Project 1: Qt GUI Development

Kaylin Punotai

2 Mar 2021

Part 1. Implementation/Assumption Notes

Implementation:

- Qt GUI connected Python that shows temperature and humidity readings
- Gets 1 or 10 readings, displays statistics (max, min, avg), and converts F to C
- Uses MariaDB to store readings and their timestamps
- Alarm text indicates when a temperature or humidity exceeds user-defined alarm values

Assumptions:

- Temperature range is [-20, 100]F
- Humidity range is [0, 100]%
- PseudoSensor code (from Week 1) randomizes data within these ranges

Part 2. Code: TempHum.py

```python
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'TempHum.ui'
#
# Created by: PyQt5 UI code generator 5.11.3
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(510, 569)
        self.CelsiusRadio = QtWidgets.QRadioButton(Dialog)
        self.CelsiusRadio.setGeometry(QtCore.QRect(320, 410, 119, 27))
        self.CelsiusRadio.setObjectName("CelsiusRadio")
        self.FahrenheitRadio = QtWidgets.QRadioButton(Dialog)
        self.FahrenheitRadio.setGeometry(QtCore.QRect(320, 380, 119, 27))
        self.FahrenheitRadio.setChecked(True)
        self.FahrenheitRadio.setObjectName("FahrenheitRadio")
        self.HumLabel = QtWidgets.QLabel(Dialog)
        self.HumLabel.setGeometry(QtCore.QRect(290, 80, 68, 22))
        self.HumLabel.setObjectName("HumLabel")
        self.TempLabel = QtWidgets.QLabel(Dialog)
        self.TempLabel.setGeometry(QtCore.QRect(80, 80, 101, 22))
        self.TempLabel.setObjectName("TempLabel")
        self.TempValue = QtWidgets.QTextBrowser(Dialog)
        self.TempValue.setGeometry(QtCore.QRect(80, 100, 151, 31))
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)
```

This code is auto-generated by Qt for TempHum.ui

The UI is set up by defining all elements and their properties

Part 2. Code: pseudoSensor.py

This code is from Week 1 and generates random humidity and temperature readings

```python
import random

class PseudoSensor:

    h_range = [0, 20, 20, 40, 40, 60, 60, 80, 80, 90, 70, 70, 50, 50, 30, 30, 10, 10]

    t_range = [-20, -10, 0, 10, 30, 50, 70, 80, 90, 80, 60, 40, 20, 10, 0, -10]

    h_range_index = 0

    t_range_index = 0

    humVal = 0

    tempVal = 0


    def __init__(self):

        self.humVal = self.h_range[self.h_range_index]

        self.tempVal = self.t_range[self.t_range_index]


    def generate_values(self):

        self.humVal = self.h_range[self.h_range_index] + random.uniform(0, 10);

        self.tempVal = self.t_range[self.t_range_index] + random.uniform(0, 10);

        self.h_range_index += 1

        if self.h_range_index > len(self.h_range) - 1:

            self.h_range_index = 0

        self.t_range_index += 1

        if self.t_range_index > len(self.t_range) - 1:

            self.t_range_index = 0

        return self.humVal, self.tempVal

```

Each index has a starter magnitude

Generated values are randomized based on the starter magnitude at each index

Part 2. Code: main.py

```
1   import sys, time
2   from datetime import datetime
3   from psuedoSensor import PseudoSensor
4   from PyQt5 import QtWidgets
5   from PyQt5.QtWidgets import QDialog, QApplication
6   from TempHum import Ui_Dialog
7   import mariadb
8
9   # Sensor readings from Week 1
10  ps = PseudoSensor()
11
12  # Alarms messages that show when readings exceed chosen maximums
13  TempAlarmText = "Temperature Alarm!"
14  HumAlarmText = "Humidity Alarm!"
15
16  # Connection to MariaDB
17  db = mariadb.connect(
18                  user="root",
19                  password="pi",
20                  host="localhost",
21                  port=3306,
22                  database="TempHum")
23  cur = db.cursor()
24  # Create a temporary table to log index, timestamp, temp, and humidity
25  cur.execute("CREATE TEMPORARY TABLE results (idx INT, timestamp DATE, temperature FLOAT, humidity FLOAT)")
26  idx = 0
```

