```python
#!/usr/bin/python3

#Para obter Data/Hora
import datetime

#SENSOR SCD30
import struct
import smbus2
from smbus2 import i2c_msg
import numpy as np
import time
from collections import namedtuple

#SENSORES MQ
#import smbus
#import time

#SENSOR CCS811
#import time
import board
import busio
import adafruit_ccs811

#SENSOR SGP30
#import time
#import board
#import busio
import adafruit_sgp30

#SENSOR TSL2591
#import time
#import board
#import busio
import adafruit_tsl2591

class SCD30():
    def __init__(self, measurement_intervalx, auto_calibrationx, pressurex):
        super(SCD30, self).__init__()

        self.i2c_address = 0x61
        self.bus_number = smbus2.SMBus(1)

        self.set_measurement_interval(measurement_intervalx)
        time.sleep(0.1)
        self.set_auto_calibration(auto_calibrationx)
        time.sleep(0.1)
        self.start_periodic_measurement(pressurex)
        time.sleep(0.1)

    # Set measurement interval
    def set_measurement_interval(self, interval):
        try:
            cmd_set_measurement_interval = self.create_cmd(interval, 0x00)
            #print ("cmd_set_measurement_interval=%s"%cmd_set_measurement_interval)
            self.bus_number.write_i2c_block_data(self.i2c_address, 0X46, cmd_set_measurement_interval)
        except OSError as e:
            print("set_measurement_interval: " + str(e))

    # Start periodic measurement
    def start_periodic_measurement(self, pressure):
        try:
            cmd_start_periodic_measurement = self.create_cmd(pressure, 0x10)
            #print ("cmd_start_periodic_measurement=%s"%cmd_start_periodic_measurement)
            self.bus_number.write_i2c_block_data(self.i2c_address, 0X00, cmd_start_periodic_measurement)
            time.sleep(0.1)
            #print("%7.2f hPa 2" % pressure)
        except OSError as e:
            print("start_periodic_measurement: " + str(e))

    # Set auto calibration
    def set_auto_calibration(self, on_off):
        try:
            cmd_set_auto_calibration = self.create_cmd(on_off, 0x06)
            #print ("cmd_set_auto_calibration=%s"%cmd_set_auto_calibration)
            self.bus_number.write_i2c_block_data(self.i2c_address, 0x53, cmd_set_auto_calibration)
        except OSError as e:
            print("set_auto_calibration: " + str(e))

    # Get Data ready
    def data_ready(self):
        try:
            self.bus_number.write_byte_data(self.i2c_address, 0x02, 0x02)
            time.sleep(0.1)
            data_ready = i2c_msg.read(self.i2c_address, 3)
            time.sleep(0.1)
            self.bus_number.i2c_rdwr(data_ready)

            return bool(np.array(list(data_ready))[1])
        except OSError as e:
            print("OSError in scd30 data_ready: " + str(e))
            return False

    # Read measurement buffer
    def get_scd30_measurements(self, ):
        try:
            self.bus_number.write_byte_data(self.i2c_address, 0x03, 0x00)
            time.sleep(0.1)

            measurement = i2c_msg.read(self.i2c_address, 18)
            self.bus_number.i2c_rdwr(measurement)

            co2_list = [0, 1, 2, 3, 4, 5]
            co2_measurement = self.get_measurement(measurement, co2_list)
            #print("co2_measurement=%s"%co2_measurement)

            temperature_list = [6, 7, 8, 9, 10, 11]
            temperature_measurement = self.get_measurement(measurement, temperature_list)
