```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
'''Ce programme réalise le côté coordonnateur d'une communication I2C.

Version du programme ex_i2c_coord gérant deux (2) noeuds. La gestion
des exceptions/erreurs repose sur le style orienté-objet de Python.

Contexte
--=-=--=-
Dans cet exemple, les noeuds (Arduino) effectuent:
  1) la lecture de la température du CPU;
  2) incrémenter le numéro d'échantillon.
selon la période d'échantillonnage NEW_TS programmé par le coordonnateur (Pi).
À toutes les SAMPLING_TIME, le coordonnateur lit ces deux valeurs des noeuds
et les affiche à la sortie standard (terminal Python). On a NEW_TS != SAMPLE_TIME
pour montrer l'asynchronisme entre les noeuds et le coordonnateur.

(voir les notes de cours "I2C: Pi et Arduino")

GPA788 Conception et intégration des objets connectés
T. Wong
Juin 2018
Juillet 2020
'''

# ---------------------------------------------------------------------
# Les modules utiles
# ---------------------------------------------------------------------
import smbus            # pour la communication I2C
import time             # pour sleep()
import struct           # pour la conversion octet -> float
from datetime import datetime # pour l'horodatage des échantillons
from CoordI2C import CoordCommunication, CoordException
import constants as cst # constants du programme
import requests  #pour l'envoie des données avec thinkspeak

# ---------------------------------------------------------------------
# Structures pour la conversion des données reçues
# ---------------------------------------------------------------------
''' bytearray
    Liste d'octets qui servira à la conversion octets -> int (numéro d'échantillon)
    et octets -> float (températire interne de l'ATmega328P)

    dictionnaire de dictionnaires
    Structure servant à entreposer les valeurs (no. d'échantillon et tempéraure)
    de chaque des noeuds. Les dictionnaires permettent l'indexage numérique et
    alphanumérique des champs. Ainsi, on utilisera l'adresse I2C des noeuds
    comme l'index principal. Cela facilitera la programmation des accès aux données.
'''
#Décalaration des noeuds
NoeudTemp = CoordCommunication(cst.I2C_ADDRESS[0], smbus.SMBus(1))
NoeudSon = CoordCommunication(cst.I2C_ADDRESS[1], smbus.SMBus(1))

ListNoeud = [NoeudTemp, NoeudSon]


# -----------------------------------------------------
# Fonction principale
# -----------------------------------------------------
```

```python
def main():

  # Stocker les données reçues dans un dictionnaire:
  #    clés -> adresses I2C des noeuds
  # valeurs -> dictionnaires contenant deux chmaps 'Température' et 'Sample_Num'
  Sensor_Data = {
    cst.I2C_ADDRESS[0] : {'Temperature' : -1.0, 'Humidité' : 1.0, 'Sample_Num' : 0},
    cst.I2C_ADDRESS[1] : {'Intensité dB' : -1.0, 'Sample_Num' : 0}
  }

  # Bon. Indiquer que le coordonnateur est prêt...
  print("Coordonnateur (Pi) en marche avec Ts =", cst.SAMPLING_TIME, "sec.")
  print("ctrl-c pour terminer le programme.")

  # Instancier un objet de type SMBus et le lié au port i2c-1
  try:
    # 1) C'est une bonne pratique d'arrêter le noeud avant de
    #    lancer des commandes.
    print('Arrêter les noeuds.')
    for noeud in ListNoeud:
        noeud.send_stop()

    # 2) Régler le temps d'échantillonnage du noeud à NEW_TS secondes et régler nombre de Vrms et Li
    print("Assigner une nouvelle Ts =",  cst.NEW_TS, "au DHT11")
    print("Assigner des nouvelles valeurs Li =", cst.New_Li, "  et Vrms =", cst.NEW_Vrms, "au MAX4466")
    for noeud in ListNoeud:
      if noeud == NoeudTemp:
        noeud.send_Ts(cst.NEW_TS)
      elif noeud == NoeudSon:
        noeud.send_Nb_Vrms(cst.NEW_Vrms)
        noeud.send_Nb_Li(cst.New_Li)
      time.sleep(0.1)         # attendre avant de continuer l'écriture

    # 3) Ok. Demander au noeud de démarrer/continuer son échantillonnage
    print("Demander aux noeuds de démarrer l'échantillonnage.")
    for noeud in ListNoeud:
        noeud.send_go()

    # 4) Le coordonnateur demande et reçoit des données du noeud
    #    jusqu'à ce que l'utilisateur arrête le programme par ctrl-c.
    while True:         # boucle infinie
      # 4.1) attendre la fin de la période d'échantillonnage
      #      du coordonateur
      time.sleep(cst.SAMPLING_TIME)

      #-------------------------------------------------------------------------
      # Gestion de nos fonctions utilitaire:
      #       - Permet de définir une heure pour mettre le système en pause
      #          et le redémarre automatiquement lorsque la pause est terminé.
      #-------------------------------------------------------------------------
      if (datetime.now().minute == None) & (datetime.now().hour == None): # Veuiller changer les valeurs 'None' pour l'heure et la minute que vous désirez faire une pause
        print('PAUSE DEMANDÉ')
        for noeud in ListNoeud:
          noeud.send_Pause()

        while True:
```

```
116           if (datetime.now().minute == None) & (datetime.now().hour == None): # Veuiller changer les valeurs 'None' pour l'heure et la minute que vous désirez redémarrer le système
117             for noeud in ListNoeud:
118               noeud.send_Restart()
119             break
120
121
122       # 4.2) Lire la température, humidité et l'intensité des noeuds.
