```
"""
#######################################################################################################
Project: Sensor Subsystem

Authors: Team Spatium Lucis

Version: v2.0

Target Device: Raspberry Pi 3

Files: circadian.py, pir_sensor.py, rgb_sensor.py, send_circadian_values.py, usr_sensor.py,
wait_for_cmd.py, start.py, stop.py, compensate.txt, config.txt, sensor_data.txt

Last edited: June 15, 2017

#######################################################################################################

(I) start.py:
This is the init script for the sensor subsystem. Usage:

$ python start.py

*********DO NOT USE sudo python!!!

(II) circadian.py:
This file serves as a custom Python module that houses functions that are common to all the various sensor subsystem
files.

    (a) Module imports:
        (1) import subprocess
        (2) import socket
        (3) import MySQLdb
        (4) import time
        (5) import math
        (6) import datetime

    (b) Functions:
        (1) def init_circadian_table():
        This function creates base MASTER_CIRCADIAN_TABLE. Returns a list of lists containing RGB brightnesses
        percentages for each minute of the day for 7 AM wake up and 11 PM sleep. Example index (not relative to the
        actual table):
        MASTER_CIRCADIAN_TABLE[420] == [ 30, 40, 50 ]. Therefore, MASTER_CIRCADIAN_TABLE[420][0] == 30.

        (2) def init_offset_table():
        This function creates the base MASTER_OFFSET_TABLE. Returns a list of lists containing rgb sensor offset
        values for each minute of the day. These are needed because the rgb sensor cannot properly pick up the values
        of the lights so they are offset with these. This function calls init_red_offset(), init_green_offset(),
        and init_blue_offset() to generate the values for the MASTER_OFFSET_TABLE. This table is for 7 AM to 11 PM
        cycle. Example index (not relative to the actual table ):
        MASTER_OFFSET_TABLE[420] == [ 120, 130, 140 ]. Therefore, MASTER_OFFSET_TABLE[420][0] == 120.

        (3) def init_red_offset(MASTER_OFFSET_TABLE):
        This function generates the red offset values for the MASTER_OFFSET_TABLE.

        (4) def init_green_offset(MASTER_OFFSET_TABLE):
        This function generates the green offset values for the MASTER_OFFSET_TABLE.

```

(5) def init_blue_offset(MASTER_OFFSET_TABLE):
This function generates the blue offset values for the MASTER_OFFSET_TABLE.

(6) def init_master_lux_table():
This function creates the lux offset values for the MASTER_LUX_TABLE. Returns a list containing lux offset values. These offset values are needed for the rgb sensor to calculate lux values because it doesn't naturally do so. The values are for every minute of the day base on 7 AM/11 PM cycle. Example index (not relative to the actual table):
MASTER_LUX_TABLE[420] == 50.

(7) def calc_user_tables(WAKE_UP_TIME, MASTER_CIRCADIAN_TABLE, MASTER_OFFSET_TABLE, MASTER_LUX_TABLE):
This function takes the user WAKE_UP_TIME, MASTER_CIRCADIAN_TABLE, MASTER_OFFSET_TABLE, and the MASTER_LUX_TABLE then shifts these tables based on the WAKE_UP_TIME. Returns a tuple with the newly shifted tables.

(8) def calc_Illuminance(lux, distance, angle):
This function takes a lux value, distance (in meters), and a viewing angle (in degrees). Returns the lumens value. Uses toArea(), toSr(), and toRad() in calculation. Source:

(9) def get_pids():
Returns a list with all of the process ids (pids) of the following scripts:
pir_sensor.py, rgb_sensor.py, usr_sensor.py, wait_for_cmd.py, and send_circadian_values.py

(10) def get_system_time():
Returns the system time in mintues.

(11) def get_ip():
Returns the local IP address of the Raspberry Pi.

(12) def create_log(cursor, db, message, user_name):
This function takes a database cursor object, database object, a message string, and username string. Stores the message into the database using execute_dB_query().

