1. Software

To-date, out accomplishments include the completion of our RS-485 serial communications code, which both the Raspberry Pi and the Arduinos will us to communicate:

RS485Protocol.h

```
#ifndef RS485Protocol_h
#   define RS485Protocol_h
#   include <stdio.h>
#endif

#ifdef (ARDUINO)
#   include <Arduino.h>
#   include <ArduinoSetup.h>
#elif (RASPBERRY_PI)         /*
#   include <wiringPi.h>
#   include <RaspberryPiSetup.h>
#   include <wiringSerial.h> */
#endif

typedef unsigned char byte;
typedef void (*WriteByte) (const byte data);
typedef int (*AvailableByte) ();
typedef int (*ReadByte) ();

void sendMsg (WriteByte, const byte*, const byte);

byte recvMsg (AvailableByte, ReadByte, byte*,
              const byte /*, unsigned long timeOut = 500*/);
```

RS485Protocol.c

```
#include <RS485Protocol.h>

#define TIMEOUT 500

const byte START = 0x55;
const byte END = 0xAA;

// creates a checksum for use as CRC byte @ end of forward/reverse frames
static unsigned char crc (const byte *addr, byte length) {
    byte c = 0;

    while (length--) {
        byte in = *addr++;
        for (byte i = 8; i; i--) {
            byte q = (c ^ in) & 0x01;
            c >>= 1;
            if (q) c^= 0x8C;
            in >>= 1;
        }
    }
    return c;
}

// sends each byte after splitting into two nibbles and transmitting first
// nibble w/ its complement, then the second nibble w/ its complement
void sendComp (WriteByte wByte, const byte data) {

    byte a;

    a = data >> 4;
    wByte ((a << 4) | (a ^ 0x0F));

    a = data & 0x0F;
    wByte ((a << 4) | (a ^ 0x0F));
}

void sendMsg (WriteByte wByte, const byte *sData, const byte len) {
    wByte (START);
    for (byte i = 0; i < len; i++) sendComp (wByte, sData[i]);
    wByte (END);
    sendComp (wByte, crc (sData, len));
}

byte recvMsg (AvailableByte avByte, ReadByte rByte,
```

```
          byte *sData, const byte len) {

    unsigned long startTime = millis();

    bool hasStart = false,
         hasEnd = false,
         first;

    byte inCh, current;

    while (millis() - startTime < TIMEOUT) {
        if ( avByte() > 0) {
            byte inputByte = rByte();

            switch (inputByte) {
                case START:
                    hasStart = true;
                    hasEnd = false;
                    inCh = 0;
                    first = true;
                    startTime = millis();
                    break;

                case END:
                    hasEnd = true;
                    break;

                default:
                    if (!hasStart) break;   // no start byte
                    if ((inputByte >> 4) != (inputByte & 0x0F) ^ 0x0F)
                        return -1;          // bad data
                    inputByte >>= 4;

                    if (first) { current = inputByte; first = false; break; }
                    current <<= 4;
                    current |= inputByte;
                    first = true;

                    if (hasEnd) {           // if ended, next byte = CRC
                        if (crc (sData, inCh) != current)
                            return 0;       // bad CRC
                        return inCh;
                    }
                                            // if not full, keep adding data
                    if (inCh < len) sData[inCh++] = current;
                    else return -2;          // means overflow
                    break;
            }
        }
    }
    return 0;

}
```

Intensity Sensor Software Code

```
int sensorPin = A0;    // select the input pin for the potentiometer

float rawRange = 1024; // 3.3v
float logRange = 5.0; // 3.3v = 10^5 lux

void setup()
{
  analogReference(EXTERNAL); //
  Serial.begin(9600);
  Serial.println("Adafruit Analog Light Sensor Test");
}

void loop()
{
  // read the raw value from the sensor:
  int rawValue = analogRead(sensorPin);

  Serial.print("Raw = ");
  Serial.print(rawValue);
  Serial.print(" - Lux = ");
  Serial.println(RawToLux(rawValue));
  delay(1000);
}

float RawToLux(int raw)
{
  float logLux = raw * logRange / rawRange;
  return pow(10, logLux);
}
```

Spectrum Software Sensor

```
#include <Wire.h>

#include "AS7265X.h"


AS7265X::AS7265X(uint8_t intPin)
{
  _intPin = intPin;
}

void AS7265X::init(uint8_t gain, uint8_t mode, uint8_t intTime)
{
    i2cm_AS72xx_write(AS72651_DEV_SEL, 0);
    i2cm_AS72xx_write(AS72651_LED_CONFIG, 0x00 );  // turn off led indication on device 1

    // enable interrupt (bit 6), set gain and mode
    i2cm_AS72xx_write(AS72651_CONTROL_SETUP, 0x40 | gain << 4 | mode << 2 );

    // set integration time
    i2cm_AS72xx_write(AS72651_INT_TIME, intTime );

}

uint8_t AS7265X::getStatus()
{

  uint8_t c = i2cm_AS72xx_read(AS72651_CONTROL_SETUP);
  return c;
}


void AS7265X::readRawData(int16_t * destination)
{
  uint8_t rawData[2];

  // collect R,S,T,U,V, W data
  i2cm_AS72xx_write(AS72651_DEV_SEL, 0);
  for(int i = 0; i < 6; i++)
  {
    for(int j = 0; j < 2; j++)
    {
    rawData[j] = i2cm_AS72xx_read(AS72651_RAW_VALUE_0_H + 2*i + j);
    }

    destination[i] = (int16_t) ( ((int16_t) rawData[0] << 8) | rawData[1]);
```

```
  }

