"""

Title: weather-station.py

Date: January 2016

Author: Jonathan Simpson

Description: This python script collects and displays the data from an Arduino weather station

(see http://ru.wordpress.fos.auckland.ac.nz/arduino-weather-station/). Some daily maximum and minimum

values are also recorded and displayed. Please note the format of data outputted by Arduino (integer for

an anemometer revolution, and 'P', 'T', or 'H' for other data). The architecture and serial port functions

are based upon those developed for the school seismology seismometer software PyjAmaseis (https://github.com/PALab/pyjamaseis).

This script was written on a computer running Ubuntu 14.

"""

import numpy as np

import sys

import time as Time

import platform

from multiprocessing import Queue

from threading import Thread

import serial

import glob

import tkinter as tk

from tkinter import messagebox as msgbx

import datetime as dt

### Class responsible for the main display of the weather station.

class mainWin(tk.Frame):

def \_\_init\_\_(self, queue1, queue2, queue3):

self.root = tk.Tk()

self.root.protocol('WM\_DELETE\_WINDOW', lambda: window\_close(self))

tk.Frame.\_\_init\_\_(self)

self.root.title('Weather Station')

self.root.minsize(width=435, height=580)

self.root.maxsize(width=435, height=580)

self.define\_variables()

self.numberfont = 'Ubuntu Condensed '

self.textfill = 'black'

self.textfont = 'Purisa'

self.headersize = 16

self.extrsize = 8

self.leftalign = 95

self.tempalign = 58

self.windalign = 183

self.pressalign = 309

self.pressoffset = 165

self.extralign = 393

self.mainFrame = tk.Canvas(self.root,width = 435, height=580, highlightthickness=0)

self.backgroundImage = tk.PhotoImage(master=self.mainFrame,file='weather-bg2.gif',width = 435, height=580)

self.mainFrame.create\_image((0,0),image=self.backgroundImage, anchor='nw')

self.draw\_constants()

self.do\_temp()

self.do\_wind()

self.do\_humid()

self.do\_pressure()

self.do\_mgust\_tod()

self.do\_mgust\_lst()

self.do\_max\_temp()

self.do\_min\_temp()

self.mainFrame.grid()

def define\_variables(self):

global ctemp, cwind, chumid, cpress, mgusttod, mguslst, maxtemp, mintemp, gusts, gusttimes

ctemp, cwind, chumid, cpress, mgusttod, mguslst, maxtemp, mintemp = 'N/A','N/A','N/A','N/A','N/A','N/A','N/A','N/A'

gusts, gusttimes = [], []

def draw\_constants(self):

self.mainFrame.create\_text((self.leftalign,self.tempalign),text='Temperature',fill=self.textfill, anchor='nw', font=(self.textfont,self.headersize))

self.mainFrame.create\_text((self.leftalign,self.windalign),text='Wind Speed',fill=self.textfill, anchor='nw', font=(self.textfont,self.headersize))

self.mainFrame.create\_text((self.leftalign+self.pressoffset,self.pressalign),text='Pressure',fill=self.textfill, anchor='nw', font=(self.textfont,self.headersize-2))

self.mainFrame.create\_text((self.leftalign,self.pressalign),text='Humidity',fill=self.textfill, anchor='nw', font=(self.textfont,self.headersize-2))

self.mainFrame.create\_text((self.leftalign,self.extralign),text='Today\'s Extremes',fill=self.textfill, anchor='nw', font=(self.textfont,self.headersize))

self.mainFrame.create\_text((self.leftalign-30,self.extralign+38),text='Max Gust',fill=self.textfill, anchor='nw', font=(self.textfont,self.headersize-4))

self.mainFrame.create\_text((self.leftalign-30,self.extralign+72),text='Today:',fill=self.textfill, anchor='nw', font=(self.textfont,self.extrsize+2))

self.mainFrame.create\_text((self.leftalign-30,self.extralign+100),text=' Last',fill=self.textfill, anchor='nw', font=(self.textfont,self.extrsize+2))

self.mainFrame.create\_text((self.leftalign-30,self.extralign+112),text='10 min:',fill=self.textfill, anchor='nw', font=(self.textfont,self.extrsize+2))

self.mainFrame.create\_text((self.leftalign+self.pressoffset-20,self.extralign+38),text='Temperatures',fill=self.textfill, anchor='nw', font=(self.textfont,self.headersize-4))