Import required classes for timing, pseudoSensor data, Qt connection, and MariaDB connection

These alarm messages show blank on the UI until the alarm is triggered

A temporary table in MariaDB is used to store readings

Stored data includes index, timestamp, temperature, and humidity

```
27
28  class AppWindow(QDialog):
29      def __init__(self):
30          # Initialize UI
31          super().__init__()
32          self.ui = Ui_Dialog()
33          self.ui.setupUi(self)
34          self.show()
35
36          # Button click connects
37          CloseButton = self.ui.CloseButton
38          CloseButton.clicked.connect(QApplication.instance().quit)
39
40          SingleButton = self.ui.SingleButton
41          SingleButton.clicked.connect(lambda: self.getValues(1))
42
43          MultiButton = self.ui.MultiButton
44          MultiButton.clicked.connect(lambda: self.getValues(10))
45
46          StatsButton = self.ui.StatsButton
47          StatsButton.clicked.connect(self.calcStats)
48
49          Fradio = self.ui.FahrenheitRadio
50          Fradio.toggled.connect(self.convertUnits)
51
```

Initialize UI and obtain defs

Connect all buttons to functions

```
52      # Get values either once or 10 times, depending on button click
53      def getValues(self, iterations):
54          for i in range(iterations):
55              # Generate readings from Week 1 Pseudosensor
56              h,t = ps.generate_values()
57              self.storeValue(h,t)
58              # After each reading, execution pauses for 1 sec
59              time.sleep(1)
60              # For multiple readings, only final reading will show on the UI
61
62          # Report humidity
63          hum = str(h)
64          self.ui.HumValue.setText(str(hum))
65          maxH = float(self.ui.AlarmHum.text())
66          # If humidity exceeds maximum humidity, an alarm will show
67          if h > maxH:
68              self.ui.HumAlarm.setText(HumAlarmText)
69          else:
70              self.ui.HumAlarm.setText("")
71
72
73          # Report temperature, converting units if necessary
74          if self.ui.FahrenheitRadio.isChecked():
75              temp = str(t)
76          elif self.ui.CelsiusRadio.isChecked():
77              tempC = (t-32)*5/9
78              temp = str(tempC)
79
80          self.ui.TempValue.setText(str(temp))
81          maxT = float(self.ui.AlarmTemp.text())
82          # If temperature exceeds maximum temp, an alarm will show
83          if t > maxT:
84              self.ui.TempAlarm.setText(TempAlarmText)
85          else:
86              self.ui.TempAlarm.setText("")
```

getValues is used for single (1) and multi (10) data grabs

With each iteration, readings are generated and stored in DB

For the 10 data reads, there is a 1 second sleep pause between readings

Humidity reading is displayed

If it exceeds the user-defined alarm maximum, the alarm message will display

Temperature units are checked and converted first based on radio button checks

Then the same happens as the humidity reading

storeValue adds the time and readings to DB

```
88      # Each reading is stored to MariaDB
89      def storeValue(self, h, t):
90          global idx
91          currentTime = datetime.now()
92          cur.execute("INSERT INTO results VALUES (?, ?, ?, ?)", (idx, currentTime, t, h))
93          idx += 1
94
```

A global idx keeps track of table rows

calcStats pulls 10 entries from the DB and calculates the min, max, and average

```
95      # Calculate min, max, and average of all readings
96      def calcStats(self):
97          temp = 0.0
98          hum = 0.0
99          tempSum = 0.0
100         humSum = 0.0
101         tempMax = -20.0
102         humMax = 0.0
103         tempMin = 100.0
104         humMin = 100.0
105         i = 0
106         numRows = 0
```

