valeur_inversee * 100
    # 3. Conversion en pourcentage
    pourcentage_final = max(0, min(100, pourcentage))
    # 4. Sécuriser pour rester dans l'interval [0, 100], utile en cas de bruit électrique ou si la tension dépasse légèrement les bornes
    return pourcentage_final

# Déclaration de l'objet console pour gérer les couleurs et mise en page
console = Console()

# ==========
# Boucle principale
# ==========
# Lecture des capteurs toutes les 20 secondes
```

```
# Calcul des indicateurs (point de rosée, humidex, humidité du sol)
# Affichage avec couleurs et icônes
while True:
    humidity = dhtDevice.humidity
    temperature = dhtDevice.temperature
    capteur_humidite_1 = AnalogIn(mcp, MCP.P0)
    voltage = capteur_humidite_1.voltage
    pression = bmp280.pressure
    if humidity is not None and temperature is not None:
        now = datetime.now()
        date_heure = now.strftime("%d-%m-%Y %H:%M:%S")
        point_de_rosee = calculer_point_de_rosee(temperature, humidity)
        humidex = calculer_humidex(temperature, point_de_rosee)
        humidity_pct = convertir_tension_en_pourcentage(voltage)
        console.print(f"\U0001f4c5 [bold cyan]Date et heure :[/bold cyan] {date_heure}")
        console.print(f"\U0001f321 [bold red]Température :[/bold red] {round(temperature,
1)}°C")
        console.print(f"\U0001f4a7 [bold blue]Humidité :[/bold blue] {round(humidity, 1)}%")
        console.print(f"\U00002746 [bold magenta]Point de rosée :[/bold magenta]
{round(point_de_rosee, 1)}°C")
        console.print(f"\U0001f525 [bold yellow]Humidex :[/bold yellow] {round(humidex, 1)}")
        console.print(f"\U0001f32c [bold green]Pression atmosphérique :[/bold green]
{round(pression, 2)} hPa")
        console.print(f"\U0001f331 [bold navajo_white1]Capteur d'humidité du sol 1 (tension)
:[/bold navajo_white1] {round(voltage, 1)} V")
        console.print(f"\U0001f9ea [bold gold3]Capteur d'humidité du sol 1 (résultat exprimé en
pourcentage) :[/bold gold3] {round(humidity_pct, 1)}%")
        print("----")
    else:
        print("Échec de la lecture du capteur")

    #Pause de 20 secondes
    time.sleep(20)
```

Conversion de la tension en pourcentage d’humidité du sol

La sonde d’humidité du sol **Gravity** est un capteur **analogique** :
elle envoie une **tension électrique comprise entre 0 et 3,3V**, proportionnelle à l’humidité détectée.

- **0 V → sol très humide (100% d’humidité)**

- **3,3 V → sol très sec (0% d’humidité)**

Or, une tension exprimée en volts n’est pas très parlante pour un utilisateur.

La fonction Python **convertir_tension_en_pourcentage(voltage)** traduit directement cette tension en un pourcentage d’humidité du sol, beaucoup plus intuitif à lire.

La bibliothèque du MCP3008 permet aussi de récupérer une **valeur brute** (par ex. entre 0 et 65535).
Mais cette valeur dépend directement du convertisseur, pas du capteur, et elle est **moins parlante** pour l’utilisateur.
C’est pourquoi nous ne retenons que :

- la **tension en volts** (utile pour les tests techniques),
- et le **pourcentage d’humidité**, compréhensible par tout le monde.

Modification du script capteur.py

```
# ============================
# Bibliothèques pour les cpteurs DHT22 (température, humidité) et BMP280 (pression
atmosphérique)
# ============================
import adafruit_dht
import adafruit_bmp280

# ==========
# Bibliothèques pour le convertisseur analogique-numérique MCP3008
# qui permet de lire la valeur de la sonde d'humidité du sol (analogique - numérique)
# ==========
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn

# ==========
# Bibliothèque pour gérer la broche CS (chip select) utilisée avec le protocole SPI
# ==========
import digitalio

# ==========
# Bibliothèque pour gérer les bus de communication (I2C pour BMP280, SPI pour MCP3008)
# =========
import busio
```