  // collect J,I,G,H,K, L data
  i2cm_AS72xx_write(AS72651_DEV_SEL, 1);
  for(int i = 0; i < 6; i++)
  {
    for(int j = 0; j < 2; j++)
    {
    rawData[j] = i2cm_AS72xx_read(AS72651_RAW_VALUE_0_H + 2*i + j);
    }

    destination[i + 6] = (int16_t) ( ((int16_t) rawData[0] << 8) | rawData[1]);

  }

  //collect D,C,A,B,E, F data
  i2cm_AS72xx_write(AS72651_DEV_SEL, 2);
  for(int i = 0; i < 6; i++)
  {
    for(int j = 0; j < 4; j++)
    {
    rawData[j] = i2cm_AS72xx_read(AS72651_RAW_VALUE_0_H + 2*i + j);
    }

   destination[i + 12] = (int16_t) ( ((int16_t) rawData[0] << 8) | rawData[1]);

  }

}


void AS7265X::readCalData(float * destination)
{
  uint8_t rawData[4];

  // collect R,S,T,U,V, W data
  i2cm_AS72xx_write(AS72651_DEV_SEL, 0);
  for(int i = 0; i < 6; i++)
  {
    for(int j = 0; j < 4; j++)
    {
    rawData[j] = i2cm_AS72xx_read(AS72651_CAL_CHAN0_0 + 4*i + j);
    }

    uint32_t x = ((uint32_t) rawData[0] << 24) | ((uint32_t) rawData[1] << 16) | ((uint32_t) rawData[2] <<
8) | rawData[3];
    destination[i] = *(float*)&x;
```

```
  }

  // collect J,I,G,H,K, L data
  i2cm_AS72xx_write(AS72651_DEV_SEL, 1);
  for(int i = 0; i < 6; i++)
  {
    for(int j = 0; j < 4; j++)
    {
    rawData[j] = i2cm_AS72xx_read(AS72651_CAL_CHAN0_0 + 4*i + j);
    }

    uint32_t x = ((uint32_t) rawData[0] << 24) | ((uint32_t) rawData[1] << 16) | ((uint32_t) rawData[2] <<
8) | rawData[3];
    destination[i + 6] = *(float*)&x;

  }

  //collect D,C,A,B,E, F data
  i2cm_AS72xx_write(AS72651_DEV_SEL, 2);
  for(int i = 0; i < 6; i++)
  {
    for(int j = 0; j < 4; j++)
    {
    rawData[j] = i2cm_AS72xx_read(AS72651_CAL_CHAN0_0 + 4*i + j);
    }

    uint32_t x = ((uint32_t) rawData[0] << 24) | ((uint32_t) rawData[1] << 16) | ((uint32_t) rawData[2] <<
8) | rawData[3];
    destination[i + 12] = *(float*)&x;

  }

}


uint8_t AS7265X::getDevType()
{
  uint8_t c = i2cm_AS72xx_read(AS72651_DEVICE_TYPE);
  return c;
}

uint8_t AS7265X::getHWVersion()
{
  uint8_t c = i2cm_AS72xx_read(AS72651_HW_VERSION);
  return c;
}
```

```
uint16_t AS7265X::getFWMajorVersion()
{
  uint8_t rawData[2] = {0, 0};
  i2cm_AS72xx_write(AS72651_FW_VERSION_H, 0x01);
  rawData[1] = i2cm_AS72xx_read(AS72651_FW_VERSION_H);
  i2cm_AS72xx_write(AS72651_FW_VERSION_L, 0x01);
  rawData[0] = i2cm_AS72xx_read(AS72651_FW_VERSION_L);
  uint16_t c = (uint16_t)( ( (uint16_t) rawData[1] << 8) | rawData[0]);
  return c;
}


uint16_t AS7265X::getFWPatchVersion()
{
  uint8_t rawData[2] = {0, 0};
  i2cm_AS72xx_write(AS72651_FW_VERSION_H, 0x02);
  rawData[1] = i2cm_AS72xx_read(AS72651_FW_VERSION_H);
  i2cm_AS72xx_write(AS72651_FW_VERSION_L, 0x02);
  rawData[0] = i2cm_AS72xx_read(AS72651_FW_VERSION_L);
  uint16_t c = (uint16_t)( ( (uint16_t) rawData[1] << 8) | rawData[0]);
  return c;
}


uint16_t AS7265X::getFWBuildVersion()
{
  uint8_t rawData[2] = {0, 0};
  i2cm_AS72xx_write(AS72651_FW_VERSION_H, 0x03);
  rawData[1] = i2cm_AS72xx_read(AS72651_FW_VERSION_H);
  i2cm_AS72xx_write(AS72651_FW_VERSION_L, 0x03);
  rawData[0] = i2cm_AS72xx_read(AS72651_FW_VERSION_L);
  uint16_t c = (uint16_t)( ( (uint16_t) rawData[1] << 8) | rawData[0]);
  return c;
}


float AS7265X::getTemperature(uint8_t devNum)
{
   i2cm_AS72xx_write(AS72651_DEV_SEL, devNum);
   float c = i2cm_AS72xx_read(AS72651_DEV_TEMP);
   return c;
}


void AS7265X::configureLed(uint8_t ledIndCurrent, uint8_t ledDrvCurrent, uint8_t devNum)
{
     i2cm_AS72xx_write(AS72651_DEV_SEL, devNum);
     i2cm_AS72xx_write(AS72651_LED_CONFIG, ledDrvCurrent << 4 | ledIndCurrent < 1);
```

```
}


void AS7265X::enableIndLed(uint8_t devNum)
{
  i2cm_AS72xx_write(AS72651_DEV_SEL, devNum);
  uint8_t c = i2cm_AS72xx_read(AS72651_LED_CONFIG);
  i2cm_AS72xx_write(AS72651_LED_CONFIG, c | 0x01 );
}


void AS7265X::disableIndLed(uint8_t devNum)
{
  i2cm_AS72xx_write(AS72651_DEV_SEL, devNum);
  uint8_t c = i2cm_AS72xx_read(AS72651_LED_CONFIG);
  i2cm_AS72xx_write(AS72651_LED_CONFIG, c &  ~(0x01) );
}


void AS7265X::enableDrvLed(uint8_t devNum)
{
  i2cm_AS72xx_write(AS72651_DEV_SEL, devNum);
  uint8_t c = i2cm_AS72xx_read(AS72651_LED_CONFIG);
  i2cm_AS72xx_write(AS72651_LED_CONFIG, c | 0x08 );
}


void AS7265X::disableDrvLed(uint8_t devNum)
{
  i2cm_AS72xx_write(AS72651_DEV_SEL, devNum);
  uint8_t c = i2cm_AS72xx_read(AS72651_LED_CONFIG);
  i2cm_AS72xx_write(AS72651_LED_CONFIG, c & ~(0x08) );
}


void AS7265X::i2cm_AS72xx_write(uint8_t virtualReg, uint8_t d)
 {
 volatile uint8_t status;
 while (1)
  {
  // Read slave I2C status to see if we can write the reg address.
  status = readByte(AS72651_ADDRESS, I2C_AS72XX_SLAVE_STATUS_REG);
  if ((status & I2C_AS72XX_SLAVE_TX_VALID) == 0)
  // No inbound TX pending at slave. Okay to write now.
  break ;
  }

 // Send the virtual register address
```

```
  // (setting bit 7 to indicate a pending write).
  writeByte(AS72651_ADDRESS, I2C_AS72XX_SLAVE_WRITE_REG, (virtualReg | 0x80)) ;
  while (1)
    {
    // Read the slave I2C status to see if we can write the data byte.
    status = readByte(AS72651_ADDRESS, I2C_AS72XX_SLAVE_STATUS_REG) ;
    if ((status & I2C_AS72XX_SLAVE_TX_VALID) == 0)
    // No inbound TX pending at slave. Okay to write data now.
