```
#! /usr/bin/python

"""  Chatterbox7.py

Chatterbox7.py was edited by M Wollenberg from a script of the same
name (Chatterbox.py) written by Peter Balfe
(U. Birmingham)and retrieved from his lab's website during the spring
of 2017.

Chatterbox7.py is coded in Python 2.7 to run on a Raspberry Pi (RPi)
that has a slice of radio
and SRF chip (made by Ciseco in 2016 and sold under the product name
RF01 from PrivateEyePi in 2016)
attached to port ttyAMA0. The SRF chip detects RF communication from
other SRF chips (sold as product
RF22 from PrivateEyePi) at 9600 baud via the serial port of the RPi.
Each of these other chips are
attached to one 10K thermistor (NTCLE100E3103JB0) that measure
temperature of -80C freezers in
our department - these temp chips have their firmware reset for Analog
functionality.
The thermistors return voltage measurements to the script. These
voltage measurements were empirically calibrated
for each SRF/thermistor combination.

The script monitors thermistor voltage measurement communication from
the SRF chip(s) and completes
three actions with these readings.

1) Each reading is collected, converted to temperature, and used to
create a graph of temperature over time
for each freezer - this graph is emailed as an attachment to a gmail
account on a weekly basis.

2) Every thermistor/temp reading from every sensor is tweeted to a
twitter account.

3) If the temperature readings go above (the freezer warms to a
temperature above) a certain
threshold value, the script sends a warning email to a gmail account.
This account has been set
up to forward particular emails to particular email addresses and/or
cell phones. Currently, rather than a local variable storing
the gmail password, I've opted to write the script such that the user
has to put the pass into the script four or so times, separately...
I feel this would make it marginally less likely for some random user
to grab the password, but obviously, it is still visible.

"""
```

```
#----------------------------------
# IMPORTS
#
# sudo apt-get install python-matplotlib to install
# use matplotlib for the figure :
import matplotlib as mpl
# turn off plotting window
mpl.use('Agg')
# pull in the plot library
import matplotlib.pyplot as plt
#
# if you need to install pyserial, use the command: sudo apt-get
install python-serial
# use serial for serial port communication
import serial
# import time functions
import datetime
from datetime import date
import time
# use smtplib for sending email alerts & base64/email for plot mime
encoding
import smtplib
#import base64
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText
from email.MIMEBase import MIMEBase
from email import encoders

# use twython to handle temperature tweets
import twython
from twython import Twython

# sudo apt-get install python-prettytable to install
# use prettytable for easy table creation
from prettytable import PrettyTable

# use os to write plotfile to remote websites (requires RSA keypair
encryption to be set up for root!)
# not necessary - MSW
#import os
#import csv
#
#-----------------------------
# SETTINGS
#
# If = 1 then alarm function works
DoAlarm = 1
#
# set up serial port for temperature readings
DEVICE = '/dev/ttyAMA0'
```

```
BAUD = 9600
#
# Reading collection and plotting scheduling.
# Scheduling depends on the length of time between readings of the
temp.
# Length of time between readings is set via LLAP (use setCycle.py)!!!
#
# (default collection settting = 336 = one read per hour, 24 hours, 7
days)
collection = 336
# Day of the week that the report is sent 0 = Monday 6 = Sunday; use
int
DayWeek = 6
# Hour of the day the report is sent = 0-23 hours (12am through 11pm);
use int
HourDay = 20
#

# Set SMTP mailing details for email(s).
# For multiple addresses, use a list and the below code.
whereto = 'smtp.gmail.com'
fromaddr = #gmail address goes here
toaddr = #gmail address goes here
# Credential for username
username = #gmail username goes here

# Twitter settings stored in the twitter variable via Twython
twitter = Twython(
    # Twython settings with single quotes ' '
    #ck, cks, at, ats
)

# Setting the different linear models for voltage to temp best fit for
each SRF/thermistor pairing, as necessary - two given.

#a01 first pairing
m1 = -2.9747
b1 = 5825

#a02 second pairing
m2 = -6.4615
b2 = 12755

#
# END OF SETTINGS
#

# This function sends a .png matplotlib plot as an attachment
# via email to the above address(es) via the smtp server above.
```

```
def sendplot(MyMsg, channel):
    filename = "Reader"+channel+".png"
    msg = MIMEMultipart()
    msg ['From'] = fromaddr
    msg ['To'] = toaddr
    msg ['Subject'] = MyMsg
    body = MyMsg+' weekly report with attached .png file.'

