```
#! /usr/bin/env python
# Program that is called from UI.py to show the user real-time sensor data
# Group 10: Automatic Brewer
# Group Members: Robert Bower (EE), Alonzo Ubilla (ME/EE), Kleber Valencia (EE), David Rodriguez (CE)

#imports
from Tkinter import*
import Tkinter
import tkMessageBox
import tkFont
import sys
import smbus
import time
import subprocess
import MySQLdb
import sqlite3
import smtplib
from PIL import Image, ImageTk
from email.mime.text import MIMEText

root = Tkinter.Tk()
root.attributes('-fullscreen', True)
root.config(bg='black')
root.title("Current System Sensor Data Readings")
name_label = Tkinter.Label(root, text='BREW STATS', font='Arial 40 bold', fg='gold', bg='black')
name_label.pack(side=TOP, pady=5)
canvas = Tkinter.Canvas(root, bg='snow')
canvas.pack(fill=BOTH, expand=YES)

# initiate the bus and the chip addresses
bus = smbus.SMBus(1)
chip1_addr = 0x10
chip2_addr = 0x20

# declare the global variable needed for our generated clock function
timevar = float
```

```
# Function to draw the main rectangles where the data will go in
def create_boxes():
    canvas.create_rectangle(75, 50, 526, 400, fill='dark grey')
    canvas.create_rectangle(585, 50, 1036, 400, fill='dark grey')
    canvas.create_rectangle(1096, 50, 1548, 400, fill='dark grey')

# Create the boil kettle GUI elements
def boil_kettle_gui():
    kettle_label = Tkinter.Label(canvas, text='BOIL KETTLE', font='Verdana 30 bold', fg='gold', bg='black', width=16)
    kettle_label.place(x=75, y=50)
    kettle_temp = Tkinter.Label(canvas, text='TEMP(C)', font='Verdana 28 bold underline', bg='dark grey')
    kettle_temp.place(x=220, y=130)
    kettle_level = Tkinter.Label(canvas, text='LEVEL(%)', font='Verdana 28 bold underline', bg='dark grey')
    kettle_level.place(x=205, y=240)

# Create the mash tun GUI elements
def mash_tun_gui():
    mash_label = Tkinter.Label(canvas, text='MASH TUN', font='Verdana 30 bold', fg='gold', bg='black', width=16)
    mash_label.place(x=584, y=50)
    mash_temp = Tkinter.Label(canvas, text='TEMP(C)', font='Verdana 28 bold underline', bg='dark grey')
    mash_temp.place(x=725, y=130)

# Create the cooling tank GUI elements
def cooling_tank_gui():
    cooling_label = Tkinter.Label(canvas, text='COOLING RETURN', font='Verdana 30 bold', fg='gold',  bg='black', width=16)
    cooling_label.place(x=1096, y=50)
    cooling_temp = Tkinter.Label(canvas, text='TEMP(C)', font='Verdana 28 bold underline', bg='dark grey')
    cooling_temp.place(x=1250, y=130)

# Create the label to display how much overall time there's left for end of system cycle
def currentStep():
    step_label = Tkinter.Label(canvas, text='STEP:', font='Verdana 25 underline bold', bg='white')
    step_label.place(x=350, y=420)

# Create the main exit button that will be displayed in the bottom of window
def exit_btn():
    exit_btn = Tkinter.Button(root, width=10, text="EXIT", font='Verdana 14 bold', bg='gold', command=client_exit)
```

```
    exit_btn.pack(side=BOTTOM, pady=5)


def reset_btn():
    # create the load reset button to zero out the load cells prior to system start
    reset_btn = Tkinter.Button(root, width=10, text="TARE", font='Verdana 14 bold', bg='gold',command=tareBtn)
    reset_btn.place(x=225, y=440)

def tareBtn():
    tkMessageBox.askquestion("CAUTION!", "Are you sure you want to tare?", icon='warning')
    bus.write_byte(chip2_addr,0xB3)
    bus.write_byte(chip2_addr,0x00)