Max temp and max hum are initially set at the minimum of their range

Min temp and min hum are initially set at the maximum of their range

```
108         # If there are fewer than 10 entries, get stats from all entries
109         if idx < 10:
110             for i in range(idx):
111                 cur.execute("SELECT temperature FROM results WHERE idx=?", (i,))
112                 for temperature in cur:
113                     tempstr = str(temperature).strip("(,)")
114                     if tempstr.startswith("-"):
115                         tempstr1 = tempstr.strip("-")
116                         temp = -1*float(tempstr1)
117                     else:
118                         temp = float(tempstr)
119                     # Check max
120                     if temp > tempMax:
121                         tempMax = temp
122                     # Check min
123                     if temp < tempMin:
124                         tempMin = temp
125                     # Add for average
126                     tempSum += temp
127                 cur.execute("SELECT humidity FROM results WHERE idx=?", (i,))
128                 for humidity in cur:
129                     humstr = str(humidity).strip("(,)")
130                     hum = float(humstr)
131                     # Check max
132                     if hum > humMax:
133                         humMax = hum
134                     # Check min
135                     if hum < humMin:
136                         humMin = hum
137                     # Add for average
138                     humSum += hum
```

When there are <10 entries, every entry is used to calculate stats

Each entry is queried and cast as FLOAT

The hyphen in string does not translate well to float, so negative entries have to be formatted

Each entry is compared to the existing max and min

Total sum is added for average calculation at the end

Same manipulations are done for humidity

```
139         # If there are more than 10 entries, only get stats from last 10 entries
140         else:
141             cur.execute("SELECT COUNT(*) from results")
142             for count in cur:
143                 numRows = int(str(count).strip("(,)"))
144             for i in range(10):
145                 cur.execute("SELECT temperature FROM results WHERE idx=?", (numRows-i-1,))
146                 for temperature in cur:
147                     tempstr = str(temperature).strip("(,)")
148                     if tempstr.startswith("-"):
149                         tempstr1 = tempstr.strip("-")
150                         temp = -1*float(tempstr1)
151                     else:
152                         temp = float(tempstr)
153                     # Check max
154                     if temp > tempMax:
155                         tempMax = temp
156                     # Check min
157                     if temp < tempMin:
158                         tempMin = temp
159                     # Add for average
160                     tempSum += temp
161                 cur.execute("SELECT humidity FROM results WHERE idx=?", (numRows-i-1,))
162                 for humidity in cur:
163                     humstr = str(humidity).strip("(,)")
164                     hum = float(humstr)
165                     # Check max
166                     if hum > humMax:
167                         humMax = hum
168                     # Check min
169                     if hum < humMin:
170                         humMin = hum
171                     # Add for average
172                     humSum += hum
```

For >10 entries, last 10 entries are used for stats

The number of entries is queried, then the indexing queries starting with the latest entry

Same manipulations as <10 entries above for temp and hum

At the end, averages are calculated based on total sums and number of entries used

```
173
174         # Display results in UI
175         avgTemp = tempSum/idx
176         avgHum = humSum/idx
177         self.ui.TempAvg.setText(str(avgTemp))
178         self.ui.HumAvg.setText(str(avgHum))
179         self.ui.TempMax.setText(str(tempMax))
180         self.ui.HumMax.setText(str(humMax))
181         self.ui.TempMin.setText(str(tempMin))
182         self.ui.HumMin.setText(str(humMin))
```

UI texts are updated with data

```python
    # Convert temp units whenever radio buttons are toggled
    def convertUnits(self):
        try:
            t = float(self.ui.TempValue.toPlainText())
        except:
            t = 1000
        try:
            maxT = float(self.ui.TempMax.toPlainText())
            minT = float(self.ui.TempMin.toPlainText())
            avgT = float(self.ui.TempAvg.toPlainText())
        except:
            maxT = 1000
            minT = 1000
            avgT = 1000
        alarmT = float(self.ui.AlarmTemp.text())
        unit = ""

        if self.ui.FahrenheitRadio.isChecked():
            unit = "°F"
            tconv = 9/5*t+32
            alarmTconv = 9/5*alarmT+32
            maxTconv = 9/5*maxT+32
            minTconv = 9/5*minT+32
            avgTconv = 9/5*avgT+32
        elif self.ui.CelsiusRadio.isChecked():
            unit = "°C"
            tconv = (t-32)*5/9
            alarmTconv = (alarmT-32)*5/9
            maxTconv = (maxT-32)*5/9
            minTconv = (minT-32)*5/9
            avgTconv = (avgT-32)*5/9