```
# ==========
# Bibliothèque qui simplifie l'accès auc broches GPIO (par exemple board.D4)
# ==========
import board

# ==========
# Bibliothèques Python standards
# ==========
import math
from datetime import datetime

# ==========
# Déclaration des capteurs :
# ==========

# Capteur BMP280 branché en I2C (pression atmosphérique)
i2c = busio.I2C(board.SCL, board.SDA)
bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c, address=0x76)

# ============================
# Fonctions de lecture capteurs
# ============================

# Fonction de lecture de la sonde DHT22
def lire_donnees_capteur():
    try:
        dhtDevice = adafruit_dht.DHT22(board.D4)
        # Initialisation de la sonde DHT22
        humidity = dhtDevice.humidity
        # Récupération de l'humidité
        temperature = dhtDevice.temperature
        # Récupération de la temperature
        dhtDevice.exit()
        if humidity is not None and temperature is not None:
            return round(humidity, 1), round(temperature, 1)
        else:
            return None, None

    except RuntimeError as error:
        print("Erreur de lecture :", error)
```

```
        return None, None

    except Exception as error:
        dhtDevice.exit()
        # Libérer le GPIO même en cas de crash
        raise error
        # Le programme crashe, car c'est une erreur critique

# Fonction de lecture de la sonde BMP280
def lire_pression():
    try:
        pression = bmp280.pressure
        return round(pression, 1)
    except Exception as error:
        print("Erreur de lecture BMP280 :", error)
        return None

# ==========
# Fonctions de calcul météo
# ==========
def calculer_point_de_rosee(temperature, humidity):
    # Calcule le point de rosée à partir de la température et de l'humidité relative.
    # Le point de rosée indique à quelle température l'air devient saturé en humidité
(condensation)
    alpha = 17.27
    beta = 237.7
    gamma = (alpha * temperature) / (beta + temperature) + math.log(humidity / 100)
    point_de_rosee = (beta * gamma) / (alpha - gamma)
    return round(point_de_rosee, 1)

def calculer_humidex(temperature, point_de_rosee):
    # Calcule l'indice Humidex à partir de la température et du point de rosée.
    # L'humidex est un indicateur de confort thermique développé au Canada
    # Il combine la température et le point de rosée pour extimer la sensation de chaleur
ressentie par le corps humain :
    # Un humidex de 30 indique une chaleur lourde
    # Au-delà de 40, la chaleur devient dangereuse pour la santé
    humidex = temperature + (5/9) * (6.11 * math.exp(5417.7530 * ((1/273.16) - (1/273.15 +
point_de_rosee))) - 10)
    return round(humidex, 1)
```

```
# ============================
# Humidité du sol (via MCP3008)
# ============================
def lire_humidite_sol():
    """
    Lecture de la sonde d’humidité du sol (canal CH0 du MCP3008).
    Retourne (tension en Volts, pourcentage estimé d’humidité).
    """

    try:
        # Convertisseur MCP3008 branché en SPI (permet de lire la sonde Gravity analogique)
        spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)
        cs = digitalio.DigitalInOut(board.D5)
        mcp = MCP.MCP3008(spi, cs)

        capteur_humidite = AnalogIn(mcp, MCP.P0)
        # Capteur branché sur CH0

        voltage = capteur_humidite.voltage

        # Conversion en pourcentage (0V = 100% humidité, 3.3V = 0%)
        valeur_normalisee = voltage / 3.3
        valeur_inversee = 1 - valeur_normalisee
        pourcentage = valeur_inversee * 100
        pourcentage_final = max(0, min(100, round(pourcentage, 1)))

        return round(voltage, 2), pourcentage_final

    except Exception as e:
        print("Erreur lecture MCP3008 :", e)
        return None, None

    finally:
        # Libération propre de la broche CS
        cs.deinit()

# ============================
# Récupération de la date et de l'heure
# ============================
```

```
def recuperer_date_heure():
```

Pourquoi utiliser `try/except/finally` ?

- `try` : on exécute le code normal (lecture de la sonde).
- `except` : si une erreur survient (mauvais branchement, problème de communication SPI, etc.), le programme n'arrête pas brutalement : on affiche un message d'erreur et on continue.
- `finally` : cette partie est toujours exécutée, qu'il y ait une erreur ou non.
  Ici, elle sert à **libérer la broche CS (`cs.deinit()`)**, exactement comme on libère la sonde **DHT22 avec `.exit()`**.

Modification du script app.py

