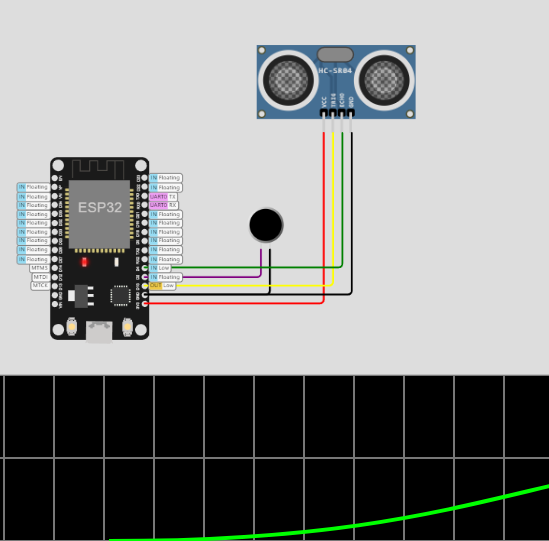
**NOISE POLLUTION MONITORING SYSYTEM**

**PHASE-5:**

****

***ESP32 Simulation***

*The ESP32 is a popular WiFi and Bluetooth-enabled microcontroller, widely used for IoT Projects. Wokwi simulates the ESP32, ESP32-C3, ESP32-S2, ESP32-S3, ESP32-C6 (beta), and ESP32-H2 (alpha).*

*VIN GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN 3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23ESP32*

***ESP32 boards***[***​***](https://docs.wokwi.com/guides/esp32#esp32-boards)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Chip*** | ***Description*** |
| *ESP32 DevKit v1* | *ESP32* | *Popular ESP32 development board* |
| *ESP32-S2-DevKitM-1* | *ESP32-S2* | *Entry-level ESP32-S2 development board* |
| *Franzininho WiFi* | *ESP32-S2* | *Board by the Franzininho Community* |
| *Wemos S2 mini* | *ESP32-S2* | *Small ESP32-S2 board by Wemos* |
| *ESP32-S3-DevKitC-1* | *ESP32-S3* | *Entry-level ESP32-S3 development board* |
| *ESP32-C3-DevKitM-1* | *ESP32-C3* | *Entry-level ESP32-C3 development board* |
| *Rust Board ESP32-C3* | *ESP32-C3* | *ESP32-C3 board designed for Rust trainings* |
| *ESP32-C6-DevKitC-1* | *ESP32-C6* | *Entry-level ESP32-C6 development board (beta)* |
| *ESP32-H2-DevKitM-1* | *ESP32-H2* | *Entry-level ESP32-H2 development board (alpha)* |

***Getting Started***[***​***](https://docs.wokwi.com/guides/esp32#getting-started)

*You can use the ESP32 simulator to run different kinds of applications:*

1. *ESP32 Arduino Core projects (including ESP-IDF projects)*
2. *MicroPython and CircuitPython projects*
3. *Rust projects*
4. *Custom application firmware files (e.g. applications built using the ESP-IDF)*

***Arduino Core***

*Start from the Arduino-ESP32 Project Template, or from the ESP32 Blink Example.If you want to use third-party Arduino libraries, add a libraries.txt file with the list of libraries that you use.*

***MicroPython***

*Start from the MicroPython ESP32 Project Template, or from the MicroPython ESP32 Blink Example.*

*Note: While the simulation is running, press Ctrl+C inside the Serial Terminal to get into the MicroPython REPL. Alternatively, you can edit the Blink Example code and remove the while loop. For more information, check out the MicroPython Guide.*

***Custom Application Firmware***

*Open the ESP32 custom application project template, and press "F1" in the code editor. Then choose "Upload Firmware and Start Simulation…". Choose any .bin, .elf or .uf2 file from your computer and the simulation will start.*

