Sound Reactive NeoPixel Peace Pendant

Created by Lalindra Jayatilleke

https://learn.adafruit.com/sound-reactive-neopixel-peace-pendant

Last updated on 2023-08-29 03:04:02 PM EDT

Table of Contents

Overview

This Peace pendant project is a great way to hone your 3D printing + electronics skills.

The circuitry for this wearable project isn't that complex, and the 3D printed enclosure will provide a compact pendant that you can wear to your next party or event.

This guide was written for the 'original' Gemma board, but can be done with either the original or M0 Gemma. We recommend the Gemma M0 as it is easier to use and is more compatible with modern computers!

Features:

- Sound reactive
- Rechargeable
- On/Off switch
- Programmable

Before you begin, please read the following prerequisite guides:

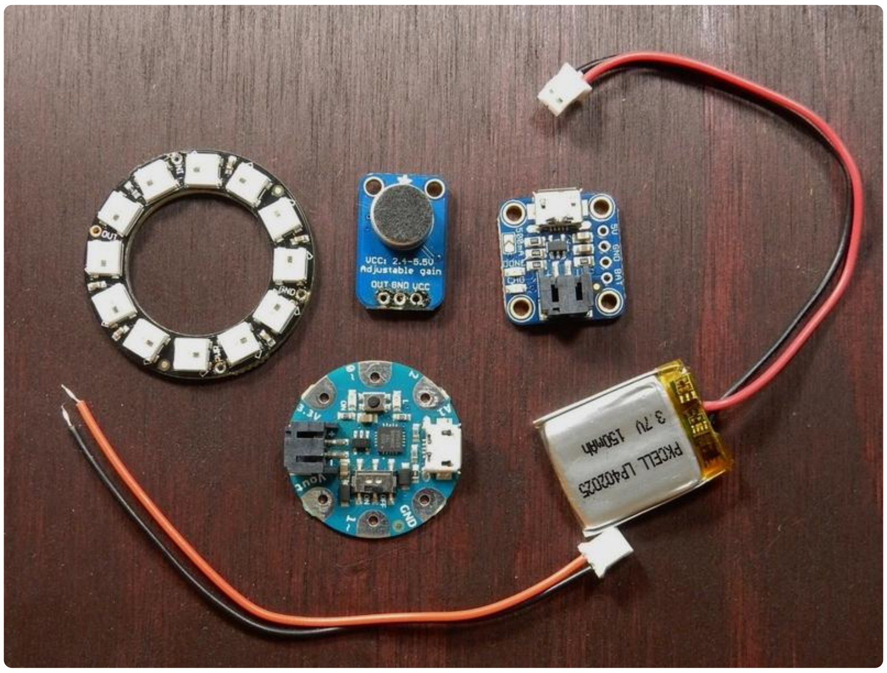
- Adafruit Guide To Excellent Soldering ()
- Battery Powering your Wearable Electronics ()
- Introducing Gemma () or Introducing Gemma M0 ()

Parts:

- Gemma M0 () or Gemma v2 (http://adafru.it/1222) (M0 type is recommended!)
- NeoPixel Ring (12 LEDs). (http://adafru.it/1643)
- Switch. (http://adafru.it/805)
- #4 3/8 inch Screws (x2) () or similar.
- Li-Po battery 3.7V 150mAh. ()
- Lipoly battery charger. (http://adafru.it/1904)
- Mic Amp [MAX4466] (http://adafru.it/1063) or Mic Amp [MAX9814] ()either will work
- Necklace and ring.
- Micro USB cable for programming/charging.
- 22 AWG hook-up wire (http://adafru.it/290) or any other suitable wire.

Tools and Supplies:

