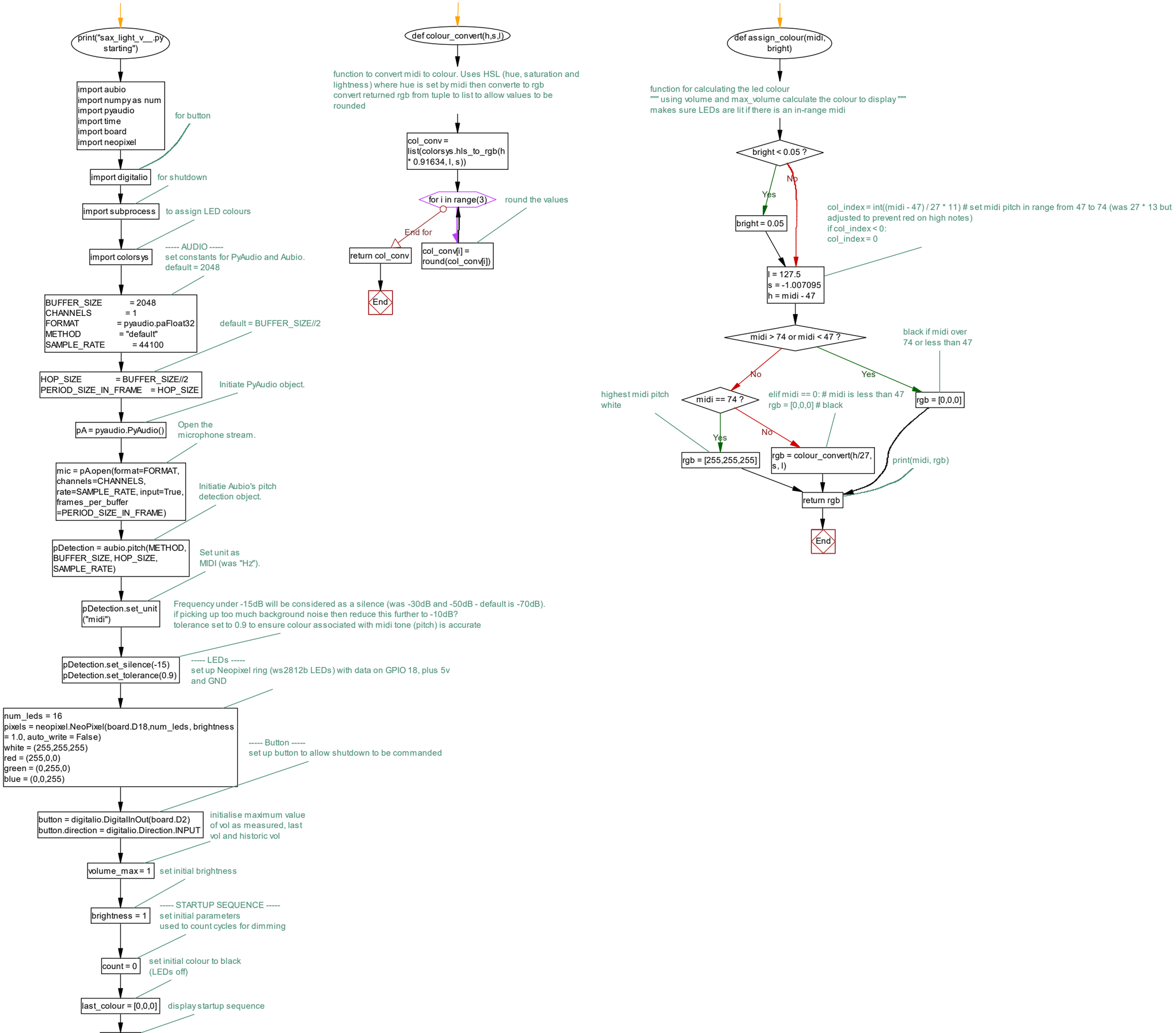
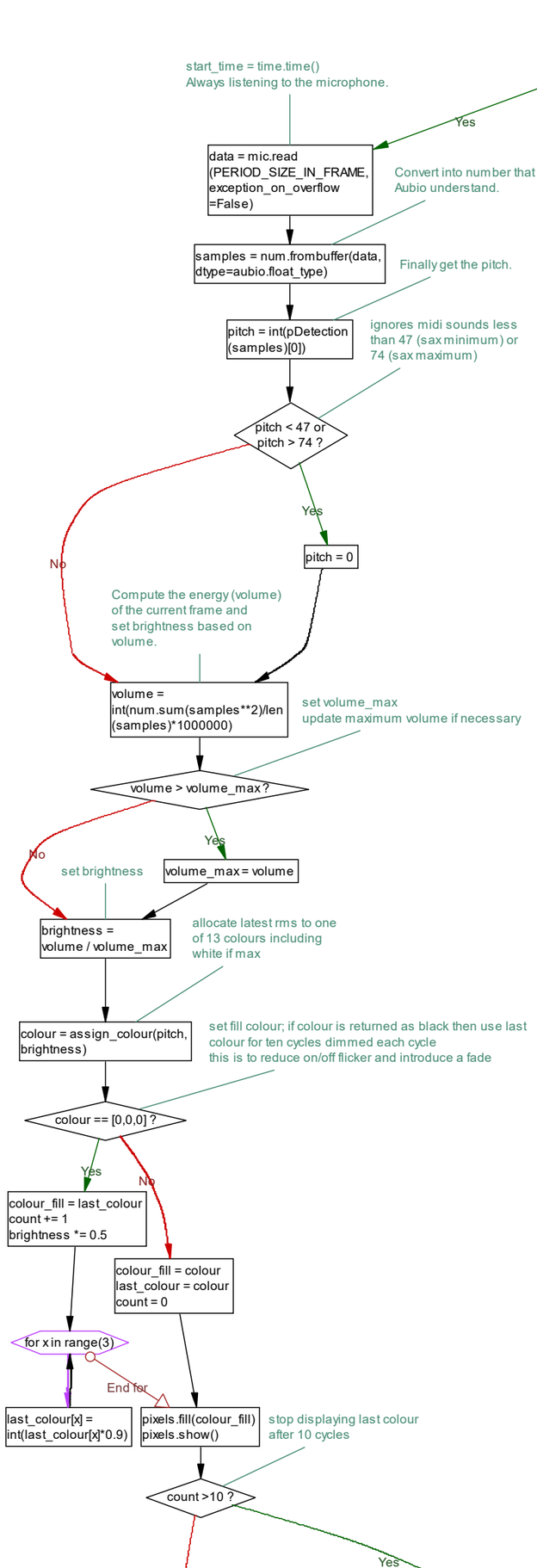
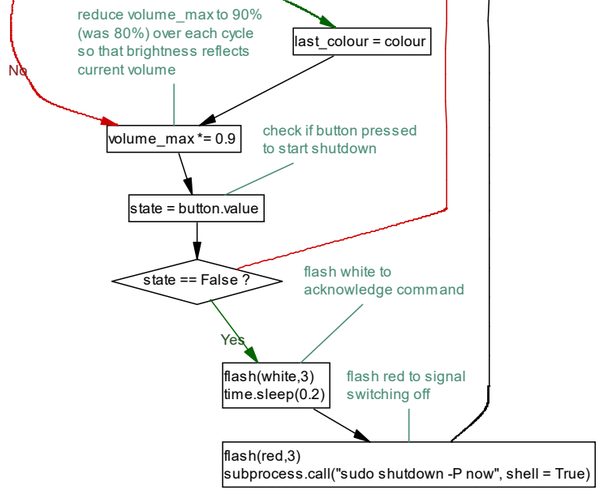
