```
# FEB_report_fun.py

# Author: Luca Ghislotti
# Version: 1.7.1

import re
import math
from unittest import case
from matplotlib import lines
import os
import textwrap
from fpdf import FPDF
import csv
import re
from datetime import date

# configuration import
def read_config_file():
    counter = 0
    lines = []

    nation_letter = ""
    doc_version = ""
    data = ""
    author = ""
    nation_word = ""

    # acquire configuration data from config.conf file
    with open("../configuration/config.conf") as f:
        lines = f.readlines()

        # nation letter (I or U)
        nation_letter = re.search(
            "nation_letter = '(.*?)'        # nationality letter identifier",
            lines[0],
        ).group(1)

        # document version
        doc_version = re.search(
            "doc_version = '(.*?)'        # document version",
            lines[1],
        ).group(1)

        # date (if left empyt, date is set to today)
        data = re.search(
            "date = '(.*?)'        # if empty date is set to current date",
            lines[2],
        ).group(1)

        # date formatting
        if data == "":
            today = date.today()
            today = today.strftime("%d.%m.%Y")
            data = today

        # author (N. Surname)
        author = re.search(
            "author = '(.*?)'     # report author",
            lines[3],
        ).group(1)

        # nation word (Italian or US)
        nation_word = re.search(
            "nation_word = '(.*?)'     # nationality identifier word",
            lines[4],
        ).group(1)

        counter = counter + 1
```

```
    return [nation_letter, doc_version, data, author, nation_word]


# report_output.txt
dir_txt = os.path.dirname(__file__)
file_txt = os.path.join(dir_txt, "../output/FEB_report_log.txt")
ftxt_w = open(file_txt, "w+")
ftxt_a = open(file_txt, "a")
ftxt_r = open(file_txt, "r")

# temperature_ADC.txt
dir_txt1 = os.path.dirname(__file__)
file_txt1 = os.path.join(dir_txt1, "../output/temperature_ADC.csv")
ftxt_w1 = open(file_txt1, "a")

# temperature_C.txt
dir_txt2 = os.path.dirname(__file__)
file_txt2 = os.path.join(dir_txt2, "../output/temperature_C.csv")
ftxt_w2 = open(file_txt2, "a")

# bias readings
def get_bias_data(module_number):
    module_data = []

    # open .csv file containing bias measurements
    with open("../CSV_tables/FEB_testing - Multimeter.csv") as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")
        line_count = 0

        for row in csv_reader:
            if line_count == 0:
                line_count += 1
            else:
                if row[0] == str(module_number):
                    for i in range(1, 11):
                        row[i] = row[i].replace(",", ".")

                    # read data with correct number of decimals
                    module_data.append(row[0])
                    module_data.append(format(float(row[1]), ".3f"))
                    module_data.append(format(float(row[2]), ".1f"))
                    module_data.append(format(float(row[3]), ".3f"))
                    module_data.append(format(float(row[4]), ".0f"))
                    module_data.append(format(float(row[5]), ".3f"))
                    module_data.append(format(float(row[6]), ".0f"))
                    module_data.append(format(float(row[7]) / 100, ".2f"))
                    module_data.append(format(float(row[8]), ".3f"))
                    module_data.append(format(float(row[9]), ".3f"))
                    module_data.append(format(float(row[10]), ".3f"))

                    flag = True

                line_count += 1

    # write to log file
    if len(module_data) != 0:
        if flag:
            ftxt_a.write("\n*** BIAS ***\n\n")
            ftxt_a.write("  AVDD [V]: " + str(module_data[1]) + "\n")
            ftxt_a.write(" IVDD [mA]: " + str(module_data[2]) + "\n")
            ftxt_a.write("  DVDD [V]: " + str(module_data[3]) + "\n")
            ftxt_a.write("IDVDD [mA]: " + str(module_data[4]) + "\n")
            ftxt_a.write("    3V3 [V]: " + str(module_data[5]) + "\n")
            ftxt_a.write(" I3V3 [mA]: " + str(module_data[6]) + "\n")
            ftxt_a.write("Ibias [mA]: " + str(module_data[7]) + "\n")
            ftxt_a.write(" VCMSH [V]: " + str(module_data[8]) + "\n")
            ftxt_a.write("    VCM [V]: " + str(module_data[9]) + "\n")
```

```
            ftxt_a.write("  RVCM [V]: " + str(module_data[10]) + "\n")
            ftxt_a.write("\n")

            flag = False

        return module_data
    else:
        return ["Error: module #" + str(module_number) + " not found"]


# temperature reading and mean calculation
def report_temperature(file_temp):
    with open(file_temp) as f:
        lines_tm = f.readlines()

    sum = 0
    count = 0

    for i in lines_tm[45:]:
        str_tm = i.split("\t")
        sum = sum + int(str_tm[1][0:4])
        count = count + 1

    # write to log file
    ftxt_a.write("\n*** TEMPERATURE SENSOR ***\n")
    ftxt_a.write("\nTemperature sensor [ADC]: " + str(int(sum / count)) + "\n")

    # temperature calculation [ADC_code > Â°C]
    ADC_code = int(sum / count)
    V_T = 0.9 * 1000 - (ADC_code - 1024) * 1.72 / (3.87)
    T = 30 + (5.506 - math.sqrt((-5.506) ** 2 + 4 * 0.00176 * (870.6 - V_T))) / (
        2 * (-0.00176)
    )

    # write to log file
    ftxt_a.write(
        " Temperature sensor [Â°C]: " + str(format(round(T, 3), ".3f")) + "\n\n"
    )

    return [str(int(sum / count)), str(format(round(T, 3), ".3f"))]


# noise ENC data
def report_ENC(file_ENC):