```
# ============================
# Importations
# ============================
from flask import Flask, render_template_string
from capteur import (
    lire_donnees_capteur,
    calculer_point_de_rosee,
    calculer_humidex,
    recuperer_date_heure,
    lire_pression,
    lire_humidite_sol
)

# ============================
# Application Flask
# ============================
app = Flask(__name__)

@app.route('/')
def index():
    humidity, temperature = lire_donnees_capteur()
    if humidity is not None and temperature is not None:
        point_de_rosee = calculer_point_de_rosee(temperature, humidity)
        humidex = calculer_humidex(temperature, point_de_rosee)
        date_heure = recuperer_date_heure()
        pression = lire_pression()
        voltage_sol, humidite_sol = lire_humidite_sol()
```

```
        html = f"""
        <h1>Données météo locales</h1>
        <ul>
            <li>Date et heure : {date_heure}</li>
            <li>Température : {temperature} °C</li>
            <li>Humidité : {humidity} %</li>
            <li>Point de rosée : {point_de_rosee} °C</li>
            <li>Humidex : {humidex}</li>
            <li>Pression atmosphérique : {pression} hPa</li>
        </ul>
        <h2>Humidité du sol de la sonde 1 :</h2>
        <ul>
            <li>Humidité du sol (tension) : {voltage_sol} V</li>
            <li>Humidité du sol (approx.) : {humidite_sol} %</li>
        </ul>
        """
    else:
        html = "<p>Erreur de lecture du capteur.</p>"

    return render_template_string(html)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Partie XV

Serveur web Apache2 et Dokuwiki

Qu’est-ce qu’Apache ?

Apache HTTP Server est l’un des serveurs web les plus utilisés au monde.
Un serveur web est un logiciel qui **écoute les requêtes HTTP** (venant d’un navigateur comme Firefox ou Chromium) et qui **renvoie des pages web**.

En d’autres termes, Apache est l’intermédiaire qui permet à vos fichiers (pages HTML, scripts PHP, wiki, etc.) d’être consultés via une simple adresse dans un navigateur :
`http://raspberry.local/wiki`

Pourquoi Apache est important pour DokuWiki ?

- **DokuWiki** est un moteur de wiki léger, qui fonctionne avec **PHP** mais **sans base de données**.
- Cela le rend particulièrement adapté à notre station autonome : pas besoin de MySQL ou MariaDB, juste du stockage en fichiers plats.
- Pour que PHP puisse fonctionner et afficher correctement les pages du wiki, il a besoin d’un serveur web comme **Apache**.

En résumé :

- Flask = pour la météo et les capteurs.
- Apache + PHP = pour héberger DokuWiki.

Les deux peuvent coexister sur le Raspberry Pi.

Installation d’Apache et PHP sur Raspberry Pi OS

Exécutez ces commandes :

```
sudo apt update
sudo apt install apache2 php libapache2-mod-php
```

- **Apache2** → le serveur web.
- **PHP** → le langage qui fait tourner DokuWiki.
- **libapache2-mod-php** → module qui permet à Apache d’interpréter le code PHP.

Installation de DokuWiki

Téléchargez et installez la dernière version stable de DokuWiki :

```
cd /var/www/html/
sudo wget https://download.dokuwiki.org/src/dokuwiki/dokuwiki-stable.tgz
sudo tar xvf dokuwiki-stable.tgz
sudo mv dokuwiki_nom_dossier wiki (Attention ici, le nom dépendra de la version de dokuwiki)
sudo chown -R www-data:www-data /var/www/html/wiki
sudo systemctl restart apache2
```

Accéder au wiki

Une fois installé, ouvrez un navigateur connecté au Wi-Fi de la station et rendez-vous sur :

`http://IP_RASPBERRY/wiki`

Vous aurez alors accès à l’interface d’installation de DokuWiki et pourrez commencer à créer la documentation du jardin.

Avantage majeur : tout fonctionne **hors-ligne**.
Les participant·es peuvent consulter et enrichir la documentation du jardin **même sans Internet**.