***Simulator Examples***

***Arduino Examples***

* *Blink*
* *Seven segment counter*
* *FastLED NeoPixel Blink*
* *WiFi Scanning*

***MicroPython Examples***

* *SSD1306 Example*
* *NeoPixels*
* *AES256 Encryption*
* *WiFi Scanning*
* *ESP-IDF Examples*

***The following examples use the ESP-IDF functions. They are compiled using Arduino ESP32 Core****:*

* *Blink using FreeRTOS API*
* *Binary LED counter using FreeRTOS tasks*
* *GPIO button input + interrupts*
* *WiFi Example*

***Simulation Features***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Peripheral* | *ESP32* | *S2* | *S3* | *C3* | *C6* | *Notes* |
| *Processor core(s)* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* |  |
| *GPIO* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* | *Interrupts supported* |
| *IOMUX* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* |  |
| *PSRAM* | *✔️* | *✔️* | *✔️* | *—* | *❌* | *4MB of external SRAM \** |
| *UART* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* |  |
| *USB* | *—* | *✔️* | *✔️* | *❌* | *❌* | *Support for UART over USB (CDC)* |
| *I2C* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* | *Master only. 10-bit addressing not supported.* |
| *I2S* | *❌* | *❌* | *❌* | *❌* | *❌* | *Open for voting* |
| *SPI* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* |  |
| *TWAI* | *❌* | *❌* | *❌* | *❌* | *❌* |  |
| *RMT* | *🟡* | *🟡* | *🟡* | *🟡* | *🟡* | *Transmit-only, use to control NeoPixels* |
| *LEDC PWM* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* | *Used by analogWrite(), Servo, Buzzer, etc.* |
| *MCPWM* | *❌* | *—* | *❌* | *—* | *❌* |  |
| *DMA* | *🟡* | *🟡* | *❌* | *❌* | *❌* |  |
| *WiFi* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* | *See the ESP32 WiFi Guide* |
| *Bluetooth* | *❌* | *—* | *❌* | *❌* | *❌* | *Open for voting* |
| *Timers* | *🟡* | *✔️* | *✔️* | *✔️* | *✔️* |  |
| *Watchdog* | *❌* | *❌* | *❌* | *❌* | *❌* |  |
| *RTC* | *🟡* | *🟡* | *🟡* | *🟡* | *🟡* | *Only RTC Pull-up / Pull-down resistors* |
| *ADC* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* | *Note: analogRead() returns values up to 4095* |
| *RNG* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* | *Random Number Generator* |
| *AES Accelerator* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* |  |
| *SHA Accelerator* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* |  |
| *RSA Accelerator* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* |  |
| *Hall Effect Sensor* | *❌* | *—* | *❌* | *—* | *—* |  |
| *ULP Processor* | *❌* | *❌* | *❌* | *—* | *✔️* |  |
| *GDB Debugging* | *✔️* | *✔️* | *✔️* | *✔️* | *✔️* | *Works with Wokwi for VS Code* |

***hc-sr04 Reference***

*HC-SR04 Ultrasonic Distance Sensor*

***Pin names***

|  |  |
| --- | --- |
| *Name* | *Description* |
| *VCC* | *Voltage supply (5V)* |
| *TRIG* | *Pulse to start the measurement* |
| *ECHO* | *Measure the high pulse length to get the distance* |
| *GND* | *Ground* |

***Attributes***

|  |  |  |
| --- | --- | --- |
| *Name* | *Description* | *Default value* |
| *Distance Initial distance value, in centimeters* | *"400"* |  |

***Operation***

*To start a new distance measurement set the TRIG pin to high for 10uS or more. Then wait until the ECHO pin goes high, and count the time it stays high (pulse length). The length of the ECHO high pulse is proportional to the distance. Use the following table to convert the ECHO pulse length in microseconds into centimeters / inches:*

|  |  |
| --- | --- |
| *Unit* | *Distance* |
| *Centimeters* | *PulseMicros / 58* |
| *Inches* | *PulseMicros / 148* |