123       for adr in cst.I2C_ADDRESS:
124           if adr == cst.I2C_ADDRESS[0]:
125             Sensor_Data[adr]['Temperature'] = ListNoeud[0].read_Value(cst.I2C_NODE_TEMP_LSB0)
126             Sensor_Data[adr]['Humidité'] = ListNoeud[0].read_Value(cst.I2C_NODE_HUM_LSB0)
127           elif adr == cst.I2C_ADDRESS[1]:
128             Sensor_Data[adr]['Intensité dB'] = ListNoeud[1].read_Value(cst.I2C_NODE_TEMP_LSB0)
129
130
131       # 4.3) Lire le temps local comme l'horodatage (timestamp)
132       #      N'oubliez pas de régler le temps du Pi s'il n'est pas relié au réseau.
133       temps = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
134
135       # 4.4) Lire le nombre d'échantillon acquis par le noeud
136       for adr in cst.I2C_ADDRESS:
137         if adr == cst.I2C_ADDRESS[0]:
138           Sensor_Data[adr]['Sample_Num'] = ListNoeud[0].read_SNumber()
139         elif adr == cst.I2C_ADDRESS[1]:
140           Sensor_Data[adr]['Sample_Num'] = ListNoeud[1].read_SNumber()
141
142
143
144       # 4.5) Afficher les données reçues à la sortie standard
145       print("\n<Temps: ", temps, ">")
146       for adr in cst.I2C_ADDRESS:
147         if adr == cst.I2C_ADDRESS[0]:
148           print("Noeud: {0}, Échantillon: {1}, Température: {2:.2f}, Humidité: {3:.2f}".format(hex(adr),
149           Sensor_Data[adr]['Sample_Num'], Sensor_Data[adr]['Temperature'], Sensor_Data[adr]['Humidité']))
150
151         if adr == cst.I2C_ADDRESS[1]:
152           print("Noeud: {0}, Échantillon: {1}, Intensité dB: {2:.2f}".format(hex(adr),
153           Sensor_Data[adr]['Sample_Num'], Sensor_Data[adr]['Intensité dB']))
154
155     #REST
156       try:
157         print("Écrire {0}, {1} et {2:.2f} dans les champs...".format(Sensor_Data[cst.I2C_ADDRESS[0]]['Temperature'],
158         Sensor_Data[cst.I2C_ADDRESS[0]]['Humidité'], Sensor_Data[cst.I2C_ADDRESS[1]]['Intensité dB']) )
159         resp = requests.get(cst.THINGSPK_URL,
160                             # 10 secondes pour connection et read timeout
161                             timeout = (10, 10),
162                             # Paramètres de cette requête
163                             params = { "api_key" : cst.THINGSPK_API_KEY,
164                                        "field1"  : Sensor_Data[cst.I2C_ADDRESS[0]]['Temperature'],
```

```
                                    "field2"  : Sensor_Data[cst.I2C_ADDRESS[0]]['Humidité'],
                                    "field3"  : Sensor_Data[cst.I2C_ADDRESS[1]]['Intensité dB']}
                            )

        print(f"ThingSpeak GET response: {resp.status_code}")
        # Vérifier la réponse de ThingSpeak
        if resp.status_code != 200:
          print("Erreur de communication détectée!")

        # Attendre 20 secondes (licence gratuite a un délai de 15 secondes)
        time.sleep(cst.DELAY)

      except requests.ConnectionError:
        print('Erreur de connexion')
        return
      except requests.Timeout:
        print("Exception de timeout reçue (connexion ou écriture)")
        return
      except requests.HTTPError:
        print('Erreur au niveau du protocole HTTP')
        return

  except IOError as io_e:
    print("Erreur détectée sur le bus i2c.")
    print("Message d'erreur: ", io_e)
  except struct.error as conv_e:
    print("Erreur détectée lors de la conversion des données.")
    print("Message d'erreur: ", conv_e)
  except CoordException as ce:
    print("Problème détecté dans l'utilisation des fonctions.")
    print(F"Message d'erreur: {ce}")


#
# Il faut aussi gérer les autres exceptions!
#


# -----------------------------------------------------
# Programme principal
# -----------------------------------------------------
if __name__ == '__main__':
  main()
```