# Event call function from exit button when pressed
def client_exit():
    answer = tkMessageBox.askquestion("CAUTION!", "Are you sure you want to EXIT?", icon='warning')
    if answer == 'yes':
        root.destroy()
        bus.write_byte(chip1_addr, 0x00)
        bus.write_byte(chip1_addr, 0x00)
        bus.write_byte(chip2_addr, 0x00)
        bus.write_byte(chip2_addr, 0x00)
    else:
        return

# Function to draw out all the valve labels
def system_valves():
    valve1 = Tkinter.Label(canvas, text='PUMP INLET VALVE', font='Verdana 25 ', bg='white')
    valve1.place(x=400, y=520)
    valve2 = Tkinter.Label(canvas, text='MASH INLET VALVE', font='Verdana 25 ', bg='white')
    valve2.place(x=400, y=570)
    valve3 = Tkinter.Label(canvas, text='SPARGE INLET VALVE', font='Verdana 25 ', bg='white')
    valve3.place(x=400, y=620)
    valve4 = Tkinter.Label(canvas, text='SPARGE DRAIN VALVE', font='Verdana 25 ', bg='white')
    valve4.place(x=400, y=670)
    valve5 = Tkinter.Label(canvas, text='MASH DRAIN VALVE', font='Verdana 25 ', bg='white')
```

```
    valve5.place(x=400, y=720)
    valve6 = Tkinter.Label(canvas, text='COOLING INLET VALVE', font='Verdana 25 ', bg='white')
    valve6.place(x=900, y=520)
    valve7 = Tkinter.Label(canvas, text='CARBOY FILL VALVE', font='Verdana 25 ', bg='white')
    valve7.place(x=900, y=570)
    wort_pump = Tkinter.Label(canvas, text='TRANSFER PUMP', font='Verdana 25 ', bg='white')
    wort_pump.place(x=900, y=620)
    cool_pump = Tkinter.Label(canvas, text='COOL PUMP', font='Verdana 25 ', bg='white')
    cool_pump.place(x=900, y=670)
    heating_element = Tkinter.Label(canvas, text='HEATING ELEMENT', font='Verdana 25 ', bg='white')
    heating_element.place(x=900, y=720)

def timer():
    global minute_counter
    seconds = time.strftime('%S')
    if (seconds >= '59'):
        minute_counter = minute_counter + 1
        time.sleep(1)

def step0():
    global sys_kettle_level
    global stepInfo
    stepInfo = "0"
    step_label.config(text="Fill Kettle")
    read_variables()
    while (sys_kettle_level < 80):
        read_variables()
        sendBluetooth()
    step1()

# step1: initiate the process
def step1():
    global sys_kettle_temp
    global user_mash_temp
    global stepInfo
    stepInfo = "1"
    step_label.config(text="HEAT MASH WATER")
```

```
    root.update()
    read_variables()
    # ***Heating element will be turned on by chip1 after the start command is sent***
    # call start step 1 command using 0xC0 for chip1
    bus.write_byte(chip1_addr, 0xC0)
    bus.write_byte(chip1_addr, 0x00)
    # call start step 1 command using 0xC0 for chip2 (idle in this step)
    bus.write_byte(chip2_addr, 0xC0)
    bus.write_byte(chip2_addr, 0x00)
    while (sys_kettle_temp < user_mash_temp):
        read_variables()
        sendBluetooth()
    system_notification("Mash water heated!")
    step2()


# step2: fill the mash tun
def step2():
    global sys_kettle_level
    global stepInfo
    stepInfo = "2"
    step_label.config(text="FILL MASH TUN")
    root.update()
    read_variables()
    # call start step 2 command using 0xC1 for chip2
    bus.write_byte(chip2_addr, 0xC1)
    bus.write_byte(chip2_addr, 0x00)
    time.sleep(5)
    # call start step 2 command using 0xC1 for chip1
    bus.write_byte(chip1_addr, 0xC1)
    bus.write_byte(chip1_addr, 0x00)
    while (sys_kettle_level > 40):
        read_variables()
        sendBluetooth()
    system_notification("Mash tun filled!")
