```python
"""Gas detection for Raspberry Pi using ADS1x15 and MQ-2 sensors."""

# pylint: disable=C0103
# pylint: disable=R0201

import math
import time

import board
import busio

from adafruit_ads1x15.analog_in import AnalogIn
from adafruit_ads1x15 import ads1015
from adafruit_ads1x15 import ads1115

# Convertors
ADS1015 = ads1015.ADS1015
ADS1115 = ads1115.ADS1115

# Pins
P0 = 0
P1 = 1
P2 = 2
P3 = 3

# Address of convertor
ADDRESS = 0x48

class GasDetection:
    """Gas detection class."""

    # Load resistance on the board in kilo ohms
    LOAD_RESISTANCE = 5

    # Clean air factor from the chart in the datasheet
    CLEAN_AIR_FACTOR = 9.6

    # Identification of gases
    CO_GAS = 0
    H2_GAS = 1
    CH4_GAS = 2
    LPG_GAS = 3
    PROPANE_GAS = 4
    ALCOHOL_GAS = 5
    SMOKE_GAS = 6

    # Calculated logarithm for the left point and
    # slope from the curve, using ten logarithm
    # and the two-point form
    #
    # More details about calculating:
    # https://tutorials-raspberrypi.com/configure-and-read-out-the-raspberry-pi-gas-sensor-mq-
    CO_CURVE = [2.30775, 0.71569, -0.33539]
    H2_CURVE = [2.30776, 0.71895, -0.33539]
    CH4_CURVE = [2.30987, 0.48693, -0.37459]
    LPG_CURVE = [2.30481, 0.20588, -0.46621]
    PROPANE_CURVE = [2.30366, 0.23203, -0.46202]
    ALCOHOL_CURVE = [2.30704, 0.45752, -0.37398]
    SMOKE_CURVE = [2.30724, 0.53268, -0.44082]

    # Number of samples and time between them in miliseconds for calibration
    CALIBARAION_SAMPLE_NUMBER = 50
    CALIBRATION_SAMPLE_INTERVAL = 500

    # Number of samples and time between them in miliseconds for reading
    READ_SAMPLE_NUMBER = 5
    READ_SAMPLE_INTERVAL = 50

    # Analog to digital channel
    channel = None

    # Ro value of the sensor
    ro = None

    def __init__(self, convertor=ADS1115, pin=P0, address=ADDRESS, ro=None):
```

```
        Initialize the class.

        Input:
        -- convertor -- Convertor to use. Must be one of ADS1x15. Default is ADS1115.
        -- pin -- Pin of ADC convertor to use. Must be one of supported pins. Default is P0.
        -- address -- Address of ADC convertor. Must be one of I2C addresses. Default is 0x48
        -- ro -- Ro value of the sensor. Must be valid ro value. Default is to calibrate it.
        """

        i2c = busio.I2C(board.SCL, board.SDA)
        adc = convertor(i2c=i2c, address=address)

        self.channel = AnalogIn(adc, pin)

        if ro:
            self.ro = ro
        else:
            self.ro = self.calibrate()

    def __read(self, number=None, interval=None):
        """
        Read sensor resistence from ADC voltage.

        This function uses :func:`~gas_detection.GasDetection.__calculate_resistance` to
        caculate the sensor resistence (Rs). The resistence changes as the sensor is in the
        different consentration of the target gas.

        Input:
        -- number -- Number of samples. Default is 5.
        -- interval -- Time between samples in miliseconds. Default is 50.

        Output:
        -- Sensor resistence.
        """

        number = number if number else self.READ_SAMPLE_NUMBER
        interval = interval if interval else self.READ_SAMPLE_INTERVAL

        rs = 0

        for _ in range(number):
            rs += self.__calculate_resistance(self.channel.voltage)
            time.sleep(interval / 1000)

        rs = rs / number

        return rs

    def __calculate_resistance(self, voltage, resistance=None):
        """
        Calculate sensor resistence from ADC voltage.

        The sensor and the load resistor forms a voltage divider. Given the voltage
        across the load resistor and its resistance, the resistance of the sensor
        could be derived.

        Input:
        -- voltage -- Voltage from ADC convertor.
        -- resistance -- Load resistance on the board in kilo ohms. Default is 5.

        Output:
        -- Calculated sensor resistance.
        """

        resistance = resistance if resistance else self.LOAD_RESISTANCE

        return float(resistance * (1023.0 - voltage) / float(voltage))

    def __calculate_percentage(self, ratio, curve):
        """
        Calculate percentage from gas curve.

        Calculate percentage from gas curve by using the slope and a point of the
        line. The x (logarithmic value of ppm) of the line could be derived if y
        (`ratio`) is provided. As it is a logarithmic coordinate, power of 10 is
        used to convert the result to non-logarithmic value.