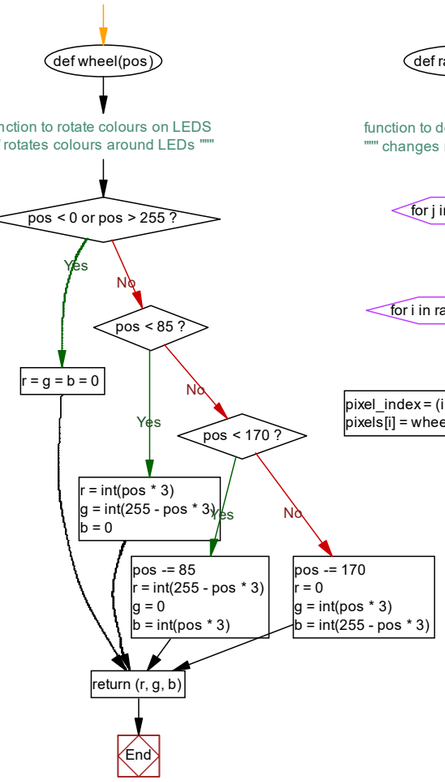
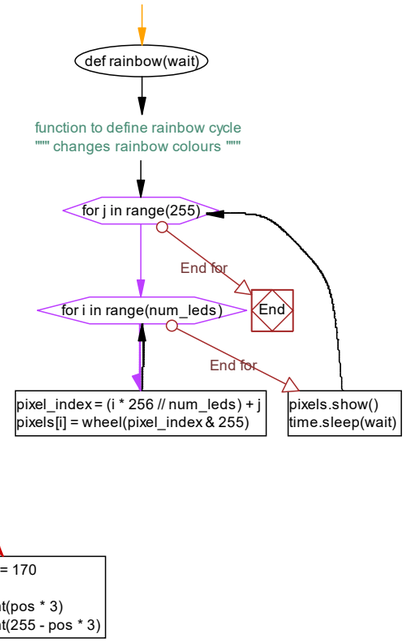
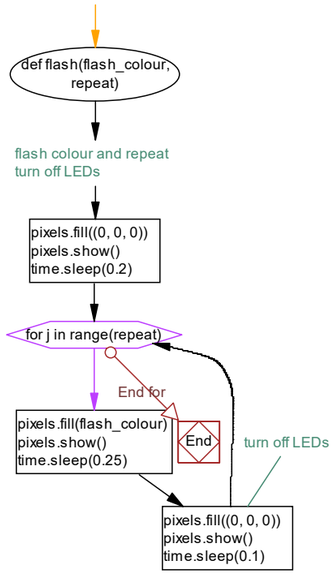

