Temperature and Humidity Measurement

with pyserial and matplotlib animation on a Raspbery Pi

Leonard Samuelson
and
Michael Handler

LB Foster/Salient Systems, Inc.

(based on an idea by Shane Lanham)

Motivation

- We needed to monitor temp and humidity at our CM’s facility while boards were baked in an oven
- CM didn’t have equipment and wouldn’t buy it
- Improperly prepared circuit boards have high probability of failure
- Board designer had the idea to use the iButtonLink temp/humidity sensor

Constraints

- Cheap
- Quick development
- We had management complaining about the time we were spending on it

Requirements

- Inexpensive hardware
- Easy connection to monitored oven
- Easy connection to Internet
- Local display for CM
- Remote display for salient

Implementation at CM

- Raspberry Pi running Raspbian
- Python
- Matplotlib
- iButtonLinkTH temp/humidity sensor
- USB-serial connection to the iButtonLink

Remote Monitoring Implementation

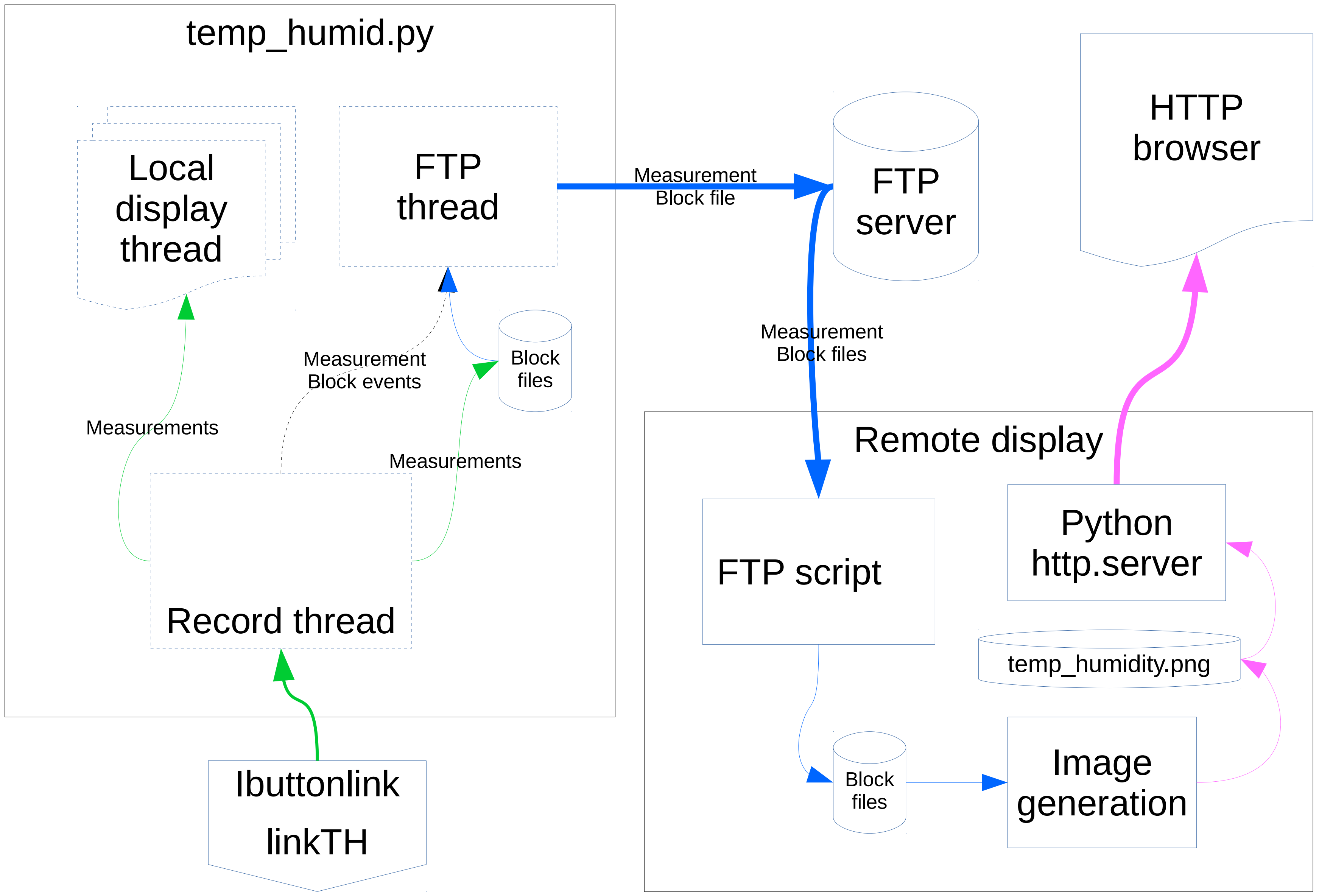
- FTP from CM to Salient FTP server
- FTP from Salient FTP server to local storage
- Bash script for data transformation
- Gnuplot for display at Salient

Division of Labor

- Len:
  - Serial connection to iButtonLink
  - Queuing of sensor data
  - FTP transfer
  - Remote static display
- Mike
  - Local animated display

SENSOR DATA ACQUISITION AND REMOTE DISPLAY

Basic System Structure



Parse, convert

Input format:

`260A8AE1010000B5 19,21.21,70.12,50,00:00:30.2`

`EOD`

`260A8AE1010000B5` – Device (One-wire) ID

`19` – Device type: This is a “linkTH”

`21.21` – Temperature, C

`70.12` – Temperature, F

`50` – Relative humidity, %

`00:00:30.2` – Device uptime, HH:MM:SS.t

`EOD` – End Of Data (cycle)

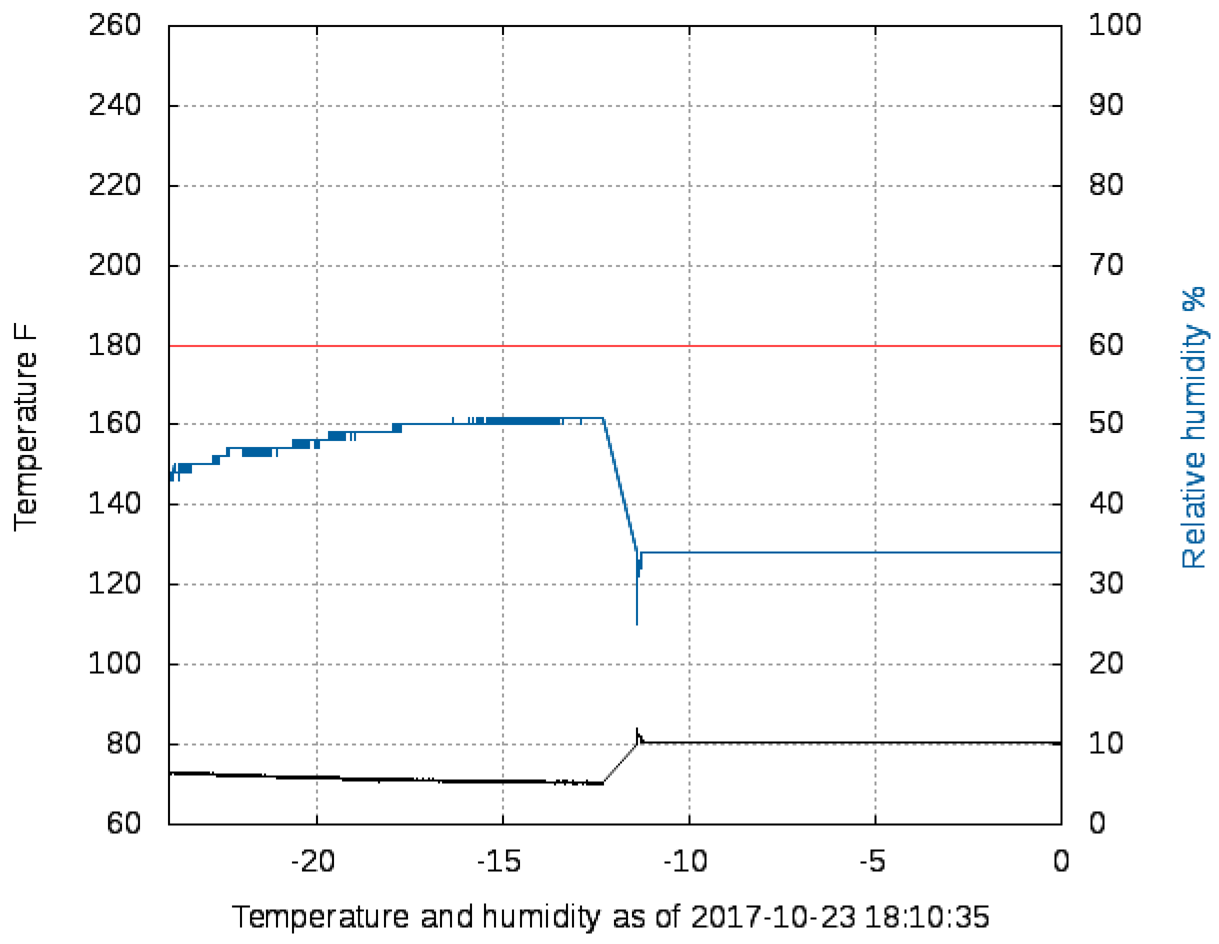
Parsed measurements format (python list):