            #print("temperature_measurement=%s"%temperature_measurement)

            humidity_list = [12, 13, 14, 15, 16, 17]
            humidity_measurement = self.get_measurement(measurement, humidity_list)
            #print("humidity_measurement=%s"%humidity_measurement)

            if co2_measurement and humidity_measurement and temperature_measurement:
                return self.to_name_tuple(co2_measurement, humidity_measurement, temperature_measurement)
            else:
                return None

        except OSError as e:
            print("OSError in scd30 get_scd30_measurements: " + str(e))
            return None

    def extract_measurement(self, i, measurement):
        return np.array(list(measurement))[i]

    def pack_struct(self, i, i1, i2, i3, measurement):
        return struct.pack('4B', self.extract_measurement(i, measurement), self.extract_measurement(i1, measurement),
                           self.extract_measurement(i2, measurement), self.extract_measurement(i3, measurement))

    def crc8_update(self, b, crc):
        crc = crc ^ b
        for i in range(8):
            if (crc & 0x80) == 0x80:
                crc = (crc << 1) ^ 0x131
            else:
                crc <<= 1
        return crc


    def calc_crc(self, d):
        crc = self.crc8_update((d >> 8) & 0x00FF, 0xff)
        crc = self.crc8_update((d & 0x00FF), crc)
        return crc

    def get_crc(self, i, j, measurement):
        crc = self.crc8_update(np.array(list(measurement))[i], 0xff)
        crc = self.crc8_update(np.array(list(measurement))[j], crc)
        return crc

    def get_measurement(self, measurement, val_list):
        crc = self.get_crc(val_list[0], val_list[1], measurement)
        unpacked_measurement = None

        if crc == self.extract_measurement(val_list[2], measurement):
            crc = self.get_crc(val_list[3], val_list[4], measurement)
            if crc == self.extract_measurement(val_list[5], measurement):
                tmp = self.pack_struct(val_list[0], val_list[1], val_list[3], val_list[4], measurement)
                unpacked_measurement = struct.unpack('>f', tmp)
        return unpacked_measurement

    def create_cmd(self, payload, x_):
        ff = 0x00FF
        i = 8
        return [x_, (payload >> i) & ff, (payload & ff), self.calc_crc(payload)]

    def to_name_tuple(self, co2_measurement, humidity_measurement, temperature_measurement):
        Data = namedtuple('Data', ['CO2', 'temperature', 'humidity'])
        return Data(co2_measurement[0], temperature_measurement[0], humidity_measurement[0])

#SENSORES MQ
class MQ():
    def __init__(self, endereco):
        super(MQ, self).__init__()
        self.i2c_address = endereco
        self.bus_number = smbus2.SMBus(1)

    def valorDigital(self):
        data = self.bus_number.read_i2c_block_data(self.i2c_address, 0x00, 2)
        raw_adc = (data[0] & 0x0F) * 256 + data[1]
        return raw_adc

#SENSOR SCD30
scd30 = SCD30(5, 0, 1013)
dados_scd30 = None

#SENSORES MQ
mq9 = MQ(0x50)
mq5 = MQ(0x52)
mq3 = MQ(0x51)
mq2 = MQ(0x54)

#SENSOR CCS811
i2c = busio.I2C(board.SCL, board.SDA)
ccs811 = adafruit_ccs811.CCS811(i2c)

#SENSOR SGP30
#i2c = busio.I2C(board.SCL, board.SDA)
# Create library object on our I2C port
sgp30 = adafruit_sgp30.Adafruit_SGP30(i2c)
sgp30.iaq_init()
sgp30.set_iaq_baseline(0x8973, 0x8aae)
sgp30_elapsed_sec = 0

