UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING
Centre for Advanced Coating Technologies

A quick guide on…

MoistureX Software

USER'S MANUAL

By: ***A. Khakpour, M. Gibbons, S. Chandra***

June 3, 2019

Table of Contents

Objective and Mechanism of Action

GVA LED units that are equipped with a pressure relief fabric vent can absorb moisture in humid environments. Moisture concentration may build up within the LED unit and reach saturation point which leads to condensation that can potentially damage the electric components. MoistureX is developed to predict the relative humidity conditions inside the LED units based on the climate of the serving location and the light's operation schedule. The objective of this software is to detect the high-risk hours of condensation throughout a year.

This code initially askes the user to specify the vent's active area, and the scheduled switch-on and switch-off time for the lights. Then it asks the user to enter the name for the climate data file for which the simulation shall refer to. Afterwards, the code accesses the data from climate file and begins processing them row by row.

Programming Language

Python 3.7 language was used to develop the code. Open source software Anaconda was used as the compiler. It is recommended to download Anaconda, since it comes with many libraries such as Math, NumPy, Openpyxl and so forth. Anaconda can be installed from this link: ***https://www.anaconda.com/distribution/***

System Requirements

This software works on Windows 7 or later with 32- or 64-bit architecture and requires a minimum available RAM of 256MBs. Almost every computer manufactured within the past ten years is capable of running this code.

MoistureX Coding

MoistureX contains seven functions each of which perform a certain calculation on the inputs and return the outputs. All components of the code along with their functionality is explained in the table below.

Python Script | ***Description***

```
import openpyxl
import math as mat
from openpyxl import Workbook
from openpyxl.styles import Color, PatternFill, Font, Border
from openpyxl.styles import colors
from openpyxl.cell import Cell
```

Importing the required python functions. Make sure your python version is installed with math and openpyxl modules as well. Or you can install free version of Anacnda from this website: https://www.anaconda.com/distribution/

```
print("PLEASE NOTE THAT THIS CODE IS ONLY VALID FOR VE8 SERIES VENTS DUE TO SIMILAR
THICKNESS")

print("DROPLET CONDENSATION ON THE VENTS OR CONTAMINATION AFFECTS THE PERMEABILITY
PROPERTIES\nOF THE VENTS AND MAKES THIS FORECAST A LOT LESS ACCURATE")

print("___________________________________________________________________________")

print("PLEASE ENTER THE VENT'S ACTIVE AREA IN SQUARE MILIMETER \n(NOTE: THE CURRENT LED MODEL
USES GORETEX VENT VE8-0308 WITH AN ACTIVE VENTING AREA OF 8.55 MM^2) \n\nAREA = ")

area = float(input())
CORR = area * (0.1169590643)

print("\nPLEASE NOTE THAT THE FILE NAME IS CASE SENSITIVE. FAILURE TO ENTER THE \nEXACT FILE
NAME WILL RESULT IN AN EXECUTION ERROR. \nALSO NOTE THAT THERE MUST BE ONLY ONE FILE WITH THIS
NAME\nON YOUR SYSTEM.\n")
print("*(A)* PLEASE ENTER THE EXACT FILE NAME FOR THE CLIMATE DATA FILE FOLLOWED BY .xlsx = ")
filenom = str(input())
print("\nLoading File... This may take a while...\n")
cfd = openpyxl.load_workbook(filenom)
```

The following lines obtains "vent area", and "climate file name" and "sheet name" from the user, in order for the software to find the file to access its data.

This correction factor is used to update the vapor passage based on the vent area. For the tested vents of area 8.55mm^2, this factor is 1.

Python Script ***Description***

```
print("\n\nPLEASE NOTE THAT THE SHEET NAME IS CASE SENSITIVE. FAILURE TO ENTER THE \nEXACT
SHEET NAME WILL RESULT IN AN EXECUTION ERROR. \n")
print("**(B)** PLEASE ENTER THE EXACT SHEET NAME OF THE CLIMATE EXCEL FILE ON WHICH THE HOUR
\nBY HOUR DATA IS STORED = ")
sheetnom = str(input())
sheet0 = cfd.get_sheet_by_name(sheetnom)
print("___________________________________________________________________________")
```

```
confirm = 'N'
while (confirm == 'N' or confirm == 'n' or confirm == 'No' or confirm == 'no'):
    valid = 0
    while (valid == 0):
        print("\nNOTE: ALL HOURS ENTERED SHALL BE IN 24H FORMAT REAL NUMBER W/DECIMALS,\nFOR
        EXAMPLE 7.00 REFERES TO 7:00 AM OR 15.75 REFERS TO 3:45 PM")
        print("\nENTER THE HOUR THAT THE LED IS SCHEDULED TO TURN ON: ")
        h_on = float(input())
        if (h_on < 24 and h_on >= 0):
            valid = 1
        elif (h_on >= 24):
            print("\nINVALID ENTRY, PLEASE TRY AGAIN ACCORDING TO INSTRUCTIONS")
            valid = 0
        elif (h_on < 0):
            print("\nINVALID ENTRY, PLEASE TRY AGAIN ACCORDING TO INSTRUCTIONS")
            valid = 0
        else:
            print("\nINVALID ENTRY, PLEASE TRY AGAIN ACCORDING TO INSTRUCTIONS")
            valid = 0

    valid = 0
    while (valid == 0):
        print("\nENTER THE HOUR THAT THE LED IS SCHEDULED TO TURN OFF: ")
```

This loop will ask the user the LED switch on and switch off time and will keep asking until the user confirms the input.

Python Script	Description

```
    h_off = float(input())
    if (h_off < 24 and h_off >= 0):
        valid = 1
    elif (h_off >= 24):
        print("INVALID ENTRY, PLEASE TRY AGAIN ACCORDING TO INSTRUCTIONS")
        valid = 0
    elif (h_off < 0):
        print("INVALID ENTRY, PLEASE TRY AGAIN ACCORDING TO INSTRUCTIONS")
        valid = 0
    else:
        print("INVALID ENTRY, PLEASE TRY AGAIN ACCORDING TO INSTRUCTIONS")
        valid = 0
h_on_hh = (int(h_on))
if h_on_hh == 0:
    h_on_hh = '00'
elif h_on_hh == 1:
    h_on_hh = '01'
elif h_on_hh == 2:
    h_on_hh = '02'
elif h_on_hh == 3:
    h_on_hh = '03'
elif h_on_hh == 4:
    h_on_hh = '04'
elif h_on_hh == 5:
    h_on_hh = '05'
elif h_on_hh == 6:
    h_on_hh = '06'
elif h_on_hh == 7:
    h_on_hh = '07'
elif h_on_hh == 8:
    h_on_hh = '08'
elif h_on_hh == 9:
```

Python Script	Description

```
        h_on_hh = '09'

    if (h_on >= 1):
        h_on_mm = int((h_on % int(h_on)) * 60)
    else:
        h_on_mm = int((h_on) * 60)

    if h_on_mm == 0:
        h_on_mm = '00'
    elif h_on_mm == 1:
        h_on_mm = '01'
    elif h_on_mm == 2:
        h_on_mm = '02'
    elif h_on_mm == 3:
        h_on_mm = '03'
    elif h_on_mm == 4:
        h_on_mm = '04'
    elif h_on_mm == 5:
        h_on_mm = '05'
    elif h_on_mm == 6:
        h_on_mm = '06'
    elif h_on_mm == 7:
        h_on_mm = '07'
    elif h_on_mm == 8:
        h_on_mm = '08'
    elif h_on_mm == 9:
        h_on_mm = '09'

    h_off_hh = (int(h_off))
    if h_off_hh == 0:
        h_off_hh = '00'
    elif h_off_hh == 1:
```

Python Script	*Description*

```
        h_off_hh = '01'
    elif h_off_hh == 2:
        h_off_hh = '02'
    elif h_off_hh == 3:
        h_off_hh = '03'
    elif h_off_hh == 4:
        h_off_hh = '04'
    elif h_off_hh == 5:
        h_off_hh = '05'
    elif h_off_hh == 6:
        h_off_hh = '06'
    elif h_off_hh == 7:
        h_off_hh = '07'
    elif h_off_hh == 8:
        h_off_hh = '08'
    elif h_off_hh == 9:
        h_off_hh = '09'

    if (h_off >= 1):
        h_off_mm = int((h_off % int(h_off)) * 60)
    else:
        h_off_mm = int((h_off) * 60)

    if h_off_mm == 0:
        h_off_mm = '00'
    elif h_off_mm == 1:
        h_off_mm = '01'
    elif h_off_mm == 2:
        h_off_mm = '02'
    elif h_off_mm == 3:
        h_off_mm = '03'
    elif h_off_mm == 4:
```

Python Script ***Description***

```
        h_off_mm = '04'
    elif h_off_mm == 5:
        h_off_mm = '05'
    elif h_off_mm == 6:
        h_off_mm = '06'
    elif h_off_mm == 7:
        h_off_mm = '07'
    elif h_off_mm == 8:
        h_off_mm = '08'
    elif h_off_mm == 9:
        h_off_mm = '09'

    print("\n___________________________________________________________________________")
    print("HOUR THAT LED IS TURNED ON = ", h_on, " (OR) In HH:MM:SS format = ", h_on_hh, ":",
     h_on_mm, ": 00")
    print("HOUR THAT LED IS TURNED OFF = ", h_off, " (OR) In HH:MM:SS format = ", h_off_hh,
     ":", h_off_mm, ": 00")
    print("\nDO YOU CONFIRM THIS INFO. ?")
    print("Enter Y for Yes, or N for No")
    confirm = str(input())
    print("\n***************************************************************************")
print("***************************************************************************")
print("**********CALCULATION IN PROGRESS: THIS MAY TAKE SEVERAL MINUTES***********")
print("***************************************************************************")
print("***************************************************************************")
```

```
def VCRPM_CALC (delta_C):
    d_C = abs(delta_C)  #yields the positive value (magnitude)
    VCRPS = CORR * 0.0000376342 * mat.exp(0.15292548311 * d_C
    VCRPM = 60  * VCRPS
    return VCRPM
```

The following function will approximate the water vapor concentration rate of change in gr/m^3-minutes based on the difference in vapor concentrations in the two

Python Script	*Description*
	environments and based on experimental results. The actual rates may "significantly" vary due to air pollution and possible clogging over extended use in moist environments.