[hour, datetime, ID, type, tempC, tempF, humidity, uptime, unix-secs]

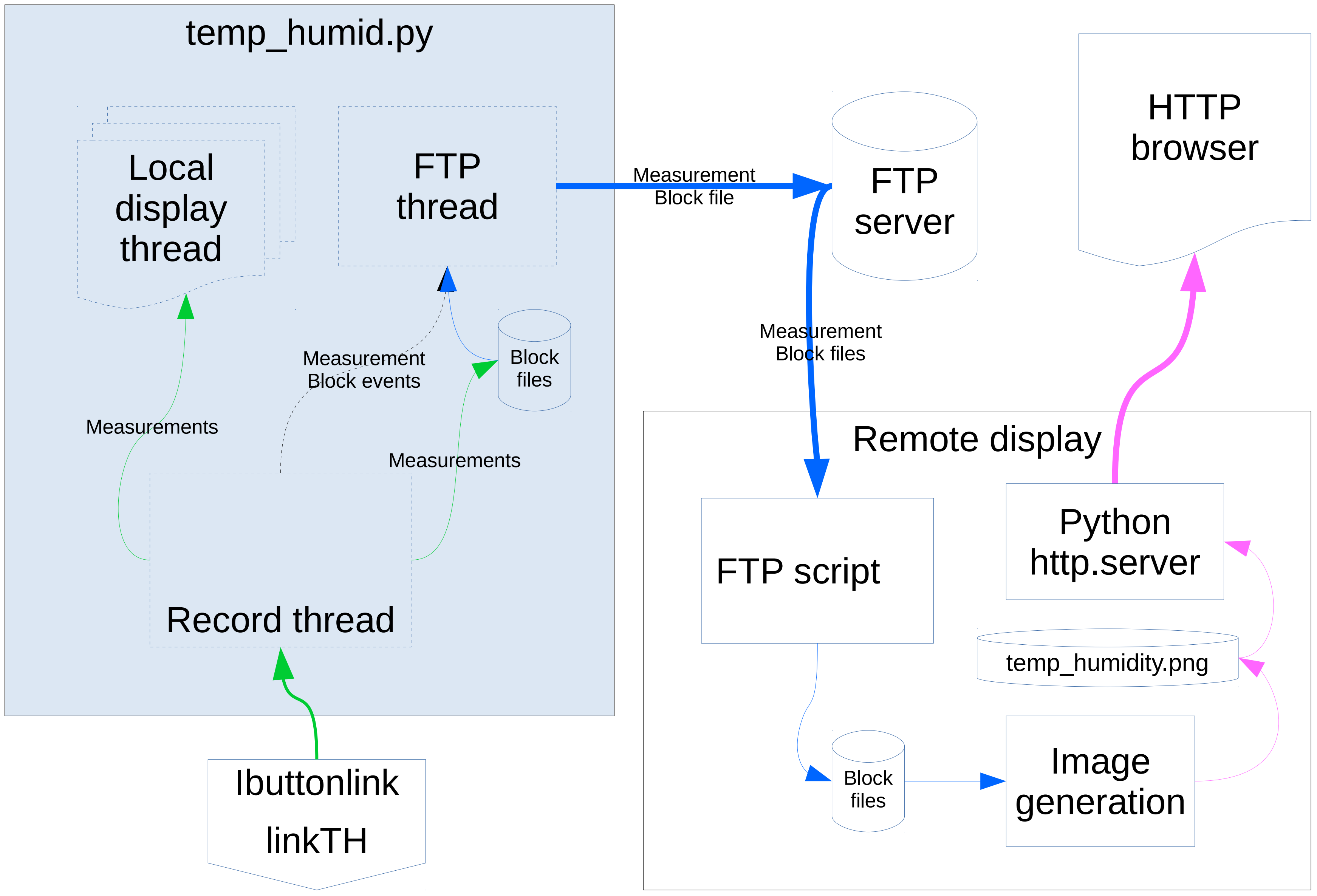
Hour – Simplify measurement block splitting

Unix-secs – Simplify remote display gnuplot image

Remote image



System Structure:
temp_humid.py, on the Pi



Record thread: Read Parse Store Notify

- Outer loop: Generate measurement block files forever
  - Open recording file
    - Filename is related to current hour
    - Append mode, in case of interruption
  - Run recording cycle
    - Report cycle start to FTP thread on *recordqueue*
    - Execute *record_cycle* to collect and store data
      - Ends on the hour or error, whichever comes first
    - Report cycle end to FTP thread on *recordqueue*
  - Close recording file

Record_thread outer loop code

```
def record_thread(recordqueue, graphqueue, datadir, portname, portretry):
    """ entry point for the data recorder, collector, distribution thread

        Manage report files and notification to the main thread for
        triggering FTP transfers.
    """
    logging.info("record_thread start dir=%s, port=%s" % (datadir, portname))
    while True:
        # Start recording
        outfilename = time.strftime('temp_monitor_%Y-%m-%d_%H.txt')
        outfilepath = os.path.join(datadir, outfilename)
        logging.info('outfile ' + outfilepath)
        outfile = open(outfilepath, 'a')
        try:
            recordqueue.put( (outfilepath, outfilename, "start") )
            record_cycle(portname, outfile, graphqueue)
            recordqueue.put( (outfilepath, outfilename, "end") )
        except serial.serialutil.SerialException:
            recordqueue.put( (outfilepath, outfilename, "error") )
            logging.critical("Warning: Serial port %s unavailable" % portname)
            time.sleep(int(portretry))
        outfile.close()
```

Recording cycle

- Open serial port
- Get measurements from recorder until “done”
  - Report measurements to display thread on *graphqueue*
  - Stop on error
  - Stop when crossing hour boundary
- Close serial port

Recording cycle code

```
def record_cycle(portname, outfile, graphqueue):
    """ Get and handle measurement data from the iButtonLink LinkTH controller.

        Manage distribution of the data to the graphqueue, and manage
        starts/stops and errors detected by the recorder.
    """
    start_hour = time.localtime().tm_hour
    try:
        sport = serial.Serial(portname, baudrate=9600)
        for measurement in recorder(sport):
            print(' '.join(measurement[1:]), file=outfile)
            outfile.flush()
            graphqueue.put(measurement)
            if measurement[0] != start_hour:
                break
    except serial.serialutil.SerialException as e:
        raise e
    except KeyboardInterrupt:
        return outfile
    finally:
        try:
            sport.close()
        except:
            pass
    return
```

iButtonLink data collection, parsing and delivery

- Classical read/parse/deliver
- Implemented as a Pythonic generator
- Repeat while moving valid data (30 second timeout)
  - Collect raw serial data
  - Locate iButtonLink data boundary
  - Transform to target measurement format
  - Yield measurement to caller
- Return (raising StopIteration)
  - Data read timeout
  - Error from serial object

Data collection: Locate data boundary

```
def recorder(serport):
    serport.timeout=1.0
    newdata = b''
    start_time = time.time()
    now = start_time
    while now - start_time < 30:
        now = time.time()
        newdata += serport.read(500)
        eodpos = newdata.find(b'EOD')
        if eodpos > 0:
            lines = newdata[:eodpos].split(b'\n')
            newdata = newdata[eodpos+3:]
            for dl in lines:
        <<< Next slide: Handle lines, including
yield/break >>>
        newdata = b''
    logging.info("timeout")
    return b'Timeout'
```