    step3()
```

```
# step3: transfer of liquid to sparge tank
def step3():
    global sys_kettle_level
    global stepInfo
    stepInfo = "3"
    step_label.config(text="FILL SPARGE TANK")
    root.update()
    read_variables()
    # call start step 3 command using 0xC2 for chip1
    bus.write_byte(chip1_addr, 0xC2)
    bus.write_byte(chip1_addr, 0x00)
    # call start step 3 command using 0xC2 for chip2
    bus.write_byte(chip2_addr, 0xC2)
    bus.write_byte(chip2_addr, 0x00)
    while sys_kettle_level > 20:
        read_variables()
        sendBluetooth()
    system_notification("Sparge tank filled!")
    step4()


# step4: soaking of the mash
def step4():
    global user_mash_time
    global minute_counter
    global stepInfo
    stepInfo = "4"
    step_label.config(text="MASH")
    root.update()
    read_variables()
    # call start step 4 command using 0xC3 for chip2
    bus.write_byte(chip1_addr, 0xC3)
    bus.write_byte(chip1_addr, 0x00)
    # call start step 4 command using 0xC3 for chip2
    bus.write_byte(chip2_addr, 0xC3)
    bus.write_byte(chip2_addr, 0x00)
```

```
    #timevar = time.time()
    while (minute_counter < user_mash_time):
        #get_time()
        timer()
        read_variables()
        sendBluetooth()
    system_notification("Mash completed!")
    minute_counter = 0
    step5()


# step5: drain the mash tun
def step5():
    global sys_kettle_level
    global stepInfo
    stepInfo = "5"
    step_label.config(text="DRAIN MASH")
    root.update()
    read_variables()
    # call start step 5 command using 0xC4 for chip1
    bus.write_byte(chip1_addr, 0xC4)
    bus.write_byte(chip1_addr, 0x00)
    # call start step 5 command using 0xC4 for chip2
    bus.write_byte(chip2_addr, 0xC4)
    bus.write_byte(chip2_addr, 0x00)
    while sys_kettle_level < 70:
        read_variables()
        sendBluetooth()
    system_notification("Mash drained!")
    step6()


# step6: boil of mash and add the hops into the mash
def step6():
    global user_boil_time
    global sys_kettle_temp
    global minute_counter
```

```
global user_boil_temp
global stepInfo
stepInfo = "6"
step_label.config(text="BOIL WORT")
root.update()
read_variables()
# use variable k to make sure we only have the hop addition command sent once
# k=1 when the hop addition is complete
k = 0
# call start step 6 command using 0xC5 for chip1
bus.write_byte(chip1_addr, 0xC5)
bus.write_byte(chip1_addr, 0x00)
# call start step 6 command using 0xC5 for chip2 (idle in this step)
bus.write_byte(chip2_addr, 0xC5)
bus.write_byte(chip2_addr, 0x00)
# sample current time
#timevar = time.time()
high_limit = user_boil_temp +1
low_limit = user_boil_temp - 1
while (minute_counter < user_boil_time):
   read_variables()
   #Update timer
   timer()
   if (sys_kettle_temp >= high_limit):
      bus.write_byte(chip1_addr, 0xB2)
      bus.write_byte(chip1_addr, 0x00)
   if (sys_kettle_temp <= low_limit):
      bus.write_byte(chip1_addr, 0xB3)
      bus.write_byte(chip1_addr, 0x00)
   elap = user_boil_time - minute_counter
   # calculate the time remaining from the end of the boil time to add the hops into the mash
   #hop_time_remaining = user_hop_addition - elap
   # if statement to check if it's time to add the hops into the mash
   if (elap <= user_hop_addition & k == 0 ):
      # send command 0xD0 to trigger hop actuator
      bus.write_byte(chip1_addr, 0xD0)
      bus.write_byte(chip1_addr, 0x00)
```

```
            k = 1
        sendBluetooth()
    system_notification("Boil complete!")