        Input:
```

```
        -- ratio -- Rs divided by Ro.
        -- curve -- The curve of the target gas.

        Output:
        -- Percentage of the target gas in ppm.
        """

        return math.pow(
            10,
            ((math.log(ratio) - curve[1]) / curve[2]) + curve[0]
        )

    def __calculate_gas_percentage(self, ratio, gas):
        """
        https://github.com/tutRPi/Raspberry-Pi-Gas-Sensor-MQ/blob/master/mq.py#L120

        Get percentage of the target gas.

        This function uses :func:`~gas_detection.GasDetection.__calculate_percentage` to
        calculate percentage in ppm (parts per million) of the target gas by it's curve.

        Input:
        -- ratio -- Rs divided by Ro.
        -- gas -- Identification of the target gas.

        Output:
        -- Percentage of the target gas in ppm.
        """

        if gas == self.CO_GAS:
            ppm = self.__calculate_percentage(ratio, self.CO_CURVE)
        elif gas == self.H2_GAS:
            ppm = self.__calculate_percentage(ratio, self.H2_CURVE)
        elif gas == self.CH4_GAS:
            ppm = self.__calculate_percentage(ratio, self.CH4_CURVE)
        elif gas == self.LPG_GAS:
            ppm = self.__calculate_percentage(ratio, self.LPG_CURVE)
        elif gas == self.PROPANE_GAS:
            ppm = self.__calculate_percentage(ratio, self.PROPANE_CURVE)
        elif gas == self.ALCOHOL_GAS:
            ppm = self.__calculate_percentage(ratio, self.ALCOHOL_CURVE)
        elif gas == self.SMOKE_GAS:
            ppm = self.__calculate_percentage(ratio, self.SMOKE_CURVE)
        else:
            ppm = 0

        return ppm

    def calibrate(self, number=None, interval=None, factor=None):
        """
        Calibrate sensor.

        This function assumes that the sensor is in clean air. It uses
        :func:`~gas_detection.GasDetection.__calculate_resistance` to
        caculate the sensor resistence (Rs) and divide it by clean
        air factor.

        Input:
        -- number -- Number of samples. Default is 50.
        -- interval -- Time between samples in miliseconds. Default is 500.
        -- factor -- The clean air factor. Default is 9.6.

        Output:
        -- The ro value of sensor.
        """

        number = number if number else self.CALIBARAION_SAMPLE_NUMBER
        interval = interval if interval else self.CALIBRATION_SAMPLE_INTERVAL
        factor = factor if factor else self.CLEAN_AIR_FACTOR

        rs = 0

        for _ in range(number):
            rs += self.__calculate_resistance(self.channel.voltage)
            time.sleep(interval / 1000)

        rs = rs / number
```

```
        :func:`~gas_detection.GasDetection.__calculate_resistance` to
        caculate the sensor resistence (Rs) and divide it by clean
        air factor.

        Input:
        -- number -- Number of samples. Default is 50.
        -- interval -- Time between samples in miliseconds. Default is 500.
        -- factor -- The clean air factor. Default is 9.6.

        Output:
        -- The ro value of sensor.
        """

        number = number if number else self.CALIBARAION_SAMPLE_NUMBER
        interval = interval if interval else self.CALIBRATION_SAMPLE_INTERVAL
        factor = factor if factor else self.CLEAN_AIR_FACTOR

        rs = 0

        for _ in range(number):
            rs += self.__calculate_resistance(self.channel.voltage)
            time.sleep(interval / 1000)

        rs = rs / number

        return rs / factor

    def percentage(self):
        """
        Get gas percentage of gases.

        This function uses :func:`~gas_detection.GasDetection.__calculate_gas_percentage` to
        the percentage of supported gases in ppm (parts per million).

        Output:
        -- Gas percentage of supported gases in ppm.
        """

        resistence = self.__read()
        ppm = {}

        ppm[self.CO_GAS] = self.__calculate_gas_percentage(
            resistence / self.ro, self.CO_GAS
        )

        ppm[self.H2_GAS] = self.__calculate_gas_percentage(
            resistence / self.ro, self.H2_GAS
        )

        ppm[self.CH4_GAS] = self.__calculate_gas_percentage(
            resistence / self.ro, self.CH4_GAS
        )

        ppm[self.LPG_GAS] = self.__calculate_gas_percentage(
            resistence / self.ro, self.LPG_GAS)

        ppm[self.PROPANE_GAS] = self.__calculate_gas_percentage(
            resistence / self.ro, self.PROPANE_GAS
        )

        ppm[self.ALCOHOL_GAS] = self.__calculate_gas_percentage(
            resistence / self.ro, self.ALCOHOL_GAS
        )

        ppm[self.SMOKE_GAS] = self.__calculate_gas_percentage(
            resistence / self.ro, self.SMOKE_GAS
        )

        return ppm
```