Data collection: parse, deliver

```
                if len(dl) < 20:
                    continue
                dl = dl.strip(b'\r').strip(b'?')
                matcher = datafmt.match(dl)
                if matcher:
                    ident =
matcher.group(1).strip().decode(encoding="ascii", errors="none")
                    if len(ident) < 16:
                        ident = None
                else:
                    ident = None
                if ident is not None:
                    now = time.time()
                    timeobj = time.localtime(now)
                    nowstr = time.strftime('%Y-%m-%d %H:%M:%S', timeobj)
                    timesecs = str(int(now))
                    measurements = [timeobj.tm_hour, nowstr, ident]

measurements.extend( [x.strip().decode(encoding="ascii", errors="none")
for x in matcher.group(2).split(b',')] )
                    measurements.append(timesecs)
                    start_time = now
                    yield measurements
```

Send reports to home base: FTP thread

- Use FTP to send measurement files
- Repeat forever:
  - Wait for timeout or item in recordqueue
  - If there’s something useful to report,
    - Run the ftpsend

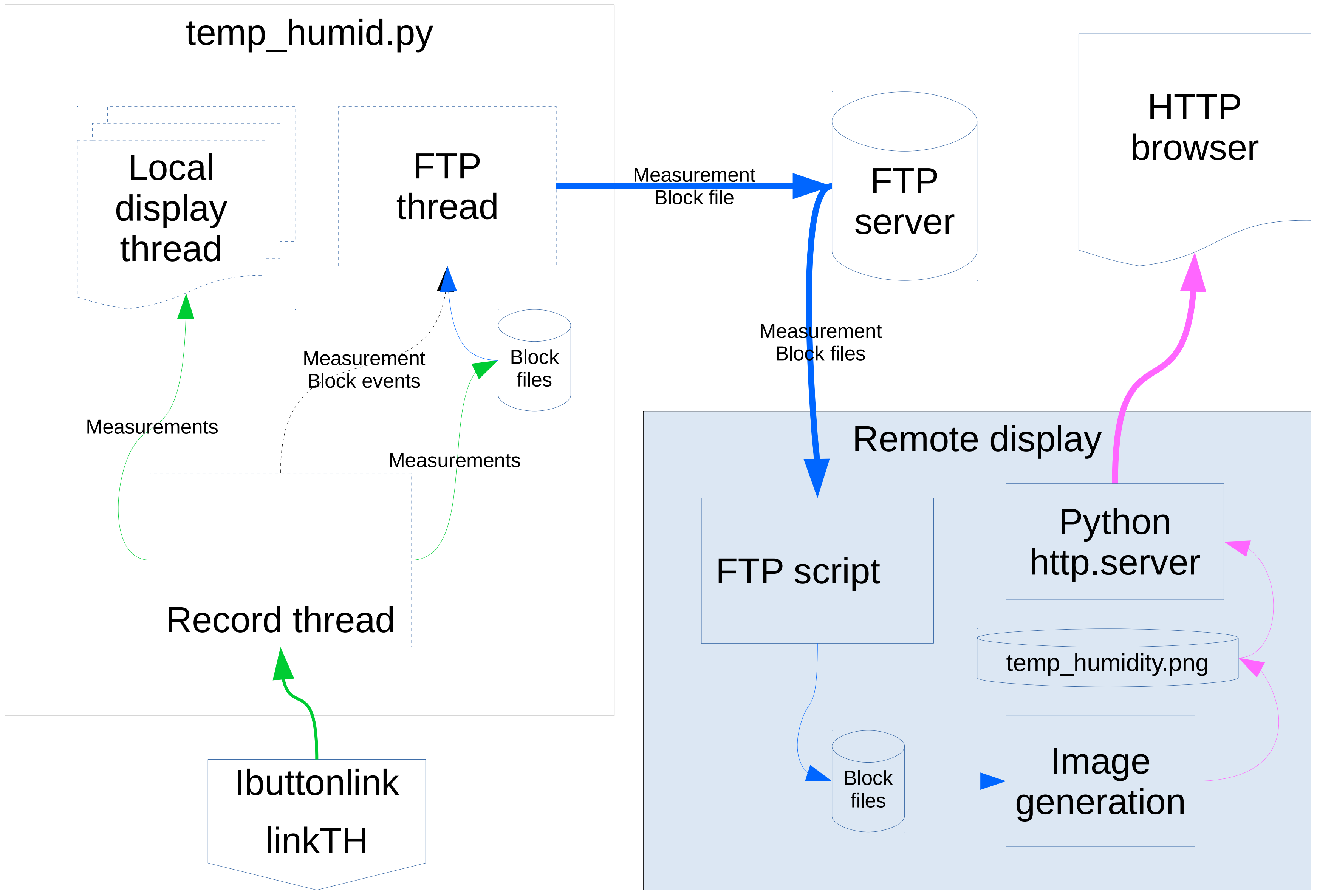
Send reports, top level

```
ftpperiod = None
def ftp_thread():
    global recordqueue, ftpperiod
    fileobj = None
    recordstate = None
    while True:
        try:
            fileobj = recordqueue.get(block=True,
timeout=ftpperiod)
            if fileobj[2] == "end":
                recordstate = "Complete"
            else:
                recordstate = None
        except queue.Empty:
            recordstate = "timeout"
            pass
        logging.info("main thread")
        if fileobj is not None and recordstate is not None:
            ftpsend(fileobj[0], fileobj[1])
```

Send reports, FTP file transfer

```
def ftpsend(filepath, filename):
    try:
        with ftplib.FTP(host="www.salientsystems.com", 
user="******", passwd="******") as ftp:
            logging.info("ftp start")
            with open(filepath, 'rb') as ftpfile:
                ftp.cwd("test_temperature")
                ftp.storbinary("STOR %s" % filename, 
ftpfile)
                ftp.quit()
                logging.info("ftp complete")
    except socket.gaierror:
        logging.critical("Bad FTP host address")
    except socket.error:
        logging.critical("FTP socket error")
    except socket.herror:
        logging.critical("FTP socket H error")
    except ftplib.Error as ftpe:
        logging.critical("FTP error")
```

System Structure: Remote display, running on workstation



Remote Display

- Cron-scheduled collect/image generator shell script
  - Run at least once per hour
  - Retrieve measurement files from FTP server
    - Use ".netrc" to automate
  - Generate PNG image with gnuplot
    - Write image file to `/tmp/temp_humidity.png`
- HTTP server
  - Pythonic command line:
    `python3 -m http.server 8001`
  - Anyone in office can see results:
    `http://helicon:8001/temp_humidity.png`

Remote collection script (1/3)

```
#!/bin/bash

if [ "$1" = "ftp" ]
then
    ftp -i www.salientsystems.com
fi

WORKFILE=/tmp/plot_temp_humid.txt

cat `ls /home/lcs/work/ibuttonlink/test_temperature/*.txt | tail
-n24 | xargs` >$WORKFILE

CURRENT="`tail -n1 $WORKFILE | cut -d ' ' -f 1,2`"
gnuplot <<PLOTEND
… Gnuplot script, see 3/3 ...
PLOTEND

rm $WORKFILE

echo Plotting temperature and humidity as of $CURRENT
```

FTP script, ~/.netrc (2/3)

```
lcs@helicon:~$ cat .netrc
machine www.salientsystems.com
login ********
password ********
macdef init
cd test_temperature
lcd
/home/lcs/work/ibuttonlink/test_temperatu
re
mget *
mdelete *
quit
lcs@helicon:~$
```

Image generation (3/3)