    ENC_data = []

    # open END data file
    with open(file_ENC) as f:
        lines_enc = f.readlines()

    # write to log file
    ftxt_a.write("*** NOISE ENC [keV] ***\n\n")

    count = 0

    # read ENC data from file (refer to file formatting)
    for i in lines_enc[7:]:
        str_enc = i.split("\t")

        time = str_enc[1][1:]
        channel = str_enc[0][1:]

        if time == "6":
            count = count + 1

            if (
                # select required channels
                (count - 1) == 0
```

```
            or (count - 1) == 7
            or (count - 1) == 15
            or (count - 1) == 16
            or (count - 1) == 23
            or (count - 1) == 31
        ):
            # write to log file
            if (count - 1) < 10:
                ftxt_a.write(
                    " Canale "
                    + str(count - 1)
                    + ": "
                    + format(float(str_enc[4]), ".3f")
                    + "\n"
                )
            else:
                ftxt_a.write(
                    "Canale "
                    + str(count - 1)
                    + ": "
                    + format(float(str_enc[4]), ".3f")
                    + "\n"
                )

            ENC_data.append(format(float(str_enc[4]), ".3f"))

    return ENC_data


# threshold dispersion data
def report_thrdisp(file_thr):
    with open(file_thr) as f:
        lines_tr = f.readlines()

    # write to log file
    str_tr = lines_tr[14]
    list_tr = str_tr.split("\t")
    ftxt_a.write("\n*** THRESHOLD DISPERSION ***\n")
    ftxt_a.write("\nBefore FT [DAC]: " + format(float(list_tr[3]), ".3f") + "\n")
    ftxt_a.write(" After FT [DAC]: " + format(float(list_tr[8]), ".3f") + "\n\n")

    return [format(float(list_tr[3]), ".3f"), format(float(list_tr[8]), ".3f")]


# pedestal mean from pedestal data samples
def report_pedestal(file_ped):
    with open(file_ped) as f:
        lines_tm = f.readlines()

    sum = 0
    count = 0

    for i in lines_tm[1:]:
        str_tm = i.split("\t")

        if int(str_tm[1]) == 6:
            sum = sum + float(str_tm[3])
            count = count + 1

    # write to log file
    ftxt_a.write("*** PEDESTAL DISPERSION ***\n")
    ftxt_a.write(
        "\nPedestal dispersion [ADC]: "
        + str(format(round(sum / count, 3), ".3f"))
        + "\n"
    )

    return str(format(round(sum / count, 3), ".3f"))
```

```
# convert report_output.txt to report_output.pdf
def text_to_pdf(text, filename):

    # PDF file configuration (paper format and text)
    a4_width_mm = 210
    pt_to_mm = 0.35
    fontsize_pt = 10
    fontsize_mm = fontsize_pt * pt_to_mm
    margin_bottom_mm = 10
    character_width_mm = 7 * pt_to_mm
    width_text = a4_width_mm / character_width_mm

    # paper layout and font
    pdf = FPDF(orientation="P", unit="mm", format="A4")
    pdf.set_auto_page_break(True, margin=margin_bottom_mm)
    pdf.add_page()
    pdf.set_font(family="Courier", size=fontsize_pt)
    splitted = text.split("\n")

    for line in splitted:
        lines = textwrap.wrap(line, width_text)

        if len(lines) == 0:
            pdf.ln()

        for wrap in lines:
            pdf.cell(0, fontsize_mm, wrap, ln=1)

    pdf.output(filename, "F")


# defect notes
def defect_notes(module_number):
    notes = ""

    # open defect notes .csv file
    with open("../CSV_tables/FEB_testing - Defects.csv") as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=",")
        line_count = 0
        flag = False

        # read .csv table
        for row in csv_reader:
            if line_count == 0:
                line_count += 1
            else:
                if row[0] == str(module_number):

                    # scratches on top
                    if row[1] == "x":
                        notes = "scratches on top"
                        flag = True

                    # exposed copper on top
                    if row[2] == "x":
                        if not (flag):
                            notes = notes + "exposed copper on top"
                            flag = True
                        else:
                            notes = notes + "\n" + "exposed copper on top"

                    # solid defect on top
                    if row[3] == "x":
                        if not (flag):
                            notes = notes + "solid defect on top"
                            flag = True
```

```
            else:
                notes = notes + "\n" + "solid defect on top"

        # scratches on bottom
        if row[4] == "x":
            if not (flag):
                notes = notes + "scratches on bottom"
                flag = True
            else:
                notes = notes + "\n" + "scratches on bottom"

        # exposed copper on bottom
        if row[5] == "x":
            if not (flag):
                notes = notes + "exposed copper on bottom"
                flag = True
            else:
                notes = notes + "\n" + "exposed copper on bottom"

        # solid defect on bottom
        if row[6] == "x":
            if not (flag):
                notes = notes + "solid defect on bottom"
                flag = True
            else:
                notes = notes + "\n" + "solid defect on bottom"

        # if "OK" in notes field, read and leave empty
        if row[7] != "OK":
            if not (flag):
                notes = notes + str(row[7])
                flag = True
            else:
                notes = notes + "\n" + str(row[7])

return notes
```