#SENSOR TSL2591
# Initialize the I2C bus.
```

```
#i2c = busio.I2C(board.SCL, board.SDA)
# Initialize the sensor.
tsl2591 = adafruit_tsl2591.TSL2591(i2c)

while True:
    try:
        #SENSOR SCD30
        if scd30.data_ready():
            dados_scd30=scd30.get_scd30_measurements()
        time.sleep(1)
        #SENSORES MQ
        mq9_val = mq9.valorDigital()
        time.sleep(0.5)
        mq5_val = mq5.valorDigital()
        time.sleep(0.5)
        mq3_val = mq3.valorDigital()
        time.sleep(0.5)
        mq2_val = mq2.valorDigital()
        time.sleep(1)
        #SENSOR CCS811
        # Wait for the sensor to be ready
        #while not ccs811.data_ready:
            #print ("PASS")
        #    pass
        if ccs811.data_ready:
            ccs811_eco2 = ccs811.eco2
            ccs811_tvoc = ccs811.tvoc
        else:
            ccs811_eco2 = None
            ccs811_tvoc = None
        time.sleep(1)
        #SENSOR SGP30
        sgp30_eCO2 = sgp30.eCO2
        sgp30_TVOC = sgp30.TVOC
        #print("eCO2 = %d ppm \t TVOC = %d ppb" % (sgp30.eCO2, sgp30.TVOC))
        time.sleep(1)
        sgp30_elapsed_sec += 1
        if sgp30_elapsed_sec > 10:
            sgp30_elapsed_sec = 0
            print("**** [SGP30] Baseline values: eCO2 = 0x%x, TVOC = 0x%x ****" % (
            sgp30.baseline_eCO2, sgp30.baseline_TVOC))
        #SENSOR TSL2591
        tsl2591_lux = tsl2591.lux
        # Infrared levels range from 0-65535 (16-bit)
        tsl2591_infrared = tsl2591.infrared
        # Visible-only levels range from 0-2147483647 (32-bit)
        tsl2591_visible = tsl2591.visible
        # Full spectrum (visible + IR) also range from 0-2147483647 (32-bit)
        tsl2591_full_spectrum = tsl2591.full_spectrum
        time.sleep(1)

        #Separador
        print ("-"*90) #desenhar linha
    except (KeyboardInterrupt, SystemExit):
        print("Stopped")
        exit("Tchau")
    except:
        print("Erro sensores!")
        time.sleep(5)
    else:
        #Data/hora
        x = datetime.datetime.now()
        datahora = x.strftime("%x %X")
        print("DataHora, %s" % (datahora))
        #SENSOR SCD30
        if dados_scd30 is not None:
            print("SCD30, CO2: %d ppm, Temperatura: %0.1f °C, Humidade: %0.1f %%"%(
            dados_scd30[0],dados_scd30[1],dados_scd30[2]))
        #SENSORES MQ
        print ("MQ9, Monoxido de carbono/LPG/CH4: %.2f ppm" % (mq9_val*1980/4096 + 20
        )) #*1000/4096 + 10
        print ("MQ5, LPG/Gas natural: %.2f ppm" % (mq5_val*9800/4096 + 200))
        #*1000/4096 + 10)
        print ("MQ3, Vapor de alcool: %.2f mg/l" % (mq3_val*9.95/4096+0.05)) #*9.95/4096
        print ("MQ2, Gas combustivel/fumo: %.2f ppm"% (mq2_val*4800/4096 + 200))
        #SENSOR CCS811
        if (ccs811_eco2 is None): eco2 = 0.0
        if (ccs811_tvoc is None): tvoc = 0.0
        print("CCS811, eCO2: %s ppm, TVOC: %s ppb"%(ccs811_eco2, ccs811_tvoc))
        #SENSOR SGP30
        print("SGP30, eCO2: %d ppm, TVOC: %d ppb" % (sgp30_eCO2, sgp30_TVOC))
        #SENSOR TSL2591
        print("TSL2591, Luz total: %d lux, Luz infravermelha: %d, Luz visivel: %d, Espectro completo: %d"%(tsl2591_lux,tsl2591_infrared,tsl2591_visible, tsl2591_full_spectrum))
        #print("Luz total: %d lux"%tsl2591_lux)
        #print('Luz infravermelha: %d'%tsl2591_infrared)
        #print('Luz visivel: %d'%tsl2591_visible)
        #print('Espectro completo (Infravermelha + visivel): %d'%tsl2591_full_spectrum)
```