    break ;
    }

  // Send the data to complete the operation.
  writeByte(AS72651_ADDRESS, I2C_AS72XX_SLAVE_WRITE_REG, d) ;
}


uint8_t AS7265X::i2cm_AS72xx_read(uint8_t virtualReg)
{
  volatile uint8_t status, d ;
  while (1)
    {
    // Read slave I2C status to see if we can write the reg address.
    status = readByte(AS72651_ADDRESS, I2C_AS72XX_SLAVE_STATUS_REG) ;
    if ((status & I2C_AS72XX_SLAVE_TX_VALID) == 0)
    // No inbound TX pending at slave. Okay to write now.
    break ;
    }

  // Send the virtual register address
  // (setting bit 7 to indicate a pending write).
  writeByte(AS72651_ADDRESS, I2C_AS72XX_SLAVE_WRITE_REG, virtualReg) ;
  while (1)
    {
    // Read the slave I2C status to see if our read data is available.
    status = readByte(AS72651_ADDRESS, I2C_AS72XX_SLAVE_STATUS_REG) ;
    if ((status & I2C_AS72XX_SLAVE_RX_VALID) != 0)
    // Read data is ready for us.
    break ;
    }
  // Read the data to complete the operation.
  d = readByte(AS72651_ADDRESS, I2C_AS72XX_SLAVE_READ_REG) ;
  return d ;
}


// I2C scan function
void AS7265X::I2Cscan()
{
```

```
// scan for i2c devices
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmission to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");

      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknown error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

}


// I2C read/write functions for the AS72651 sensor

  void AS7265X::writeByte(uint8_t address, uint8_t subAddress, uint8_t data)
{
  Wire.beginTransmission(address);  // Initialize the Tx buffer
  Wire.write(subAddress);           // Put slave register address in Tx buffer
  Wire.write(data);                 // Put data in Tx buffer
```

```
  Wire.endTransmission();          // Send the Tx buffer
}

  uint8_t AS7265X::readByte(uint8_t address, uint8_t subAddress)
{
  uint8_t data = 0;                    // `data` will store the register data
  Wire.beginTransmission(address);         // Initialize the Tx buffer
  Wire.write(subAddress);                // Put slave register address in Tx buffer
  Wire.endTransmission(false);           // Send the Tx buffer, but send a restart to keep connection alive
  Wire.requestFrom(address, 1);           // Read one byte from slave register address
  data = Wire.read();                   // Fill Rx buffer with result
  return data;                         // Return data read from slave register
}
```

Temperature Software Sensor

```
#include <OneWire.h>
#include <DallasTemperature.h>

// Data wire is conntec to the Arduino digital pin 4
#define ONE_WIRE_BUS 4

// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);

void setup(void)
{
  // Start serial communication for debugging purposes
  Serial.begin(9600);
  // Start up the library
  sensors.begin();
}

void loop(void){
  // Call sensors.requestTemperatures() to issue a global temperature and Requests to all devices on the
bus
  sensors.requestTemperatures();

  Serial.print("Celsius temperature: ");
  // Why "byIndex"? You can have more than one IC on the same bus. 0 refers to the first IC on the wire
  Serial.print(sensors.getTempCByIndex(0));
  Serial.print(" - Fahrenheit temperature: ");
  Serial.println(sensors.getTempFByIndex(0));
  delay(1000);
}
```

Thermocouple Software Code

```
#include <Adafruit_MAX31856.h>

// Use software SPI: CS, DI, DO, CLK
Adafruit_MAX31856 maxthermo = Adafruit_MAX31856(10, 11, 12, 13);
// use hardware SPI, just pass in the CS pin
//Adafruit_MAX31856 maxthermo = Adafruit_MAX31856(10);
const int selectPins[3]={7,8,9};
const int muxEnable=16;
const int thermFault=15;
int tempArray[8];
byte tArray[8];

void setup() {
  Serial.begin(115200);
  Serial.println("MAX31856 thermocouple test");
  pinMode(muxEnable, OUTPUT);
  pinMode(thermFault, INPUT);
  digitalWrite(muxEnable,HIGH);
  for(int i=0; i<3; i++)
  {
    pinMode(selectPins[i], OUTPUT);
    digitalWrite(selectPins[i], LOW);
  }

  maxthermo.begin();

  maxthermo.setThermocoupleType(MAX31856_TCTYPE_T);

  Serial.print("Thermocouple type: ");
  switch (maxthermo.getThermocoupleType() ) {
    case MAX31856_TCTYPE_B: Serial.println("B Type"); break;
    case MAX31856_TCTYPE_E: Serial.println("E Type"); break;
    case MAX31856_TCTYPE_J: Serial.println("J Type"); break;
    case MAX31856_TCTYPE_K: Serial.println("K Type"); break;
    case MAX31856_TCTYPE_N: Serial.println("N Type"); break;
    case MAX31856_TCTYPE_R: Serial.println("R Type"); break;
    case MAX31856_TCTYPE_S: Serial.println("S Type"); break;
    case MAX31856_TCTYPE_T: Serial.println("T Type"); break;
    case MAX31856_VMODE_G8: Serial.println("Voltage x8 Gain mode"); break;
    case MAX31856_VMODE_G32: Serial.println("Voltage x8 Gain mode"); break;
    default: Serial.println("Unknown"); break;
  }


}
```

```
void loop() {
  for(byte pin=0; pin<=7; pin++)
  {
    selectMuxPin(pin);
    tArray[pin]=thermTemp();
    tempArray[static_cast<int>(pin)] = tArray[pin];
  }
  // Check and print any faults
  uint8_t fault = maxthermo.readFault();
  if (fault) {
    if (fault & MAX31856_FAULT_CJRANGE) Serial.println("Cold Junction Range Fault");
    if (fault & MAX31856_FAULT_TCRANGE) Serial.println("Thermocouple Range Fault");
    if (fault & MAX31856_FAULT_CJHIGH)  Serial.println("Cold Junction High Fault");
    if (fault & MAX31856_FAULT_CJLOW)   Serial.println("Cold Junction Low Fault");
    if (fault & MAX31856_FAULT_TCHIGH)  Serial.println("Thermocouple High Fault");
    if (fault & MAX31856_FAULT_TCLOW)   Serial.println("Thermocouple Low Fault");
    if (fault & MAX31856_FAULT_OVUV)    Serial.println("Over/Under Voltage Fault");
    if (fault & MAX31856_FAULT_OPEN)    Serial.println("Thermocouple Open Fault");
  }

  for(int i = 0; i < 8; i++){
    Serial.println("temp:");
    Serial.print(tempArray[i]);
  }
  delay(1000);
}

void selectMuxPin(byte pin)
{
  for(int i=0; i<3; i++)
  {
    if (pin &(1<<i)) digitalWrite (selectPins[i], HIGH);
    else
      digitalWrite(selectPins[i], LOW);
  }
}
int thermTemp()
{
  Serial.print("Cold Junction Temp: ");
  Serial.println(maxthermo.readCJTemperature());

  Serial.print("Thermocouple Temp: ");
  Serial.println(maxthermo.readThermocoupleTemperature());
return maxthermo.readThermocoupleTemperature();
}
```

Graphical User Interface – Python

```
from tkinter import Tk, Label, OptionMenu, StringVar, Toplevel, LabelFrame
from tkinter import Entry, END, Frame, Listbox, Scrollbar, Canvas, Button
import sqlite3
import tkinter.messagebox as tmb
import tkinter.ttk as ttk

# MAIN LOOP
root = Tk()
root.configure(background='#9DD9F9')
root.title('Home     -     Hyperion Intelligent Lighting System')
"""root.iconbitmap('NASA.ico')"""

# set the number of crew and/or plant zones
crew_zones = 1
plant_zones = 1

crew_zones_list = []