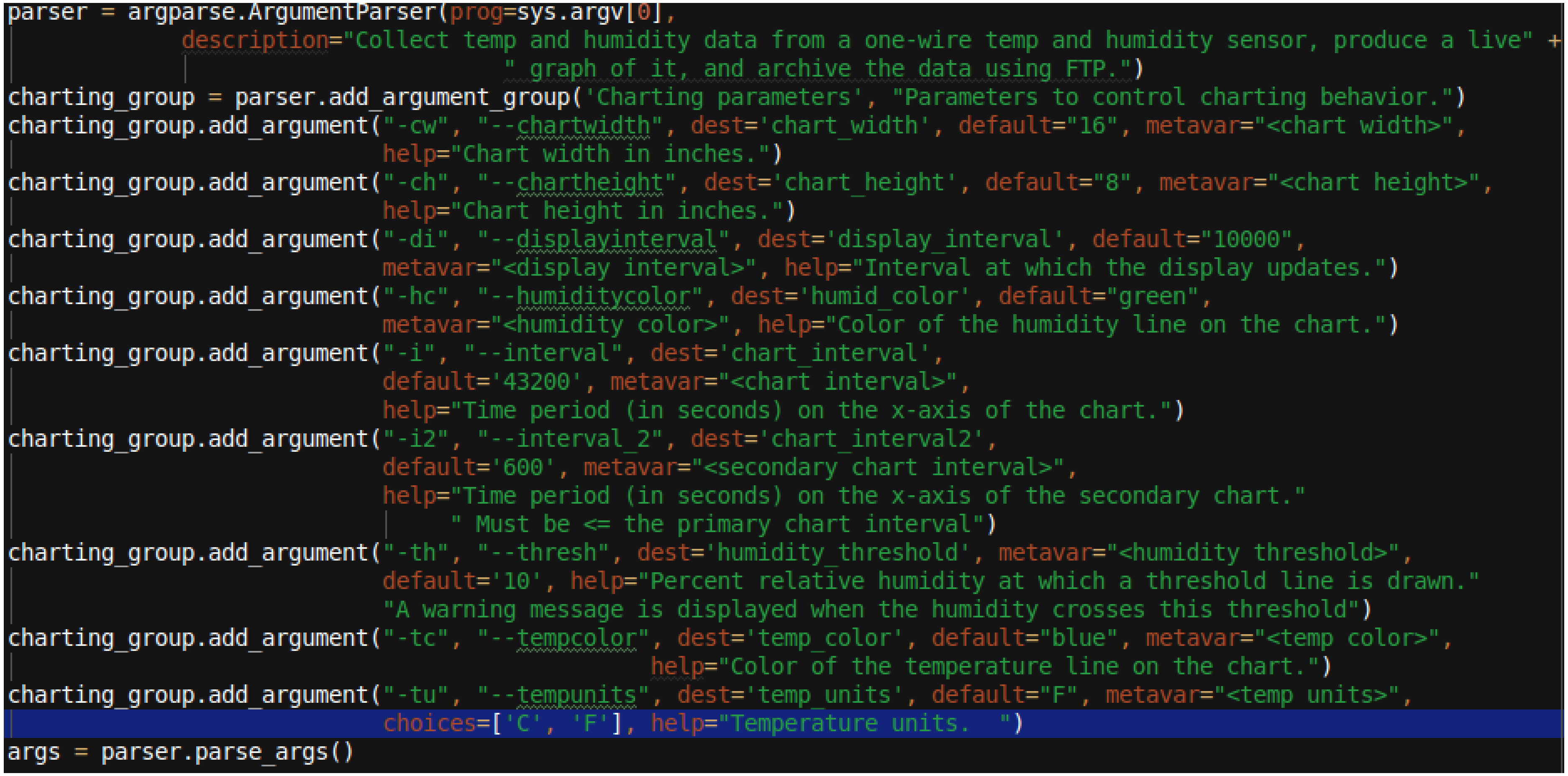
```
set terminal png size 800,600
set output "/tmp/temp_humidity.png"
set multiplot
stats "$WORKFILE" using 9 name 'T'
set xrange [-24:0]
set xlabel "Temperature and humidity as of " . "$CURRENT"
set yrange [60:260]
set y2range [0:100]
set ylabel "Temperature F"
set y2label "Relative humidity %" textcolor rgb "#0060a0"
set ytics 20
set y2tics 10

set grid
plot "$WORKFILE" using ((\$9 - T_max)/3600):7 axes x1y2 with
lines lc rgb "#0060a0" notitle
set nogrid
replot 60 axes x1y2 with lines lc rgb "#ff4040" notitle
replot "$WORKFILE" using ((\$9 - T_max)/3600):6 axes x1y1
with lines lc rgb "#000000" notitle
```

Animated Display Graph

Requirements

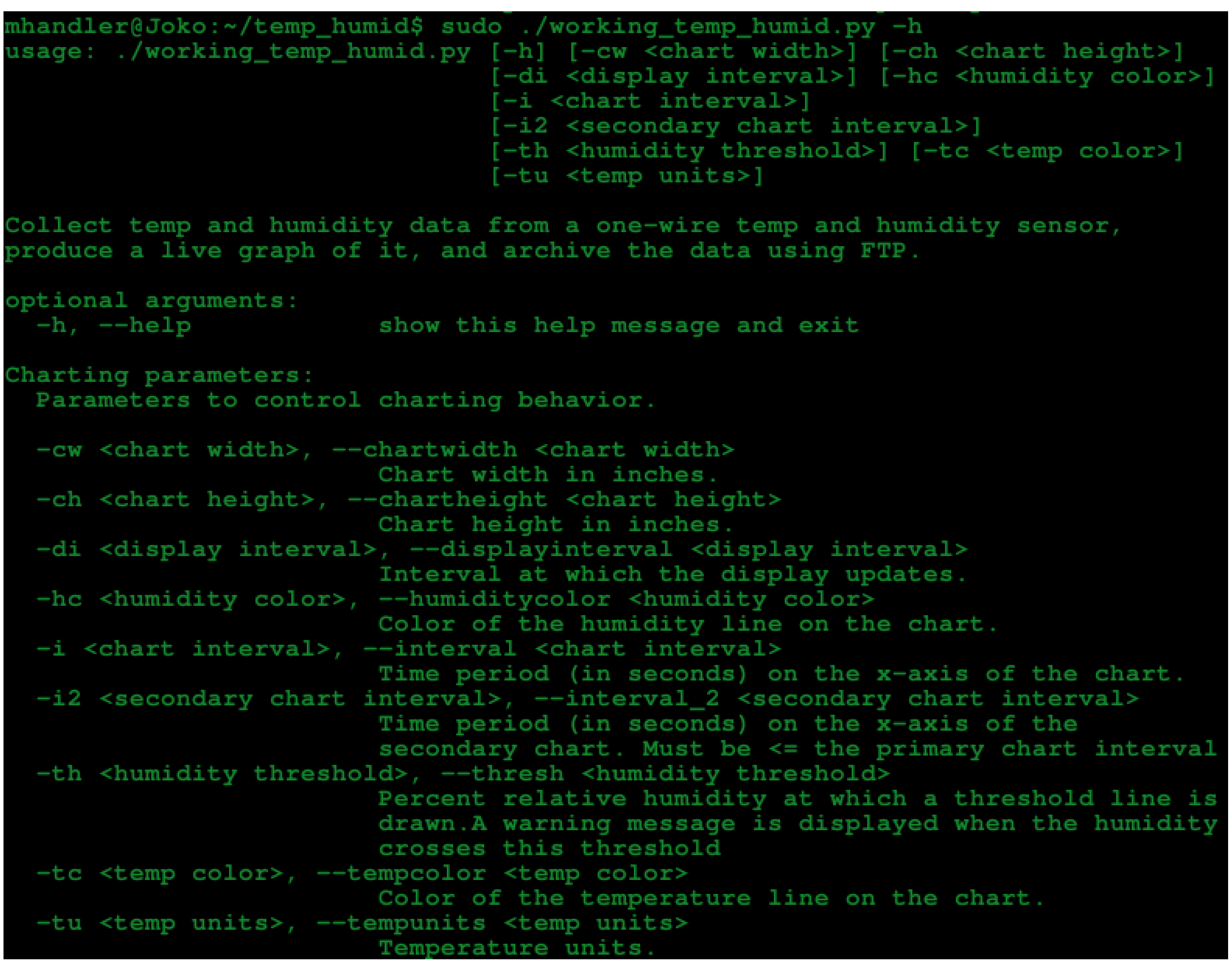
- Initial
  - Animated chart with one 12 hour graph showing temp and humidity
  - Dual y-axes
- Later
  - Two charts:
    - 12 hours
    - 10 minutes
    - Line showing humidity threshold
    - Threshold exceeded notification