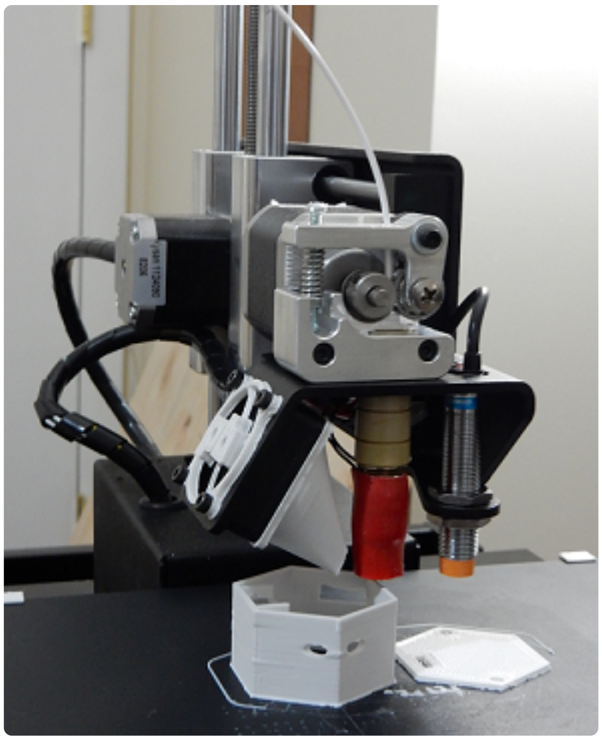
- Access to a 3D printer with filament.
- Soldering Iron with solder.
- Wire Strippers and pliers.
- Hot glue gun and super glue.



3D Printing

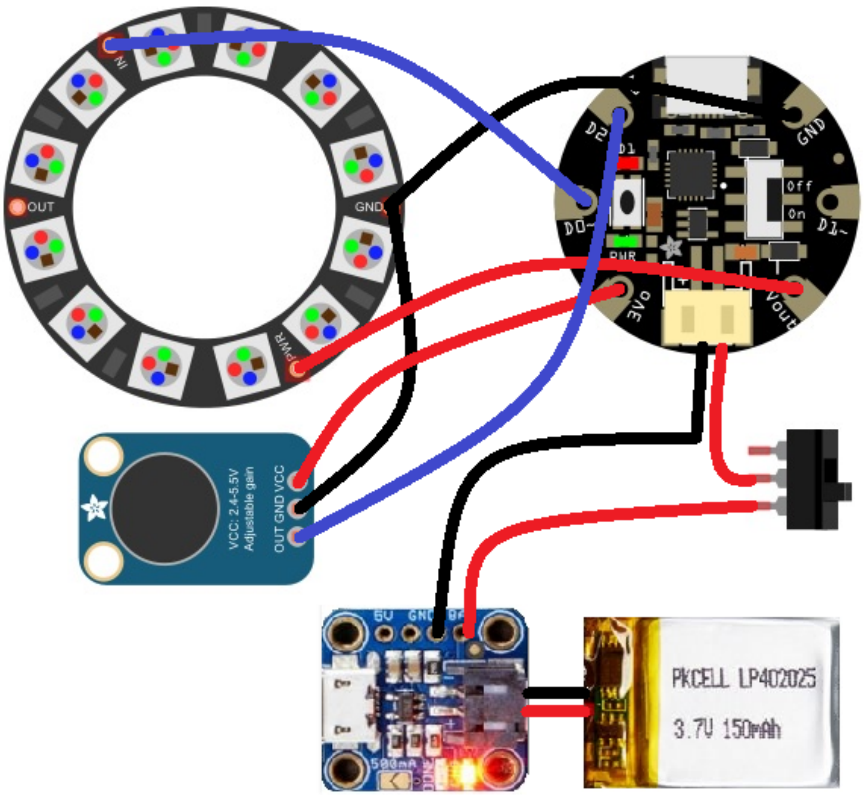
Please download the .stl files and print out the enclosure and lid. Some support material is required, but nothing intense.

Click here to download the .STL files from Thingiverse



Assembly

The pendant assembly shouldn't be that complicated. The enclosure was designed to comfortably hold the electronics without much hassle. Please see the video () for an assembly summary.



This diagram uses the original Gemma but you can also use the Gemma M0 with the exact same wiring!

Circuit Summary:

- Gemma D0 -> NeoPixel ring IN
- Gemma D2 -> Mic OUT
- Gemma GND -> NeoPixel ring GND
- Gemma 3V -> Mic VCC
- Gemma Vout -> NeoPixel ring PWR
- NeoPixel ring GND-> Mic GND
- LiPoly charger GND-> Black wire on JST cable
- LiPoly charger BAT-> Switch corner pin
- Switch middle pin-> Red wire on JST cable

1) Start by soldering short lengths of hook-up wire to the LED ring. The front face of the enclosure was designed with holes aligned to the LED ring wire solder points. Mount the ring onto the face of the enclosure and slide the wires into the enclosure. You could use some adhesive to anchor the ring, but this isn't required since the fit is tight enough to hold it in place.

2) Solder a short piece of wire to one of the far-end pins on the slide switch, and connect that to the BAT of the LiPo charger.