plant_zones_list = []
# creating lists out of the number of crew and plant zones
for i in range(0, crew_zones):
   crew_zones_list.append('CREW ZONE 'f'{i + 1}')
for i in range(0, plant_zones):
   plant_zones_list.append('PLANT ZONE 'f'{i + 1}')

# database created for the crew profiles - name/ID (column 0) and respective time offsets from
UTC/GMT time (in hours, column 0)
conn = sqlite3.connect('crew_profile_db.db')
c = conn.cursor()

# database created for the plant profiles - name (column 0) and respective photoperiod (in hours,
column 1)
conn2 = sqlite3.connect('plant_profile_db.db')
c2 = conn2.cursor()


## separate databases for crew and plant profiles
## already created so they are both commented out
## ********** IF THEY AREN'T CREATED YET, REMOVE THE COMMENTS, RUN THE PROGRAM, AND
THEN PLACE THE COMMENTS BACK **********

#c.execute("""CREATE TABLE crew_profile_db (
#          profile name text,
#          time offset integer
#     )""")
```

```
#c2.execute("""CREATE TABLE plant_profile_db (
#          profile name text,
#          photoperiod integer
#      )""")

# ********** REMOVE COMMENTS UP TO HERE **********
###Alerts Portion of GUI###
def alerts_button():
    # This is the warnings interface portion of the GUI
    # Must have the following functions: Display latest Alert on the main menu,
    # display a full list of uncleared alerts when prompted, display a full list
    # of cleared alerts (all logs), and alert settings(?)

    """import tkinter as tk  # using tkinter library to create GUI
    import tkinter.messagebox as tmb  # message box tool
    import tkinter.ttk as ttk  # used for tree view of database
    import sqlite3  # database tool"""

    def home():  # home button function; terminates the program
        main.destroy()

    # create database or connect to one
    conn = sqlite3.connect( 'all_alerts.db' )

    # create a cursor
    cursor = conn.cursor()

    # create table
    # ******* take comments if table is not made yet, run the code, put the comments back *******
    #cursor.execute( """CREATE TABLE alerts_table (
    #            alert_type text,
    #            zone int,
    #            info float,
    #            record_id int
    #            )""" )

    #cursor.execute( """CREATE TABLE logs_table (
    #            alert_type text,
    #            zone int,
    #            info float,
    #            record_id int,
    #            log_entry str
    #            )""" )"""

    # create update function (pressing the Update Alerts button)

    def update():
        # create database or connect to one
```

```
    tree.delete( *tree.get_children() )
    conn = sqlite3.connect( 'all_alerts.db' )

    # create a cursor
    cursor = conn.cursor()
    record_id = delete_box.get()
    cursor.execute( """UPDATE alerts_table SET
            alert_type = :alert,
            zone = :zone,
            info = :info

            WHERE oid= :oid""",
               {'alert': type_editor.get(),
                'zone': zone_editor.get(),
                'info': info_editor.get(),
                'oid': record_id} )

    # commit changes
    conn.commit()

    # close connection
    conn.close()
    editor.destroy()

def all_alerts():  # all alerts button function; displays all uncleared alerts

    def delete_alert():
        # create database or connect to one
        conn = sqlite3.connect( 'all_alerts.db' )

        # create a cursor
        cursor = conn.cursor()

        cursor.execute( "DELETE from alerts_table WHERE oid=" + delete_box.get() )

        delete_box.delete( 0, tk.END )
        enter_log_box.delete( 0, tk.END )

        # commit changes
        conn.commit()

        # close connection
        conn.close()

    def log_entry():
        i_d = delete_box.get()
        log = enter_log_box.get()
```

```
    conn = sqlite3.connect( "all_alerts.db" )
    cursor = conn.cursor()

    cursor.execute( "SELECT * from alerts_table WHERE oid=" + delete_box.get() )
    data = cursor.fetchall()

    data_list = list( data )

    # Insert links into table
    for i in data_list:
        cursor.execute( "INSERT INTO logs (alert_type, zone, info) VALUES(?, ?, ?)", (i) )
    conn.commit()
    delete_alert()

    cursor.execute( "INSERT INTO logs (record_id, log_entry) VALUES (?,?)",
                (delete_box.get(), enter_log_box.get()) )

    print( log )
    # print(i_d)

    enter_log_box.delete( 0, tk.END )
    delete_box.delete( 0, tk.END )

    conn.commit()
    conn.close()

all_alerts_window = Toplevel( bg='#9DD9f9' )
all_alerts_window.title( 'All Alerts     -     Hyperion Intelligent Lighting System ' )
all_alerts_window.iconbitmap( 'nasa_PLL_icon.ico' )
all_alerts_window.geometry( "800x600" )

all_alerts_frame = Frame( all_alerts_window, bg='#9DD9f9' )
all_alerts_frame.grid( row=0, column=0, padx=5, pady=5, ipadx=10 )

# create database or connect to one
conn = sqlite3.connect( 'all_alerts.db' )

# create a cursor
cursor = conn.cursor()

# create submit button

delete_box_label = Label( all_alerts_frame, text='Enter ID' )
delete_box_label.grid( row=0, column=0, padx=3, pady=5 )
delete_box = Entry( all_alerts_frame, width=5 )
delete_box.grid( row=0, column=1, padx=3, pady=5 )
enter_log_label = Label( all_alerts_frame, text='Enter Log' )
enter_log_label.grid( row=1, column=0, padx=3, pady=5 )
```

```
enter_log_box = Entry( all_alerts_frame, width=50 )
enter_log_box.grid( row=1, column=1, padx=3, pady=5 )

log_alert_button = Button( all_alerts_frame, text='Log Alert', command=log_entry )
log_alert_button.grid( row=2, column=1, padx=5, pady=5, ipadx=8 )

delete_btn = Button( all_alerts_frame, text='Delete Alert', command=delete_alert )
delete_btn.grid( row=3, column=1, padx=5, pady=5, ipadx=8 )

info_label = Label( all_alerts_window, text='Below is a list of all unaddressed alerts.\n'
                                'Enter the ID of the alert you would like to alter into \n'
                                'the Enter ID field. Then, click the "Log Alert " button \n'
                                'to clear/send alert to log book, or click the "Delete Alert"  \n'
                                'button to completely remove the alert from the system.\n'
                                'Click the Return button to go back to the menu.',

                width=46, )

info_label.grid( row=4, column=0, padx=5, pady=5, ipadx=10 )

# Gets the requested values of the height and width
windowWidth_1 = all_alerts_window.winfo_reqwidth()
windowHeight_1 = all_alerts_window.winfo_reqheight()

# Gets both half the screen width/height and window width/height
positionRight = int( all_alerts_window.winfo_screenwidth() / 2 - windowWidth_1 / 2 )
positionDown = int( all_alerts_window.winfo_screenheight() / 2 - windowHeight_1 / 2 )