    minute_counter = 0
    step7()


# step7: initiate cool down process
def step7():
    global sys_kettle_temp
    global user_cool_temp
    global stepInfo
    stepInfo = "7"
    step_label.config(text="COOL DOWN")
    root.update()
    read_variables()
    # call start step 8 command using 0xC6 for chip2
    bus.write_byte(chip2_addr, 0xC6)
    bus.write_byte(chip2_addr, 0x00)
    time.sleep(2)
    # call start step 8 command using 0xC6 for chip1
    bus.write_byte(chip1_addr, 0xC6)
    bus.write_byte(chip1_addr, 0x00)
    while sys_kettle_temp > user_cool_temp:
        read_variables()
        sendBluetooth()
    system_notification("Cool down completed!")
    step8()


# step8: last step, drain into carboy
def step8():
    global sys_kettle_level
    global stepInfo
    stepInfo = "8"
    step_label.config(text="CARBOY FILL")
    root.update()
```

```
    root.update()
    read_variables()
    # call start step 7 command using 0xC7 for chip1
    bus.write_byte(chip1_addr, 0xC7)
    bus.write_byte(chip1_addr, 0x00)
    # call start step 7 command using 0xC7 for chip2
    bus.write_byte(chip2_addr, 0xC7)
    bus.write_byte(chip2_addr, 0x00)
    while sys_kettle_level > 20:
        read_variables()
        sendBluetooth()
    # send the end ack 0xC8 to chip1
    bus.write_byte(chip1_addr, 0xC8)
    bus.write_byte(chip1_addr, 0x00)
    # send the end ack 0xC8 to chip2
    bus.write_byte(chip2_addr, 0xC8)
    bus.write_byte(chip2_addr, 0x00)
    #subprocess.call(["obexftp","-b","00:EE:BD:6D:7F:49","-c","ABE","-p","brewprog.txt"])
    system_notification("Cycle Completed!")
    step_label.config(text="FINISHED")
    read_variables()
    # Display a warning message to the user to let them know the process has finished
    tkMessageBox.showwarning("System Warning","Brew has Finished!", icon='warning')


# Function to constantly update the system readings that will be displayed in the GUI
# Depending on the step the system is in we will be reading:
# mash temperature, kettle temperature, kettle level, & cooling temperature
def read_variables():
    global sys_kettle_temp
    global sys_mash_temp
    global sys_cooling_temp
    global sys_kettle_level
    # send 0xB0 to chip1 to request the system kettle temperature
    bus.write_byte(chip1_addr, 0xB0)
    bus.write_byte(chip1_addr, 0x00)
    sys_kettle_temp = bus.read_byte(chip1_addr)
```

```
    # send 0xB1 to chip1 to request the system mash temperature
    bus.write_byte(chip1_addr, 0xB1)
    bus.write_byte(chip1_addr, 0x00)
    sys_mash_temp = bus.read_byte(chip1_addr)
    #send 0xB2 to chip2 to request the system cooler temperature
    bus.write_byte(chip2_addr, 0xB2)
    bus.write_byte(chip2_addr, 0x00)
    sys_cooling_temp = bus.read_byte(chip2_addr)
    # send 0xB0 to chip2 to request the system kettle level
    bus.write_byte(chip2_addr, 0xB0)
    bus.write_byte(chip2_addr, 0x00)
    sys_kettle_level = bus.read_byte(chip2_addr)
    if (sys_kettle_level > 110):
        sys_kettle_level = 0
    # update the labels on the GUI to reflect the received data
    sys_kettle_temp_label.config(text=sys_kettle_temp)
    sys_kettle_level_label.config(text=sys_kettle_level)
    sys_mash_temp_label.config(text=sys_mash_temp)
    sys_cool_temp_label.config(text=sys_cooling_temp)
    # send 0xB7 to refresh analog data each time from chip1
    bus.write_byte(chip1_addr, 0xB7)
    bus.write_byte(chip1_addr, 0x00)
    # send 0xB7 to refresh analog data each time from chip2
    bus.write_byte(chip2_addr, 0xB7)