`def V_CONCENTRATION_CALC (rh , t):`
`    a = 0.00000003230010727 * rh - 0.0000000003012727272`
`    b = 0.00000144926 * rh + 0.00000003169090909`
`    c = 0.00007509705455 * rh + 0.0000002527272727`
`    d = 0.003926678182 * rh - 0.00005416363636`
`    e = 0.05287762545 * rh + 0.00008172727273`
`    vc = ((a)*(t*t*t*t)) + ((b)*(t*t*t)) + ((c)*(t*t)) + ((d)*(t)) + e`
`    if vc <= 0:`
`        vc = 0`
`    vc = round(vc, 5)`
`    return vc`
	This function is a psychometric function that estimates the vapor mass concentration based on the relative humidity and dry bulb temperature.

`def RH_CALC (vc , t):`
`    mrx = (0.000000001102596701*(t*t*t*t)) - (0.000000476784208667*(t*t*t)) +`
`   (0.000128305099896850*(t*t)) - (0.030008109869684000*t) + (1.314954264157040000)`
`    mry = 10 ** mrx`
`    rh = mry * vc`
`    rh = round(rh, 3)`
`    return rh`
	The following function is a psychometric function that estimates the relative humidity based on the vapor concentration and dry bulb temperature.

`def OPERATION_STAT(hour, h_on, h_off):`
`    if (h_on <= h_off):`
`        day_factor = int(hour/24)`
`        cum_hour_on = h_on + (day_factor * 24)`
	This function aims to detect the operative status of the LEDs throughout the entire

Python Script | ***Description***

```
        cum_hour_off = h_off + (day_factor * 24)
        if (hour < cum_hour_on):
            OP = 0
        elif (hour > cum_hour_off):
            OP = 0
        else:
            OP = 1
    if (h_on > h_off):
        day_factor = int(hour/24)
        cum_hour_on = h_on + (day_factor * 24)
        cum_hour_off = h_off + (day_factor * 24)
        if (hour < cum_hour_on and hour > cum_hour_off):
            OP = 0
        else:
            OP = 1
    return OP
```

hours in a year, and based on the user-defined switch-on and switch-off time.

The value of returned variable OP is 1, for hours in which the LED is working and 0 when it is off.

```
def VC_UPDATER (int_vc, ext_vc):
    for iror in range (1, 60):
        delta_C = int_vc - ext_vc
        VCRPM = VCRPM_CALC(delta_C)
        if (ext_vc > (int_vc)):
            int_vc = int_vc + VCRPM
        elif (ext_vc <= (int_vc)):
            int_vc = int_vc - VCRPM
    return int_vc
```

This function updates the internal vapor concentration for the next hour based on the previous internal vapor concentration, and the ambient (external) vapor concentration.

To increase accuracy (and since the rate of vapor transfer is a function of difference in vapor concentrations) a minute by minute iterative approach is used, and the result at the end of the 60 iterations is reported as the next hour's internal vapor concentration.

Python Script	***Description***

`def RH_Retriever (hour):`
`    i = int(hour) + 1`
`    rh = sheet0.cell (row = i, column = 6).value`
`    return rh`
	This function is used to retrieve the ambient relative humidity value for each hour from the climate data file excel file.

`def Temp_Retriever (hour):`
`    i = int(hour) + 1`
`    t = sheet0.cell(row=i, column=4).value`
`    return t`
	This function is used to retrieve the ambient temperature value for each hour from the climate data file excel file.

`hour = 1`
`ihf = openpyxl.Workbook()`
`ihf.create_sheet(index = 0 , title = 'LED Internal Humidity Condition Forecast')`
`sheet1 = sheet1 = ihf.get_sheet_by_name('LED Internal Humidity Condition Forecast')`
`sheet1['A1'] = 'HOUR'`
`sheet1['B1'] = 'EXT TEMP'`
`sheet1['C1'] = 'EXT RH'`
`sheet1['D1'] = 'EXT V.CONC'`
`sheet1['E1'] = 'INT TEMP'`
`sheet1['F1'] = 'INT RH'`
`sheet1['G1'] = 'INT V.CONC'`
 	
`#This loop starts from hour 1 and ends at hour 8760, last hour of a year.`
`#the values for hour, external temp and external RH are placed in the new`
`#excel file and ambient vapor concentrations are calculated and placed in its columns.`
`for hour in range (1 , 8760):`
`    i = int(hour) + 1`
`   #row number of the excel file is hour plus one. Plus one is added since the first row is`
`   filled with titles of each column.`
`    sheet1.cell(row=i, column=1).value = hour`
`   #places hour number in column 1`
	From here on, this makes the main body of the code.
	
	It creates a new excel file with columns named as (1)Hour (2)External Temp (3)External RH (4)External Vapor Concentration (5) Internal Temp(6)Internal RH and (7)Internal Vapor Concentration.
	
	This loop starts from hour 1 and ends at hour 8760, last hour of a year. The values for hour, external temp and external RH are placed in the new excel file and ambient vapor concentrations are calculated and placed in its columns.

Python Script

```
ext_temp = Temp_Retriever (hour)
#calls temp retriever function to obtain the ambient temp from the climate data file loaded
earlier.
sheet1.cell(row=i, column=2).value = ext_temp
#places the external temperature value in column 2

ext_rh = RH_Retriever (hour)
#calls relative humidity retriever function to obtain the ambient RH from the climate data
file loaded earlier.
sheet1.cell(row=i, column=3).value = ext_rh
#places the external RH value in column 3

ext_vc = V_CONCENTRATION_CALC (ext_rh , ext_temp)
#calls the psychometric function defined earlier to calculate the external vapor
concentrations based on the external temp and RH.
sheet1.cell(row=i, column=4).value = ext_vc
#places the external vapor concentrations value in column 4

if hour == 1:
#For the first hour, all internal parameters were set equal to external (ambient)
parameters.
    int_temp = ext_temp
    #and the values for internal temp, RH and vapor concentrations were placed in columns
    5, 6 and 7 respectively.
    sheet1.cell(row=i, column=5).value = ext_temp

    int_rh = ext_rh
    sheet1.cell(row=i, column=6).value = ext_rh

    int_vc = ext_vc
    sheet1.cell(row=i, column=7).value = ext_vc
```

Description

Afterwards, the internal vapor concentrations are calculated, and the temperature of the LED is estimated based on its operation status and ambient temperature. The psychometric function then converts these inputs into relative humidity.

All data is then saved as an excel file, with high risk hours highlighted.

Python Script	*Description*

```
else:
    hourp = hour - 1
    OPP = OPERATION_STAT(hourp, h_on, h_off)
    OP = OPERATION_STAT(hour, h_on, h_off)
    #for hours greater than one:
    DIV = OP - OPP
    if (DIV == 1):
    #detects the hours just after LED switched on to account for the VC gain due to
    unstable conditions
        int_vprime = VC_UPDATER(int_vc, ext_vc)
        int_vc = int_vprime * 1.251
        #Internal vapor concentration hike post rapid transient mode (turn on)
        sheet1.cell(row=i, column=7).value = int_vc
        #internal vapor concentration then placed in column 7 for each corresponding row.
    elif (DIV == -1):
        int_vprime = VC_UPDATER(int_vc, ext_vc)
        int_vc = int_vprime * 0.796
        #Internal vapor concentration loss post rapid transient mode (turn off)
        sheet1.cell(row=i, column=7).value = int_vc
        #internal vapor concentration then placed in column 7 for each corresponding row.
    else:
        int_vc = VC_UPDATER(int_vc, ext_vc)
        #Internal vapor concentration calculater function was called to calculate the
        internal VC at the next hour, based on external conditions.
        sheet1.cell(row=i, column=7).value = int_vc
        #internal vapor concentration then placed in column 7 for each corresponding row.

    if(OP == 0):
    #After finding the internal vapor concentrations, and to detect condensation, we aim to
    calculate the internal relative humidity.
        int_temp = ext_temp
```

