```
# program for doorbell
# 25.2.17
# by Julian Rogers
# door_bell_3
# This program is to run on a Pi Zero, interfacing with a “door knocker sensor” and three LEDs
# There is also provision for switching such things as relays and monitoring other inputs
# This means the program is a bit more complex than it needs to be
# The program communicates with another Pi using UDP
# The other Pi at address IP_LIB
# The LEDs are fitted into the transparent base of the door knocker and are for “decorative effect”
# They steadily increase and decrease in brightness 30 degrees out of phase with each other (is that right?)
# I can monitor this using VPN on my PC (or Android for that matter but, curiously, not another Pi).
# Sorry about global variables and any other programming foxes' paws!


IP_LIB = "xxx.xxx.xxx.xxx"           # insert IP of receiving computer
DEST_PORT_LIB = xxxx                 # insert port on receiving computer
                                     # I pick any number between 2000 and 9000. See Wiki for better advice!
                                     # Obviously, the port set on the receiving computer has to match.

from tkinter import *   #GUI

import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
# these are for relay outputs (relay board only!)
GPIO.setup(4, GPIO.OUT)
GPIO.setup(17, GPIO.OUT)
GPIO.setup(27, GPIO.OUT)
GPIO.setup(22, GPIO.OUT)

GPIO.output(4, False)
GPIO.output(17, False)
GPIO.output(27, False)
GPIO.output(22, False)

# inputs for door annunciator plus a spare

GPIO.setup(5, GPIO.IN)
GPIO.setup(6, GPIO.IN)
GPIO.setup(13, GPIO.IN)

# these are to test pwm
GPIO.setup(21, GPIO.OUT)
GPIO.setup(20, GPIO.OUT)
GPIO.setup(16, GPIO.OUT)

# 50 Hz
pwm21 = GPIO.PWM(21, 100)
pwm20 = GPIO.PWM(20, 100)
pwm16 = GPIO.PWM(16, 100)

# initial 50% duty cycle
pwm21.start(50)
pwm20.start(50)
pwm16.start(50)

import datetime
import time
import socket           #UDP
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
import smbus
bus = smbus.SMBus(1)


global red_led, green_led, yellow_led, red_flag, yellow_flag, green_flag, count

count = 0
red_led = 0
green_led = 33
yellow_led = 66
red_flag = True
yellow_flag = True
green_flag = True

# create the root window
root = Tk()

# modify the window
root.title("Julian's IOT CONTROLLER")
```

```
#root.geometry("700x430")
root.configure(bg = "gray")
#root.attributes('-fullscreen', True)   #eliminates the title bar

w, h = root.winfo_screenwidth(), root.winfo_screenheight()
root.geometry("%dx%d+0+0" % (w, h))

# create a frame
app = Frame(root)
app.configure(bg = "gray")
app.grid()

title_lab = Label(app, text = "GPIO CONTROL- Broadcom Numbering", font = ("Arial Bold", 20), fg = "maroon", bg =
"gray")
title_lab.grid(row = 1, column = 1, columnspan = 8)

blank_lab = Label(app, bg = "gray")
blank_lab.grid(row = 2, column = 1)

sys_1_but = Button(app, text = "Sys 1(04)", font = ("Arial", 16), fg = "maroon", bg = "light blue")
sys_1_but.grid(row = 3, column = 1)

# GPIO action commented out for these three buttons

def on_off_1():
    but_col = sys_1_but.config('bg')[-1]
    if but_col == "light blue":
        but_col = "yellow"
        #GPIO.output(4, True)
    else:
        but_col = "light blue"
        #GPIO.output(4, False)

    sys_1_but.config(bg = but_col)


sys_1_but.config(command = on_off_1)

sys_2_but = Button(app, text = "Sys 2(17)", font = ("Arial", 16), fg = "maroon", bg = "light blue")
sys_2_but.grid(row = 3, column = 2)

def on_off_2():
    but_col = sys_2_but.config('bg')[-1]
    if but_col == "light blue":
        but_col = "yellow"
        #GPIO.output(17, True)
    else:
        but_col = "light blue"
        #GPIO.output(17, False)

    sys_2_but.config(bg = but_col)


sys_2_but.config(command = on_off_2)

sys_3_but = Button(app, text = "Sys 3(27)", font = ("Arial", 16), fg = "maroon", bg = "light blue")
sys_3_but.grid(row = 3, column = 3)

def on_off_3():
    but_col = sys_3_but.config('bg')[-1]
    if but_col == "light blue":
        but_col = "yellow"
        #GPIO.output(27, True)
    else:
        but_col = "light blue"
        #GPIO.output(27, False)

    sys_3_but.config(bg = but_col)


sys_3_but.config(command = on_off_3)

sys_4_but = Button(app, text = "Sys 4(22)", font = ("Arial", 16), fg = "maroon", bg = "light blue")
sys_4_but.grid(row = 3, column = 4)

blank_lab2 = Label(app, bg = "gray")
blank_lab2.grid(row = 4, column = 1)

tmp_lab = Label(app, bg = "gray", fg = "maroon", text = "temp ", font = ("Arial", 16) )
tmp_lab.grid(row = 5, column = 1)
```

```
temp_lab = Label(app, bg = "gray", fg = "maroon", text = "reading...", font = ("Arial", 16) )