    bus.write_byte(chip2_addr, 0x00)
    time.sleep(.1)
    root.update()
    read_valve_status()

def sendBluetooth():
    global sys_kettle_temp
    global sys_mash_temp
    global sys_cooling_temp
    global sys_kettle_level
    global sys_valve1
    global sys_valve2
    global sys_valve3
```

```
    global sys_valve4
    global sys_valve5
    global sys_valve6
    global sys_valve7
    global sys_wort_pump
    global sys_cool_pump
    global sys_heating_element
    global stepInfo
    subprocess.call(["obexftp","-b","00:EE:BD:6D:7F:49","-c","ABE","-p","brewStats.txt"])
    file = open("brewStats.txt", "w")
    file.write("%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s" %(sys_kettle_temp, sys_mash_temp, sys_cooling_temp, sys_kettle_level, sys_valve1, sys_valve2,
sys_valve3, sys_valve4,
                                                                sys_valve5, sys_valve6, sys_valve7, sys_wort_pump, sys_cool_pump, sys_heating_element,
stepInfo))
    file.close()

# Function to update the status of the various solenoid valves laid out throughout the system
# (*Dark Green = ON, Red = OFF*)
def read_valve_status():
    global sys_valve1
    global sys_valve2
    global sys_valve3
    global sys_valve4
    global sys_valve5
    global sys_valve6
    global sys_valve7
    global sys_wort_pump
    global sys_cool_pump
    global sys_heating_element
    # Read the status of valve 1 by sending 0xE0 to MCU
    bus.write_byte(chip2_addr, 0xE0)
    bus.write_byte(chip2_addr, 0x00)
    sys_valve1 = bus.read_byte(chip2_addr)
    if sys_valve1 == 1:
        status_valve1.config(bg='dark green')
        root.update()
    else:
```

```
    status_valve1.config(bg='red')
    root.update()
# Read the status of valve 2 by sending 0xE1 to MCU
bus.write_byte(chip2_addr, 0xE1)
bus.write_byte(chip2_addr, 0x00)
sys_valve2 = bus.read_byte(chip2_addr)
# change the status if valve is open, else stays red
if sys_valve2 == 1:
    status_valve2.config(bg='dark green')
    root.update()
else:
    status_valve2.config(bg='red')
    root.update()
# Read the status of valve 3 by sending 0xE2 to MCU
bus.write_byte(chip2_addr, 0xE2)
bus.write_byte(chip2_addr, 0x00)
sys_valve3 = bus.read_byte(chip2_addr)
# change the status if valve is open, else stays red
if sys_valve3 == 1:
    status_valve3.config(bg='dark green')
    root.update()
else:
    status_valve3.config(bg='red')
    root.update()
# Read the status of valve 4 by sendig 0xE3 to MCU
bus.write_byte(chip2_addr, 0xE3)
bus.write_byte(chip2_addr, 0x00)
sys_valve4 = bus.read_byte(chip2_addr)

# change the status if valve is open, else stays red
if sys_valve4 == 1:
    status_valve4.config(bg='dark green')
    root.update()
else:
    status_valve4.config(bg='red')
    root.update()
# Read the status of valve 5 by sending 0xE4 to MCU
```

```
bus.write_byte(chip2_addr, 0xE4)
bus.write_byte(chip2_addr, 0x00)
sys_valve5 = bus.read_byte(chip2_addr)
# change the status if valve is open, else stays red
if sys_valve5 == 1:
   status_valve5.config(bg='dark green')
   root.update()
else:
   status_valve5.config(bg='red')
   root.update()
# Read the status of valve 6 by sending 0xE5 to MCU
bus.write_byte(chip2_addr, 0xE5)
bus.write_byte(chip2_addr, 0x00)
sys_valve6 = bus.read_byte(chip2_addr)
# change the status if valve is open, else stays red
if sys_valve6 == 1:
   status_valve6.config(bg='dark green')
   root.update()
else:
   status_valve6.config(bg='red')
   root.update()