Python Script	Description

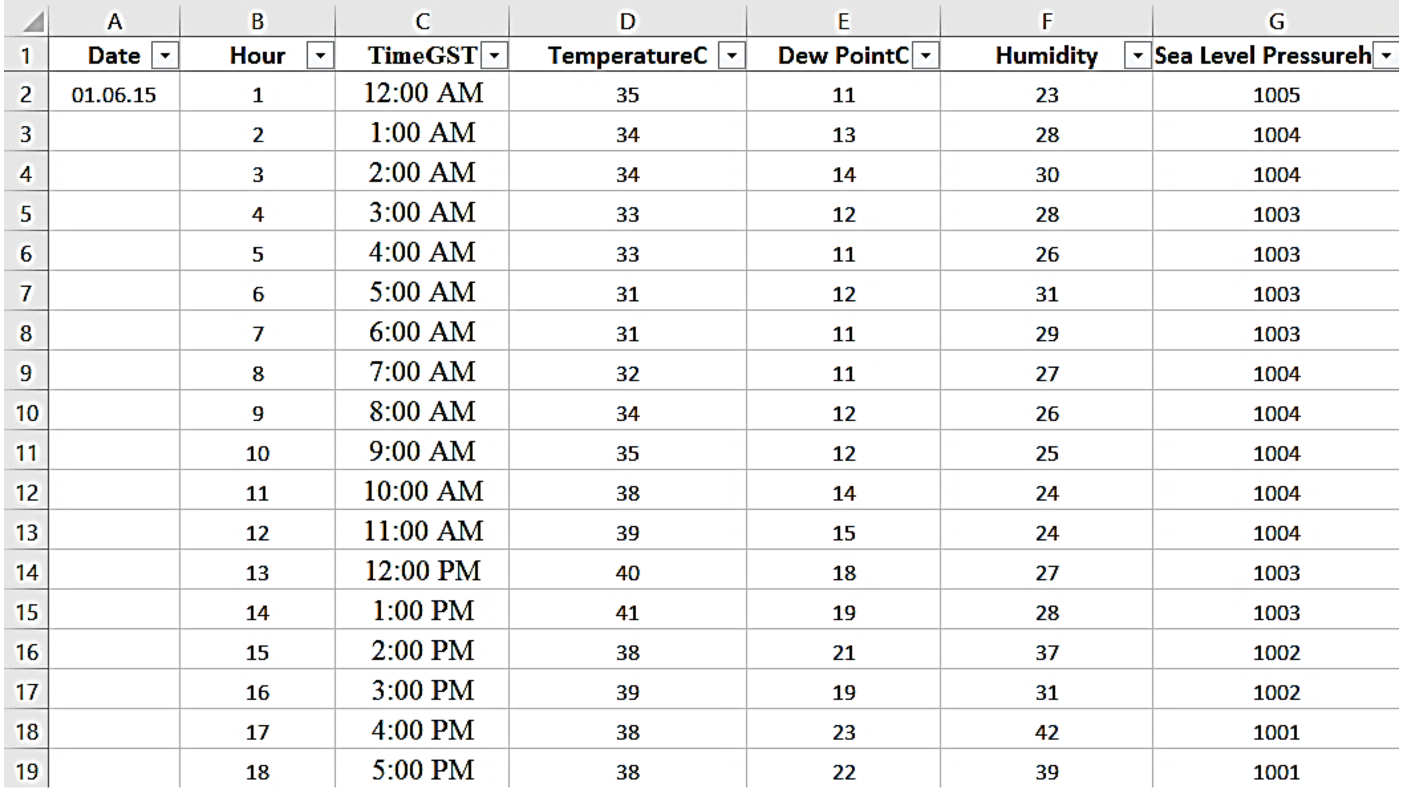
```
            #To find the RH, internal temperature must be known on top of the vapor
            concentration. Internal temperature is a function of external temperature and
            light's operative status.
        elif(OP == 1):
        #OP value for each hour is determined by calling the OPERATION_STAT function that was
        defined to detect hours for which the light is in operation.
        int_temp = (-0.002447543006*(ext_temp)*(ext_temp)) + (0.950598039598*(ext_temp)) +
        17.474593962999
        #if the ligh is in operation, its temperature can be estimated by this experimental
        equation. (Found in NO WIND testing)
        sheet1.cell(row=i, column=5).value = int_temp
        #This command places the estimated internal temp in column 5

        int_rh = RH_CALC (int_vc , int_temp)
        #Psychometric function RH_CALC is called to calculate the internal relative humidity
        based on the internal vapor concentration and LED internal temperature.
        if int_rh >= 100:
        #Since this is approximation, RH values of slightly over 100% may be returned. Thus,
        this command reports everything above hundred as 100%.
            int_rh = 100
        elif int_rh < 0:
            int_rh = 0
        sheet1.cell(row=i, column=6).value = int_rh
        #calculated RH is then placed in column 6, for the corresponding row i.
        if (int_rh >= 90):
        #If the reported RH is over 90%, the cell is highlighted in red and the risky hour is
        reported.
            sheet1.cell(row=i, column=6).fill = PatternFill("solid", fgColor="FF2F44")
            print("Elevated risk of condensation detected at hour number:", hour)

ihf.save('Internal_Humidity_File.xlsx')
#Saving the results as the output excel file.
```

Python Script	*Description*

`print("\n^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")`
`print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")`
`print("AFTER COMPLETION OF THIS SIMULATION, AN EXCEL FILE CONTAINING THE RESULTS\nWILL BE SAVED IN THE SAME DIRECTORY THE RAW CLIMATE DATA EXISTS. \nTHE FILE NAME WILL BE 'INTERNAL_HUMIDITY_FILE.xlsx'")`
`print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")`
`print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")`
	This is a message displayed to the user.

How to run a simulation?

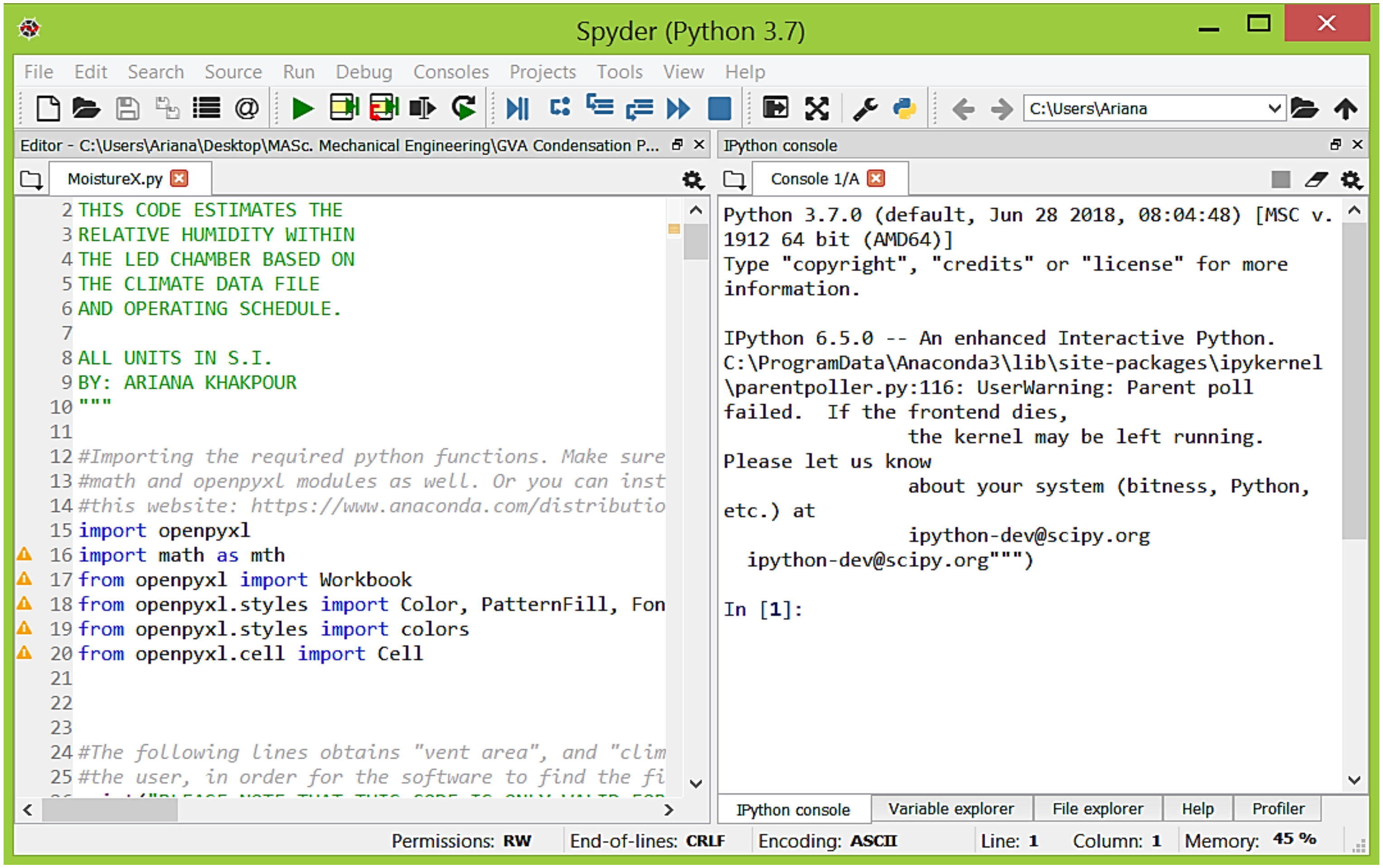
STEP 1: Before running this code, create a new folder and place the software “MoistureX.py” inside the folder. Then, place the climate data file in the same folder. Please ensure that the climate data file is an excel file with extension “.xlsx” and has the same format shown in Figure1. The columns B (hour), D (temperature) and F (relative humidity) are the only columns that are required to contain data. Column B (hour) should be in an ascending order, starting from hour 1. This code will run through hours 1 to 8760, which covers an entire year. If the climate data file contains more hours, or if the formatting is different that it’s demonstrated in Figure 1, minor modifications shall be made to the code which will be introduced in the next section.