# Positions the window in the center of the page.
all_alerts_window.geometry( "+{}+{}".format( positionRight, positionDown ) )

# displaying the database info
def show():
   for item in listBox.get_children():
      listBox.delete( item )
   tempList = []
   conn = sqlite3.connect( "all_alerts.db" )
   c = conn.cursor()
   c.execute( "SELECT *, oid FROM alerts_table" )
   records = c.fetchall()
   print( records )
   for i in records:
      tempList.append( i )
   print( tempList )
   #    for i, (name, score) in enumerate(tempList, start=1):
   #       listBox.insert("", "end", values=(i, name, score))
   for i, (alert_type, zone, info, record_id) in enumerate( tempList, start=1 ):
      listBox.insert( "", "end", values=(alert_type, zone, info, record_id) )
```

```
        # commit changes
        conn.commit()

        # close connection
        conn.close()

    alerts_view = Canvas( all_alerts_window )
    alerts_view.grid()

    label = Label( alerts_view, bg='red', text="Alerts", font=("Arial", 30) ).grid( row=0, columnspan=3 )
    # create Treeview with 4 columns
    cols = ('Alert Type', 'Zone', 'Info', 'ID')
    listBox = ttk.Treeview( alerts_view, columns=cols, show='headings' )
    # set column headings
    for col in cols:
        listBox.heading( col, text=col )
    listBox.grid( row=1, column=0, columnspan=2 )

    def return_menu():
        main.deiconify()
        all_alerts_window.destroy()

    btn2 = Button( alerts_view, width=15, text="Return", command=(return_menu) )
    btn2.grid( row=4, column=1 )

    showScores = Button( alerts_view, text="Update Alerts", width=15, command=show ).grid( row=4,
column=0 )
    closeButton = Button( alerts_view, text="Return", width=15, command=return_menu ).grid( row=4,
column=1 )

    # commit changes
    conn.commit()

    # close connection
    conn.close()

    def delete_window():
        response = tmb.askokcancel( 'Warning',
                                    'Are you sure you want to exit? You will be '
                                    'redirected to the main alerts menu.', parent=all_alerts_window )
        if response == 1:
            main.deiconify()
            all_alerts_window.destroy()

    all_alerts_window.protocol( "WM_DELETE_WINDOW", delete_window )

    all_alerts_window.attributes( '-topmost', 'true' )
```

```
    main.withdraw()

def all_logs():  # all alerts button function; displays all logs (past cleared alerts)

    # displaying the database info
    def show_logs():
        for item in listBox.get_children():
            listBox.delete( item )
        tempList = []
        conn = sqlite3.connect( "all_alerts.db" )
        c = conn.cursor()
        c.execute( "SELECT *, oid FROM logs" )
        records = c.fetchall()
        print( records )
        for i in records:
            tempList.append( i )
        print( tempList )
        #    for i, (name, score) in enumerate(tempList, start=1):
        #        listBox.insert("", "end", values=(i, name, score))
        for i, (alert_type, zone, info, record_id, log_entry) in enumerate( tempList, start=1 ):
            listBox.insert( "", "end", values=(alert_type, zone, info, record_id, log_entry) )

        # commit changes
        conn.commit()

        # close connection
        conn.close()

    all_logs_window = Toplevel( bg='#9DD9f9' )
    all_logs_window.title( 'All Logs     -     Hyperion Intelligent Lighting System' )
    #all_logs_window.iconbitmap( 'nasa_PLL_icon.ico' )
    all_logs_window.geometry( "1000x400" )

    all_logs_frame = Frame( all_logs_window, bg='#9DD9f9' )
    all_logs_frame.grid( row=0, column=0, padx=5, pady=5, ipadx=10 )

    label = Label( all_logs_frame, text="Alert Logs", font=("Arial", 30) ).grid( row=0, columnspan=3 )
    # create Treeview with 4 columns
    cols = ('Alert Type', 'Zone', 'Info', 'ID', 'Log')
    listBox = ttk.Treeview( all_logs_frame, columns=cols, show='headings' )
    # set column headings
    for col in cols:
        listBox.heading( col, text=col )
    listBox.grid( row=1, column=0, columnspan=2 )

    def return_menu():
        main.deiconify()
```

```
        all_logs_window.destroy()

    showScores = Button( all_logs_frame, text="Update Logs", width=15, command=show_logs ).grid(
row=4,
                                                                column=0 )
    closeButton = Button( all_logs_frame, text="Return", width=15, command=return_menu ).grid(
row=4, column=1 )

    def delete_window():
        main.deiconify()
        all_logs_window.destroy()

    all_logs_window.protocol( "WM_DELETE_WINDOW", delete_window )

    all_logs_window.attributes( '-topmost', 'true' )

    # commit changes
    conn.commit()

    # close connection

    main.withdraw()

  def enter_log():  # this will be the command that allows for error clearing/log entering
    print( 'Hello World' )

  main = Tk()  # creating the main page
  main.title( 'Alerts    -    Hyperion Intelligent Lighting System - ' )
  #main.iconbitmap( 'nasa_PLL_icon.ico' )  # .iconbitmap creates window icon

  canvas = Canvas( main, height=700, width=700, bg='#9DD9F9' )  # creates canvas that GUI widgets will
sit on
  canvas.grid()

  horizontal_border = Label( canvas, height=1, width=72, bg='#9DD9F9' )  # using labels to organize
widgets on page
  horizontal_border.grid( row=2, column=0 )

  warnings_list_label = Label( canvas, text='System Alerts Log', height=2, width=37, bg='#FFA3A3',
relief='solid' )
  warnings_list_label.grid( row=3, column=0 )

  latest_alert = 'This is a placeholder!\nMost recent alert type and other info should go here.\nClick
here to log.'