self.mainFrame.create\_text((self.leftalign+self.pressoffset-20,self.extralign+72),text='Max:',fill=self.textfill, anchor='nw', font=(self.textfont,self.extrsize+2))

self.mainFrame.create\_text((self.leftalign+self.pressoffset-20,self.extralign+108),text='Min:',fill=self.textfill, anchor='nw', font=(self.textfont,self.extrsize+2))

def do\_temp(self, second\_or\_more=False):

global ctemp

if second\_or\_more:

self.mainFrame.delete(self.temp), self.mainFrame.delete(self.oval1), self.mainFrame.delete(self.c1)

self.temp = self.mainFrame.create\_text((self.leftalign,self.tempalign+18),fill=self.textfill,text=ctemp, anchor='nw', font=(self.numberfont,75))

self.oval1 = self.mainFrame.create\_oval((self.mainFrame.bbox(self.temp)[2]+15,self.tempalign+94,self.mainFrame.bbox(self.temp)[2]+18,self.tempalign+97),fill='', outline='black')

self.c1 = self.mainFrame.create\_text((self.mainFrame.bbox(self.temp)[2]+20,self.tempalign+90),fill=self.textfill,text='C', anchor='nw', font='Purisa 14')

def do\_wind(self, second\_or\_more=False):

global cwind

if second\_or\_more:

self.mainFrame.delete(self.windspeed), self.mainFrame.delete(self.kph1)

self.windspeed = self.mainFrame.create\_text((self.leftalign,self.windalign+18),fill=self.textfill,text=cwind, anchor='nw', font=(self.numberfont,75))

self.kph1 = self.mainFrame.create\_text((self.mainFrame.bbox(self.windspeed)[2]+15,self.windalign+90),fill=self.textfill,text='km/hr', anchor='nw', font='Purisa 14')

def do\_humid(self, second\_or\_more=False):

global chumid

if second\_or\_more:

self.mainFrame.delete(self.humidity), self.mainFrame.delete(self.percent)

self.humidity = self.mainFrame.create\_text((self.leftalign,self.pressalign+24),fill=self.textfill,text=chumid, anchor='nw', font=(self.numberfont,24))

self.percent = self.mainFrame.create\_text((self.mainFrame.bbox(self.humidity)[2]+10,self.pressalign+36),fill=self.textfill,text='%', anchor='nw', font='Purisa 12')

def do\_pressure(self, second\_or\_more=False):

global cpress

if second\_or\_more:

self.mainFrame.delete(self.pressure), self.mainFrame.delete(self.hpa)

self.pressure = self.mainFrame.create\_text((self.leftalign+self.pressoffset,self.pressalign+24),fill=self.textfill,text=cpress, anchor='nw', font=(self.numberfont,24))

self.hpa = self.mainFrame.create\_text((self.mainFrame.bbox(self.pressure)[2]+10,self.pressalign+38),fill=self.textfill,text='hPa', anchor='nw', font='Purisa 10')

def do\_mgust\_tod(self, second\_or\_more=False):

global mgusttod

if second\_or\_more:

self.mainFrame.delete(self.maxgtod), self.mainFrame.delete(self.kph2)

self.maxgtod = self.mainFrame.create\_text((self.leftalign+27,self.extralign+74),text=mgusttod,fill=self.textfill, anchor='nw', font=(self.numberfont,self.extrsize+4))

self.kph2 = self.mainFrame.create\_text((self.mainFrame.bbox(self.maxgtod)[2]+5,self.extralign+75),fill=self.textfill,text='km/hr', anchor='nw', font='Purisa 8')

def do\_mgust\_lst(self, second\_or\_more=False):

global mguslst

if second\_or\_more:

self.mainFrame.delete(self.maxglst), self.mainFrame.delete(self.kph3)

self.maxglst = self.mainFrame.create\_text((self.leftalign+27,self.extralign+110),text=mguslst,fill=self.textfill, anchor='nw', font=(self.numberfont,self.extrsize+4))

self.kph3 = self.mainFrame.create\_text((self.mainFrame.bbox(self.maxglst)[2]+5,self.extralign+111),fill=self.textfill,text='km/hr', anchor='nw', font='Purisa 8')

def do\_max\_temp(self, second\_or\_more=False):

global maxtemp

if second\_or\_more:

self.mainFrame.delete(self.maxt), self.mainFrame.delete(self.oval2), self.mainFrame.delete(self.c2)

self.maxt = self.mainFrame.create\_text((self.leftalign+self.pressoffset+18,self.extralign+73),text=maxtemp,fill=self.textfill, anchor='nw', font=(self.numberfont,self.extrsize+4))

