camara\_UV.py

--------------------------------------------------------------------------------------------------------------------------------------------------------------

#inporta bibliotecas

import RPi.GPIO as GPIO #libreria para el manejo de los GPIO

import os

import sys

import lcddriver #importamos configuracion de manejo de LCD

from time import \*

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

GPIO.setup(14, GPIO.IN)

GPIO.setup(15, GPIO.IN)

GPIO.setup(4, GPIO.OUT)

GPIO.setup(18, GPIO.IN)

GPIO.setup(25, GPIO.OUT)

minP=0 #variable que guarda cantidad de minutos para mostrar

t1pro=0 #variable para saver cuantos minutos se programan

t2=0 #iniciar segundos

exit=0 #para cuando llegue a 0 el conteo

next=0 #salir del conteo

a=29

E=0

#inicializa le lcd

lcd = lcddriver.lcd()

GPIO.output(4, GPIO.LOW)

lcd.lcd\_display\_string(" INSOLADORA UV ",0)

lcd.lcd\_display\_string(" ITCA-FEPADE ",20)

sleep(2)

lcd.lcd\_clear()

while True:

while True:

lcd.lcd\_display\_string(" programe ",0)

lcd.lcd\_display\_string(" minutos: "+str(t1pro) ,20)

inputValue = GPIO.input(14)

if (inputValue == True):

print("pulso 1")

t1pro = t1pro + 1

minP=minP+1

sleep(0.3)

if minP >30:

minP=0

if t1pro>30:

t1pro=0

lcd.lcd\_clear()

if (GPIO.input(15) == True):

sleep(0.3)

print("pulso 2")

lcd.lcd\_clear()

break

GPIO.output(4, GPIO.HIGH)

for b in range(t1pro):

if E==1:

t1pro=0

E=0

minP=0

t2=0

break

for j in range(60):

if GPIO.input(18) == True:

print("pulso 3")

sleep(0.3)

lcd.lcd\_clear()

E=1

break

if t2==0:

t2=60

t1pro=t1pro-1

t2=t2-1

lcd.lcd\_display\_string("Time: "+str(minP)+" Min(s)",0)

if t1pro == minP:

lcd.lcd\_display\_string(" ",20)

lcd.lcd\_display\_string(str(t1pro)+ ":" +str(t2) ,24)

sleep(1)

if t1pro>=10 and t2 == 10:

a=28

lcd.lcd\_display\_string(" ",a)

else:

a=29

lcd.lcd\_display\_string(" ",a)

if t1pro <= 9 and t2 == 10:

a=27

lcd.lcd\_display\_string(" ",a)

if t1pro==0 and t2==0:

GPIO.output(4, GPIO.LOW)

lcd.lcd\_display\_string(" Expocicion ", 0)

lcd.lcd\_display\_string(" Finalizada ", 20)

GPIO.output(25, GPIO.HIGH)

sleep(1)

GPIO.output(25, GPIO.LOW)

sleep(1)

GPIO.output(25, GPIO.HIGH)

sleep(1)

GPIO.output(25, GPIO.LOW)

sleep(2)

lcd.lcd\_clear()

break

while True:

GPIO.output(4, GPIO.LOW)

lcd.lcd\_display\_string(" Standby ", 0)

lcd.lcd\_display\_string("1-OFF 2-NEW CONT", 20)

if GPIO.input(15) == True:

sleep(0.3)

print("apagar")

while True:

if GPIO.input(18) == True:

break

lcd.lcd\_clear()

lcd.lcd\_backlight("OFF")

if GPIO.input(14) == True:

sleep(0.3)

print("next")

lcd.lcd\_clear()

break

continue

-----------------------------------------------------------------------------------------------------------------------------------

lcddriver.py

-----------------------------------------------------------------------------------------------------------------------------------

import sys

sys.path.append("./lib")

import i2c\_lib

from time import \*

# LCD Address

ADDRESS = 0x27

# commands

LCD\_CLEARDISPLAY = 0x01

LCD\_RETURNHOME = 0x02

LCD\_ENTRYMODESET = 0x04

LCD\_DISPLAYCONTROL = 0x08

LCD\_CURSORSHIFT = 0x10

LCD\_FUNCTIONSET = 0x20

LCD\_SETCGRAMADDR = 0x40

LCD\_SETDDRAMADDR = 0x80

# flags for display entry mode

LCD\_ENTRYRIGHT = 0x00

LCD\_ENTRYLEFT = 0x02

LCD\_ENTRYSHIFTINCREMENT = 0x01

LCD\_ENTRYSHIFTDECREMENT = 0x00

# flags for display on/off control

LCD\_DISPLAYON = 0x04

LCD\_DISPLAYOFF = 0x00

LCD\_CURSORON = 0x02

LCD\_CURSOROFF = 0x00

LCD\_BLINKON = 0x01

LCD\_BLINKOFF = 0x00

# flags for display/cursor shift

LCD\_DISPLAYMOVE = 0x08

LCD\_CURSORMOVE = 0x00

LCD\_MOVERIGHT = 0x04

LCD\_MOVELEFT = 0x00

# flags for function set

LCD\_8BITMODE = 0x10

LCD\_4BITMODE = 0x00

LCD\_2LINE = 0x08

LCD\_1LINE = 0x00

LCD\_5x10DOTS = 0x04

LCD\_5x8DOTS = 0x00

# flags for backlight control

LCD\_BACKLIGHT = 0x08

LCD\_NOBACKLIGHT = 0x00

En = 0b00000100 # Enable bit

Rw = 0b00000010 # Read/Write bit

Rs = 0b00000001 # Register select bit

class lcd:

#initializes objects and lcd

def \_\_init\_\_(self):

self.lcd\_device = i2c\_lib.i2c\_device(ADDRESS)

self.lcd\_write(0x03)

self.lcd\_write(0x03)

self.lcd\_write(0x03)

self.lcd\_write(0x02)

self.lcd\_write(LCD\_FUNCTIONSET | LCD\_2LINE | LCD\_5x8DOTS | LCD\_4BITMODE)

self.lcd\_write(LCD\_DISPLAYCONTROL | LCD\_DISPLAYON)

self.lcd\_write(LCD\_CLEARDISPLAY)

self.lcd\_write(LCD\_ENTRYMODESET | LCD\_ENTRYLEFT)

sleep(0.2)

# clocks EN to latch command

def lcd\_strobe(self, data):

self.lcd\_device.write\_cmd(data | En | LCD\_BACKLIGHT)

sleep(.0005)

self.lcd\_device.write\_cmd(((data & ~En) | LCD\_BACKLIGHT))

sleep(.0001)

def lcd\_write\_four\_bits(self, data):

self.lcd\_device.write\_cmd(data | LCD\_BACKLIGHT)

self.lcd\_strobe(data)

# write a command to lcd

def lcd\_write(self, cmd, mode=0):

self.lcd\_write\_four\_bits(mode | (cmd & 0xF0))

self.lcd\_write\_four\_bits(mode | ((cmd << 4) & 0xF0))

#turn on/off the lcd backlight

def lcd\_backlight(self, state):

if state in ("on","On","ON"):

self.lcd\_device.write\_cmd(LCD\_BACKLIGHT)

elif state in ("off","Off","OFF"):

self.lcd\_device.write\_cmd(LCD\_NOBACKLIGHT)

else:

print "Unknown State!"

# put string function

def lcd\_display\_string(self, string, line):

if line == 0:

self.lcd\_write(0x80)

if line == 1:

self.lcd\_write(0x81)

if line == 2:

self.lcd\_write(0x82)

if line == 3:

self.lcd\_write(0x83)

if line == 4:

self.lcd\_write(0x84)

if line == 5:

self.lcd\_write(0x85)

if line == 6:

self.lcd\_write(0x86)

if line == 7:

self.lcd\_write(0x87)

if line == 8:

self.lcd\_write(0x88)

if line == 9:

self.lcd\_write(0x89)

if line == 20:

self.lcd\_write(0xC0)

if line == 21:

self.lcd\_write(0xC1)

if line == 22:

self.lcd\_write(0xC2)

if line == 23:

self.lcd\_write(0xC3)

if line == 24:

self.lcd\_write(0xC4)

if line == 25:

self.lcd\_write(0xC5)

if line == 26:

self.lcd\_write(0xC6)

if line == 27:

self.lcd\_write(0xC7)

if line == 28:

self.lcd\_write(0xC8)

if line == 3:

self.lcd\_write(0x94)

if line == 4:

self.lcd\_write(0xD4)

for char in string:

self.lcd\_write(ord(char), Rs)

# clear lcd and set to home

def lcd\_clear(self):

self.lcd\_write(LCD\_CLEARDISPLAY)

self.lcd\_write(LCD\_RETURNHOME)

-----------------------------------------------------------------------------------------------------------------------------------

i2c\_lib.py

-----------------------------------------------------------------------------------------------------------------------------------

import sys

sys.path.append("./lib")

import smbus

from time import \*

class i2c\_device:

def \_\_init\_\_(self, addr, port=1):

self.addr = addr

self.bus = smbus.SMBus(port)

# Write a single command

def write\_cmd(self, cmd):

self.bus.write\_byte(self.addr, cmd)

sleep(0.0001)

# Write a command and argument

def write\_cmd\_arg(self, cmd, data):

self.bus.write\_byte\_data(self.addr, cmd, data)

sleep(0.0001)

# Write a block of data

def write\_block\_data(self, cmd, data):

self.bus.write\_block\_data(self.addr, cmd, data)

sleep(0.0001)

# Read a single byte

def read(self):

return self.bus.read\_byte(self.addr)

# Read

def read\_data(self, cmd):

return self.bus.read\_byte\_data(self.addr, cmd)

# Read a block of data

def read\_block\_data(self, cmd):

return self.bus.read\_block\_data(self.addr, cmd)