  new_alert = 1

  if new_alert == True:
```

```
      recent_alert = Button( canvas, text=latest_alert, height=10, width=43, bg='#FE9A2E',
command=enter_log )
      recent_alert.grid( row=4, column=0 )
   else:
      recent_alert = Button( canvas, text='No alerts to show.', height=10, width=43, bg='#01DF01' )
      recent_alert.grid( row=4, column=0 )

   thick_horiz = Label( canvas, height=1, width=69, bg='#9DD9F9' )
   thick_horiz.grid( row=6, column=0 )

   settings_frame = Frame( canvas, height=5, width=70, bg='#9DD9F9' )
   settings_frame.grid( row=7, column=0 )

   home_button = Button( settings_frame, text='Home', height=3, width=15, bg='#3CA3DE',
cursor='hand2',
                         command=(home) )
   home_button.grid( row=0, column=0, padx=4 )

   all_alerts_button = Button( settings_frame, text='All Alerts', height=3, width=15, bg='#3CA3DE',
cursor='hand2',
                            command=(all_alerts) )
   all_alerts_button.grid( row=0, column=1, padx=4 )

   all_logs_button = Button( settings_frame, text='All Logs', height=3, width=15, bg='#3CA3DE',
cursor='hand2',
                           command=(all_logs) )
   all_logs_button.grid( row=0, column=2, padx=4 )

   main.mainloop()
```

###End Alerts Portion of GUI

```
# define crew profile selection screen
def crew_screen():
   # define the home command as going back to home and destroying the profile screen
   # def home():
   # close out all windows except root
   #   crew_screen.destroy()

   conn = sqlite3.connect('crew_profile_db.db')
   c = conn.cursor()

   crew_screen = Toplevel(root)
   crew_screen.configure(background='#9DD9F9')
   """crew_screen.iconbitmap('NASA.ico')"""

   # make root (Home) hidden
   def return_to_root_menu():
      root.deiconify()
      crew_screen.destroy()

   btn2 = Button(crew_screen, text="Return", command=(return_to_root_menu), bg='#966fd6')
   btn2.grid(row=3, column=0, padx=3, pady=5, ipadx=4)

   def delete_window():
      response = tmb.askokcancel('Exit?',
                        'You will be redirected to the main menu.')
      if response == 1:
         root.deiconify()
         crew_screen.destroy()

   crew_screen.protocol("WM_DELETE_WINDOW", delete_window)

   crew_screen.attributes('-topmost', 'true')

   crew_screen.focus_force()

   root.withdraw()

   crew_profile_list = []
   column_0 = [column[0] for column in c.execute('SELECT*, oid FROM crew_profile_db')]
   for profile in column_0:
      crew_profile_list.append(profile)

   crew_screen.title(f'{crew_select.get()}''    -    Hyperion Intelligent Lighting System')

   crew_listbox_label = Label(crew_screen, text=' ', width=20, bg='#9DD9F9', font=16)
   crew_listbox_label.grid(row=0, column=0)

   crew_frame = Frame(crew_screen, height=700, width=700, bg='#9DD9F9')
```

```
    crew_frame.grid(row=1, column=0, padx=5)

    crew_scrollbar = Scrollbar(crew_frame)
    crew_scrollbar.pack(side='right')

    crew_listbox = Listbox(crew_frame, height=15, yscrollcommand=crew_scrollbar.set,
highlightcolor='#b19cd9',
                           highlightthickness=2, selectmode='multiple')
    crew_listbox.insert(END, *crew_profile_list)
    crew_listbox.pack(side='left')

    crew_scrollbar.config(command=crew_listbox.yview)

    modify_frame = Frame(crew_screen, bg='#9DD9F9')
    modify_frame.grid(row=1, column=1)

    crew_profile_modify = Button(modify_frame, text='Modify Profiles', height=2, width=12,
bg='#3CA3DE',
                                 activebackground='#9DD9F9', command=modify_crew)
    crew_profile_modify.grid(row=0, column=1, padx=5)

    # system_modify=Button(modify_frame,text='Modify System',height=2,width=12,bg='#b19cd9',
    #                      activebackground='#b19cd9',fg='white',command=modify_system)
    # system_modify.grid(row=1, column=1, padx=5, pady=5)
    # crew_sensor_modify=Button(modify_frame,text='Modify Sensors',height=2,width=12,bg='#b19cd9',
    #                       activebackground='#b19cd9',fg='white',command=modify_crew_sensors)
    # crew_sensor_modify.grid(row=2,column=1,padx=5,pady=5)

    #
home_button2=Button(crew_frame,text='Home',bg='#3CA3DE',activebackground='#9DD9F9',command
=home,height=1,width=5)
    # home_button2.grid(row=3, column=0, pady=5)

    conn.commit()
    conn.close()


# define plant profile selection screen
def plant_screen():
    # define the home command as going back to home and destroying the profile screen
    # def home():
    #   plant_screen.destroy()

    conn2 = sqlite3.connect('plant_profile_db.db')
    c2 = conn2.cursor()

    plant_screen = Toplevel(root)
    plant_screen.configure(background='#9DD9F9')
```

```
"""plant_screen.iconbitmap('NASA.ico')"""

# make root (Home) hidden
def return_to_root_menu():
   root.deiconify()
   plant_screen.destroy()

btn2 = Button(plant_screen, text="Return", command=(return_to_root_menu), bg='#966fd6')
btn2.grid(row=3, column=0, padx=3, pady=5, ipadx=4)

def delete_window():
   response = tmb.askokcancel('Exit?',
                      'You will be redirected to the main menu.')
