```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
import constants as cst # constants du programme

# ---------------------------------------------------------------------
# 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.
'''
# ---------------------------------------------------------------------
# Classe d'exception
# ---------------------------------------------------------------------
class CoordException(Exception):
  '''Cette classe représente les exceptions en lien avec l'utilisation
     des fonctions de ce programme.

  Un objet de cette classe d'exception est envoyée au programme principal
  par les fonctions. Elle représente les problèmes détectés l'utilisation
  du bus I2C et les bornes fonctions MAIS pas les erreurs de communication I2C.
  '''
  def __init__(self, *args):
```

```
60      if args:
61        self.message = args[0]
62      else:
63        self.message = None
64        super().__init__(self.message)
65
66    def __str__(self):
67      if self.message:
68        return self.message
69      else:
70        return F"Exception CoordException a été lancée."
71
72
73
74  # ---------------------------------------------------------------
75  # Classe de communication
76  # ---------------------------------------------------------------
77  class CoordCommunication():
78
79    def __init__(self, address = -1, bus = None):
80
81      self.address = address
82      self.bus = bus
83
84    # -------------------------------------------------------
85    # Fonctions utilisateurs
86    # -------------------------------------------------------
87    def send_stop(self):
88      '''Envoyer la commande Arrêt sur le bus i2c au noeud à l'adresse adr.
89
90      Paramètres:
91      bus -- objet SMBUS déjà initialisé
92      adr (int) -- adresse du noeud destinataire
93
94      Retour: n/a
95      Exceptions possibles: CoordException, IOError
96      '''
97      if self.bus != None and (self.address >= cst.I2C_MIN_ADR and self.address <=
    cst.I2C_MAX_ADR):
98        self.bus.write_byte(self.address, cst.I2C_CMD_SET_STOP)
99      else:
100       raise CoordException(F"<send_stop> Bus non initié ou adresse I2C invalide.")
101
102   def send_go(self):
103     '''Envoyer la commande démarrage sur le bus i2c au noeud à l'adresse adr.
104
105     Arguments:
106     bus -- objet SMBUS déjà initialisé
107     adr (int) -- adresse du noeud destinataire
108
109     Retour: n/a
110     Exceptions possibles: CoordException, IOError
111     '''
112     if self.bus != None and (self.address >= cst.I2C_MIN_ADR and self.address <=
    cst.I2C_MAX_ADR):
113       self.bus.write_byte(self.address, cst.I2C_CMD_SET_GO)
114     else:
115       raise CoordException(F"<send_go> Bus non initié ou adresse I2C invalide.")
116
117   def send_Ts(self, ts = -1):
```

```
118     '''Envoyer la commande pour le changement de période d'échantillonnage
119     et sa nouvelle valeur.
120 
121     Arguments:
122     bus -- objet SMBUS déjà initialisé
123     adr (int) -- adresse du noeud destinataire
124     ts (int) -- nouvelle période d'échantillonnage
125 
126     Retour: n/a
127     Exceptions possibles: CoordException, IOError
128     '''
129     if self.bus != None and (self.address >= cst.I2C_MIN_ADR and self.address <=
cst.I2C_MAX_ADR):
130       # Note: la fonction write_i2c_block_data transmet une commande
131       #       et des données en bloc. Les données doivent être dans
132       #       une liste même si elle n'a qu'une seule données d'où
133       #       [ts] avec des crochets.
134       self.bus.write_i2c_block_data(self.address, cst.I2C_CMD_SET_TS, [ts])
135     else:
136       raise CoordException(F"<send_Ts> Bus non initié ou adresse I2C invalide.")
137 
138   def send_Nb_Vrms(self, nb_Vrms =-1):
139     '''Envoyer le nombre de Vrms au noeud controlant le MAX4466.
140 
141     Paramètres:
142     bus -- objet SMBUS déjà initialisé
143     adr (int) -- adresse du noeud destinataire
144     nb_Vrms -- nombre de Vrms pour configurer le noeud du MAX4466
145 
146     Retour: n/a
147     Exceptions possibles: CoordException, IOError
148     '''
149     if self.bus != None and (self.address >= cst.I2C_MIN_ADR and self.address <=
cst.I2C_MAX_ADR):
150       # Note: la fonction write_i2c_block_data transmet une commande
151       #       et des données en bloc. Les données doivent être dans
152       #       une liste même si elle n'a qu'une seule données d'où
153       #       [ts] avec des crochets.
154       self.bus.write_i2c_block_data(self.address, cst.I2C_CMD_SET_NB_VRMS,
[nb_Vrms])
155     else:
156       raise CoordException(F"<send_Nb_Vrms> Bus non initié ou adresse I2C
invalide.")
157 
158   def send_Nb_Li(self, nb_Li =-1):
159     '''Envoyer le nombre de Li au noeud controlant le MAX4466.
160 
161     Paramètres:
162     bus -- objet SMBUS déjà initialisé
163     adr (int) -- adresse du noeud destinataire
164     nb_Li -- nombre de Li pour configurer le noeud du MAX4466
165 
166     Retour: n/a
167     Exceptions possibles: CoordException, IOError
168     '''
169 
170     if self.bus != None and (self.address >= cst.I2C_MIN_ADR and self.address <=
cst.I2C_MAX_ADR):
171       # Note: la fonction write_i2c_block_data transmet une commande
172       #       et des données en bloc. Les données doivent être dans
```

```
      #       une liste même si elle n'a qu'une seule données d'où
      #       [ts] avec des crochets.
      self.bus.write_i2c_block_data(self.address, cst.I2C_CMD_SET_NB_LI, [nb_Li])
    else:
      raise CoordException(F"<send_Nb_Li> Bus non initié ou adresse I2C invalide.")