# Read the status of valve 7 by sending 0xE6 to MCU
bus.write_byte(chip2_addr, 0xE6)
bus.write_byte(chip2_addr, 0x00)
sys_valve7 = bus.read_byte(chip2_addr)
# change the status if valve is open, else stays red
if sys_valve7 == 1:
   status_valve7.config(bg='dark green')
   root.update()
else:
   status_valve7.config(bg='red')
   root.update()
# Read the status of wort pump by sending 0xE7 to MCU
bus.write_byte(chip1_addr, 0xE0)
bus.write_byte(chip1_addr, 0x00)
sys_wort_pump = bus.read_byte(chip1_addr)
# change the status if valve is open, else stays red
```

```
    if sys_wort_pump == 1:
        status_wort_pump.config(bg='dark green')
        root.update()
    else:
        status_wort_pump.config(bg='red')
        root.update()
    # Read the status of the cool pump by sending 0xE8 to MCU
    bus.write_byte(chip1_addr, 0xE1)
    bus.write_byte(chip1_addr, 0x00)
    sys_cool_pump = bus.read_byte(chip1_addr)
    # change the status if valve is open, else stays red
    if sys_cool_pump == 1:
        status_cool_pump.config(bg='dark green')
        root.update()
    else:
        status_cool_pump.config(bg='red')
        root.update()
    bus.write_byte(chip1_addr, 0xE2)
    bus.write_byte(chip1_addr, 0x00)
    sys_heating_element = bus.read_byte(chip1_addr)
    if sys_heating_element == 1:
        status_heating_element.config(bg='dark green')
        root.update()
    else:
        status_heating_element.config(bg='red')
        root.update()


# function to send the user(s) a SMS/Email notification each time a step is completed
def system_notification(stepInfo):
    msg = MIMEText('MESSAGE FROM AUTO BREW')
    msg['Subject'] = ('%s' %stepInfo)
    msg['From'] = USERNAME
    msg['To'] = MAILTO
    server = smtplib.SMTP('smtp.gmail.com:587')
    server.ehlo_or_helo_if_needed()
    server.starttls()
```

```
    server.ehlo_or_helo_if_needed()
    server.login(USERNAME, PASSWORD)
    server.sendmail(USERNAME, MAILTO, msg.as_string())
    server.sendmail(USERNAME, MAILTO2, msg.as_string())
    server.sendmail(USERNAME, MAILTO3, msg.as_string())
    server.sendmail(USERNAME, MAILTO4, msg.as_string())
    server.quit()
    return


# the main control loop that will kick off the steps 1-7
def control_loop():
    global user_mash_temp
    global user_boil_temp
    global user_cool_temp
    global user_mash_time
    global user_boil_time
    global user_hop_addition
    # send start command 0xA0
    bus.write_byte(chip1_addr, 0xA0)
    bus.write_byte(chip1_addr, 0x00)
    start_ack = bus.read_byte(chip1_addr)
    # read back the user input mash temp from 1st GUI
    bus.write_byte(chip1_addr, 0xB4)
    bus.write_byte(chip1_addr, 0x00)
    user_mash_temp = bus.read_byte(chip1_addr)
    # read back the user input boil temp from 1st GUI
    bus.write_byte(chip1_addr, 0xB5)
    bus.write_byte(chip1_addr, 0x00)
    user_boil_temp = bus.read_byte(chip1_addr)
    # read back the user input cool temp from 1st GUI
    bus.write_byte(chip1_addr, 0xB6)
    bus.write_byte(chip1_addr, 0x00)
    user_cool_temp = bus.read_byte(chip1_addr)
    # read back the user input mash time from 1st GUI
    bus.write_byte(chip1_addr, 0xA7)
    bus.write_byte(chip1_addr, 0x00)
```

```
    user_mash_time = bus.read_byte(chip1_addr)
    # read back the user input boil time from 1st GUI
    bus.write_byte(chip1_addr, 0xA8)
    bus.write_byte(chip1_addr, 0x00)
    user_boil_time = bus.read_byte(chip1_addr)
    # read back the user input hop addition time from 1st GUI
    bus.write_byte(chip1_addr, 0xA9)