(13) def execute_dB_query(cursor, db, sql, sql_args):
This function takes a database cursor object, database object, an sql string, and a tuple of lists that contain the sql query arguments and executes the query.

(14) def get_circadian_cmd(USER_CIRCADIAN_TABLE, PREV_PRIMARY_COLORS, PREV_SECONDARY_COLORS, IS_PRIMARY_DEG, IS_SEC_ON, IS_SEC_DEG):
This function takes the USER_CIRCADIAN_TABLE, PREV_PRIMARY_COLORS, PREV_SECONDARY_COLORS, IS_PRIMARY_DEG, IS_SEC_ON, and IS_SEC_DEG lists as input. Returns a tuple containing the circadian string, list for new previous primary colors, and list for new previous secondary colors.

(III) pir_sensor.py:
This file is for the usage of the PIR motion sensor.

(a) Module imports:
(1) import time
(2) import os
(3) import signal
(4) import subprocess
(5) import circadian
(6) import MySQLdb
(7) import socket
(8) import datetime
(9) import RPi.GPIO as GPIO

(b) Signal handling:

These are the signal handler setups. When a kill -<number> is issued to the LINUX system, if it is one of the following numbers then it will be handled differently in the Python script. Theses are basically software interrupts.

```
signal.signal(3, handle_change_cmd)
signal.signal(4, catch_other_signals)
signal.signal(5, catch_other_signals)
signal.signal(6, handle_send_compensation)
signal.signal(7, handle_send_circadian)
signal.signal(8, handle_wait_for_cmd_dB_connect)
signal.signal(10, catch_other_signals)
signal.signal(11, handle_rgb_dB_connect)
signal.signal(12, handle_usr_dB_connect)
signal.signal(15, handle_send_circadian_dB_connect)
```

You will notice that throughout the scripts that some signal handler functions don't do anything and that may seem redundant. This was done on purpose for (1) to catch the signal and (2) to keep the signal handling consistent among the scripts.

(c) Functions:

(1) def catch_other_signals(signum, stack):
Does nothing but catch signals. Used with signal handling.

(2) def handle_change_cmd(signum, stack):
Catches kill -3. Simply performs time.sleep(3).

(3) def handle_send_compensation(signum, stack):
Catches kill -6. Simply performs time.sleep(3).

(4) def handle_send_circadian(signum, stack):
Catches kill -7. Simply performs time.sleep(3).

(5) def handle_wait_for_cmd_dB_connect(signum, stack):
Catches kill -8. Alerts the pir_sensor.py script that the wait_for_cmd.py script has connected to the database.

(6) def handle_rgb_dB_connect(signum, stack):
Catches kill -11. Alerts the pir_sensor.py script that the rgb_sensor.py script has connected to the database.

(7) def handle_usr_dB_connect(signum, stack):
Catches kill -12. Alerts the pir_sensor.py script that the usr_sensor.py script has connected to the database.

(8) def handle_send_circadian_dB_connect(signum, stack):
Catches kill -15. Alerts the pir_sensor.py script that the send_circadian_values.py script has connected to the database.

(9) def handle_motion_detection(PIR_PIN):
This function is the hardware interrupt handler for the PIR sensor.

(IV) rgb_sensor.py:

This file is for the usage of the RGB sensor.

(a) Module imports;

(1) import time
(2) import os
(3) import signal
(4) import subprocess
(5) import circadian
(6) import MySQLdb
(7) import socket
(8) import datetime

(9) import smbus

(b) Signal handling: (See section in pir_sensor.py for more info on signal handling.)

```
signal.signal(3, handle_change_cmd)
signal.signal(4, handle_sleep_mode)
signal.signal(5, handle_wake_up)
signal.signal(6, catch_other_signals)
signal.signal(7, handle_send_circadian)
signal.signal(8, handle_wait_for_cmd_dB_connect)
signal.signal(10, handle_pir_dB_connect)
signal.signal(11, catch_other_signals)
signal.signal(12, handle_usr_dB_connect)
signal.signal(15, handle_send_circadian_dB_connect)
```

(c) Functions:

(1) def catch_other_signals(signum, stack):
Does nothing but catch signals. Used with signal handling.