   if response == 1:
      root.deiconify()
      plant_screen.destroy()

plant_screen.protocol("WM_DELETE_WINDOW", delete_window)
plant_screen.attributes('-topmost', 'true')
plant_screen.focus_force()

root.withdraw()

plant_profile_list = []
column_0 = [column[0] for column in c2.execute('SELECT*, oid FROM plant_profile_db')]
for profile in column_0:
   plant_profile_list.append(profile)

plant_screen.title(f'{plant_select.get()}''    -    Hyperion Intelligent Lighting System')

plant_listbox_label = Label(plant_screen, text=' ', width=20, bg='#9DD9F9', font=16)
plant_listbox_label.grid(row=0, column=0)

plant_frame = Frame(plant_screen, height=700, width=700, bg='#9DD9F9')
plant_frame = Frame(plant_screen)
plant_frame.grid(row=1, column=0, padx=5)

plant_scrollbar = Scrollbar(plant_frame)
plant_scrollbar.pack(side='right')

plant_listbox = Listbox(plant_frame, height=15, yscrollcommand=plant_scrollbar.set,
                  highlightcolor='#b19cd9', highlightthickness=2, selectmode='multiple')
plant_listbox.insert(END, *plant_profile_list)
plant_listbox.pack(side='left')

plant_scrollbar.config(command=plant_listbox.yview)

modify_frame = Frame(plant_screen, bg='#9DD9F9')
```

```
    modify_frame.grid(row=1, column=1)

    plant_profile_modify = Button(modify_frame, text='Modify Profiles', height=2, width=12,
bg='#3CA3DE',
                                  activebackground='#9DD9F9', command=modify_plant)
    plant_profile_modify.grid(row=0, column=1, padx=5)

    # system_modify=Button(modify_frame,text='Modify System',height=2,width=12,bg='#b19cd9',
    #                  activebackground='#b19cd9',fg='white',command=modify_system)
    # system_modify.grid(row=1,column=1,padx=5,pady=5)
    # sensor_modify=Button(modify_frame,text='Modify Sensors',height=2,width=12,bg='#b19cd9',
    #                   activebackground='#b19cd9',fg='white',command=modify_plant_sensors)
    # sensor_modify.grid(row=2,column=1,padx=5,pady=5)

    conn2.commit()
    conn2.close()

def modify_crew():
    modify_crew = Toplevel(root)
    modify_crew.configure(background='#9DD9F9')
    modify_crew.title('Modify Crew Profiles     -     Hyperion Intelligent Lighting System')
    """modify_crew.iconbitmap('NASA.ico')"""

    # make root (Home) hidden
    def return_to_root_menu():
        root.deiconify()
        modify_crew.destroy()

    btn2 = Button(modify_crew, text="Return", command=(return_to_root_menu), bg='#966fd6')
    btn2.grid(row=10, column=2, padx=3, pady=3, ipadx=4)

    def delete_window():
        response = tmb.askokcancel('Exit?',
                                   'You will be redirected to the main menu.')
        if response == 1:
            root.deiconify()
            modify_crew.destroy()

    modify_crew.protocol("WM_DELETE_WINDOW", delete_window)
    modify_crew.attributes('-topmost', 'true')
    modify_crew.focus_force()

    root.withdraw()

    # create delete function to delete information in the database
    def delete():
        conn = sqlite3.connect('crew_profile_db.db')
        c = conn.cursor()
```

```
    c.execute('DELETE from crew_profile_db WHERE oid = ' + delete_record.get())

    conn.commit()
    conn.close()

    # clear the entry box
    delete_record.delete(0, END)

  # create submit function for entering the information into the database
  def submit():
    conn = sqlite3.connect('crew_profile_db.db')
    c = conn.cursor()

    # insert entry values into database
    c.execute('INSERT INTO crew_profile_db VALUES (:profile_name,:time_offset)',
          {
            'profile_name': profile_name.get(),
            'time_offset': time_offset.get()
          })
    conn.commit()
    conn.close()

    # clear the entry boxes
    profile_name.delete(0, END)
    time_offset.delete(0, END)

  # creating title label for adding profiles
  add_profiles_label = Label(modify_crew, bg='#9DD9F9', relief='ridge', bd=3,
                     text='ADD CREW PROFILE(S)-ENTER YOUR NAME/ID AND OFFSET FROM UTC TIME')
  add_profiles_label.grid(row=0, column=0, columnspan=5, padx=15, pady=15)

  # creating entry boxes for adding profiles
  profile_name = Entry(modify_crew, width=30)
  profile_name.grid(row=1, column=1, pady=10)
  time_offset = Entry(modify_crew, width=30)
  time_offset.grid(row=2, column=1, pady=10)

  # creating labels for entry boxes above
  profile_name_label = Label(modify_crew, text='Profile Name/ID:', bg='#9DD9F9')
  profile_name_label.grid(row=1, column=0, pady=10)
  time_offset_label = Label(modify_crew, text='Offset in Hours:', bg='#9DD9F9')
  time_offset_label.grid(row=2, column=0, pady=10)

  # creating cutton to add profiles to the database
  # entry boxes will clear for further use
  add_to_db_btn = Button(modify_crew, text='Add information to database', command=submit,
padx=20,
```

```
                bg='#3CA3DE', activebackground='#9DD9F9')
add_to_db_btn.grid(row=3, column=1, pady=10)

# creating title label for deleting profiles
delete_profiles_label = Label(modify_crew, bg='#9DD9F9', relief='ridge', bd=3,
                      text='DELETE CREW PROFILE(S)-ENTER YOUR PROFILE # FROM RECORDS')
delete_profiles_label.grid(row=4, column=0, columnspan=5, padx=15, pady=(20, 15))

# creating entry box for deleting profiles
delete_record = Entry(modify_crew, width=30)
delete_record.grid(row=5, column=1, pady=10)

# creating label for entry box above
delete_record_label = Label(modify_crew, text='Profile # to Delete:', bg='#9DD9F9')
delete_record_label.grid(row=5, column=0, pady=10)

def show():
   conn = sqlite3.connect('crew_profile_db.db')
   c = conn.cursor()

   # creating a list consisting of the rows in the crew database
   # this list will be shown when the 'Show Records' button is clicked on the modify_crew screen
   profile_list = []
   profile_rows = [row[:] for row in c.execute('SELECT*, oid FROM crew_profile_db')]
   for row in profile_rows:
      profile_list.append(row)

   show_frame = Frame(modify_crew, height=4, bg='#9DD9F9')
   show_frame.grid(row=10, column=0)

   crew_scrollbar = Scrollbar(show_frame)
   crew_scrollbar.pack(side='right')

   crew_listbox = Listbox(show_frame, height=5, highlightcolor='#b19cd9',
                    highlightthickness=2, yscrollcommand=crew_scrollbar.set)
   crew_listbox.insert(END, *profile_list)
   crew_listbox.pack(side='left')

   crew_scrollbar.config(command=crew_listbox.yview)

   conn.commit()
   conn.close()

show_button = Button(modify_crew, text='Show Records', bg='black', fg='white', command=show)
show_button.grid(row=6, column=0, pady=10)

delete_button = Button(modify_crew, text='Delete Record', bg='#3CA3DE',
                activebackground='#9DD9F9', command=delete)
```

```
    delete_button.grid(row=6, column=1, pady=10)


def modify_plant():
    modify_plant = Toplevel(root)
    modify_plant.configure(background='#9DD9F9')
    modify_plant.title('Modify Plant Profiles     -     Hyperion Intelligent Lighting System')
    """modify_plant.iconbitmap('NASA.ico')"""

    # make root (Home) hidden
    def return_to_root_menu():
        root.deiconify()
        modify_plant.destroy()

    btn2 = Button(modify_plant, text="Return", command=(return_to_root_menu), bg='#966fd6')
    btn2.grid(row=10, column=2, padx=3, pady=3, ipadx=4)

    def delete_window():
        response = tmb.askokcancel('Exit?',
                                   'You will be redirected to the main menu.')