print("sax_light_v__.py starting")
import aubio
import numpy as num
import pyaudio
import time
import board
import neopixel
for button
import digitalio
for shutdown
import subprocess
to assign LED colours
import colorsys
----- AUDIO -----
set constants for PyAudio and Aubio.
default = 2048
BUFFER_SIZE = 2048
CHANNELS = 1
FORMAT = pyaudio.paFloat32
METHOD = "default"
SAMPLE_RATE = 44100
default = BUFFER_SIZE//2
HOP_SIZE = BUFFER_SIZE//2
PERIOD_SIZE_IN_FRAME = HOP_SIZE
Initiate PyAudio object.
pA = pyaudio.PyAudio()
Open the microphone stream.
mic = pA.open(format=FORMAT, channels=CHANNELS, rate=SAMPLE_RATE, input=True, frames_per_buffer =PERIOD_SIZE_IN_FRAME)
Initiatie Aubio's pitch detection object.
pDetection = aubio.pitch(METHOD, BUFFER_SIZE, HOP_SIZE, SAMPLE_RATE)
Set unit as MIDI (was "Hz").
pDetection.set_unit ("midi")
Frequency under -15dB will be considered as a silence (was -30dB and -50dB - default is -70dB).
if picking up too much background noise then reduce this further to -10dB?
tolerance set to 0.9 to ensure colour associated with midi tone (pitch) is accurate
pDetection.set_silence(-15)
pDetection.set_tolerance(0.9)
----- LEDs -----
set up Neopixel ring (ws2812b LEDs) with data on GPIO 18, plus 5v and GND
num_leds = 16
pixels = neopixel.NeoPixel(board.D18,num_leds, brightness = 1.0, auto_write = False)
white = (255,255,255)
red = (255,0,0)
green = (0,255,0)
blue = (0,0,255)
----- Button -----
set up button to allow shutdown to be commanded
button = digitalio.DigitalInOut(board.D2)
button.direction = digitalio.Direction.INPUT
initialise maximum value of vol as measured, last vol and historic vol
volume_max = 1
set initial brightness
brightness = 1
----- STARTUP SEQUENCE -----
set initial parameters
used to count cycles for dimming
count = 0
set initial colour to black (LEDs off)
last_colour = [0,0,0]
display startup sequence
def colour_convert(h,s,l)
function to convert midi to colour. Uses HSL (hue, saturation and lightness) where hue is set by midi then converte to rgb
convert returned rgb from tuple to list to allow values to be rounded
col_conv = list(colorsys.hls_to_rgb(h * 0.91634, l, s))
for i in range(3)
round the values
End for
col_conv[i] = round(col_conv[i])
return col_conv
End
def assign_colour(midi, bright)
function for calculating the led colour
""" using volume and max_volume calculate the colour to display """
makes sure LEDs are lit if there is an in-range midi
bright < 0.05 ?
Yes
No
bright = 0.05
col_index = int((midi - 47) / 27 * 11) # set midi pitch in range from 47 to 74 (was 27 * 13 but adjusted to prevent red on high notes)
if col_index < 0:
col_index = 0
l = 127.5
s = -1.007095
h = midi - 47
midi > 74 or midi < 47 ?
black if midi over 74 or less than 47
No
Yes
rgb = [0,0,0]
highest midi pitch white
midi == 74 ?
elif midi == 0: # midi is less than 47
rgb = [0,0,0] # black
Yes
No
rgb = [255,255,255]
rgb = colour_convert(h/27, s, l)
print(midi, rgb)
return rgb
End