self.oval2 = self.mainFrame.create\_oval((self.mainFrame.bbox(self.maxt)[2]+4,self.extralign+75,self.mainFrame.bbox(self.maxt)[2]+7,self.extralign+78),fill='', outline='black')

self.c2 = self.mainFrame.create\_text((self.mainFrame.bbox(self.maxt)[2]+10,self.extralign+74),fill=self.textfill,text='C', anchor='nw', font='Purisa 8')

def do\_min\_temp(self, second\_or\_more=False):

global mintemp

if second\_or\_more:

self.mainFrame.delete(self.mint), self.mainFrame.delete(self.oval3), self.mainFrame.delete(self.c3)

self.mint = self.mainFrame.create\_text((self.leftalign+self.pressoffset+18,self.extralign+109),text=mintemp,fill=self.textfill, anchor='nw', font=(self.numberfont,self.extrsize+4))

self.oval3 = self.mainFrame.create\_oval((self.mainFrame.bbox(self.mint)[2]+4,self.extralign+111,self.mainFrame.bbox(self.mint)[2]+7,self.extralign+114),fill='', outline='black')

self.c3 = self.mainFrame.create\_text((self.mainFrame.bbox(self.mint)[2]+10,self.extralign+110),fill=self.textfill,text='C', anchor='nw', font='Purisa 8')

#### This method gets all the active usb ports and selects the port that the TC1 is connected to by doing property comparisons that are unique to the TC1 connected port

def getSerialPort():

try:

activePorts = serial\_ports()

for port in activePorts:

serialPort = serial.Serial(port)

if (serialPort.baudrate == 9600):

if (serialPort.parity == 'N'):

if (serialPort.timeout == None):

if (serialPort.xonxoff == False):

if platform.system() == 'Linux': #new v2.0. TC1 will be a /dev/ttyACM\* port on linux.

if serialPort.port.find('/dev/ttyACM') != -1:

serialPort.close()

return port

else:

serialPort.close()

return port

#if(serialPort.inWaiting() != 0):

# return port

except:

print("Device not found")

#### Method Returns all active usb ports

def serial\_ports():

"""Lists serial ports

:raises EnvironmentError:

On unsupported or unknown platforms

:returns:

A list of available serial ports

"""

if sys.platform.startswith('win'):

ports = ['COM' + str(i + 1) for i in range(256)]

elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):

# this is to exclude your current terminal "/dev/tty"

ports = glob.glob('/dev/tty[A-Za-z]\*')

elif sys.platform.startswith('darwin'):

ports = glob.glob('/dev/tty.\*')

else:

raise EnvironmentError('Unsupported platform')

result = []

for port in ports:

try:

s = serial.Serial(port)

s.close()

result.append(port)

except (OSError, serial.SerialException):

pass

return result

def get\_serial\_port():

serialPort = getSerialPort()

while serialPort == None:

redundantRoot = tk.Tk() #Parent for error dialog to display on top of. This is done so it can then be hidden and destroyed.

redundantRoot.withdraw()

yes\_or\_no = msgbx.askokcancel(message="Please Connect Arduino Weather Station", title="Error", parent=redundantRoot)

redundantRoot.destroy()

if yes\_or\_no:

serialPort = getSerialPort()

else:

#sys.exit()

return False

return serialPort

def window\_close(gui):

global threadLooping

threadLooping = False

Time.sleep(0.2)

gui.root.destroy()

def Collecting(queue1,queue2,queue3, port):

global threadLooping

port.flushInput()

port.flushOutput()

while threadLooping:

try:

reading = port.readline()

queue1.put(reading[:-1])

except:

exc\_type = sys.exc\_info()[0]

if str(exc\_type).find('SerialException') != -1:

msgbx.showerror("Error", "A weather station connection error has been detected.Please exit and reconnect weather station.")