***Setting the distance***

*To change the distance while the simulation is running, click on the HC-SR04 drawing in the diagram and use the slider to set the distance value. You can choose any value between 2cm and 400cm.*

***Arduino code example***

*#define PIN\_TRIG 3*

*#define PIN\_ECHO 2*

*void setup() {*

*Serial.begin(115200);*

*pinMode(PIN\_TRIG, OUTPUT);*

*pinMode(PIN\_ECHO, INPUT);*

*}*

*void loop() {*

*// Start a new measurement:*

*digitalWrite(PIN\_TRIG, HIGH);*

*delayMicroseconds(10);*

*digitalWrite(PIN\_TRIG, LOW);*

*// Read the result:*

*int duration = pulseIn(PIN\_ECHO, HIGH);*

*Serial.print("Distance in CM: ");*

*Serial.println(duration / 58);*

*Serial.print("Distance in inches: ");*

*Serial.println(duration / 148);*

*delay(1000);*

*}*

***Microphone:***

***What is a microphone?***

*A microphone is a device that translates sound vibrations in the air into electronic signals and scribes them to a recording medium or over a loudspeaker.Microphones enable many types of audio recording devices for purposes including communications of many kinds, as well as music vocals, speech and sound recording.Microphones can be standalone or embedded in devices such as headsets and telephones.*

***How do microphones work?***

*The most common type of microphone, the dynamic microphone, uses a coil suspended in a magnetic field that may be attached to multiple membranes for extended frequency response.Dynamic microphones use electrical energy in the form of induction to produce the audio signal. These microphones are well suited to stage performance.The microphone capsule contains a small diaphragm connected to a moving coil. When sound waves hit the diaphragm, it vibrates. This causes the coil to move back and forth in the magnet's field, generating an electrical current.However, how a microphone works ultimately varies depending on its designed purpose.One of the main considerations, aside from the type of device, is what is being recorded. The directionality of microphones is another consideration in microphone design.*

***Code Definition:***

*It then defines a variable for the Wi-Fi credentials, which is defined as wifi\_ssid and wifi\_password.The code then connects to the Wi-Fi network.It waits until the connection has been established.The code will connect to Wi-Fi, wait for it to be connected. This code takes advantage of the urequests library.The ujson library provides JSON parsing capabilities in Python.The code defines 2 variables called "ultrasonic\_trig" and "ultrasonic\_echo" which will be used to trigger the ultrasonic sensor and read from the ultrasonic sensor.*

*The code then defines a constant called "calibration\_constant" which represents how much time it takes for an echo to return after being triggered by an ultrasound pulse. It's set at 2 seconds and we can change it whenever need.There is a variable called "noise\_threshold", which sets the threshold of noise in dB (decibel).The code defines the pins for the ultrasonic sensor and microphone. The code also defines the calibration constant, noise threshold, and a variable.The code then uses machine.Pin(15, machine.Pin.OUT) to define a pin that is connected to an ultrasonic sensor.This pin is then used with machine.Pin(4, machine.Pin.IN) to define a pin that is connected to an analog-to-digital converter on digital input 4 .It will be used as an input for calculating sound levels from an audio source coming in.*

*The code is trying to measure the distance between two points.It will use a function called "analyze".The code then uses this function and some math to calculate how far away one point is from another point.def analyze(x1, y1, x2, y2): Analyzes two points on a 2D plane and returns their distance in meters as an integer value.It calculates the distance between two points in the world.*

*The code uses the Google Maps API to determine the latitude and longitude of two points, then calculates their distance using Pythagorean theorem.*