	A	B	C	D	E	F	G
1	Date	Hour	TimeGST	TemperatureC	Dew PointC	Humidity	Sea Level Pressureh
2	01.06.15	1	12:00 AM	35	11	23	1005
3		2	1:00 AM	34	13	28	1004
4		3	2:00 AM	34	14	30	1004
5		4	3:00 AM	33	12	28	1003
6		5	4:00 AM	33	11	26	1003
7		6	5:00 AM	31	12	31	1003
8		7	6:00 AM	31	11	29	1003
9		8	7:00 AM	32	11	27	1004
10		9	8:00 AM	34	12	26	1004
11		10	9:00 AM	35	12	25	1004
12		11	10:00 AM	38	14	24	1004
13		12	11:00 AM	39	15	24	1004
14		13	12:00 PM	40	18	27	1003
15		14	1:00 PM	41	19	28	1003
16		15	2:00 PM	38	21	37	1002
17		16	3:00 PM	39	19	31	1002
18		17	4:00 PM	38	23	42	1001
19		18	5:00 PM	38	22	39	1001

Figure 1. Standard format for the climate data file required to run this simulation

STEP 2: Once the climate data file was checked to ensure the proper format is followed, open the program file named “MoistureX.py” using Python 3.7 or Anaconda. For this demonstration, Anaconda is used. If Anaconda is installed properly, a code editor application “Spyder” would exist on your system. To run the code, open Spyder, and on the top bar, click on “File”, then “Open”. In the file explorer that appears, find the “MoistureX.py” from its saved location and press open. A window containing the code appears as can be seen in Figure 2.



Editor - C:\Users\Ariana\Desktop\MASc. Mechanical Engineering\GVA Condensation P...

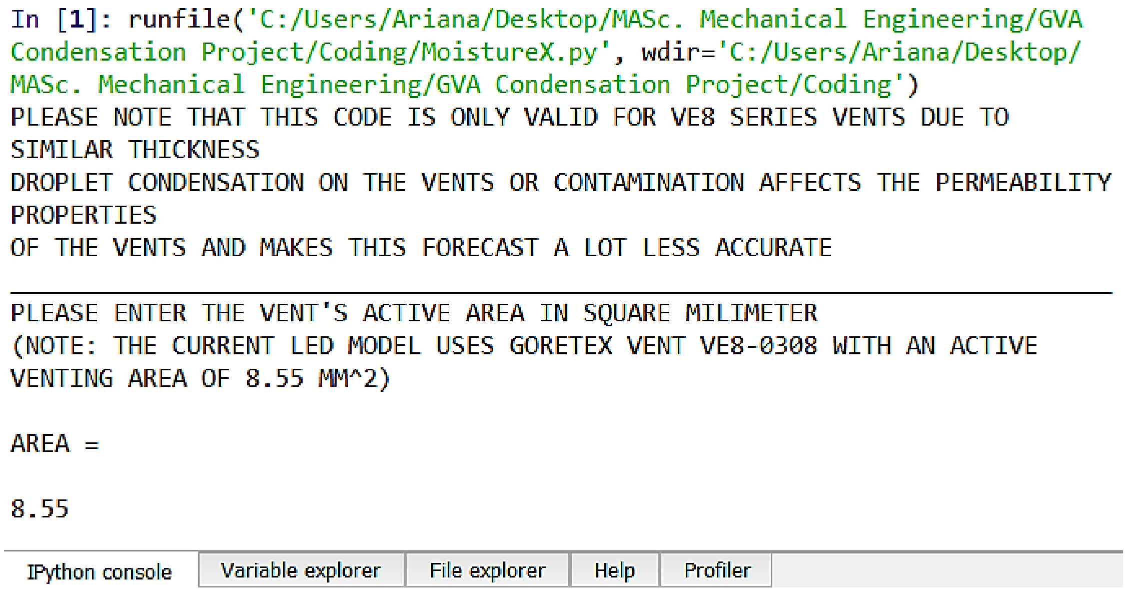
```
THIS CODE ESTIMATES THE
RELATIVE HUMIDITY WITHIN
THE LED CHAMBER BASED ON
THE CLIMATE DATA FILE
AND OPERATING SCHEDULE.

ALL UNITS IN S.I.
BY: ARIANA KHAKPOUR
"""

#Importing the required python functions. Make sure
#math and openpyxl modules as well. Or you can inst
#this website: https://www.anaconda.com/distributio
import openpyxl
import math as mth
from openpyxl import Workbook
from openpyxl.styles import Color, PatternFill, Fon
from openpyxl.styles import colors
from openpyxl.cell import Cell



#The following lines obtains "vent area", and "clim
#the user, in order for the software to find the fi
```

```
Python 3.7.0 (default, Jun 28 2018, 08:04:48) [MSC v.
1912 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more
information.

IPython 6.5.0 -- An enhanced Interactive Python.
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel
\parentpoller.py:116: UserWarning: Parent poll
failed.  If the frontend dies,
                the kernel may be left running.
Please let us know
                about your system (bitness, Python,
etc.) at
                ipython-dev@scipy.org
  ipython-dev@scipy.org""")

In [1]:
```

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 1 Column: 1 Memory: 45 %

Figure 2. The code opened in Spyder (Python 3.7) displayed in this window

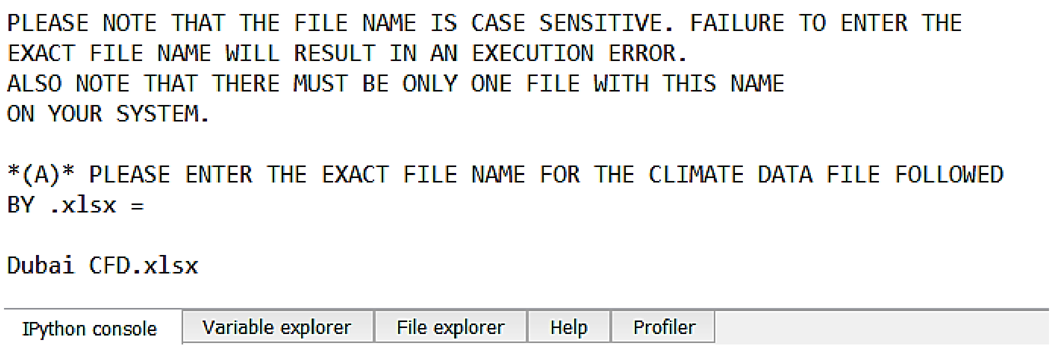
STEP 3: After opening the code on the python editor, press the "run" bottom on the toolbar on the top side of the window, or simply press F5 on your keyboard. After a few seconds, a message is displayed on the Ipython Console, on the right-hand side of the screen, asking for vent's active area in millimetres squared (Figure 3). Below the "AREA = " type in the vent's active area, and press enter. For this example, we will just enter 8.55.



```
In [1]: runfile('C:/Users/Ariana/Desktop/MASc. Mechanical Engineering/GVA
Condensation Project/Coding/MoistureX.py', wdir='C:/Users/Ariana/Desktop/
MASc. Mechanical Engineering/GVA Condensation Project/Coding')
PLEASE NOTE THAT THIS CODE IS ONLY VALID FOR VE8 SERIES VENTS DUE TO
SIMILAR THICKNESS
DROPLET CONDENSATION ON THE VENTS OR CONTAMINATION AFFECTS THE PERMEABILITY
PROPERTIES
OF THE VENTS AND MAKES THIS FORECAST A LOT LESS ACCURATE
___________________________________________________________________________
PLEASE ENTER THE VENT'S ACTIVE AREA IN SQUARE MILIMETER
(NOTE: THE CURRENT LED MODEL USES GORETEX VENT VE8-0308 WITH AN ACTIVE
VENTING AREA OF 8.55 MM^2)

AREA =

8.55
```

Figure 3. The first message displayed after running the code

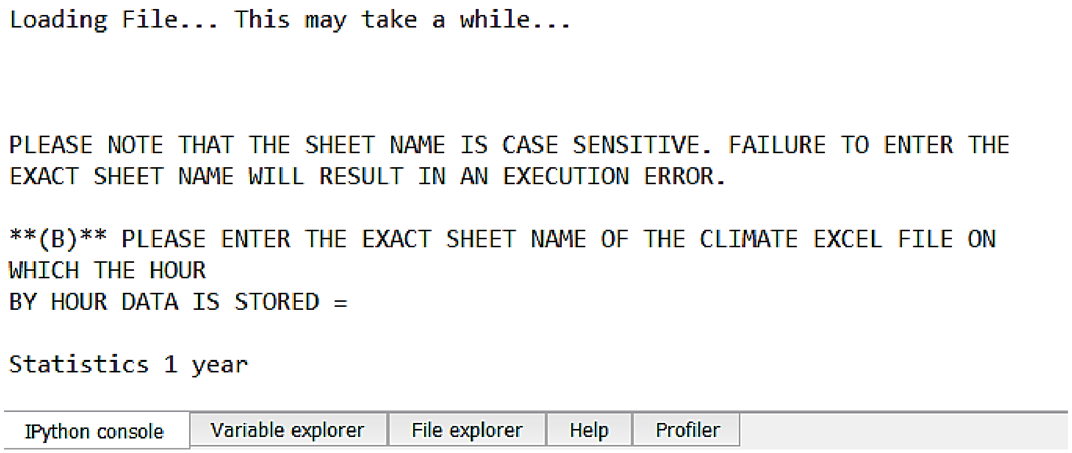
STEP 4: After entering the area, a second message is shown on the Console (Figure 4), asking the user to enter the exact name for the climate data file of interest. Please note that the input is case sensitive, and the file extension should be included as well. For this demonstration, the climate data file's name was "Dubai CDF.xlsx" and the exact name was typed in. After typing the file name, press enter to progress to the next step. Please note that there should be ONLY one file with that name on your system, and the excel file should not be running, while doing the simulation.



```
PLEASE NOTE THAT THE FILE NAME IS CASE SENSITIVE. FAILURE TO ENTER THE
EXACT FILE NAME WILL RESULT IN AN EXECUTION ERROR.
ALSO NOTE THAT THERE MUST BE ONLY ONE FILE WITH THIS NAME
ON YOUR SYSTEM.

*(A)* PLEASE ENTER THE EXACT FILE NAME FOR THE CLIMATE DATA FILE FOLLOWED
BY .xlsx =

Dubai CFD.xlsx
```