temp_lab.grid(row = 5, column = 2)

def on_off_4():
    but_col = sys_4_but.config('bg')[-1]
    if but_col == "light blue":
        but_col = "yellow"
        #GPIO.output(22, True)
    else:
        but_col = "light blue"
        #GPIO.output(22, False)

    sys_4_but.config(bg = but_col)


sys_4_but.config(command = on_off_4)

blank_lab3 = Label(app, bg = "gray")
blank_lab3.grid(row = 6, column = 1)

pwm_lab = Label(app, bg = "gray", fg = "maroon", text = "PWM value: ", font = ("Arial", 16) )
pwm_lab.grid(row = 7, column = 1)

var1 = StringVar(app)
var1.set("50")      #initial value
pwm_opt_men = OptionMenu(app, var1, "0", "10", "20", "30", "40", "50", "60", "70", "80", "90", "100")
pwm_opt_men.config(font = ("Arial", 14), fg = "black", bg = "white", width = 2)
pwm_opt_men.grid(row = 7, column = 2,)

blank_lab4 = Label(app, bg = "gray")
blank_lab4.grid(row = 8, column = 1)

var2 = Scale(app, from_=0, to=100, bg = "light blue", fg = "maroon",font = ("Arial", 14) )
var2.grid(row = 9, column = 1)

blank_lab5 = Label(app, bg = "gray")
blank_lab5.grid(row = 10, column = 1)

ind_but1 = Button(app, fg = "white", bg = "white", text = "X", font = ("Arial", 20))
ind_but1.grid(row = 11, column = 1)

ind_but2 = Button(app, fg = "white", bg = "white", text = "X", font = ("Arial", 20))
ind_but2.grid(row = 11, column = 2)

ind_but3 = Button(app, fg = "white", bg = "white", text = "X", font = ("Arial", 20))
ind_but3.grid(row = 11, column = 3)

blank_lab6 = Label(app, bg = "gray")
blank_lab6.grid(row = 12, column = 1)

lbl_feedback =Label(app, fg = "black", bg = "white")
lbl_feedback.grid(row = 13, column = 1)

lbl_feedback2 =Label(app, fg = "black", bg = "white")
lbl_feedback2.grid(row = 13, column = 2)


def send_UDP():
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        sock.sendto(bytes("BELL", "utf-8") , (IP_LIB, DEST_PORT_LIB))
       # sock wants bytes not Unicode (is that correct?)

        sock.settimeout(1)

        response = sock.recv(100)
        lbl_feedback.config(text = response)
        global count
        count = 0

    except socket.error:
        lbl_feedback.config(text = "error...")


    sock.close()

    #response = str(response, "utf-8")

def do_stuff():
    # if nothing else available:
```

```
#somethingtodo = 0
#some error handling needed here

# this is for i2c tmp102 sensor
#data = bus.read_i2c_block_data(0x48,0)
#msb = data[0]
#lsb = data[1]
#temperature = (((msb << 8) | lsb) >> 4) * 0.0625
#temp_lab.config(text = round(temperature,1))

# this is for option menu
#pwm_num = int(var1.get())

# this is for slider
pwm_num = var2.get()
var1.set(str(pwm_num))

# adjust pwm
#pwm21.ChangeDutyCycle(pwm_num)
#pwm20.ChangeDutyCycle(pwm_num)
#pwm16.ChangeDutyCycle(pwm_num)

global red_led, green_led, yellow_led, red_flag, yellow_flag, green_flag, count

if red_flag:
    red_led = red_led + 1
else:
    red_led = red_led -1

if red_led > 100:
    red_led = 100
    red_flag = False

if red_led < 0:
    red_led = 0
    red_flag = True

if green_flag:
    green_led = green_led + 1
else:
    green_led = green_led -1

if green_led > 100:
    green_led = 100
    green_flag = False

if green_led < 0:
    green_led = 0
    green_flag = True

if yellow_flag:
    yellow_led = yellow_led + 1
else:
    yellow_led = yellow_led -1

if yellow_led > 100:
    yellow_led = 100
    yellow_flag = False


if yellow_led < 0:
    yellow_led = 0
    yellow_flag = True



ind_but1.config(text = red_led)
ind_but2.config(text = green_led)
ind_but3.config(text = yellow_led)


# adjust pwm
pwm21.ChangeDutyCycle(red_led)
pwm20.ChangeDutyCycle(green_led)
pwm16.ChangeDutyCycle(yellow_led)


# check inputs

if GPIO.input(5) == 1:
        ind_but1.config(bg = "red", fg = "red")
        send_UDP()

    else:
        ind_but1.config(bg = "white", fg = "white")

    if GPIO.input(6) == 1:
        ind_but2.config(bg = "red", fg = "red")

    else:
        ind_but2.config(bg = "white", fg = "white")

    if GPIO.input(13) == 1:
        ind_but3.config(bg = "red", fg = "red")

    else:
        ind_but3.config(bg = "white", fg = "white")



    count = count + 1
    lbl_feedback2.config(text = count)
    if count > 30:
        lbl_feedback.config(text = "")
        count = 0


# calls a function after given time
def after(self, ms, func = None, *args):
    """call function after a given time"""

# updates screen every 0.1 seconds
def task():

    do_stuff()
    root.after(100, task)

# calls the screen update every 0.1 seconds
root.after(100, task)

#------------------------------------------------------------------------------------------
# kick off the window's event-loop
root.mainloop()
```