Solder the RED lead of the JST-PH lead onto the middle pin of the slide switch. Please see the circuit diagram.

3) Push the slide switch into the opening on the side of the enclosure. Secure the switch in place by using some glue.

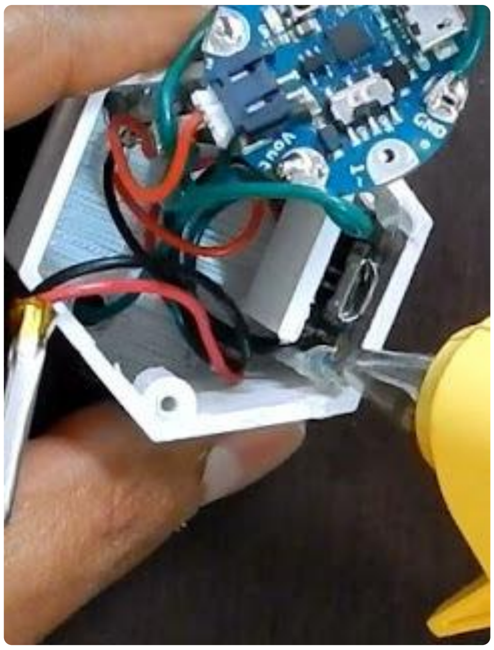
Please be careful not to encircle the entire switch with glue or else the switch won't work! You could also use hot glue on the inside to hold the switch in place.

4) Solder the Black GND lead of the JST-PH lead to the GND on the LiPo charger. Please ensure the wires soldered to the LiPo charger are attached to the TOP side of the charger. This ensures the charger would sit flush when inserted into the enclosure.

5) Solder short lengths of hook-up wire to the MIC amp. The connections are shown in the circuit picture above.

6) Solder the LED ring wires to the Gemma according to the circuit diagram.

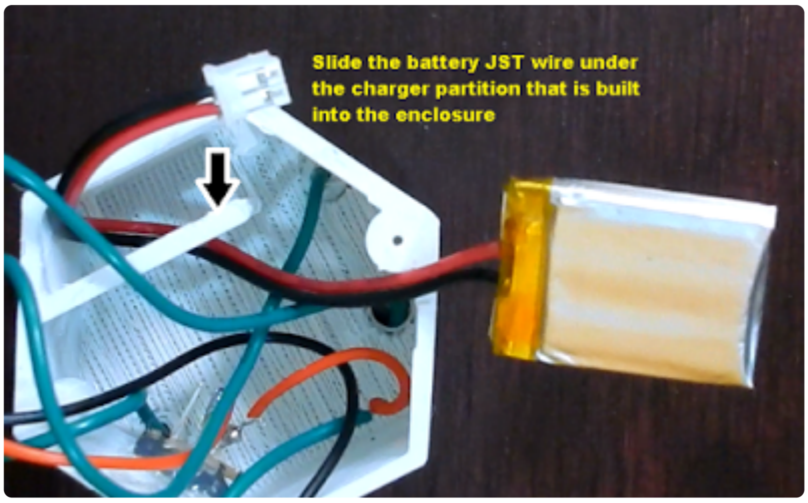
7) Insert the battery lead into the LiPo charger and seat the charger in the enclosure. Use some hot glue to secure the charger on the enclosure. Ensure that this is properly done because the USB cable will constantly be attached and detached for charging.



8) At this point, the soldering is complete. Lets program the Gemma before final assembly. You will need the Adafruit NeoPixel library () for the sketch. Grab the Arduino sketch () and load it onto the Gemma. Putting the Gemma into the bootloader mode is documented here ().

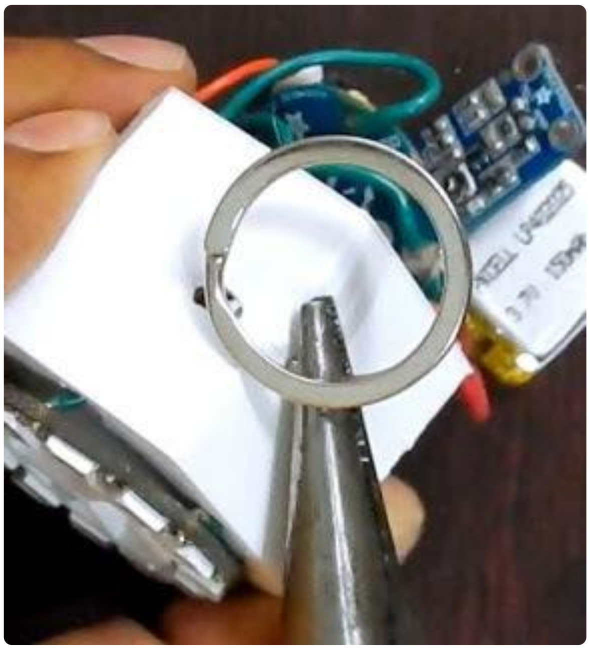
Once the sketch is loaded, test the flashing of the LEDs by tapping the mic. The LEDs should react to the sound. If it does not work, please check your wiring and ensure your Arduino sketch is loaded correctly.