(2) def handle_change_cmd(signum, stack):
Catches kill -3. Handles when the user changes a parameter on the website.

(3) def handle_sleep_mode(signum, stack):
Catches kill -4. Updates database with sensor reading of 0 when system goes into sleep mode.

(4) def handle_wake_up(signum, stack):
Catches kill -5. Used in waking from sleep mode.

(5) def handle_send_circadian(signum, stack):
Catches kill -7. Simply does time.sleep(3).

(6) def handle_wait_for_cmd_dB_connect(signum, stack):
Catches kill -8. Tells rgb_sensor.py that wait_for_cmd.py connected to the database.

(7) def handle_pir_dB_connect(signum, stack):
Catches kill -10. Tells rgb_sensor.py that pir_sensor.py connected to the database.

(8) def handle_usr_dB_connect(signum, stack):
Catches kill -12. Tells rgb_sensor.py that usr_sensor.py connected to the database.

(9) def handle_send_circadian_dB_connect(signum, stack):
Catches kill -15. Tells rgb_sensor.py that sends_circadian_values.py connected to the database.

(V) send_circadian_values.py:
This file is responsible for sending circadian values to the lighting subsystem. This sends compensation values as well.

(a) Module imports:

(1) import time
(2) import os
(3) import signal
(4) import subprocess
(5) import circadian
(6) import MySQLdb
(7) import socket
(8) import datetime