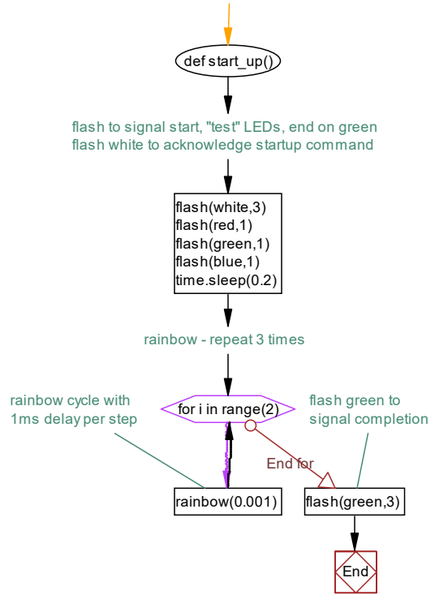

start_up()
True ?
Yes
No
End
start_time = time.time()
Always listening to the microphone.
data = mic.read
(PERIOD_SIZE_IN_FRAME,
exception_on_overflow
=False)
Convert into number that
Aubio understand.
samples = num.frombuffer(data,
dtype=aubio.float_type)
Finally get the pitch.
pitch = int(pDetection
(samples)[0])
ignores midi sounds less
than 47 (sax minimum) or
74 (sax maximum)
pitch < 47 or
pitch > 74 ?
Yes
No
pitch = 0
Compute the energy (volume)
of the current frame and
set brightness based on
volume.
volume =
int(num.sum(samples**2)/len
(samples)*1000000)
set volume_max
update maximum volume if necessary
volume > volume_max ?
Yes
No
volume_max = volume
set brightness
brightness =
volume / volume_max
allocate latest rms to one
of 13 colours including
white if max
colour = assign_colour(pitch,
brightness)
set fill colour; if colour is returned as black then use last
colour for ten cycles dimmed each cycle
this is to reduce on/off flicker and introduce a fade
colour == [0,0,0] ?
Yes
No
colour_fill = last_colour
count += 1
brightness *= 0.5
colour_fill = colour
last_colour = colour
count = 0
for x in range(3)
End for
last_colour[x] =
int(last_colour[x]*0.9)
pixels.fill(colour_fill)
pixels.show()
stop displaying last colour
after 10 cycles
count >10 ?
Yes


reduce volume_max to 90%
(was 80%) over each cycle
so that brightness reflects
current volume
last_colour = colour
No
volume_max *= 0.9
check if button pressed
to start shutdown
state = button.value
state == False ?
flash white to
acknowledge command
Yes
flash(white,3)
time.sleep(0.2)
flash red to signal
switching off
flash(red,3)
subprocess.call("sudo shutdown -P now", shell = True)


def wheel(pos)
nction to rotate colours on LEDS
rotates colours around LEDs """
pos < 0 or pos > 255 ?
Yes
No
r = g = b = 0
pos < 85 ?
No
Yes
r = int(pos * 3)
g = int(255 - pos * 3)
b = 0
pos < 170 ?
Yes
No
pos -= 85
r = int(255 - pos * 3)
g = 0
b = int(pos * 3)
pos -= 170
r = 0
g = int(pos * 3)
b = int(255 - pos * 3)
return (r, g, b)
End


def rainbow(wait)
function to define rainbow cycle
""" changes rainbow colours """
for j in range(255)
End for
End
for i in range(num_leds)
End for
pixel_index = (i * 256 // num_leds) + j
pixels[i] = wheel(pixel_index & 255)
pixels.show()
time.sleep(wait)


def flash(flash_colour, repeat)
flash colour and repeat
turn off LEDs
pixels.fill((0, 0, 0))
pixels.show()
time.sleep(0.2)
for j in range(repeat)
End for
pixels.fill(flash_colour)
pixels.show()
time.sleep(0.25)
End
turn off LEDs
pixels.fill((0, 0, 0))
pixels.show()
time.sleep(0.1)


def start_up()
flash to signal start, "test" LEDs, end on green
flash white to acknowledge startup command
flash(white,3)
flash(red,1)
flash(green,1)
flash(blue,1)
time.sleep(0.2)
rainbow - repeat 3 times
rainbow cycle with
1ms delay per step
for i in range(2)
flash green to
signal completion
End for
rainbow(0.001)
flash(green,3)
End