Script Arguments (1 of 2)



```
parser = argparse.ArgumentParser(prog=sys.argv[0],
             description="Collect temp and humidity data from a one-wire temp and humidity sensor, produce a live" +
                                    " graph of it, and archive the data using FTP.")
charting_group = parser.add_argument_group('Charting parameters', "Parameters to control charting behavior.")
charting_group.add_argument("-cw", "--chartwidth", dest='chart_width', default="16", metavar="<chart width>",
                            help="Chart width in inches.")
charting_group.add_argument("-ch", "--chartheight", dest='chart_height', default="8", metavar="<chart height>",
                            help="Chart height in inches.")
charting_group.add_argument("-di", "--displayinterval", dest='display_interval', default="10000",
                            metavar="<display interval>", help="Interval at which the display updates.")
charting_group.add_argument("-hc", "--humiditycolor", dest='humid_color', default="green",
                            metavar="<humidity color>", help="Color of the humidity line on the chart.")
charting_group.add_argument("-i", "--interval", dest='chart_interval',
                            default='43200', metavar="<chart interval>",
                            help="Time period (in seconds) on the x-axis of the chart.")
charting_group.add_argument("-i2", "--interval_2", dest='chart_interval2',
                            default='600', metavar="<secondary chart interval>",
                            help="Time period (in seconds) on the x-axis of the secondary chart."
                                 " Must be <= the primary chart interval")
charting_group.add_argument("-th", "--thresh", dest='humidity_threshold', metavar="<humidity threshold>",
                            default='10', help="Percent relative humidity at which a threshold line is drawn."
                            "A warning message is displayed when the humidity crosses this threshold")
charting_group.add_argument("-tc", "--tempcolor", dest='temp_color', default="blue", metavar="<temp color>",
                                                    help="Color of the temperature line on the chart.")
charting_group.add_argument("-tu", "--tempunits", dest='temp_units', default="F", metavar="<temp units>",
                            choices=['C', 'F'], help="Temperature units.  ")
args = parser.parse_args()
```

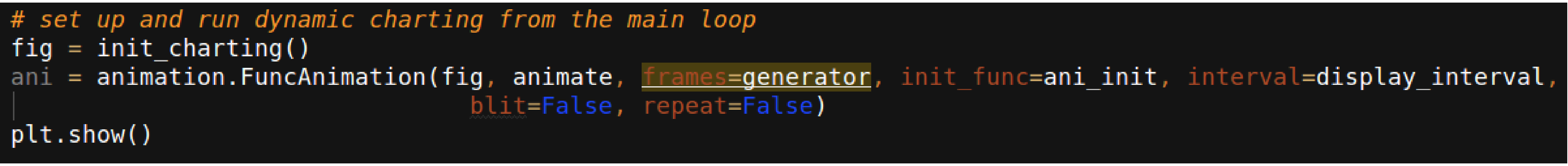
Script Arguments (2 of 2)



```
mhandler@Joko:~/temp_humid$ sudo ./working_temp_humid.py -h
usage: ./working_temp_humid.py [-h] [-cw <chart width>] [-ch <chart height>]
                               [-di <display interval>] [-hc <humidity color>]
                               [-i <chart interval>]
                               [-i2 <secondary chart interval>]
                               [-th <humidity threshold>] [-tc <temp color>]
                               [-tu <temp units>]

Collect temp and humidity data from a one-wire temp and humidity sensor,
produce a live graph of it, and archive the data using FTP.

optional arguments:
  -h, --help            show this help message and exit

Charting parameters:
  Parameters to control charting behavior.

  -cw <chart width>, --chartwidth <chart width>
                        Chart width in inches.
  -ch <chart height>, --chartheight <chart height>
                        Chart height in inches.
  -di <display interval>, --displayinterval <display interval>
                        Interval at which the display updates.
  -hc <humidity color>, --humiditycolor <humidity color>
                        Color of the humidity line on the chart.
  -i <chart interval>, --interval <chart interval>
                        Time period (in seconds) on the x-axis of the chart.
  -i2 <secondary chart interval>, --interval_2 <secondary chart interval>
                        Time period (in seconds) on the x-axis of the
                        secondary chart. Must be <= the primary chart interval
  -th <humidity threshold>, --thresh <humidity threshold>
                        Percent relative humidity at which a threshold line is
                        drawn.A warning message is displayed when the humidity
                        crosses this threshold
  -tc <temp color>, --tempcolor <temp color>
                        Color of the temperature line on the chart.
  -tu <temp units>, --tempunits <temp units>
                        Temperature units.
```

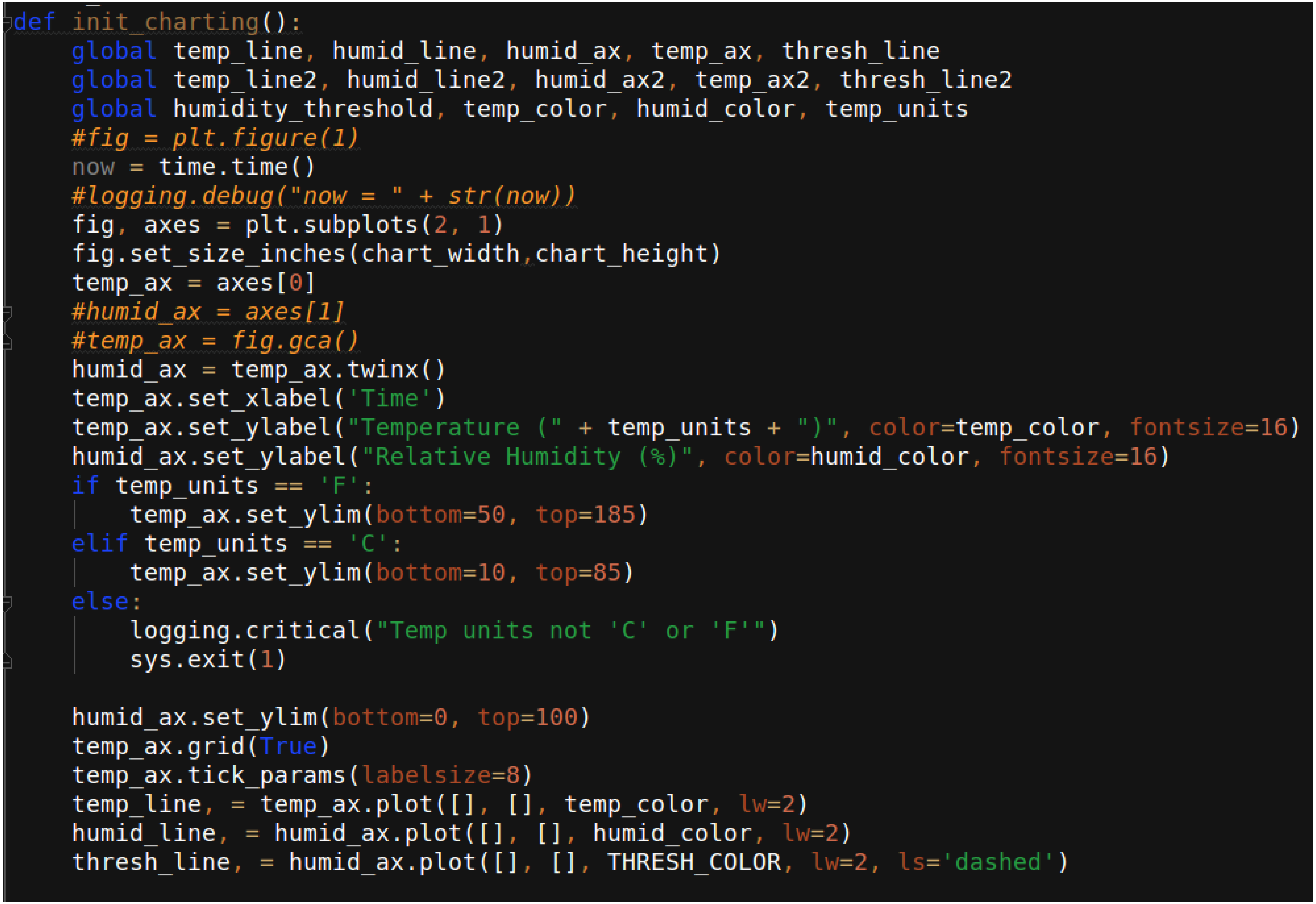
Animation

- Matplotlib.animation module
- Matplotlib.animation.FuncAnimation function
- https://matplotlib.org/api/_as_gen/matplotlib.animation.FuncAnimation.html#matplotlib.animation.FuncAnimation
- All the examples on the web were canned animations, *e. g.*, …
- https://brushingupscience.wordpress.com/2016/06/21/matplotlib-animations-the-easy-way/