***Code:***

*import machine*

*import time*

*import urequests*

*import ujson*

*import network*

*import math*

*# Define your Wi-Fi credentials*

*wifi\_ssid = 'Wokwi-GUEST'*

*wifi\_password = '' # Replace with the actual Wi-Fi password*

*# Connect to Wi-Fi*

*wifi = network.WLAN(network.STA\_IF)*

*wifi.active(True)*

*wifi.connect(wifi\_ssid, wifi\_password)*

*# Wait for Wi-Fi connection*

*while not wifi.isconnected():*

*pass*

*# Define ultrasonic sensor pins (Trig and Echo pins)*

*ultrasonic\_trig = machine.Pin(15, machine.Pin.OUT)*

*ultrasonic\_echo = machine.Pin(4, machine.Pin.IN)*

*# Define microphone pin*

*microphone = machine.ADC(2)*

*calibration\_constant = 2.0*

*noise\_threshold = 60 # Set your desired noise threshold in dB*

*# Firebase Realtime Database URL and secret*

*firebase\_url = 'https://noise-pollution-bd0ab-default-rtdb.asia-southeast1.firebasedatabase.app/' # Replace with your Firebase URL*

*firebase\_secret = 'nBsgyQFTqHUe4qExlaZX6VL3mpf5gn6BlpnMiuR0' # Replace with your Firebase secret*

*def measure\_distance():*

*# Trigger the ultrasonic sensor*

*ultrasonic\_trig.value(1)*

*time.sleep\_us(10)*

*ultrasonic\_trig.value(0)*

*# Measure the pulse width of the echo signal*

*pulse\_time = machine.time\_pulse\_us(ultrasonic\_echo, 1, 30000)*

*# Calculate distance in centimeters*

*distance\_cm = (pulse\_time / 2) / 29.1*

*return distance\_cm*

*def measure\_noise\_level():*

*# Read analog value from the microphone*

*noise\_level = microphone.read()*

*noise\_level\_db = 20 \* math.log10(noise\_level / calibration\_constant)*

*return noise\_level, noise\_level\_db*

*# Function to send data to Firebase*

*def send\_data\_to\_firebase(distance, noise\_level\_db):*

*data = {*

*"Distance": distance,*

*"NoiseLevelDB": noise\_level\_db*

*}*

*url = f'{firebase\_url}/sensor\_data.json?auth={firebase\_secret}'*

*try:*

*response = urequests.patch(url, json=data) # Use 'patch' instead of 'put'*

*if response.status\_code == 200:*

*print("Data sent to Firebase")*

*else:*

*print(f"Failed to send data to Firebase. Status code: {response.status\_code}")*

*except Exception as e:*

*print(f"Error sending data to Firebase: {str(e)}")*

*try:*

*while True:*

*distance = measure\_distance()*

*noise\_level, noise\_level\_db = measure\_noise\_level()*

*print("Distance: {} cm, Noise Level: {:.2f} dB".format(distance, noise\_level\_db))*

*if noise\_level\_db > noise\_threshold:*

*print("Warning: Noise pollution exceeds threshold!")*

*# Send data to Firebase*

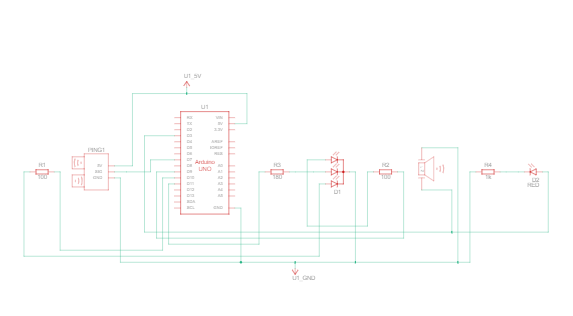
*send\_data\_to\_firebase(distance, noise\_level\_db)*