9) Once the circuit is verified to be working, button-up your project by putting the LiPo battery in the enclosure and pass the JST lead under the charger partition that is built into the enclosure. Please see below.


Slide the battery JST wire under
the charger partition that is built
into the enclosure

10) If you want to insert a ring to hang the enclosure on a laynard, now would be the time before final assembly.

Please see below.



11) Lets do the final assembly by inserting the battery first, then the Gemma.

Now insert a small plastic spacer before putting in the MIC to prevent any short circuits. You can use a piece of plastic. I used a scrap piece of 3D printed plastic and placed it on top of the Gemma, then inserted the MIC on top of that.

Close the enclosure using 2 x #4 3/8 inch screws with the 3D printed lid. Ensure that the USB charging port lines up with the opening. Attach a laynard, beaded necklace or whatever you like for a necklace.

Arduino Code

The Arduino code presented below works equally well on all versions of GEMMA: v2 and M0. But if you have an M0 board, consider using the CircuitPython code on the next page of this guide, no Arduino IDE required! Click to Download the NeoPixel Library

Installing Arduino libraries is a frequent stumbling block. If this is your first time, or simply needing a refresher, please read the All About Arduino Libraries () tutorial. ()If the library is correctly installed (and the Arduino IDE is restarted), you should be able to navigate through the “File” rollover menus as follows:

File→Sketchbook→Libraries→Adafruit_NeoPixel→strandtest

```
// SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <Adafruit_NeoPixel.h>

#define N_PIXELS  12  // Number of pixels you are using
#define MIC_PIN   A1  // Microphone is attached to Trinket GPIO #2/Gemma D2 (A1)
#define LED_PIN    0  // NeoPixel LED strand is connected to GPIO #0 / D0
#define DC_OFFSET  0  // DC offset in mic signal - if unusure, leave 0
#define NOISE     100  // Noise/hum/interference in mic signal
#define SAMPLES   60  // Length of buffer for dynamic level adjustment
#define TOP       (N_PIXELS +1) // Allow dot to go slightly off scale

byte
  peak      = 0,      // Used for falling dot
  dotCount  = 0,      // Frame counter for delaying dot-falling speed
  volCount  = 0;      // Frame counter for storing past volume data

int
  vol[SAMPLES],       // Collection of prior volume samples
  lvl       = 10,     // Current "dampened" audio level
  minLvlAvg = 0,      // For dynamic adjustment of graph low & high
  maxLvlAvg = 512;

Adafruit_NeoPixel  strip = Adafruit_NeoPixel(N_PIXELS, LED_PIN, NEO_GRB +
NEO_KHZ800);

void setup() {
  memset(vol, 0, sizeof(vol));
  strip.begin();
}
void loop() {
  uint8_t  i;
  uint16_t minLvl, maxLvl;
  int      n, height;
  n   = analogRead(MIC_PIN);                 // Raw reading from mic
  n   = abs(n - 512 - DC_OFFSET);            // Center on zero
  n   = (n <= NOISE) ? 0 : (n - NOISE);      // Remove noise/hum
  lvl = ((lvl * 7) + n) >> 3;    // "Dampened" reading (else looks twitchy)

  // Calculate bar height based on dynamic min/max levels (fixed point):
  height = TOP * (lvl - minLvlAvg) / (long)(maxLvlAvg - minLvlAvg);
  if(height < 0L)       height = 0;      // Clip output
  else if(height > TOP) height = TOP;
  if(height > peak)     peak   = height; // Keep 'peak' dot at top

  // Color pixels based on rainbow gradient
  for(i=0; i<N_PIXELS; i++) {
    if(i >= height)
       strip.setPixelColor(i,   0,   0, 0);
    else
       strip.setPixelColor(i,Wheel(map(i,0,strip.numPixels()-1,30,150)));
    }

   strip.show(); // Update strip

  vol[volCount] = n;                      // Save sample for dynamic leveling
  if(++volCount >= SAMPLES) volCount = 0; // Advance/rollover sample counter

  // Get volume range of prior frames
  minLvl = maxLvl = vol[0];
  for(i=1; i<SAMPLES; i++) {
    if(vol[i] < minLvl)      minLvl = vol[i];
    else if(vol[i] > maxLvl) maxLvl = vol[i];
  }
  // minLvl and maxLvl indicate the volume range over prior frames, used
  // for vertically scaling the output graph (so it looks interesting
  // regardless of volume level).  If they're too close together though
  // (e.g. at very low volume levels) the graph becomes super coarse
  // and 'jumpy'...so keep some minimum distance between them (this
  // also lets the graph go to zero when no sound is playing):
  if((maxLvl - minLvl) < TOP) maxLvl = minLvl + TOP;
  minLvlAvg = (minLvlAvg * 63 + minLvl) >> 6; // Dampen min/max levels
  maxLvlAvg = (maxLvlAvg * 63 + maxLvl) >> 6; // (fake rolling average)
}

// Input a value 0 to 255 to get a color value.
// The colors are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  if(WheelPos < 85) {
   return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  } else if(WheelPos < 170) {
   WheelPos -= 85;
   return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  } else {
   WheelPos -= 170;
   return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
}
```