    msg.attach(MIMEText(body, 'plain'))

    attachment = open(filename, "rb")
    p = MIMEBase('application', 'octet-stream')
    p.set_payload((attachment).read())
    encoders.encode_base64(p)
    p.add_header('Content-Disposition', 'attachment; filename=%s' %
filename)
    msg.attach(p)

    server = smtplib.SMTP(whereto, 587)
    # login required by gmail server
    server.ehlo()
    server.starttls()
    server.ehlo()
    server.login(username,#gmail password goes here in single quotes '
'
                 )
    server.sendmail(fromaddr, toaddr, msg.as_string())
    server.quit()
    now = datetime.datetime.now()
    print "\nThe plot email, "+body+" was sent on "+str(now)

"""

TEMPERATURE MONITORING PROGRAM.

Data collection program for thermistor devices previously sent into
cyclic sleep mode by program "Set-cycle.py". Use "CTRL+Z" to stop/quit
this program.

You often get a buffer-full of readings when the program starts up or
when the battery readings are sent, this is normal.

"""

# Text from above that was removed...is copied below
# Sometimes the monitor fails to connect properly after the first
reading
# so this screen just stays frozen. The cause is unclear, but the
solution is to quit and relaunch.
# In extremis a full reboot might be needed!
```

```
# Parameter reader from a textfile of freezer variables;
# Reads in device settings from file "settings.txt".
# The variables must have the below format.
# ID, Freezer Name, Min, Max
# AA, "My Freezer", -85, -60
#
#print "\nOpening the settings.txt file..."
#f = open('settings.txt', 'rb')
#try :
    #csvFile = csv.reader(f) for row in csvFile:
    #monitors[Col1].append(row[0])
    #monitors[Col2].append(row[1])
    #monitors[Col3].append(row[2])
    #monitors[Col4].append(row[3])
#finally :
    #f.close()

#Make a dictionary of freezer variables
Col1 = 'ID'
Col2 = 'Name'
Col3 = 'low'
Col4 = 'high'

# Manual entry of device settings; spacing is for aesthetic reasons
only.
monitors={Col1:['01', '02'], Col2:['Olaf     (Left)', 'J.Frost
(Right)'], Col3:[-90, -90], Col4:[-50, -50]}
tTable = PrettyTable(["ID","Name","Low","High","Mid"])

# All of the below is in service of creating a nice table (and nice
graphs in matplotlib)
#
# Create two lists, length as long as the number of IDs in monitors
middlevalue = [0] * len(monitors[Col1])
y_series = [0] * len(monitors[Col1])
i=0
while i < len(monitors[Col1]) :

    # The " " string below is 30 characters long to explicitly blank
end of the string.
    # The below two lines are not important wiht prettytable.
    # not sure why the index starts at 1
    #monitors[Col2][i] = monitors[Col2][i][0:len(monitors[Col2][i])+1]
+ "                "
    #monitors[Col2][i] = monitors[Col2][i][0:15]

    # middlevalue is the average of the low and high values for a
particular ID/Name
```

```
    middlevalue[i] = int(monitors[Col4][i]) - (int(monitors[Col4][i])
- int(monitors[Col3][i]))/2
    # For each device in the above list : multiply the number of
readings by the middle value and store in y-series
    y_series[i] = [middlevalue[i]] * collection
    # Add a row to the table that includes middlevalue as well as all
the other variables.
    tTable.add_row([monitors[Col1][i],monitors[Col2][i],monitors[Col3]
[i],monitors[Col4][i],middlevalue[i]])
    #print monitors[Col1][i],monitors[Col2][i],monitors[Col3]
[i],monitors[Col4][i],middlevalue[i]
    i=i+1
    # finished loop

# Display the table settings to the terminal
print "\nThermistor Settings Table :"
print tTable
#
# Create a list of x axis values that ascend from 0 to (1 -
collection).
# In the case of weekly monitoring every hour, this will be 0 to 335
x_series = range(collection)
# Declare a 1 element array for temperature (needed for array
addition)
temp = [0]
# Set battery level strings to "????"
battlevels = ["?????"] * (len(monitors[Col1]))
# End variable set up

# Read the time
now = datetime.datetime.now()
# Set the clocks deliberately wrong to trigger the first thermometer
recordings.
# This allows for different refresh rates for each channel to be set
if we wish.