(b) Signal handling: (See section in pir_sensor.py for more info on signal handling.)

```
signal.signal(3, handle_change_cmd)
signal.signal(4, handle_sleep_mode)
signal.signal(5, handle_wake_up)
signal.signal(6, handle_send_compensation)
signal.signal(7, catch_other_signals)
signal.signal(8, handle_wait_for_cmd_dB_connect)
signal.signal(10, handle_pir_dB_connect)
```

```
        signal.signal(11, handle_rgb_dB_connect)
        signal.signal(12, handle_usr_dB_connect)
        signal.signal(15, catch_other_signals)

    (c) Functions:
        (1) def catch_other_signals(signum, stack):
        Catches signals. Does nothing else.

        (2) def handle_change_cmd(signum, stack):
        Catches kill -3. Used for when a user changes something from the website.

        (3) def handle_sleep_mode(signum, stack):
        Catches kill -4. Sends values to the lighting subsystem to put it to sleep.

        (4) def handle_wake_up(signum, stack):
        Catches kill -5. Makes the script send a value to wake the lights up.

        (5) def handle_send_compensation(signum, stack):
        Catches kill -6. Sends compensation values to the lighting subsystem.

        (6) def handle_wait_for_cmd_dB_connect(signum, stack):
        Catches kill -8. Alerts the send_circadian_values.py that wait_for_cmd.py
        connected to the DB.

        (7) def handle_pir_dB_connect(signum, stack):
        Catches kill -10. Alerts the send_circadian_values.py that pir_sensor.py
        connected to the DB.

        (8) def handle_rgb_dB_connect(signum, stack):
        Catches kill -11. Alerts the send_circadian_values.py that rgb_sensor.py
        connected to the DB.

        (9) def handle_usr_dB_connect(signum, stack):
        Catches kill -12. Alerts the send_circadian_values.py that usr_sensor.py
        connected to the DB.

(VI) usr_sensor.py:
This file is responsible for the usage of the ultra sonic range sensor.

    (a) Module imports:
        (1)  import time
        (2)  import os
        (3)  import signal
        (4)  import subprocess
        (5)  import circadian
        (6)  import MySQLdb
        (7)  import socket
        (8)  import datetime
        (9)  import math
        (10) import RPi.GPIO as GPIO

    (b) Signal handling: (See section in pir_sensor.py for more info on signal handling.)
        signal.signal(3, catch_other_signals)
        signal.signal(4, catch_other_signals)
        signal.signal(5, catch_other_signals)
        signal.signal(6, catch_other_signals)
        signal.signal(7, catch_other_signals)
        signal.signal(8, handle_wait_for_cmd_dB_connect)
        signal.signal(10, handle_pir_dB_connect)
        signal.signal(11, handle_rgb_dB_connect)
        signal.signal(12, catch_other_signals)
        signal.signal(15, handle_send_circadian_dB_connect)

    (c) Functions:
        (1) def catch_other_signals(signum, stack):
        Catches signals. Does nothing else.

        (2) def handle_wait_for_cmd_dB_connect(signum, stack):
        Catches kill -8. Tells usr_sensor.py that wait_for_cmd.py connected to the DB.
        (3) def handle_pir_dB_connect(signum, stack):
        Catches kill -10. Tells usr_sensor.py that pir_sensor.py connected to the DB.

        (4) def handle_rgb_dB_connect(signum, stack):
        Catches kill -11. Tells usr_sensor.py that rgb_sensor.py connected to the DB.

        (5) def handle_send_circadian_dB_connect(signum, stack):
        Catches kill -15. Tells usr_sensor.py that send_circadian_values.py connected to the DB.

(VII) wait_for_cmd.py:
This file is responsible for receiving commands from the website.

    (a) Module imports:
        (1) import time
        (2) import os
        (3) import signal
        (4) import subprocess
        (5) import circadian
        (6) import MySQLdb
        (7) import socket
        (8) import datetime

    (b) Signal handling: (See section in pir_sensor.py for more info on signal handling.)
        signal.signal(3, catch_other_signals)
        signal.signal(4, handle_sleep_mode)
        signal.signal(5, handle_wake_up)
        signal.signal(6, catch_other_signals)
        signal.signal(7, catch_other_signals)
        signal.signal(8, catch_other_signals)
        signal.signal(10, handle_pir_dB_connect)
        signal.signal(11, handle_rgb_dB_connect)
        signal.signal(12, handle_usr_dB_connect)
        signal.signal(15, handle_send_circadian_dB_connect)

    (c) Functions:
        (1) def catch_other_signals(signum, stack):
        Catches signals and does nothing else.

        (2) def handle_pir_dB_connect(signum, stack):
        Catches kill -10. Tells wait_for_cmd.py that pir_sensor.py connected to the DB.

        (3) def handle_rgb_dB_connect(signum, stack):
        Catches kill -11. Tells wait_for_cmd.py that rgb_sensor.py connected to the DB.

        (4) def handle_usr_dB_connect(signum, stack):
        Catches kill -12. Tells wait_for_cmd.py that usr_sensor.py connected to the DB.

        (5) def handle_send_circadian_dB_connect(signum, stack):
        Catches kill -15. Tells wait_for_cmd.py that send_circadian_values.py connected to the DB.

        (6) def handle_sleep_mode(signum, stack):
        Catches kill -4. Tells the script that the system entered sleep mode.

        (7) def handle_wake_up(signum, stack):
        Catches kill -5. Tells the script that the system exited sleep mode.

        (8) def boot_up():
        Initial database check to pair with the lighting subsystem and retrieve previous sensor subsystem settings if
        any.

(VIII) stop.py
This script is used for killing the sensor subsystem. Usage:

$ python stop.py
```

****DO NOT USE sudo python!!!

(IX) pause.py
This script is used for suspending the sensor and lighting subsystems. Usage:

```
$ python pause.py
```

****DO NOT USE sudo python!!!

(X) compensate.txt:
Holds the compensation data.

(XI) sensor_data.txt:
Holds the sensor readings. Uses in compensation.