    bus.write_byte(chip1_addr, 0x00)
    user_hop_addition = bus.read_byte(chip1_addr)
    # make sure we received 0x1A
    if start_ack==0x1A:
        step0()
    else:
        tkMessageBox.showwarning("System Warning","ERROR! System start did not initiate!", icon='warning')


# declare the variables needed to get the SMS messages out
USERNAME = "autobrewucf@gmail.com"
PASSWORD = "autobrew2015"
MAILTO = "3216529332@vtext.com"
MAILTO2 = "4079293732@tmomail.net"
MAILTO3 = "7724462655@tmomail.net"
MAILTO4 = "9174536187@vtext.com"
# declare step number as a global variable to pass to the system_notification function
stepInfo = StringVar()


# declare all the system variables that will be read throughout the process
sys_kettle_temp = StringVar()
sys_kettle_level = StringVar()
sys_mash_temp = StringVar()
sys_cooling_temp = StringVar()
sys_valve1 = StringVar()
sys_valve2 = StringVar()
sys_valve3 = StringVar()
sys_valve4 = StringVar()
sys_valve5 = StringVar()
```

```
sys_valve6 = StringVar()
sys_valve7 = StringVar()
sys_wort_pump = StringVar()
sys_cool_pump = StringVar()
sys_heating_element = StringVar()
minute_counter = 0


# declare variables that will be read back immediately from the 1st GUI (user inputs)
user_mash_temp = StringVar()
user_mash_time = StringVar()
user_boil_temp = StringVar()
user_boil_time = StringVar()
user_cool_temp = StringVar()
user_hop_addition = StringVar()


# Declare the valve labels outside any fucntion so that they can be updated all
# throughout our process by different calling functions
status_valve1 = Tkinter.Label(canvas, text='      ')
status_valve1.place(x=350, y=530)
status_valve2 = Tkinter.Label(canvas, text='      ')
status_valve2.place(x=350, y=580)
status_valve3 = Tkinter.Label(canvas, text='      ')
status_valve3.place(x=350, y=630)
status_valve4 = Tkinter.Label(canvas, text='      ')
status_valve4.place(x=350, y=680)
status_valve5 = Tkinter.Label(canvas, text='      ')
status_valve5.place(x=350, y=730)
status_valve6 = Tkinter.Label(canvas, text='      ')
status_valve6.place(x=850, y=530)
status_valve7 = Tkinter.Label(canvas, text='      ')
status_valve7.place(x=850, y=580)
status_wort_pump = Tkinter.Label(canvas, text='      ')
status_wort_pump.place(x=850, y=630)
status_cool_pump = Tkinter.Label(canvas, text='      ')
status_cool_pump.place(x=850, y=680)
```

```
status_heating_element = Tkinter.Label(canvas, text='       ')
status_heating_element.place(x=850, y=730)
step_label = Tkinter.Label(canvas, font='Verdana 28 bold', bg='white')
step_label.place(x=350, y=460)


# Declare the system variable labels outside any fucntion so that they can be updated
# all throughout our process by different calling functions
sys_kettle_temp_label = Tkinter.Label(canvas, text='', font='Verdana 35 bold', bg='dark grey')
sys_kettle_temp_label.place(x=245, y=180)
sys_kettle_level_label = Tkinter.Label(canvas, text='', font='Verdana 35 bold', bg='dark grey')
sys_kettle_level_label.place(x=245, y= 290)
sys_mash_temp_label = Tkinter.Label(canvas, text='', font='Verdana 35 bold', bg='dark grey')
sys_mash_temp_label.place(x=750, y=180)
sys_cool_temp_label = Tkinter.Label(canvas, text='', font='Verdana 35 bold', bg='dark grey')
sys_cool_temp_label.place(x=1275, y=180)

# call functions to start the GUI
create_boxes()
boil_kettle_gui()
mash_tun_gui()
cooling_tank_gui()
currentStep()
system_valves()
exit_btn()
reset_btn()
root.update()
var = StringVar()
read_variables()
control_loop()

# displays all items on the root window
root.mainloop()
```