return

def Calculating(queue1, queue2):

global threadLooping

values = []

prev\_speed = 0

anemometer\_constant = 2.25 #For Davis Weather Instruments, Wind Cups, Large. (1600 rev/hr = 1 mph)

conversion = 1.609344 #Number of kilometres per mile

while threadLooping:

while not queue1.empty():

values.append(queue1.get())

if len(values) > 0:

for value in values:

if value[0] in ['0','1','2','3','4','5','6','7','8','9']:

value = float(value)

diff = (value-prev\_speed)/1000

if diff > 0:

wind\_speed = anemometer\_constant/diff\*conversion

if not queue2.full():

queue2.put('W'+str(wind\_speed))

prev\_speed = value

elif value[0] in ['T','P','H']:

try:

temp = float(value[1:])

queue2.put(value)

except:

queue2.put('E'+value[0])

values = []

def Displaying(queue2, gui):

global ctemp, cwind, chumid, cpress, mgusttod, mguslst, maxtemp, mintemp, gusts, gusttimes, threadLooping

first\_temp = True

lastWind = Time.time()

windspeeds = [0.0]

lasthour = dt.datetime.today().hour

max\_time = dt.datetime.today()

while threadLooping:

if Time.time()-lastWind > 2:

average\_windspeed = round(np.sum(windspeeds)/(max(1,len(windspeeds)-1)), 2)

cwind = str(average\_windspeed)

gui.do\_wind(True)

if len(gusts) > 0 and (max(gusts) > mguslst or mguslst == 'N/A'):

mguslst = max(gusts)

max\_time = gusttimes[gusts.index(mguslst)]

gusts, gusttimes = [], []

gui.do\_mgust\_lst(True)

elif (dt.datetime.today()-max\_time).total\_seconds() > 600:

if len(gusts) > 0:

mguslst = max(gusts)

index = gusts.index(mguslst)

max\_time = gusttimes[index]

if (dt.datetime.today()-max\_time).total\_seconds() > 600:

looping = True

while looping:

if len(gusts) > 0:

mguslst = max(gusts)

index = gusts.index(mguslst)

max\_time = gusttimes[index]

else:

mguslst = 0.0

max\_time = dt.datetime.today()

if (dt.datetime.today()-max\_time).total\_seconds() > 600:

gusts = gusts[index+1:]

gusttimes = gusttimes[index+1:]

elif not ((dt.datetime.today()-max\_time).total\_seconds() > 600) or len(gusts) == 0:

looping = False

gusts = gusts[index+1:]

gusttimes = gusttimes[index+1:]

else:

mguslst = 0.0

max\_time = dt.datetime.today()

gui.do\_mgust\_lst(True)

lastWind, windspeeds = Time.time(), [0.0]

if not queue2.empty():

reading = queue2.get()

if reading[0] == 'E':

print 'An error has occured:', reading[1]

elif reading[0] == 'W':

windspeed = round(float(reading[1:]),1)

gusts.append(windspeed)

gusttimes.append(dt.datetime.today())

windspeeds.append(windspeed)

if windspeed > mgusttod or mgusttod=='N/A':

mgusttod = windspeed

gui.do\_mgust\_tod(True)

elif reading[0] == 'H':

chumid = reading[1:]

gui.do\_humid(True)

elif reading[0] == 'T':

if first\_temp:

prev\_temp = float(reading[1:])

first\_temp = False

else:

first\_temp = True

ctemp = str(round(prev\_temp,2)) #Currently taking temperature from temp/humidity sensor. Add str(round((prev\_temp+float(reading[1:]))/2,2)) to average from pressure.

gui.do\_temp(True)

if maxtemp == 'N/A' or float(ctemp) > float(maxtemp):

maxtemp = ctemp

gui.do\_max\_temp(True)

if mintemp == 'N/A' or float(ctemp) < float(mintemp):

mintemp = ctemp

gui.do\_min\_temp(True)

elif reading[0] == 'P':

cpress = reading[1:]

gui.do\_pressure(True)

if dt.datetime.today().hour == 0 and lasthour == 23:

mgusttod, maxtemp, mintemp = cwind, ctemp, ctemp

gui.do\_mgust\_tod(True), gui.do\_max\_temp(True), gui.do\_min\_temp(True)

lasthour = dt.datetime.today().hour

if \_\_name\_\_ == '\_\_main\_\_':

global threadLooping

threadLooping = True

port = get\_serial\_port()

port = serial.Serial(port, timeout=None)

queue1 = Queue()

queue2 = Queue()

queue3 = Queue()

window = mainWin(queue1, queue2, queue3)

collectionProcess = Thread(target= Collecting, args=(queue1,queue2,queue3,port,))

calculatingProcess = Thread(target= Calculating, args=(queue1,queue2,))

displayingProcess = Thread(target= Displaying, args=(queue2,window,))

collectionProcess.daemon = True

calculatingProcess.daemon = True

displayingProcess.daemon = True

collectionProcess.start()

calculatingProcess.start()

displayingProcess.start()

window.root.mainloop()