# Not sure we need the below
thishour = now.hour - 1
#clocks = [thishour] * (len(monitors[Col1]))

# Open up the serial connection
print "\nOpening connection and waiting for response..."


ser = serial.Serial(DEVICE, BAUD)
ser.timeout = 0
# Clear the buffer
ser.flushInput()
print "...serial port opened.\n"
# If all goes well
```

```
msg = 'Temperature monitor initialised : ' + now.strftime("%H:%M %m-
%d-%Y")
print "Startup complete!  "+msg
#
# Initial mail sent on startup. Don't use the sendplot function as
"ReaderXX.png" doesn't exist yet!
#
# Try to email in case the server is down...
try :
    msg2 = MIMEMultipart()
    msg2['From'] = fromaddr
    msg2['To'] = toaddr
    msg2['Subject'] = 'Freezer Monitor: Temperature Startup'
    msg2.attach(MIMEText(msg))
# gmail variable is 587 here
    server = smtplib.SMTP(whereto, 587)
# secure login required by gmail smtp server, uses starttls
    server.ehlo()
    server.starttls()
    server.ehlo()
    server.login(username,#gmail password goes here in single quotes '
'
                )
    server.sendmail(fromaddr, toaddr, msg2.as_string())
    server.quit()
    now = datetime.datetime.now()
    print "\nStartup email sent to "+toaddr+" at "+str(now)
except :
    now = datetime.datetime.now()
    print "\nGmail server returned an error, STARTUP email not sent at
"+str(now)
#end of startup

# Listening program start
print "\n+++ Starting infinite temperature listening loop, use CTRL+z
to quit. +++"
#
#temporary counter for debugging
MyCounter = 0
# Start infinite while loop to listen to SRF module
while 1 :
# All SRF module read and write commands should have 12 characters
(because they are written in LLAP),
# Some LLAP out to terminal may be longer due to startup or other
quirks (else statement at endwhile)
# All SRF module read and write will begin with the letter "a"
#
# Wait for message, the 1 second pause seems to improve the reading
when several messages
# are arriving in sequence, such as: a--ANA19734-a--AWAKE----a--
```

```
BATT2.74-a--SLEEPING-
#
    time.sleep(1)
    leng1 = ser.inWaiting()
    if leng1 == 12:
        llapMsg = ser.read(ser.inWaiting())
        # display packet, helps to troubleshoot any errors
        now = datetime.datetime.now()
        print '\nReceived |'+ llapMsg + '| from ' + monitors[Col2]
[monitors[Col1].index(llapMsg[1:3])] + ' at ' + now.strftime("%H:%M
%m-%d-%Y")
        if "a" == llapMsg[0] :
            # llap msg detected - also for troubleshooting
            #print 'Received llapMsg from ' + llapMsg[1:3]

            # is it one of ours?
            if llapMsg[1:3] in monitors[Col1] :
                # Yes!
                # Identify the sensor device's array channel
                sensor = monitors[Col1].index(llapMsg[1:3])
                #use the right linear model?
                if sensor == 0:
                  m0 = m1
                  b0 = b1
                  print llapMsg[1:3] + " detected, = channel " +
str(sensor)+"mx+b = 1\n"
                elif sensor == 1:
                  m0 = m2
                  b0 = b2
                  print llapMsg[1:3] + " detected, = channel " +
str(sensor)+"mx+b = 2\n"
                # More debugging print
                #print llapMsg[1:3] + " detected, = channel " +
str(sensor)

                # Read the time
                now = datetime.datetime.now()
#
#               Check for BATT(ery) OR ANA(log)/TMP (temperature)
packet, ignore else.
#
                # Is it an ANA(log) or TMP(erature) reading?
                if ('ANA' in llapMsg) or ('TMP' in llapMsg) :
                    # some internal clocking for debugging
                    #MyCounter=MyCounter+1
                    #MyCounter2 =MyCounter%20
                    #print "MyCounter: "+str(MyCounter)+" MyCounter%20
= "+str(MyCounter2)

                    # NOT SURE THE BELOW IS NECESSARY
```

```
                    # Has an hour passed?
                    #if clocks[sensor] != now.hour :
                        # yes,- update the plots
                        #clocks[sensor] = now.hour
                        # during debugging, get this wrong
deliberately by setting:
                        #clocks[sensor] = now.minute

                    # convert string to real number/temperature for
plotting
                    if 'ANA' in llapMsg :

                    # Analog reading from low temp thermistor
                    # looking like this: a--ANA1974--
                    # This conversion is a crude guess of voltage to
temp conversion
                    # based on a linear interpolation between three
values
                    # with the default an alarm temperature occurs
when the
                    # reading falls below 1887, (1887 - 1626)/-4.3 =
-60 #

                        Volt1 = llapMsg[6:10]
                        Volt2 = float(Volt1)
                        try :
                            temp = [(m0*Volt2)+b0]
                            print "Analog reading:" + Volt1 + " =
Temperature: " +  str(round(temp[0],2)) + " C."