Figure 4. Second message displayed after entering the area

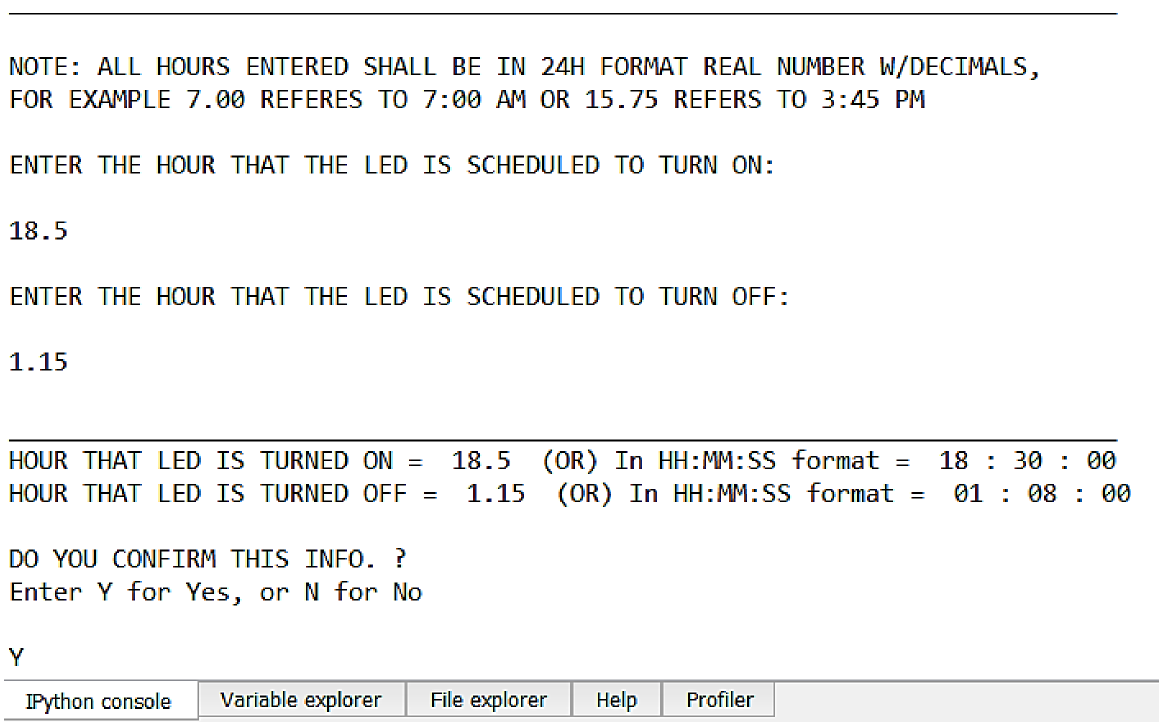
STEP 5: Once the climate data file's name is entered, another message is displayed, asking the user for the sheet's name, on which the data is stored (Figure 5). Each excel file may contain several sheets. Therefore, in order for the code to access the climate data, it has to know the name for the sheet where the data is located. The sheet name can be found on a tab at the bottom of the excel file. In this case, data was stored on the first sheet, with the name "Statistics 1 year". After typing this term, "enter" key was hit to proceed to the next step.



```
Loading File... This may take a while...


PLEASE NOTE THAT THE SHEET NAME IS CASE SENSITIVE. FAILURE TO ENTER THE
EXACT SHEET NAME WILL RESULT IN AN EXECUTION ERROR.

**(B)** PLEASE ENTER THE EXACT SHEET NAME OF THE CLIMATE EXCEL FILE ON
WHICH THE HOUR
BY HOUR DATA IS STORED =

Statistics 1 year
```

Figure 5. Message displayed asking the user to enter the excel sheet name

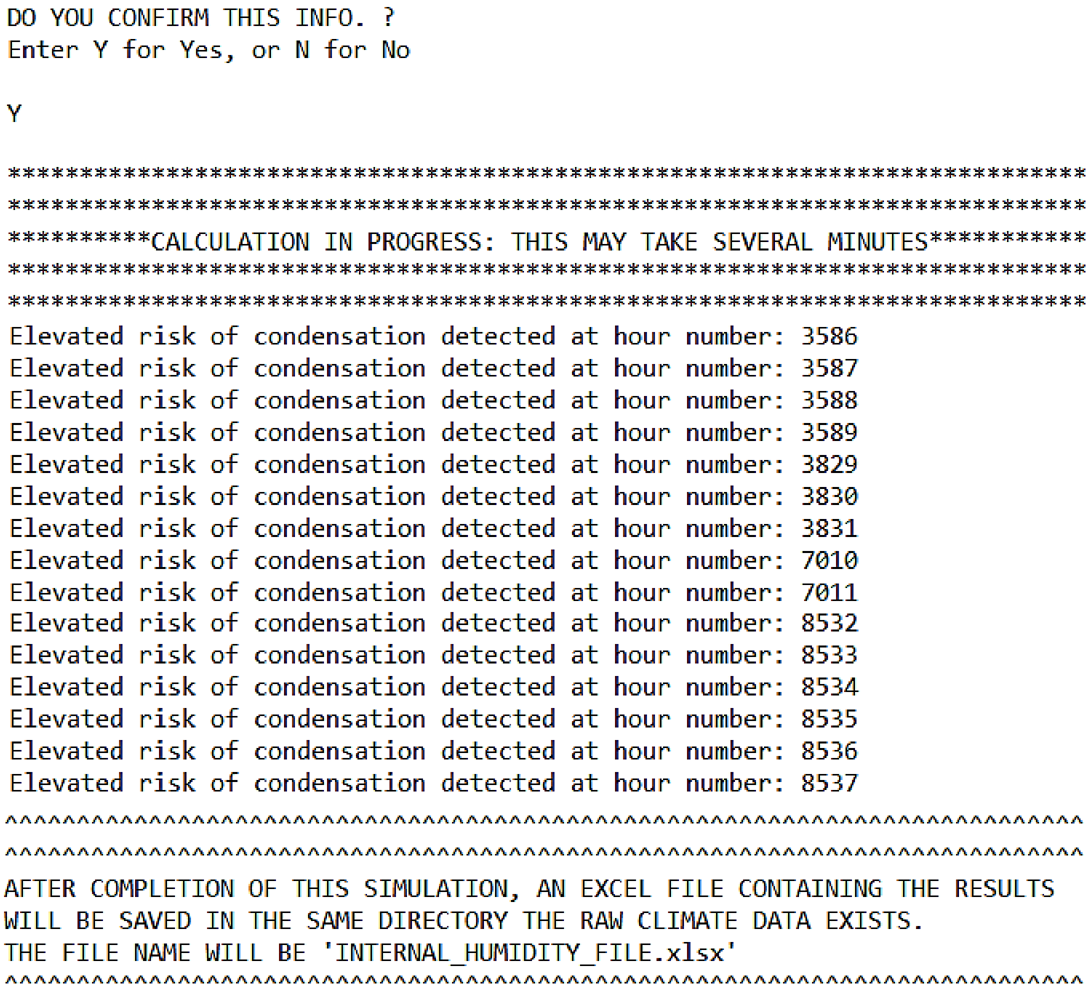
STEP 6: Now that the code accessed the climate data, it asks the user to enter the time when the LED is scheduled to turn on. Type the hour in a 24-hour decimal scheme. For example, if the LED is scheduled to turn on at 6:30 pm, simply type 18.5 and hit enter. Afterwards, another message is displayed that asks the user for the light's shut down time. Type in the time in the same fashion as the previous one and press enter. For example, we entered 1.15 to represent 1:08 AM. Once the shut-down time is entered, the software shows both the turn-on and turn-off schedule in HH: MM: SS format and asks for confirmation. If the entries are correct, type "Y" and press enter to proceed to the next step (Figure 6). Otherwise, you may re-enter the schedule by entering "N". After confirming the entries, the calculation process begins, and the software starts the simulation process.



```
_____________________________________________________________________________

NOTE: ALL HOURS ENTERED SHALL BE IN 24H FORMAT REAL NUMBER W/DECIMALS,
FOR EXAMPLE 7.00 REFERES TO 7:00 AM OR 15.75 REFERS TO 3:45 PM

ENTER THE HOUR THAT THE LED IS SCHEDULED TO TURN ON:

18.5

ENTER THE HOUR THAT THE LED IS SCHEDULED TO TURN OFF:

1.15

_____________________________________________________________________________
HOUR THAT LED IS TURNED ON =  18.5  (OR) In HH:MM:SS format =  18 : 30 : 00
HOUR THAT LED IS TURNED OFF =  1.15  (OR) In HH:MM:SS format =  01 : 08 : 00

DO YOU CONFIRM THIS INFO. ?
Enter Y for Yes, or N for No

Y
```

Figure 6. This message askes the user to enter the time when the LED is switched on, then off, on a daily basis. Once the time is entered, user can confirm or reject the inputs.