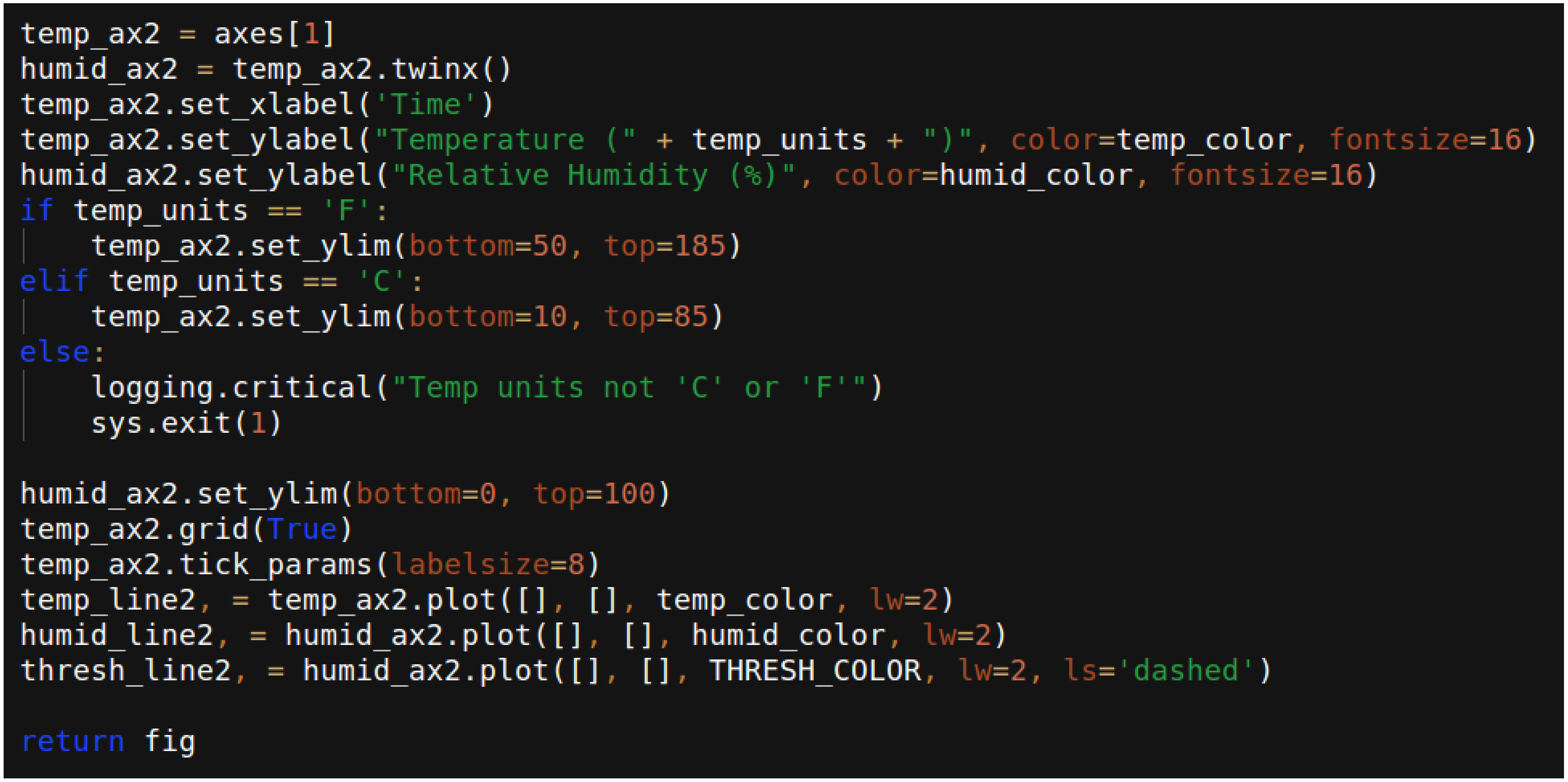
```
# set up and run dynamic charting from the main loop
fig = init_charting()
ani = animation.FuncAnimation(fig, animate, frames=generator, init_func=ani_init, interval=display_interval,
                              blit=False, repeat=False)
plt.show()
```

Matplotlib Initialization (1 of 2)



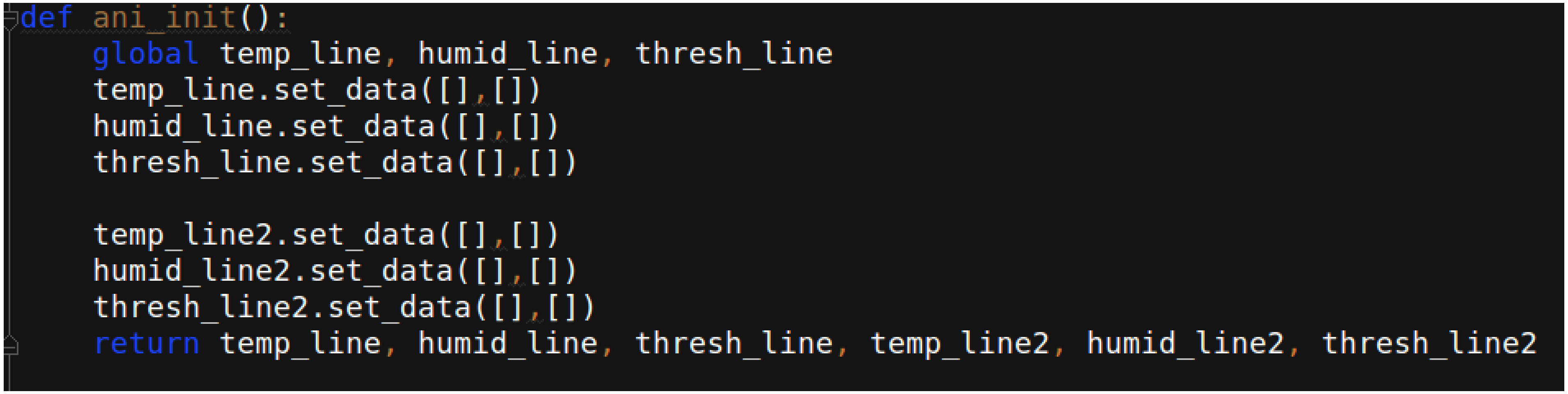
```
def init_charting():
    global temp_line, humid_line, humid_ax, temp_ax, thresh_line
    global temp_line2, humid_line2, humid_ax2, temp_ax2, thresh_line2
    global humidity_threshold, temp_color, humid_color, temp_units
    #fig = plt.figure(1)
    now = time.time()
    #logging.debug("now = " + str(now))
    fig, axes = plt.subplots(2, 1)
    fig.set_size_inches(chart_width,chart_height)
    temp_ax = axes[0]
    #humid_ax = axes[1]
    #temp_ax = fig.gca()
    humid_ax = temp_ax.twinx()
    temp_ax.set_xlabel('Time')
    temp_ax.set_ylabel("Temperature (" + temp_units + ")", color=temp_color, fontsize=16)
    humid_ax.set_ylabel("Relative Humidity (%)", color=humid_color, fontsize=16)
    if temp_units == 'F':
        temp_ax.set_ylim(bottom=50, top=185)
    elif temp_units == 'C':
        temp_ax.set_ylim(bottom=10, top=85)
    else:
        logging.critical("Temp units not 'C' or 'F'")
        sys.exit(1)

    humid_ax.set_ylim(bottom=0, top=100)
    temp_ax.grid(True)
    temp_ax.tick_params(labelsize=8)
    temp_line, = temp_ax.plot([], [], temp_color, lw=2)
    humid_line, = humid_ax.plot([], [], humid_color, lw=2)
    thresh_line, = humid_ax.plot([], [], THRESH_COLOR, lw=2, ls='dashed')
```

Matplotlib Initialization (2 of 2)