```

convertUnits converts temperature between Fahrenheit and Celsius whenever the radio buttons are toggled

If any fields are blank, I set their values to 1000 so that they can still be evaluated without errors

Check which radio button is clicked then converts all temperatures

```
216
217             if t != 1000:
218                 self.ui.TempValue.setText(str(tconv))
219             if maxT != 1000:
220                 self.ui.TempMax.setText(str(maxTconv))
221             if minT != 1000:
222                 self.ui.TempMin.setText(str(minTconv))
223             if avgT != 1000:
224                 self.ui.TempAvg.setText(str(avgTconv))
225             self.ui.AlarmTemp.setText(str(alarmTconv))
226             self.ui.TempUnits.setText(unit)
227             self.ui.TempUnits_2.setText(unit)
228             self.ui.TempUnits_3.setText(unit)
229             self.ui.TempUnits_4.setText(unit)
230             self.ui.TempUnits_5.setText(unit)
231
```

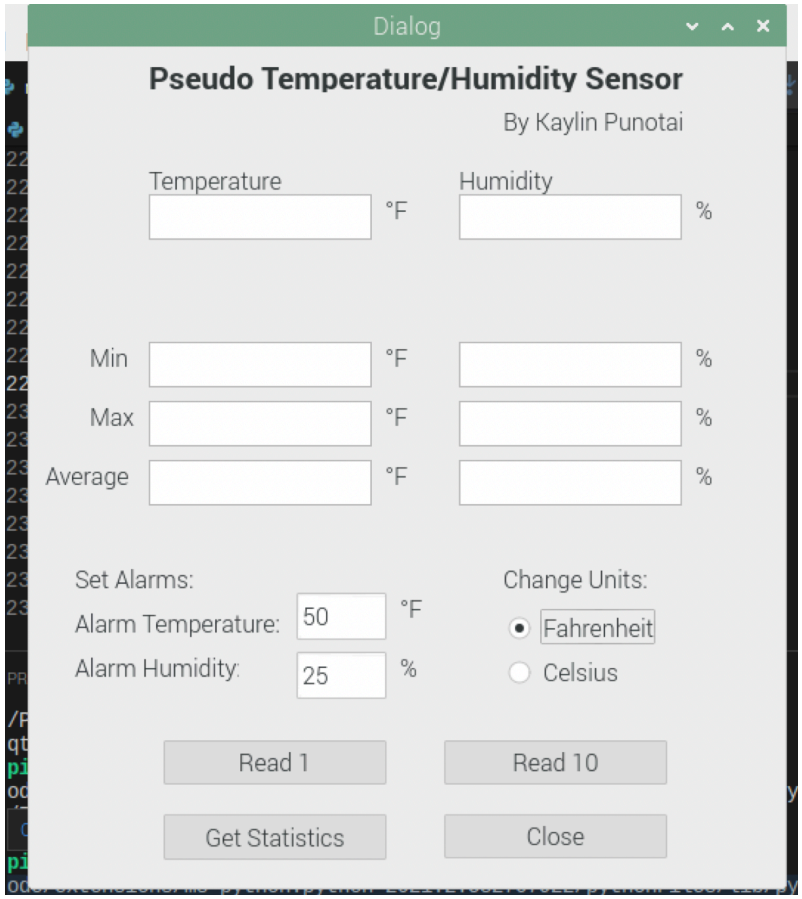
For any blank fields (which I set to 1000), they are kept blank. If they're not 1000, those temperatures are updated to their converted values

