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)
```

Révision #19
Créé 2025-08-07 16:46:55 UTC
Mis à jour 2025-08-11 09:21:15 UTC