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)
```

```python
    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)
```

Révision #11
Créé 2025-08-11 15:03:32 UTC
Mis à jour 2025-09-11 12:16:42 UTC par Olivier Carpels