                        # temp is a 1 element array, hence the "[" "]"
                        except ValueError :
                            # if float operation fails, skip bad
reading
                            print "\nBad reading of analog Temp data."

                    # Else Temperature reading from conventional
thermistor
                    # Thermistor format via LLAP: a--TMPA23.28 or a--
TEMP23.28
                    else :
                    # Simply directly convert string to number
                        print "\nTemperature reading = " +
llapMsg[7:11]
                        try :
                            temp = [float(llapMsg[7:11])]
                        except ValueError :
                            # if float operation fails, save the
reading as 0
                            temp = [00.00]
                            #
```

```
                        # shuffle the temperature list to the left,
discard oldest
                    y_series[sensor] = y_series[sensor][1:]+temp
                    nowt = datetime.datetime.now()
                    # send a tweet about the temperature
                    message = "Freezer %s is at temperature %s at %s."
% (monitors[Col2][sensor], temp, nowt)
                    try:
                        twitter.update_status(status=message)
                        print "FreezeTweetSent."
                    except :
                        print "Twitter conection not made; No Tweet -
No Cry."

                    # Check if the temperature reading is Out of Range
(and ALARM on).
                    if (int(temp[0]) < int(monitors[Col3][sensor]) or
int(temp[0]) > int(monitors[Col4][sensor])) and DoAlarm :
                    # Freezer is out of range! Send Alarm Email.
                        msg ='ALARM! Monitor reports ' +
str(round(temp[0],2)) + " C for " + monitors[Col2][sensor] + " at " +
now.strftime("%H:%M %m-%d-%Y") + ". Battery level : " +
battlevels[sensor] + " V."
                        print '\n'+msg
                        try :
                            msg2 = MIMEMultipart()
                            msg2['From'] = fromaddr
                            msg2['To'] = toaddr
                            msg2['Subject'] = 'Freezer Monitor:
TEMPERATURE ALARM!'
                            msg2.attach(MIMEText(msg))
                            server = smtplib.SMTP(whereto)
                            server = smtplib.SMTP(whereto, 587)
                            #
                            # secure login required by gmail smtp
server
                            server.ehlo()
                            server.starttls()
                            server.ehlo()
                            server.login(username,#gmail password goes
here in single quotes ' '
                                         )
                            server.sendmail(fromaddr, toaddr,
msg2.as_string())
                            server.quit()
                             print "\nALARM email was successfully
sent!!"
                        except :
                            print "\nMail server returned an error on
ALARM email!"
```

```
                # Is it a battery reading? Not very likely, but for
robustness.
                elif 'BATT' in llapMsg :
                    # Battery reading sent
                    # on our setup this seems to come in as a single
48 byte lump (see below)
                    print "Battery level for " + monitors[Col1]
[sensor] + "is " + llapMsg[7:10] + " V"
                    # Save this value for the plot file
                    battlevels[sensor] = llapMsg[7:10]
                    # new message in later SRF software!
                    if 'BATTLOW' in llapMsg :
                        msg = 'LOW BATTERY WARNING : ' + llapMsg + "
for " + monitors[Col2][sensor] + " at "+ now.strftime("%H:%M %m-%d-
%Y")
                        print msg
                        try :
                            msg2 = MIMEMultipart()
                            msg2['From'] = fromaddr
                            msg2['To'] = toaddr
                            msg2['Subject'] = 'Freezer Monitor:
Battery Low'
                            msg2.attach(MIMEText(msg))
                            server = smtplib.SMTP(whereto, 587)
                            #
                            # secure login required by gmail smtp
server
                            server.ehlo()
                            server.starttls()
                            server.ehlo()
                            server.login(username,#gmail password goes
here in single quotes ' '
                                         )
                            server.sendmail(fromaddr, toaddr,
msg2.as_string())
                            server.quit()
                            print msg
                        except :
                            print "\nMail server returned an error,
LOWBATT email not sent."