(XII) config.txt:
Holds the values that were sent by the user. Needed because DB was misbehaving.

```
######################################################################################################
```

Project: Lighting Subsystem

Authors: Team Spatium Lucis

Version: v1.0

Target Device: Raspberry Pi 3

Files: lighting_sub.py, pause.py

Last edited: August 10, 2017

Note: This code is not as polished as the sensor subsystem one. I didn't really get the time to separate the code into different scripts and stuff like the sensor code because of my summer class/work schedule. It's okay though because the bottle necks in performance came from the sensor codes various threading locks, delays, etc. and the lighting code doesn't really have these. Most of the thread are more or less independent, and the current code, although long and at points redundant, still works like a charm :)

```
######################################################################################################
```

(I) lighting_sub.py:
This is the init and main script for the lighting subsystem. Usage:

```
$ sudo python lighting_sub.py
```

NOTE: You may get some error saying that some address is already in use. If this happens then do the following:

```
    (1) Perform a CTRL + Z
    (2) Type ps -al and hit enter
    (3) Locate the 'pid' for 'sudo'
    (4) Type sudo kill -9 <pid for 'sudo'> and hit enter
    (5) Try to run the lighting_sub.py script again

    (a) Module Imports:
        (1) import time
        (2) import socket
        (3) import threading
        (4) import RPi.GPIO as GPIO
        (5) import MySQLdb
        (6) import wiringpi
        (7) import sys
        (8) import os
```

(9) import signal

(b) Section of code after imports and before the functions:

There is a chunk of code after the imports and before the functions that basically setting up the PWM pin information for the lights. The wiringpi python module was used because the RPi.GPIO module would produce this ridiculous flickering that was comparable to a camera flash when the lights were dim (like early morning/late evening values). Trust me, continue to use this module.

Because the Rpi3 only has really 1 hardware PWM pin, we used soft PWMs for the lights. These actually look pretty good but will never beat a hardware PWM.

Example soft PWM python code:

```
wiringpi.wiringPiSetupGpio()
GPIO.setmode(GPIO.BCM) # from the Rpi.GPIO module. Needed to set pin mode and for relays.
wiringpi.pinMode(17, 17)
wiringpi.softPwmCreate(17, 0, 100)
```

This will create a softPWM channel on pin 17(BCM) with frequency 100Hz.

BCM Pins:

(1) Pin 17: Primary Red
(2) Pin 27: Primary Green
(3) Pin 22: Primary Blue
(4) Pin 6: Secondary Red #the code has a comment saying pin 29 for some reason. Ignore it.
(5) Pin 13: Secondary Green
(6) Pin 26: Secondary Blue

The rest of this section of code simply establishes some mutex locks for the threads and a couple of global variables.

(c) Functions:

(1) def get_ip():
Gets the local IP address of the lighting subsystem and returns it as a string

(2) def boot_up():
Checks the database for an existing entry of the local IP address. If it exists then move on to wait for the sensor subsystem to connect. If it does not exist then it will be added to the database and wait to be paired.

(3) def begin_threading():
Creates the pir_thread (waits for sleep/wake values), delete_thread (when the system is deleted), the comp_thread (waits for compensation values), and the light_cmd thread (waits for circadian commands). Also creates some threading events to help with synchronization.

(4) def delete_cmd():
Receives a command from the sensor subsystem that the system is begin deleted. Turns the lights off then ends the script.

(5) def comp_cmd():
Receives the compensation command from the sensor subsystem (see sensor subsystem for command format) and brightens/dims the lights accordingly. There are 64 combinations depending on which lights are brightening or dimming (6 lights -> 2^6 = 64). I basically made a truth table for the lights.

(6) Handler Functions:
These functions are used for the threads that are spawned in the comp_cmd() and

```
        light_cmd() threads.
        They are used to change the primary or secondary lights to a certain brightness using the PWM.

        (7) def PIR_cmd():
        The function for the pir thread. Receives a command from the sensor subsystem to turn the lights
        off for sleep mode, or wake them up.

        (8) def light_cmd():
        This function is basically identical to the comp_cmd() function except it uses a different
        command string format. See sensor subsystem for the proper format.


(II) pause.py:
This script is used to suspend the lighting subsystem for whatever reason. Usage:

$ sudo python pause.py
"""


```