        if response == 1:
            root.deiconify()
            modify_plant.destroy()

    modify_plant.protocol("WM_DELETE_WINDOW", delete_window)

    modify_plant.attributes('-topmost', 'true')

    modify_plant.focus_force()

    root.withdraw()

    # create delete function to delete information in the database
    def delete():
        conn2 = sqlite3.connect('plant_profile_db.db')
        c2 = conn2.cursor()

        c2.execute('DELETE from plant_profile_db WHERE oid = ' + delete_record.get())

        conn2.commit()
        conn2.close()

        # clear the entry box
        delete_record.delete(0, END)

    # create submit function for entering the information into the database
    def submit():
        conn2 = sqlite3.connect('plant_profile_db.db')
```

```
    c2 = conn2.cursor()

    # insert entry values into database
    c2.execute('INSERT INTO plant_profile_db VALUES (:profile_name,:photoperiod)',
            {
               'profile_name': profile_name.get(),
               'photoperiod': photoperiod.get()
            })
    conn2.commit()
    conn2.close()

    # clear the entry boxes
    profile_name.delete(0, END)
    photoperiod.delete(0, END)

  # creating title label for adding profiles
  add_profiles_label = Label(modify_plant, bg='#9DD9F9', relief='ridge', bd=3,
                   text='ADD PLANT PROFILE(S)-ENTER THE PLANT/VEGETATION NAME AND ITS
PHOTOPERIOD')
  add_profiles_label.grid(row=0, column=0, columnspan=5, padx=15, pady=15)

  # creating entry boxes for adding profiles
  profile_name = Entry(modify_plant, width=30)
  profile_name.grid(row=1, column=1, pady=10)
  photoperiod = Entry(modify_plant, width=30)
  photoperiod.grid(row=2, column=1, pady=10)

  # creating labels for entry boxes above
  profile_name_label = Label(modify_plant, text='Profile Name:', bg='#9DD9F9')
  profile_name_label.grid(row=1, column=0, pady=10)
  photoperiod_label = Label(modify_plant, text='Photoperiod in Hours:', bg='#9DD9F9')
  photoperiod_label.grid(row=2, column=0, pady=10)

  # creating cutton to add profiles to the database
  # entry boxes will clear for further use
  add_to_db_btn = Button(modify_plant, text='Add information to database', command=submit,
                  padx=20, bg='#3CA3DE', activebackground='#9DD9F9')
  add_to_db_btn.grid(row=3, column=1, pady=10)

  # creating title label for deleting profiles
  delete_profiles_label = Label(modify_plant, bg='#9DD9F9', relief='ridge', bd=3,
                      text='DELETE PLANT PROFILE(S)-ENTER YOUR PROFILE # FROM RECORDS')
  delete_profiles_label.grid(row=4, column=0, columnspan=5, padx=15, pady=(20, 15))

  # creating entry box for deleting profiles
  delete_record = Entry(modify_plant, width=30)
  delete_record.grid(row=5, column=1, pady=10)
```

```
    # creating label for entry box above
    delete_record_label = Label(modify_plant, text='Profile # to Delete:', bg='#9DD9F9')
    delete_record_label.grid(row=5, column=0, pady=10)

    def show():
        conn2 = sqlite3.connect('plant_profile_db.db')
        c2 = conn2.cursor()

        # creating a list consisting of the rows in the plant database
        # this list will be shown when the 'Show Records' button is clicked on the modify_plant screen
        profile_list = []
        profile_rows = [row[:] for row in c2.execute('SELECT*, oid FROM plant_profile_db')]
        for row in profile_rows:
            profile_list.append(row)

        show_frame = Frame(modify_plant, height=4, bg='#9DD9F9')
        show_frame.grid(row=10, column=0)

        plant_scrollbar = Scrollbar(show_frame)
        plant_scrollbar.pack(side='right')

        plant_listbox = Listbox(show_frame, height=5, highlightcolor='#b19cd9',
                                highlightthickness=2, yscrollcommand=plant_scrollbar.set)
        plant_listbox.insert(END, *profile_list)
        plant_listbox.pack(side='left')

        plant_scrollbar.config(command=plant_listbox.yview)

        conn2.commit()
        conn2.close()

    show_button = Button(modify_plant, text='Show Records', bg='black', fg='white', command=show)
    show_button.grid(row=6, column=0, pady=10)

    delete_button = Button(modify_plant, text='Delete Record', bg='#3CA3DE',
                           activebackground='#9DD9F9', command=delete)
    delete_button.grid(row=6, column=1, pady=10)

# define selection of crew zones
# set default to text SELECT YOUR ZONE
crew_select = StringVar()
crew_select.set('SELECT YOUR CREW ZONE')

# define selection of plant zones
plant_select = StringVar()
plant_select.set('SELECT YOUR PLANT ZONE')

# create drop-downs - one with crew zones, one with plant
```

```
###root.bind('<Return>', crew_screen)
crew_menu = OptionMenu(root, crew_select, *crew_zones_list)
crew_menu.config(bg='#966fd6', activebackground='#b19cd9', font=1, anchor='w', width=40, height=1)
crew_menu.grid(row=1, column=0, padx=(18, 0), pady=15)
enter_button1 = Button(root, text='Enter', command=crew_screen, bg='#966fd6',
 activebackground='#b19cd9', width=5)
enter_button1.bind('<Button-1>', crew_screen)
enter_button1.grid(row=1, column=1, padx=7)

plant_menu = OptionMenu(root, plant_select, *plant_zones_list)
plant_menu.config(bg='#966fd6', activebackground='#b19cd9', font=1, anchor='w', width=40, height=1)
plant_menu.grid(row=2, column=0, padx=(18, 0), pady=7)
enter_button1 = Button(root, text='Enter', command=plant_screen, bg='#966fd6',
 activebackground='#b19cd9', width=5)
###enter_button1.bind('<Button-1>', plant_screen)
enter_button1.grid(row=2, column=1, padx=15)

## frame for alters/warnings for system
frame_alerts1 = LabelFrame(root, text='SYSTEM ALERTS/WARNINGS', padx=130, pady=20,
 bg='#F99D9D')
frame_alerts1.grid(row=3, column=0, columnspan=2, pady=15)
# you can add a command when created
alerts_appear1 = Button(frame_alerts1, text='Alerts Menu', bg='#F99D9D', relief='sunken',
 command=alerts_button)
alerts_appear1.grid(row=4, column=0)
# commit changes to both databases
conn.commit()
conn2.commit()
# close connection to both databases
conn.close()
conn2.close()


root.mainloop()
```