All other temps are updated, and units are changed to reflect the clicked radio button choice

```
231
232  # Open window
233  app = QApplication(sys.argv)
234  w = AppWindow()
235  w.show()
236  sys.exit(app.exec_())
237
```

Opens window to show the UI

Part 3. Qt UI Demonstration



UI at startup

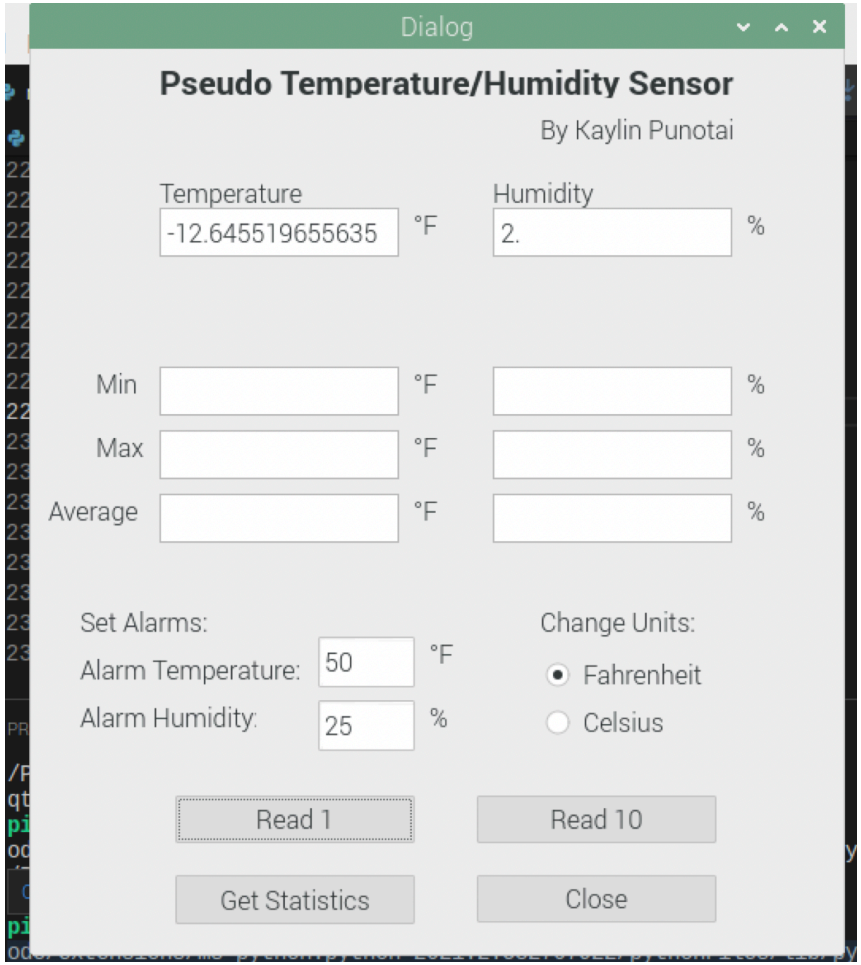
These fields cannot be edited and will be populated when data entries are requested