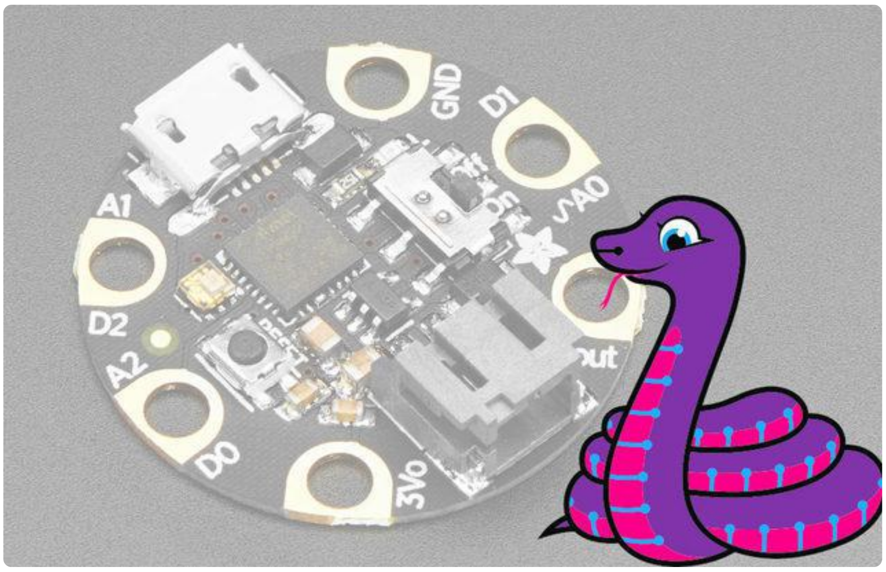
From the Tools→Board menu, select the device you are using:

- Adafruit Gemma M0
- Adafruit Gemma 8 MHz

Connect the USB cable between the computer and your device. The original Gemma (8 MHz) need the reset button pressed on the board, then click the upload button (right arrow icon) in the Arduino IDE. You do not need to press the reset on the newer Gemma M0.

When the battery is connected, you should get a light show from the LEDs. All your pixels working? Great! You can take apart this prototype and get ready to put the pixels in the collar. Refer to the NeoPixel Uberguide () for more info.

CircuitPython Code



GEMMA M0 boards can run CircuitPython — a different approach to programming compared to Arduino sketches. In fact, CircuitPython comes factory pre-loaded on GEMMA M0. If you've overwritten it with an Arduino sketch, or just want to learn the basics of setting up and using CircuitPython, this is explained in the Adafruit GEMMA M0 guide ().

These directions are specific to the "M0" GEMMA board. The original GEMMA with an 8-bit AVR microcontroller doesn't run CircuitPython…for those boards, use the Arduino sketch on the "Arduino code" page of this guide.

Below is CircuitPython code that works similarly (though not exactly the same) as the Arduino sketch shown on a prior page. To use this, plug the GEMMA M0 into USB…it should show up on your computer as a small flash drive…then edit the file "main.py" with your text editor of choice. Select and copy the code below and paste it into that file, entirely replacing its contents (don't mix it in with lingering bits of old code). When you save the file, the code should start running almost immediately (if not, see notes at the bottom of this page).