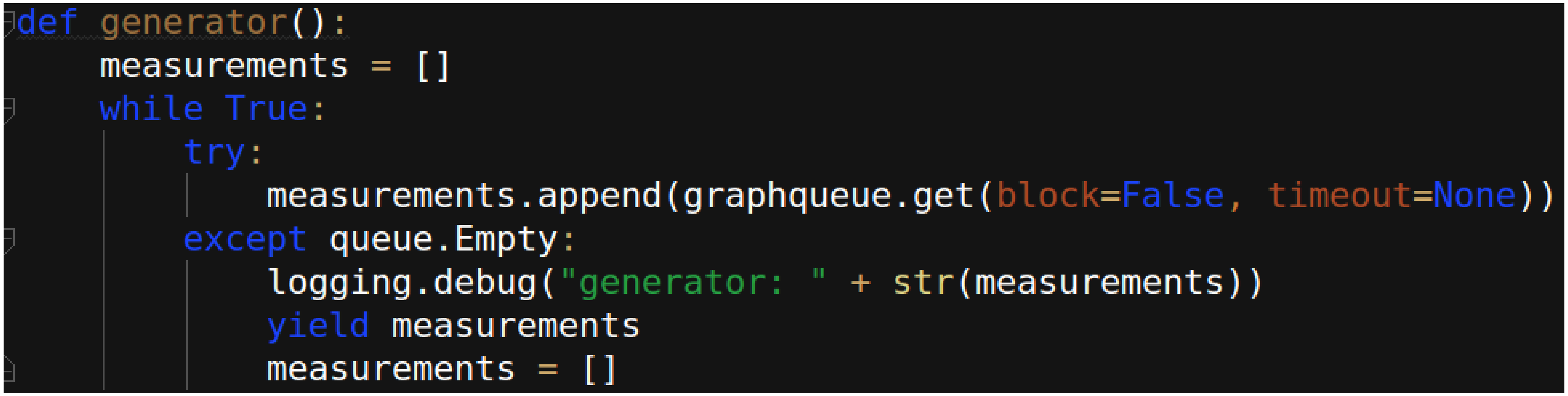
```python
temp_ax2 = axes[1]
humid_ax2 = temp_ax2.twinx()
temp_ax2.set_xlabel('Time')
temp_ax2.set_ylabel("Temperature (" + temp_units + ")", color=temp_color, fontsize=16)
humid_ax2.set_ylabel("Relative Humidity (%)", color=humid_color, fontsize=16)
if temp_units == 'F':
    temp_ax2.set_ylim(bottom=50, top=185)
elif temp_units == 'C':
    temp_ax2.set_ylim(bottom=10, top=85)
else:
    logging.critical("Temp units not 'C' or 'F'")
    sys.exit(1)

humid_ax2.set_ylim(bottom=0, top=100)
temp_ax2.grid(True)
temp_ax2.tick_params(labelsize=8)
temp_line2, = temp_ax2.plot([], [], temp_color, lw=2)
humid_line2, = humid_ax2.plot([], [], humid_color, lw=2)
thresh_line2, = humid_ax2.plot([], [], THRESH_COLOR, lw=2, ls='dashed')

return fig
```

Animation Initialization



```python
def ani_init():
    global temp_line, humid_line, thresh_line
    temp_line.set_data([],[])
    humid_line.set_data([],[])
    thresh_line.set_data([],[])

    temp_line2.set_data([],[])
    humid_line2.set_data([],[])
    thresh_line2.set_data([],[])
    return temp_line, humid_line, thresh_line, temp_line2, humid_line2, thresh_line2
```

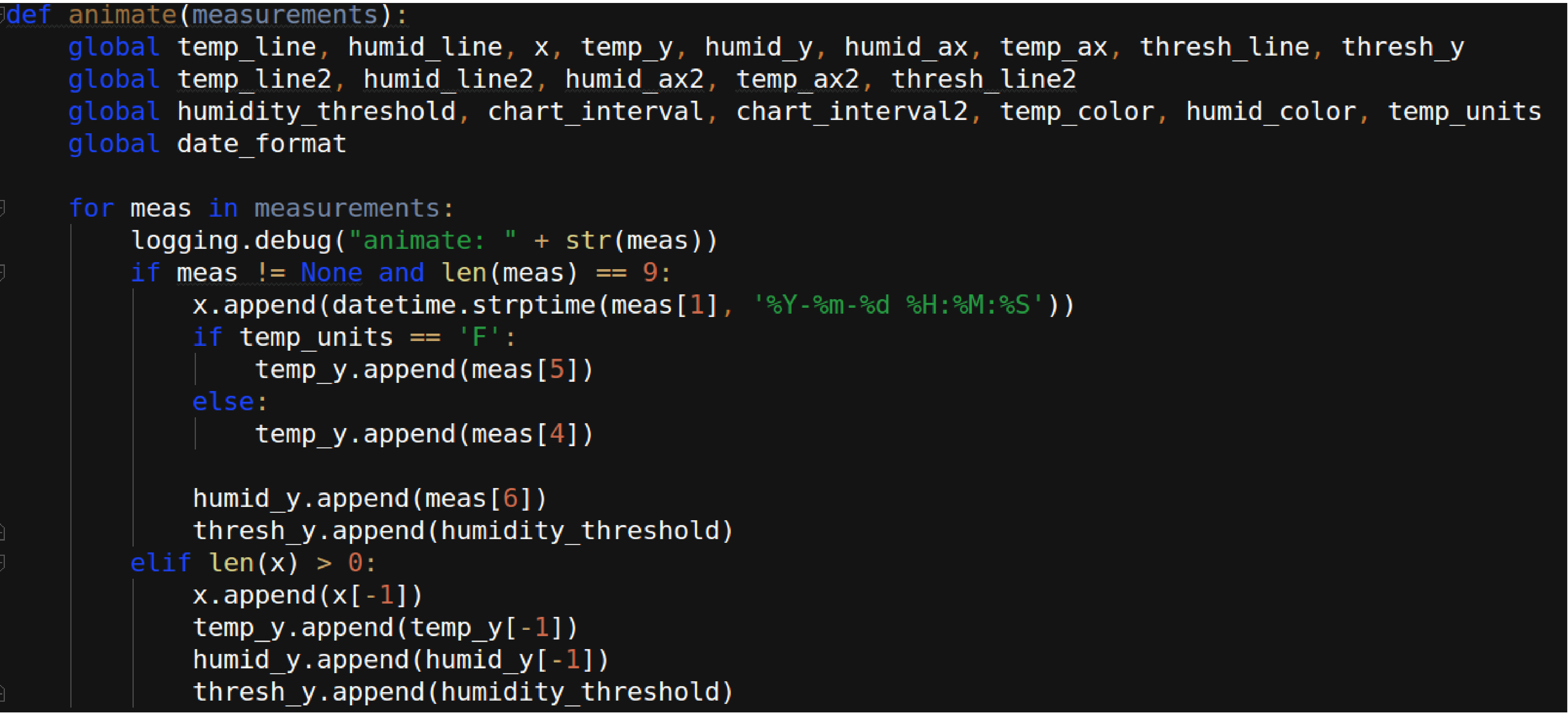
Generator Function



```
def generator():
    measurements = []
    while True:
        try:
            measurements.append(graphqueue.get(block=False, timeout=None))
        except queue.Empty:
            logging.debug("generator: " + str(measurements))
            yield measurements
            measurements = []
```

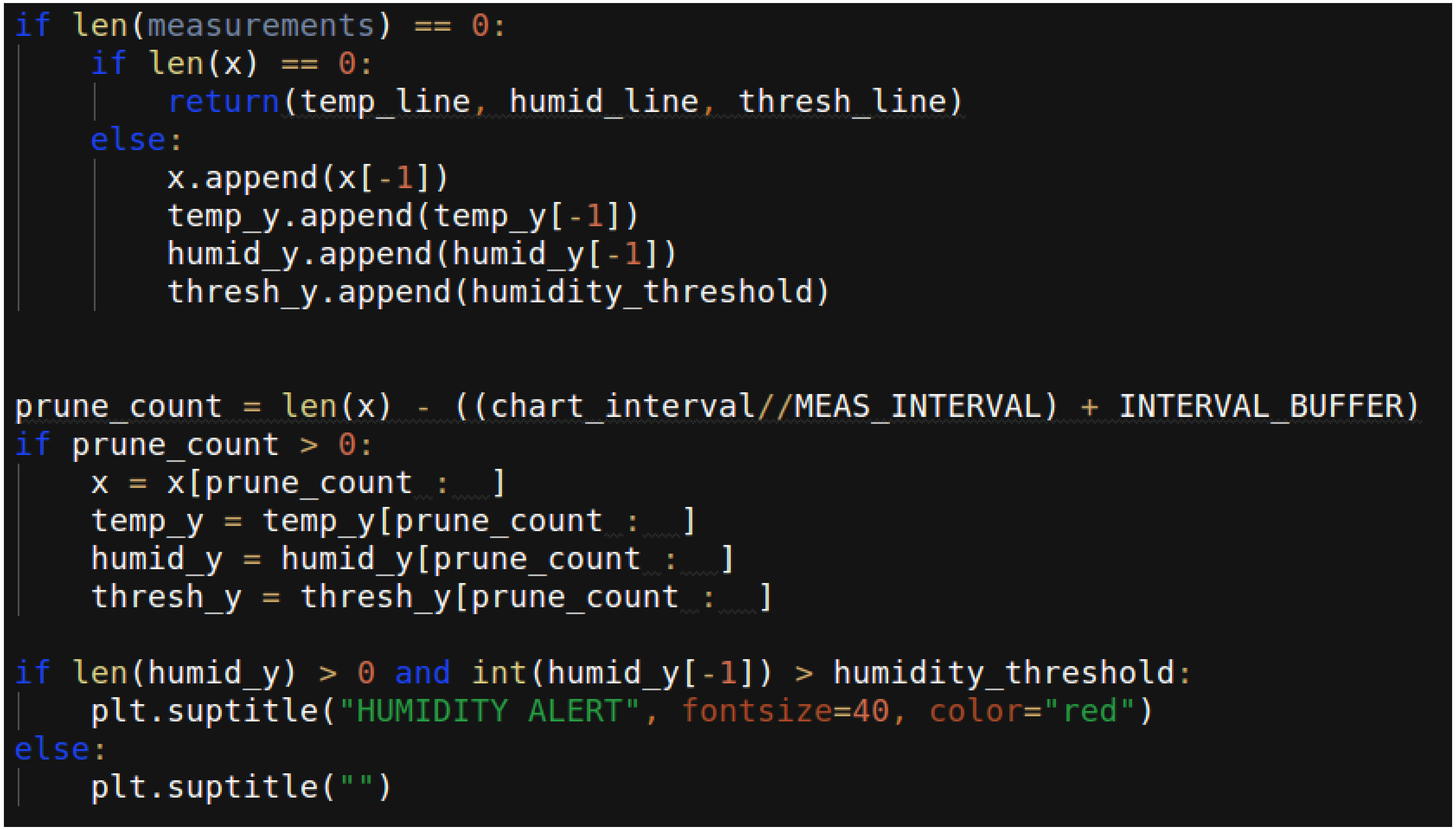
Animate Function (1 of 3)