Alarms are user-defined but have startup values

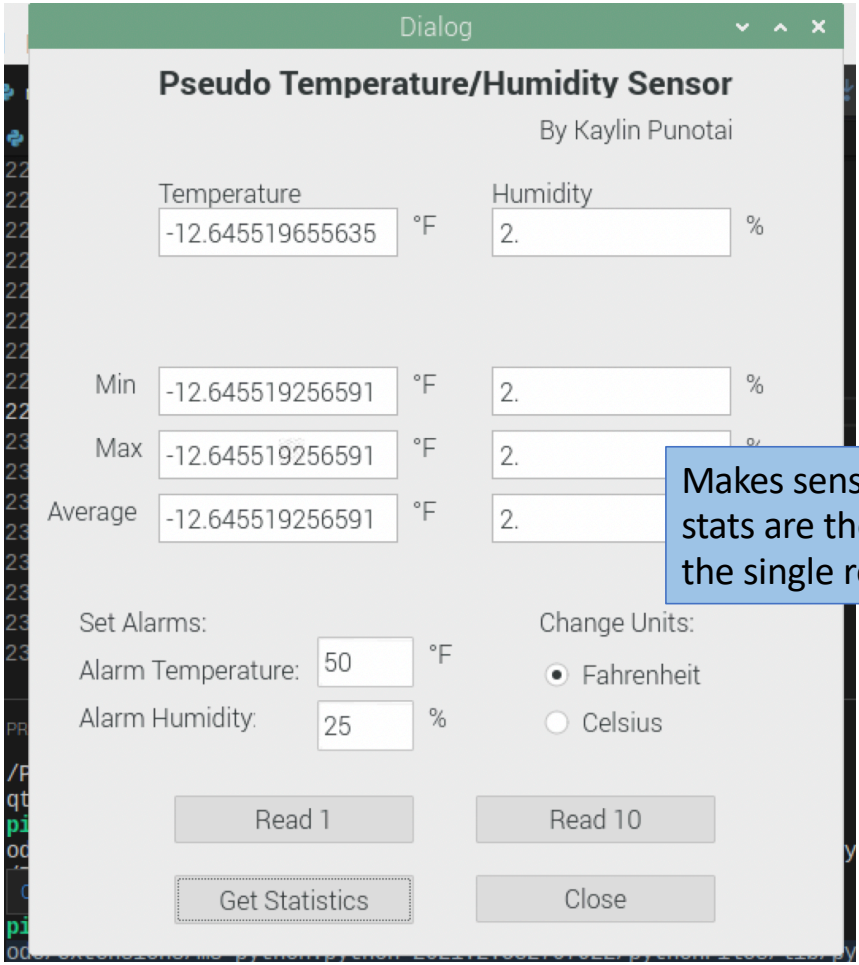
Units can be converted from F to C

Buttons for getting 1 or 10 readings, getting stats, and closing UI

Getting 1 reading:

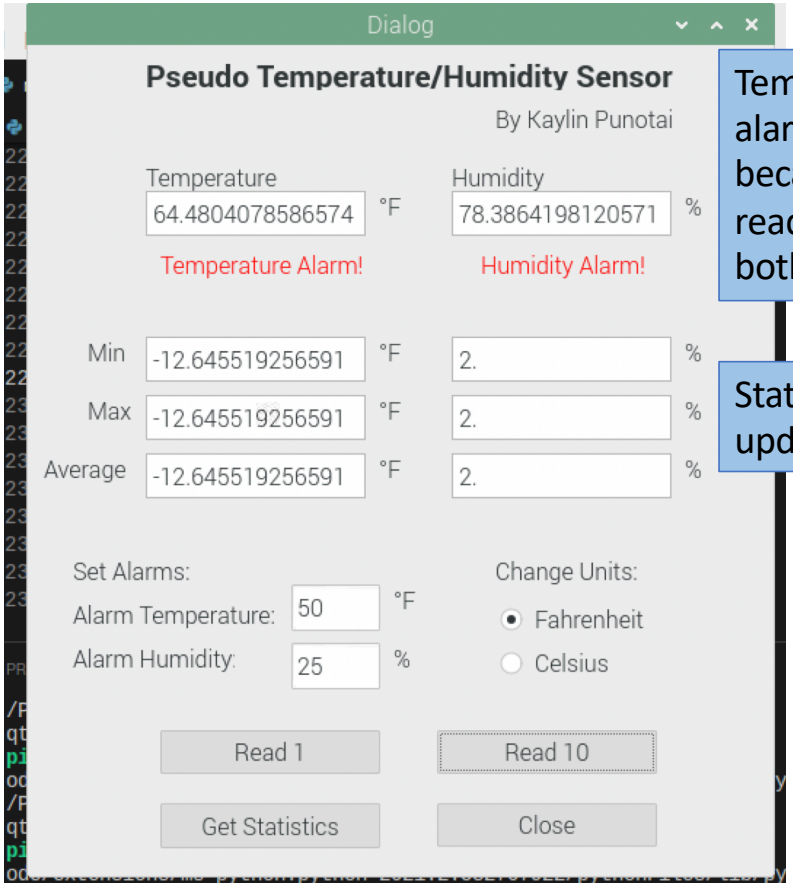


Getting stats for 1 reading:



Makes sense that all stats are the same as the single reading

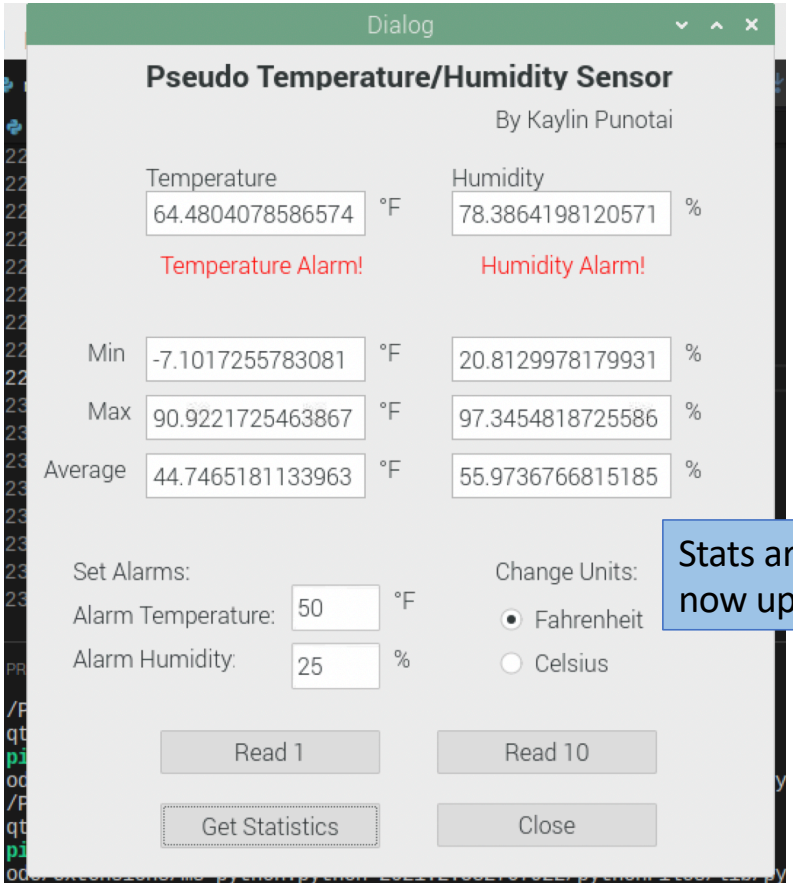
Getting 10 additional entries:



Temp and hum alarms show because the last reading exceed both alarm values

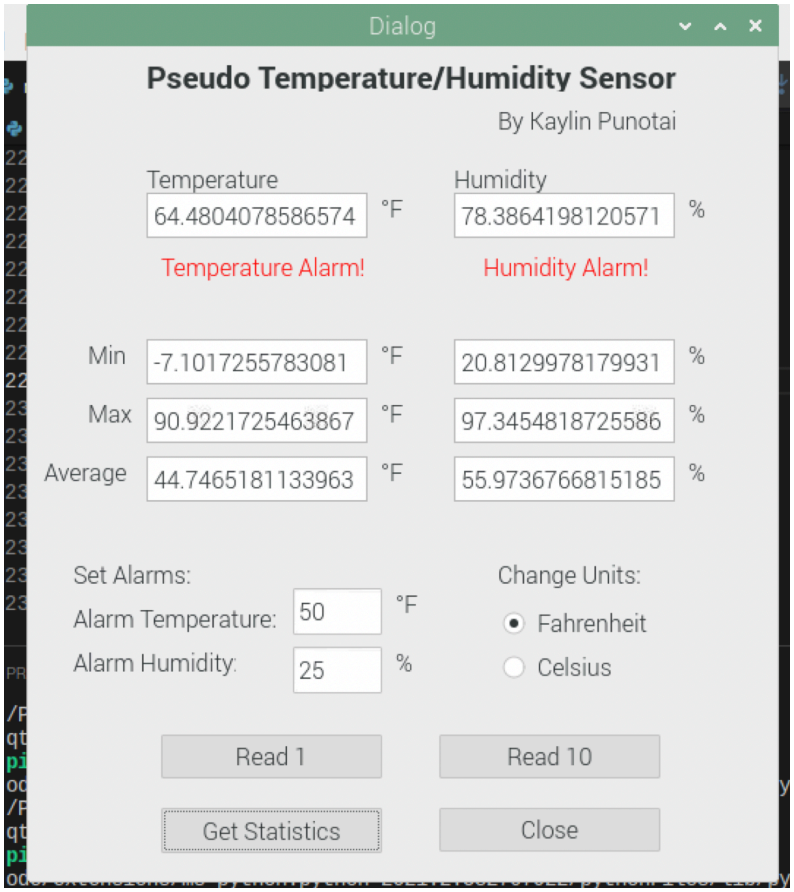
Stats are not updated yet

Getting stats for last 10 entries:



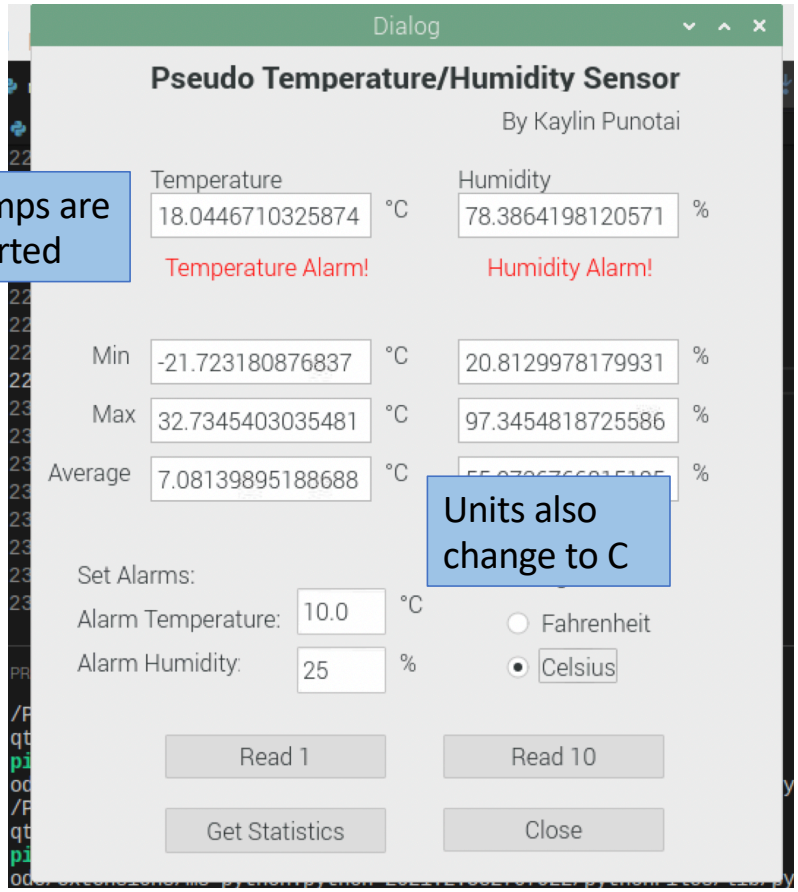
Stats are now updated

Same picture above (after getting stats for 10 entries)



Changing units to C:

All temps are converted



Units also change to C