If GEMMA M0 doesn't show up as a drive, follow the GEMMA M0 guide link above to prepare the board for CircuitPython.

```
# SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
#
```

```
# SPDX-License-Identifier: MIT

import array
from rainbowio import colorwheel
import board
import neopixel
from analogio import AnalogIn

led_pin = board.D0  # NeoPixel LED strand is connected to GPIO #0 / D0
n_pixels = 12  # Number of pixels you are using
dc_offset = 0  # DC offset in mic signal - if unusure, leave 0
noise = 100  # Noise/hum/interference in mic signal
samples = 60  # Length of buffer for dynamic level adjustment
top = n_pixels + 1  # Allow dot to go slightly off scale

peak = 0  # Used for falling dot
dot_count = 0  # Frame counter for delaying dot-falling speed
vol_count = 0  # Frame counter for storing past volume data

lvl = 10  # Current "dampened" audio level
min_level_avg = 0  # For dynamic adjustment of graph low & high
max_level_avg = 512

# Collection of prior volume samples
vol = array.array('H', [0] * samples)

mic_pin = AnalogIn(board.A1)

strip = neopixel.NeoPixel(led_pin, n_pixels, brightness=.1, auto_write=True)


def remap_range(value, leftMin, leftMax, rightMin, rightMax):
    # this remaps a value from original (left) range to new (right) range
    # Figure out how 'wide' each range is
    leftSpan = leftMax - leftMin
    rightSpan = rightMax - rightMin

    # Convert the left range into a 0-1 range (int)
    valueScaled = int(value - leftMin) / int(leftSpan)

    # Convert the 0-1 range into a value in the right range.
    return int(rightMin + (valueScaled * rightSpan))


while True:
    n = int((mic_pin.value / 65536) * 1000)  # 10-bit ADC format
    n = abs(n - 512 - dc_offset)  # Center on zero

    if n >= noise:  # Remove noise/hum
        n = n - noise

    # "Dampened" reading (else looks twitchy) - divide by 8 (2^3)
    lvl = int(((lvl * 7) + n) / 8)

    # Calculate bar height based on dynamic min/max levels (fixed point):
    height = top * (lvl - min_level_avg) / (max_level_avg - min_level_avg)

    # Clip output
    if height < 0:
        height = 0
    elif height > top:
        height = top

    # Keep 'peak' dot at top
    if height > peak:
        peak = height

        # Color pixels based on rainbow gradient
    for i in range(0, len(strip)):
        if i >= height:
            strip[i] = [0, 0, 0]
        else:
            strip[i] = colorwheel(remap_range(i, 0, (n_pixels - 1), 30, 150))

    # Save sample for dynamic leveling
    vol[vol_count] = n

    # Advance/rollover sample counter
    vol_count += 1

    if vol_count >= samples:
        vol_count = 0

        # Get volume range of prior frames
    min_level = vol[0]
    max_level = vol[0]

    for i in range(1, len(vol)):
        if vol[i] < min_level:
            min_level = vol[i]
        elif vol[i] > max_level:
            max_level = vol[i]

    # minlvl and maxlvl indicate the volume range over prior frames, used
    # for vertically scaling the output graph (so it looks interesting
    # regardless of volume level).  If they're too close together though
    # (e.g. at very low volume levels) the graph becomes super coarse
    # and 'jumpy'...so keep some minimum distance between them (this
    # also lets the graph go to zero when no sound is playing):
    if (max_level - min_level) < top:
        max_level = min_level + top

    # Dampen min/max levels - divide by 64 (2^6)
    min_level_avg = (min_level_avg * 63 + min_level) >> 6
    # fake rolling average - divide by 64 (2^6)
    max_level_avg = (max_level_avg * 63 + max_level) >> 6

    print(n)
```

This code requires the neopixel.py library. A factory-fresh board will have this already installed. If you've just reloaded the board with CircuitPython, create the "lib" directory and then download neopixel.py from Github ().

Wear It

Turn on the pendant by sliding on the switch. Test it by snapping your fingers or clapping. You can adjust the sensitivity of the pendant by playing around with the gain of the MIC (pot on the back of the MIC amp); you can also adjust the sensitivity in the Arduino sketch.

In order to charge the pendant, simply attach a micro USB cable to the LiPo charger.

arts