            # For debugging
            #if (MyCounter % 10) == 0:

            # Is it time for the weekly report? defined in first part
of script
            # Check the day of the week (i.e. Monday = 0, Sunday = 6,
Friday = 4)
            # for daily reports replace with this:
```

```
            # if (thishour == 12) :
            if (date.weekday(now) == DayWeek) and (thishour ==
HourDay) :
                print "In the printing plot loop."
                # Open plot for data
                plt.figure
                # Set plot label (timestamp)
                freezertime = "Last reading at : " + now.strftime("%H:
%M %m- %d-%Y") + ", battery = " + battlevels[sensor] + "V"
                # Plot x and y values with label
                plt.plot(x_series, y_series[sensor],
label=freezertime)
                # Add in axes legends and title
                plt.xlabel("Time (hours)")
                plt.ylabel("Temperature (approx)")
                plt.xlabel("Time (hours)")
                plt.ylabel("Temperature (approx)")
                plt.title('Past week(s): ' + monitors[Col2][sensor])
                # Set limits of the x and y axes, the extra 5 on the Y
axis
                plt.xlim(0,collection)
                plt.ylim(min(y_series[sensor])-1,
max(y_series[sensor])+5)
                # Set legend position
                plt.legend(loc="upper left")
                # Save figure as a .png graphics file
                PlotName = "Reader" + monitors[Col1][sensor] + ".png"
                plt.savefig(PlotName)
                # Close plot
                plt.clf()

                msg = 'Freezer Monitor: Weekly report ' +
now.strftime("%H:%M %m-%d-%Y") + " for " + monitors[Col2][sensor]
                try :
                    #try to send an email with the plot attached
                    sendplot(msg,monitors[Col1][sensor])
                except :
                    print "\nMail server returned an error when
attempting to send the WEEKLY report and plot."
                # If the temp is only read every hour, this day/hour
combo will occur only once # each week so we don't need to check this
hasn't been sent already.

                #Print the settings table to terminal for the user.
                print "\nThermistor Settings Table :"
                print tTable
                print "\nContinuing infinite temperature listening
loop, use CTRL+z to quit."

    else:
```

```
        # goes with first "if" statement (if ser.inWaiting() == 12: )
        # read buffer has non-12 length string in it,- empty the
buffer!
        if ser.inWaiting() > 0:
            length = ser.inWaiting()
            llapMsg = ser.read(ser.inWaiting())
            now = datetime.datetime.now()
            print '\nReceived a long message from one sensor at ' +
now.strftime("%H:%M %m-%d-%Y")+'\n'
            print llapMsg+'\n'
            # the 48 byte string :
            # a--ANA1973--a--AWAKE----a--BATT2.74-a--SLEEPING-
            # 012345678901234567890123456789012345678901234567
            # is frequently seen
            if 'BATT' in llapMsg:
                # Battery reading sent as part of wake cycle
                # is it one of ours?
                if llapMsg[1:3] in monitors[Col1] :
                    # yes
                    # identify the sensor device's array channel
                    sensor = monitors[Col1].index(llapMsg[1:3])
                    # print llapMsg[1:3] + " detected, = channel ",
sensor
                    # Save the battery reading to the sensor channel.
                    battlevels[sensor] = llapMsg[llapMsg.find('BATT')
+4:llapMsg.find('BATT')+9]
                    print "Battery level is " + battlevels[sensor] + "
V"
                    if 'BATTLOW' in llapMsg:
                        msg = 'LOW BATTERY WARNING : ' + llapMsg + "
for " + monitors[Col2][sensor] + " at "+ now.strftime("%H:%M %m-%d-
%Y")
                        print msg
                        try :
                            msg2 = MIMEMultipart()
                            msg2['From'] = fromaddr
                            msg2['To'] = toaddr
                            msg2['Subject'] = 'Freezer Monitor:
Battery Low'
                            msg2.attach(MIMEText(msg))
                            server = smtplib.SMTP(whereto, 587)
                            #
                            # secure login required by gmail smtp
server
                            server.ehlo()
                            server.starttls()
                            server.ehlo()
                            server.login(username,#gmail password goes
here in single quotes ' '
                                         )
```

```
                            server.sendmail(fromaddr, toaddr,
msg2.as_string())
                            server.quit()
                        except :
                            print "Mail server returned an error,
LOWBATT email not sent."
                ser.flushInput()
#
# For want of a better phrase, end the infinite while loop

# End of Program
```