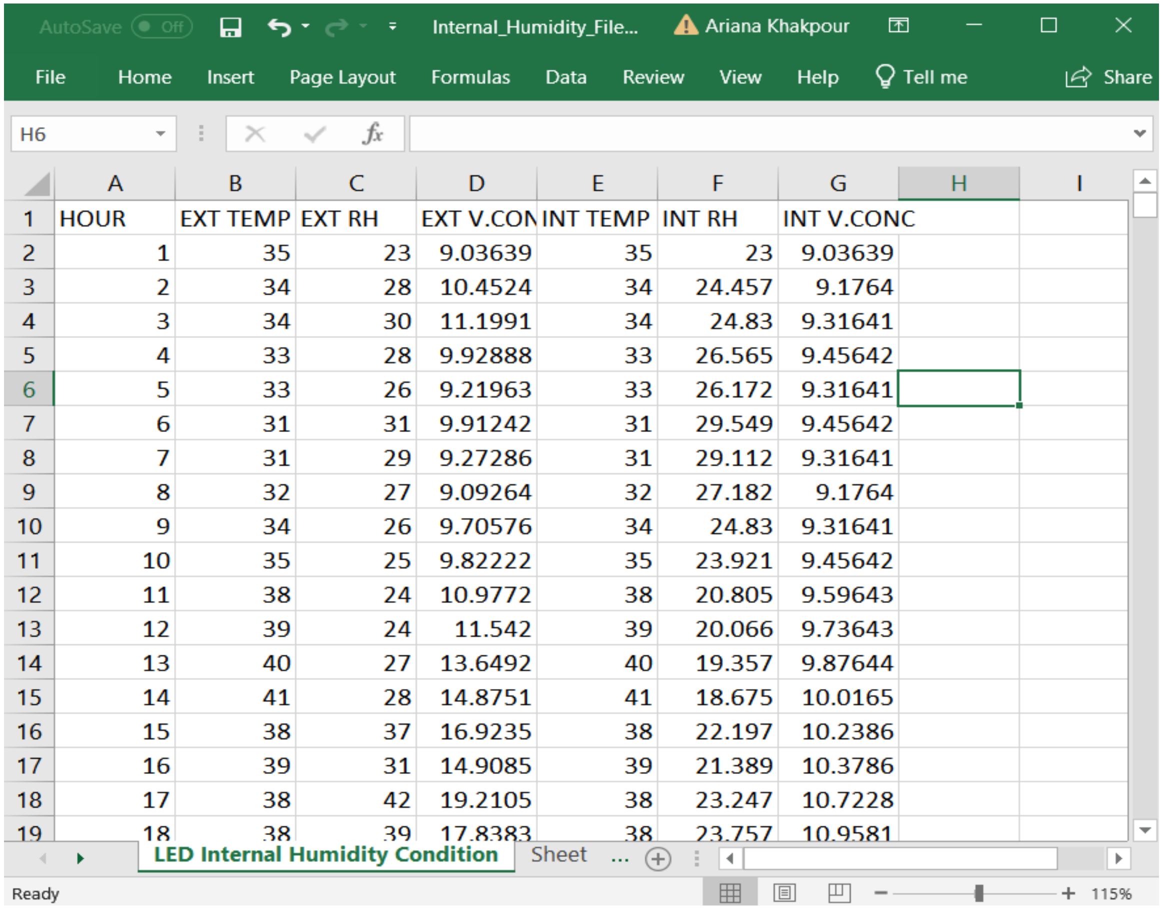
STEP 7: After confirming the LED's operative schedule, a message is displayed stating that calculation has begun. Within a few seconds (to a few minutes if the system is slow), the hours at which an internal relative humidity of over 90% is anticipated is listed (Figure 7), and an excel file containing the results is saved in the same directory where the "MoistureX.py" file is located.



```
DO YOU CONFIRM THIS INFO. ?
Enter Y for Yes, or N for No

Y

***************************************************************************
***************************************************************************
**********CALCULATION IN PROGRESS: THIS MAY TAKE SEVERAL MINUTES***********
***************************************************************************
***************************************************************************
Elevated risk of condensation detected at hour number: 3586
Elevated risk of condensation detected at hour number: 3587
Elevated risk of condensation detected at hour number: 3588
Elevated risk of condensation detected at hour number: 3589
Elevated risk of condensation detected at hour number: 3829
Elevated risk of condensation detected at hour number: 3830
Elevated risk of condensation detected at hour number: 3831
Elevated risk of condensation detected at hour number: 7010
Elevated risk of condensation detected at hour number: 7011
Elevated risk of condensation detected at hour number: 8532
Elevated risk of condensation detected at hour number: 8533
Elevated risk of condensation detected at hour number: 8534
Elevated risk of condensation detected at hour number: 8535
Elevated risk of condensation detected at hour number: 8536
Elevated risk of condensation detected at hour number: 8537
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AFTER COMPLETION OF THIS SIMULATION, AN EXCEL FILE CONTAINING THE RESULTS
WILL BE SAVED IN THE SAME DIRECTORY THE RAW CLIMATE DATA EXISTS.
THE FILE NAME WILL BE 'INTERNAL_HUMIDITY_FILE.xlsx'
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Figure 7. A summary of results displayed on the console.

STEP 8: Open the results file, which is named “INTERNAL_HUMIDITY_FILE.xlsx” and located at the same directory where the python file exists. As can be seen in Figure 8, the results file contains both the ambient conditions and the forecasted internal temperature and relative humidity. Hours at which the internal relative humidity is expected to exceed 90% are automatically highlighted red.



	A	B	C	D	E	F	G	H	I
1	HOUR	EXT TEMP	EXT RH	EXT V.CON	INT TEMP	INT RH	INT V.CONC		
2	1	35	23	9.03639	35	23	9.03639		
3	2	34	28	10.4524	34	24.457	9.1764		
4	3	34	30	11.1991	34	24.83	9.31641		
5	4	33	28	9.92888	33	26.565	9.45642		
6	5	33	26	9.21963	33	26.172	9.31641		
7	6	31	31	9.91242	31	29.549	9.45642		
8	7	31	29	9.27286	31	29.112	9.31641		
9	8	32	27	9.09264	32	27.182	9.1764		
10	9	34	26	9.70576	34	24.83	9.31641		
11	10	35	25	9.82222	35	23.921	9.45642		
12	11	38	24	10.9772	38	20.805	9.59643		
13	12	39	24	11.542	39	20.066	9.73643		
14	13	40	27	13.6492	40	19.357	9.87644		
15	14	41	28	14.8751	41	18.675	10.0165		
16	15	38	37	16.9235	38	22.197	10.2386		
17	16	39	31	14.9085	39	21.389	10.3786		
18	17	38	42	19.2105	38	23.247	10.7228		
19	18	38	39	17.8383	38	23.757	10.9581		

Figure 8. Output excel file containing the simulation results.

How to change or modify the code?

In this section a brief training will be given to enable the user to change or modify the code for various motivations. PLEASE retain the original copy of the code and perform modifications on a copied script, to avoid losing the main code.

A. Changing the number of hours

Changing the number of hours covered in the simulation is a very simple, yet useful trick. The user might be interested in analysing a certain timeslot of the year, or even for periods longer than a year. To do so, go to the line 313 and find the following statement:

```
for hour in range (1, 8760):
```

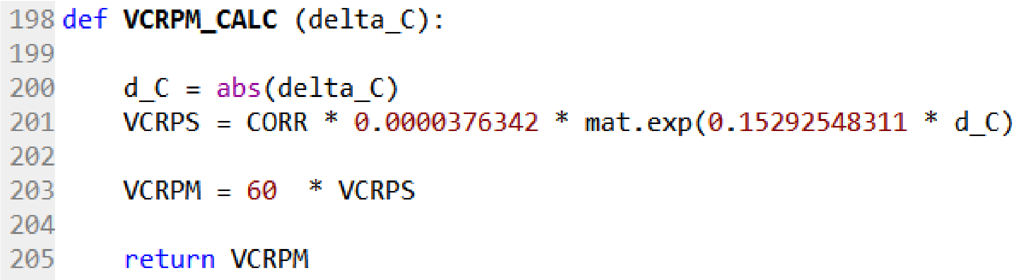
You may change the range stated in the bracket. For example, if you're interested to perform the simulation only for the first month, simply place **(1, 720)** in the bracket to cover the initial 720 hours of the year, as shown in the following statement:

```
for hour in range (1, 720):
```

Please note that if the range of hours you enter in the code does not exist on the climate file, an error will be encountered upon running the code.

B. Changing the vent's permeability coefficient

If the vent type is changed, or if further studies are performed to better evaluate the Gore-Tex vent's permeability behaviour, the code can be simply upgraded. To upgrade the permeability characteristics, simply go to line 198 and find the function responsible for calculation of vapor concentration rate of change per minute (VCRPM_Calc).



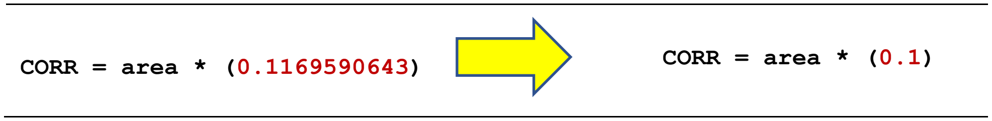
```
198 def VCRPM_CALC (delta_C):
199
200     d_C = abs(delta_C)
201     VCRPS = CORR * 0.0000376342 * mat.exp(0.15292548311 * d_C)
202
203     VCRPM = 60   * VCRPS
204
205     return VCRPM
```

The variable “delta_C” is the vapor concentration difference between the two environments (gr/m^3) and VCRPM is vapor concentration rate of change in the LED chassis (gr/m^3-s). Correction factor (CORR) is introduced to correct the results for the vent area that the user enters. To obtain the CORR, divide 1 by the area of the vent used in experiments.

Let’s suppose that experiments that were done on a metal mesh vent of area 10mm^2 have shown a linear dependency between the internal vapor concentration rate of change and difference in concentrations in the two environments. Assume that the following equation was obtained in experiments:

VCRPM = 0.01892 x delta_C

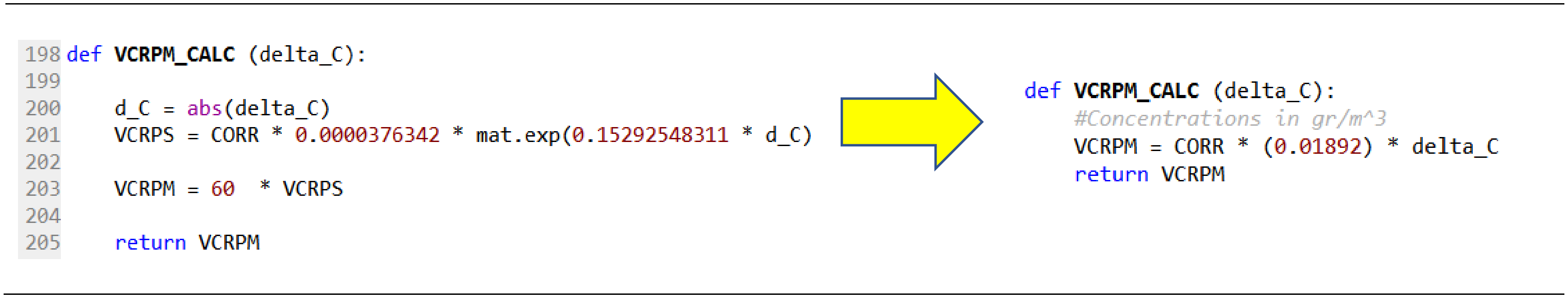
Line 47 of the code should be modified since the correction factor (CORR) is simply 1mm^2 divided by 10mm^2 (1 / 10 = 0.1) which was used in this example. The change we make in the code should look like this:



```
CORR = area * (0.1169590643)
```

```
CORR = area * (0.1)
```

Lines 198 to 205 (permeability function) should be changed as well according to the new experimental findings:



```
198 def VCRPM_CALC (delta_C):
199
200     d_C = abs(delta_C)
201     VCRPS = CORR * 0.0000376342 * mat.exp(0.15292548311 * d_C)
202
203     VCRPM = 60  * VCRPS
204
205     return VCRPM
```

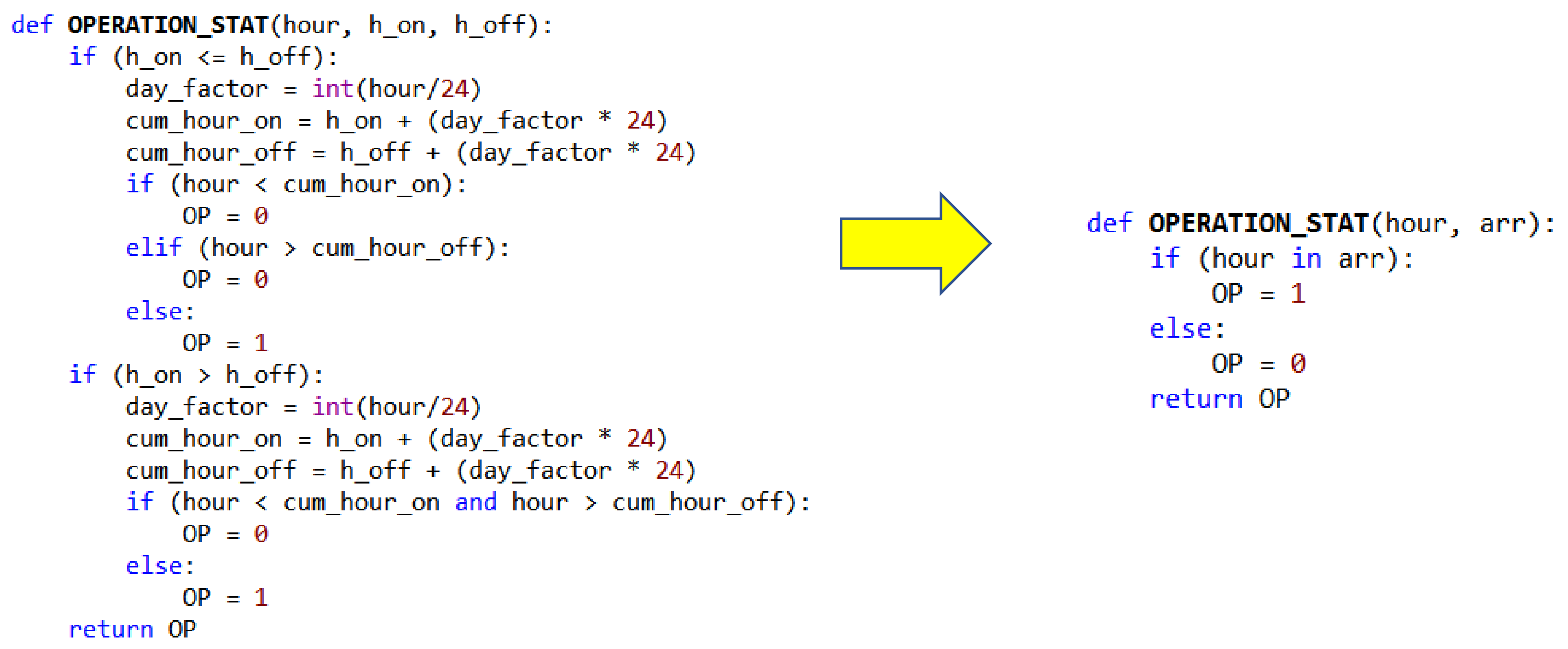
```
def VCRPM_CALC (delta_C):
    #Concentrations in gr/m^3
    VCRPM = CORR * (0.01892) * delta_C
    return VCRPM
```

C. Customizing operative hours

This code applies a constant daily schedule for an entire year. To change this default assumption, you may introduce an array of hours were the LED is expected to be in operation. To do so, you will need to remove lines 51 to 191 that are liable for obtaining and validating the time schedule. Furthermore, the "OPERATION_STAT()" function shall be modified. Let us assume that we want to have the LED on for hours 33, 34, 35, 93, 94, 95, 155, 156, 157, 450, 451. After omitting lines 51 through 191, we write this command therein:

```
arr = [33, 34, 35, 93, 94, 95, 155, 156, 157, 450, 451]
```

Afterwards, we update the "OPERATION_STAT()" function (lines 236 through 255) , as shown below:

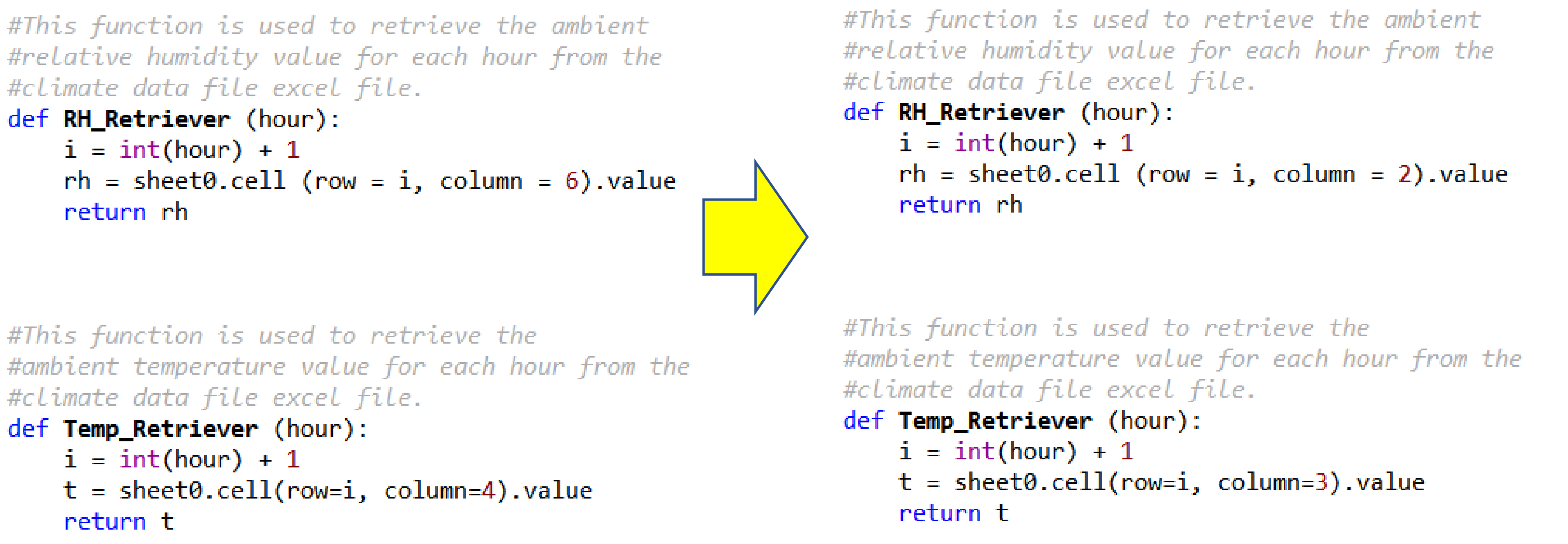