  def send_Restart(self):
    '''Envoyer une commande de redémarrage aux noeuds après une pause.

    Paramètres:
    bus -- objet SMBUS déjà initialisé
    adr (int) -- adresse du noeud destinataire

    Retour: n/a
    Exceptions possibles: CoordException, IOError
    '''

    if self.bus != None and (self.address >= cst.I2C_MIN_ADR and self.address <=
cst.I2C_MAX_ADR):
      # Note: la fonction write_i2c_block_data transmet une commande
      #       et des données en bloc. Les données doivent être dans
      #       une liste même si elle n'a qu'une seule données d'où
      #       [ts] avec des crochets.
      self.bus.write_byte(self.address, cst.I2C_CMD_SET_RESTART)
    else:
      raise CoordException(F"<send_Restart> Bus non initié ou adresse I2C
invalide.")

  def send_Pause(self):
    '''Envoyer une pause aux noeuds pour arrêter l'accumulation des données.

    Paramètres:
    bus -- objet SMBUS déjà initialisé
    adr (int) -- adresse du noeud destinataire

    Retour: n/a
    Exceptions possibles: CoordException, IOError
    '''
    if self.bus != None and (self.address >= cst.I2C_MIN_ADR and self.address <=
cst.I2C_MAX_ADR):
      # Note: la fonction write_i2c_block_data transmet une commande
      #       et des données en bloc. Les données doivent être dans
      #       une liste même si elle n'a qu'une seule données d'où
      #       [ts] avec des crochets.

      self.bus.write_byte(self.address, cst.I2C_CMD_SET_PAUSE)

    else:
      raise CoordException(F"<send_Pause> Bus non initié ou adresse I2C invalide.")


  def read_Value(self, startByte = 0):
    '''Lire les valeurs des noeuds à l'adresse adr.

    Arguments:
    bus -- objet SMBUS déjà initialisé
    adr (int) -- adresse du noeud destinataire

    Retour (float): Valeur de températeur, humidité ou intensité
    Exceptions possibles: CoordException, IOError, struct.error
```

```
230     '''
231     if self.bus != None and (self.address >= cst.I2C_MIN_ADR and self.address <=
    cst.I2C_MAX_ADR):
232       # On fait la lecture de la température octet par octet.
233       # MSB -> adresse haute, LSB -> adresse basse
234       # (c'est l'ordre little Endian)
235       T = bytearray([0x00, 0x00, 0x00, 0x00])
236       #if startByte == 3:
237       self.bus.write_byte(self.address, startByte + 3)
238       T[3] = self.bus.read_byte(self.address)
239       self.bus.write_byte(self.address, startByte + 2)
240       T[2] = self.bus.read_byte(self.address)
241       self.bus.write_byte(self.address, startByte + 1 )
242       T[1] = self.bus.read_byte(self.address)
243       self.bus.write_byte(self.address, startByte)
244       T[0] = self.bus.read_byte(self.address)
245 
246       # Convertir les 4 octets représentant la température de format binary32
247       # en nombre virgule flottante
248       value = struct.unpack('f', T)
249       # Note: la conversion par struct produit des listes d'objets dans nb_echan
250       #       et temperature. C'est pourquoi on prend uniquement le premier élément
251       #       avec la syntaxe [0].
252       return round(value[0],2)
253     else:
254       raise CoordException(F"<read_Temp> Bus non initié ou adresse I2C invalide.")
255 
256   def read_SNumber(self):
257     '''Lire le numéro de l'échantillon de la température du noeud à l'adresse adr.
258 
259     Arguments:
260     bus -- objet SMBUS déjà initialisé
261     adr (int) -- adresse du noeud destinataire
262 
263     Retour (int): Numéro de l'échantillon
264     Exceptions possibles: CoordException, IOError, struct.error
265     '''
266     if self.bus != None and (self.address >= cst.I2C_MIN_ADR and self.address <=
    cst.I2C_MAX_ADR):
267       # Lire le numéro d'échantillon acquis par le noeud depuis son démarrage.
268       # On fait cette lecture octet par octet (ordre little Endian)
269       N = bytearray([0x00, 0x00])
270       self.bus.write_byte(self.address, cst.I2C_NODE_NS_MSB)
271       N[1] = self.bus.read_byte(self.address)
272       self.bus.write_byte(self.address, cst.I2C_NODE_NS_LSB)
273       N[0] = self.bus.read_byte(self.address)
274       # Convertir les 2 octets du numéro d'échantillon en valeur entière
275       nb_echan = struct.unpack('H', N)
276       # Note: la conversion par struct produit des listes d'objets dans nb_echan
277       #       et temperature. C'est pourquoi on prend uniquement le premier élément
278       #       avec la syntaxe [0].
279       return nb_echan[0]
280     else:
281       raise CoordException(F"<read_SNumber> Bus non initié ou adresse I2C
    invalide.")
282 
283 
284 
```