[18, '2017-10-23 18:16:35', '26B55411020000FF', '19', '20.62', '69.06', '94', '00:00:32.2', '1508796995']



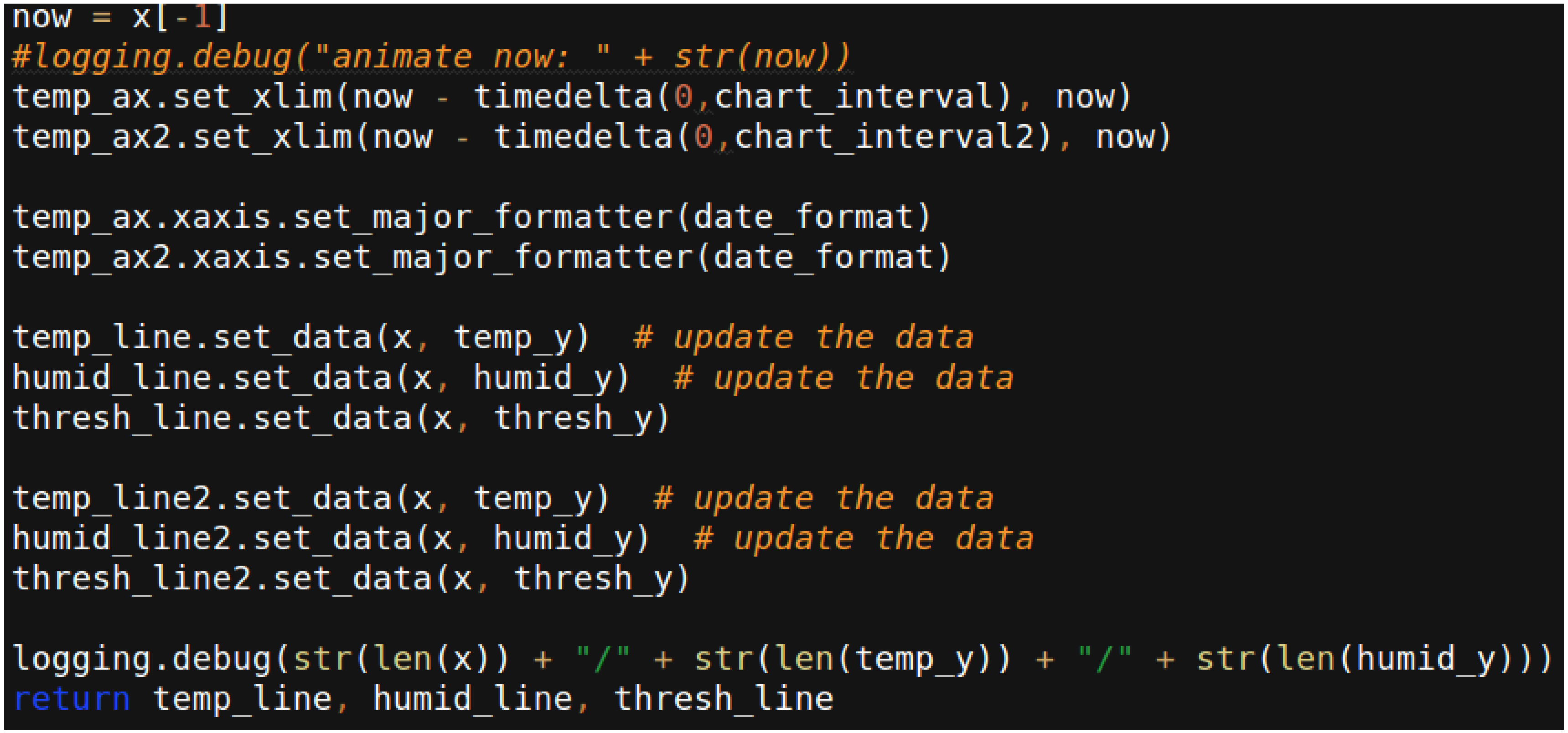
```python
def animate(measurements):
    global temp_line, humid_line, x, temp_y, humid_y, humid_ax, temp_ax, thresh_line, thresh_y
    global temp_line2, humid_line2, humid_ax2, temp_ax2, thresh_line2
    global humidity_threshold, chart_interval, chart_interval2, temp_color, humid_color, temp_units
    global date_format

    for meas in measurements:
        logging.debug("animate: " + str(meas))
        if meas != None and len(meas) == 9:
            x.append(datetime.strptime(meas[1], '%Y-%m-%d %H:%M:%S'))
            if temp_units == 'F':
                temp_y.append(meas[5])
            else:
                temp_y.append(meas[4])

            humid_y.append(meas[6])
            thresh_y.append(humidity_threshold)
        elif len(x) > 0:
            x.append(x[-1])
            temp_y.append(temp_y[-1])
            humid_y.append(humid_y[-1])
            thresh_y.append(humidity_threshold)
```

Animate Function (2 of 3)



```python
if len(measurements) == 0:
    if len(x) == 0:
        return(temp_line, humid_line, thresh_line)
    else:
        x.append(x[-1])
        temp_y.append(temp_y[-1])
        humid_y.append(humid_y[-1])
        thresh_y.append(humidity_threshold)


prune_count = len(x) - ((chart_interval//MEAS_INTERVAL) + INTERVAL_BUFFER)
if prune_count > 0:
    x = x[prune_count :  ]
    temp_y = temp_y[prune_count :  ]
    humid_y = humid_y[prune_count :  ]
    thresh_y = thresh_y[prune_count :  ]

if len(humid_y) > 0 and int(humid_y[-1]) > humidity_threshold:
    plt.suptitle("HUMIDITY ALERT", fontsize=40, color="red")
else:
    plt.suptitle("")
```

Animate Function (3 of 3)



```
now = x[-1]
#logging.debug("animate now: " + str(now))
temp_ax.set_xlim(now - timedelta(0,chart_interval), now)
temp_ax2.set_xlim(now - timedelta(0,chart_interval2), now)

temp_ax.xaxis.set_major_formatter(date_format)
temp_ax2.xaxis.set_major_formatter(date_format)

temp_line.set_data(x, temp_y)  # update the data
humid_line.set_data(x, humid_y)  # update the data
thresh_line.set_data(x, thresh_y)

temp_line2.set_data(x, temp_y)  # update the data
humid_line2.set_data(x, humid_y)  # update the data
thresh_line2.set_data(x, thresh_y)

logging.debug(str(len(x)) + "/" + str(len(temp_y)) + "/" + str(len(humid_y)))
return temp_line, humid_line, thresh_line
```

Q&A

Any Questions?...Any Answers?