```
def OPERATION_STAT(hour, h_on, h_off):
    if (h_on <= h_off):
        day_factor = int(hour/24)
        cum_hour_on = h_on + (day_factor * 24)
        cum_hour_off = h_off + (day_factor * 24)
        if (hour < cum_hour_on):
            OP = 0
        elif (hour > cum_hour_off):
            OP = 0
        else:
            OP = 1
    if (h_on > h_off):
        day_factor = int(hour/24)
        cum_hour_on = h_on + (day_factor * 24)
        cum_hour_off = h_off + (day_factor * 24)
        if (hour < cum_hour_on and hour > cum_hour_off):
            OP = 0
        else:
            OP = 1
    return OP
```

```
def OPERATION_STAT(hour, arr):
    if (hour in arr):
        OP = 1
    else:
        OP = 0
    return OP
```

D. Getting it to work with different climate data file formats

The only data that is of interest for this simulation is “Temperature” and “Relative Humidity” for a consecutive list of hours. If a climate data file does not follow the format demonstrated in Figure 1, user have to change the code in order to access that information. In the standard CDF, temperature values lay in column 4 and relative humidity data is in column 6. If the climate file follows a different pattern, “RH_Retriever” and “Temp_Retriever()” functions (lines 279 through 282, and 288 through 291 respectively) should be changed. For example, if there’s a climate file with relative humidity located in column 2, and temperature in column 3, these functions shall be changed as shown below:

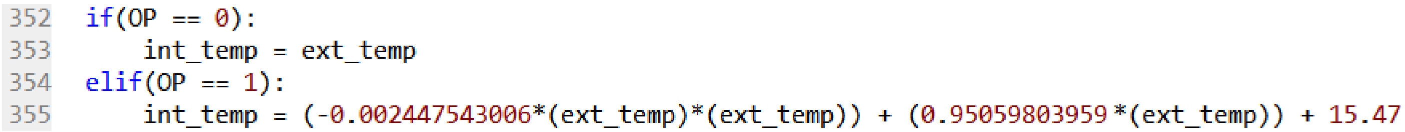


```
#This function is used to retrieve the ambient
#relative humidity value for each hour from the
#climate data file excel file.
def RH_Retriever (hour):
    i = int(hour) + 1
    rh = sheet0.cell (row = i, column = 6).value
    return rh


#This function is used to retrieve the
#ambient temperature value for each hour from the
#climate data file excel file.
def Temp_Retriever (hour):
    i = int(hour) + 1
    t = sheet0.cell(row=i, column=4).value
    return t
```

```
#This function is used to retrieve the ambient
#relative humidity value for each hour from the
#climate data file excel file.
def RH_Retriever (hour):
    i = int(hour) + 1
    rh = sheet0.cell (row = i, column = 2).value
    return rh


#This function is used to retrieve the
#ambient temperature value for each hour from the
#climate data file excel file.
def Temp_Retriever (hour):
    i = int(hour) + 1
    t = sheet0.cell(row=i, column=3).value
    return t
```

E. Modifying the thermal profile of the LED unit

Knowing the thermal profile of the LED is vital, since the model calculates the changes in vapour concentration, as per theory of mass diffusion through a permeable medium. To convert the vapour concentrations into relative humidity, knowing the temperature is a must. Temperature profile of the LED unit was characterized experimentally and was used in the code (lines 352 to 355). Lines 353 and 355 are used to estimate the terminal air temperature within the LED housing (int_temp) as a function of the ambient air temperature (ext_temp) when the LED is off, and on, respectively.

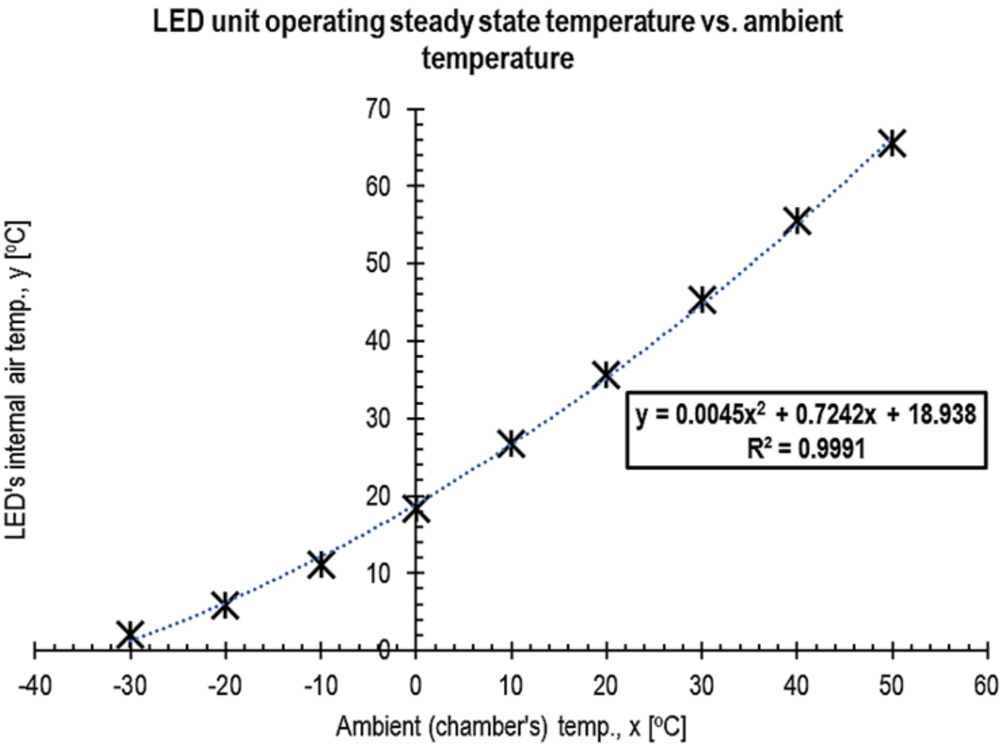


```
352  if(OP == 0):
353      int_temp = ext_temp
354  elif(OP == 1):
355      int_temp = (-0.002447543006*(ext_temp)*(ext_temp)) + (0.95059803959 *(ext_temp)) + 15.47
```

There might be several motivations to alter the temperature profile used in the code. For example, if the LED's power output is changed or the number of LED lights within the LED array is changed, the thermal profile must be acquired experimentally, and the code must be updated accordingly. Another example can be detecting the local condensation on a specific electric component on the PCB board of the LED. If the user is interested in finding the local relative humidity around a certain electric component, the thermal profile of the component should be studied first, and then reflected in the code.

But, how can you obtain the thermal profile of an LED unit (or an electric component inside the unit) experimentally? To do so, a controlled environment chamber shall be used. The LED must be turned on and placed in the chamber with a specific temperature. If the chamber's air circulation is significant, use a thermally conductive wind protection cover around the LED. Wait for 30 minutes to ensure the steady state terminal temperature is reached within the LED housing. Then, record the steady state LED temperature (or record the temperature of an electric component inside the unit) at various chamber temperatures and tabulate the results as shown below. Afterwards, create a scatter plot that shows the LED's internal temperature (or temperature of an electric component inside the unit) as a function of the ambient (chamber's) temperature. From the plot, find the best mathematical equation that fits the data points with a reliability of at least **R^2 >= 0.95**. After obtaining the mathematical equation (model), place that into line 355 of the code. For example, for the sample profile shown below, the command that is inserted in line 355 would be: **int_temp = (0.0045 * ext_temp * ext_temp) + (0.7242 * ext_temp) + 18.938**.

Experiment No.	Chamber's Temperature [°C]	LED's Internal Air Temperature [°C]
1	-30	+2.1
2	-20	+5.9
3	-10	+11.2
4	0	+18.4
5	+10	+26.8
6	+20	+35.7
7	+30	+45.4
8	+40	+55.6
9	+50	+65.5



Advantages and Limitations

The code is simple, quick and robust and can simulate the internal relative humidity with a good accuracy. However, the vapor permeability behaviour of the Gore-Tex® vents were observed to change when exposed to high humidity environments for a prolonged duration of time. The thermal profile of the LEDs that was used for this code does not account for wind. Therefore, the error is expected to increase with blow of an inbound or cross wind conditions.

Future Support

If you faced any issues using this software, or you have any questions or concerns regarding the code, please do not hesitate to contact us through email or phone.

Sanjeev Chandra, PhD, P.Eng., FAAAS, FASME

Professor, Mechanical Engineering

Laboratory: Centre for Advanced Coating Technologies (CACT)

Email: chandra@mie.utoronto.ca | **Tel:** 416-978-5742 | **Office:** BA8254

Website: http://www.mie.utoronto.ca/faculty/chandra