*time.sleep(1) # Adjust the sleep duration as needed*

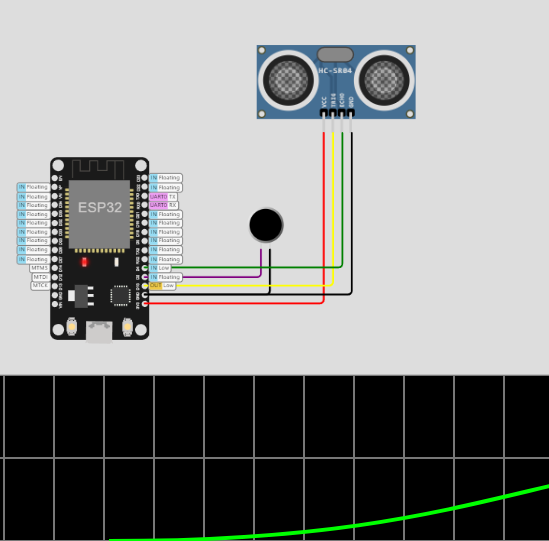
*except KeyboardInterrupt:*

*print("Monitoring stopped")*

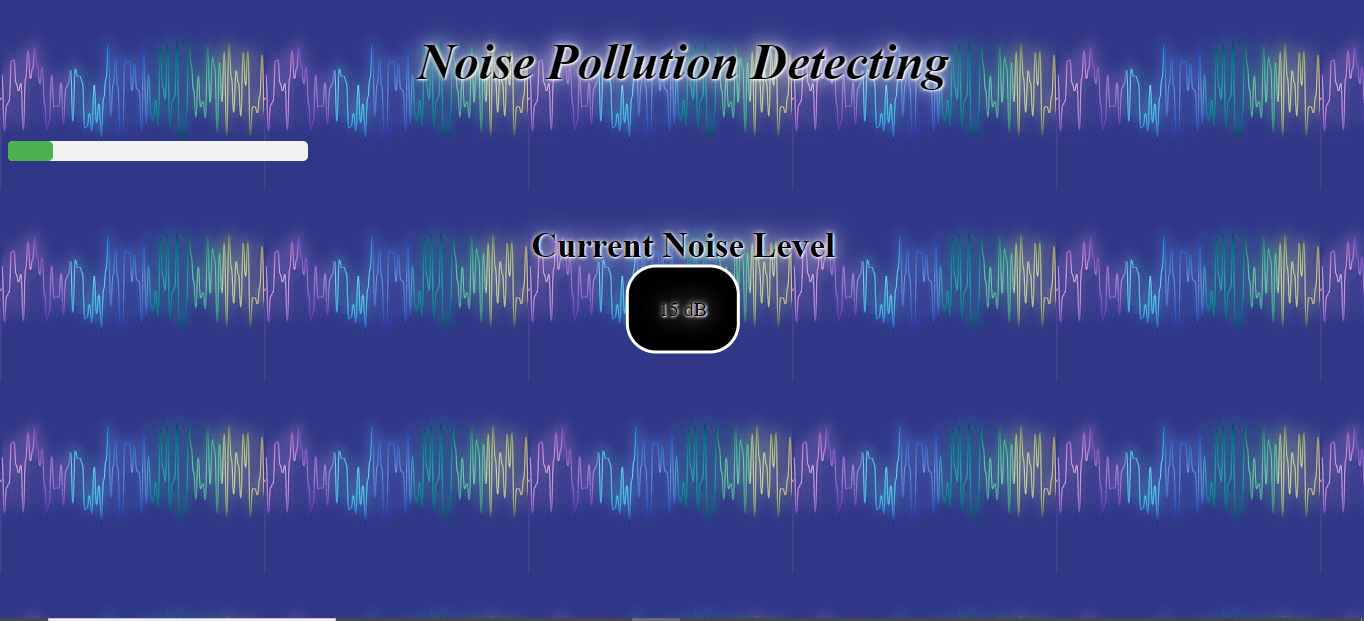
***Schematic Diagram:***

**

***IOT SENSOR :***

******

***Noise Pollution Information:***

**

***How the real-time noise level monitoring system promotes public awareness and contributes to noise pollution mitigation.***

***Real-time Data Collection and Display:*** *These systems consist of sensors strategically placed in different locations to continuously monitor noise levels. The data collected is then processed and displayed in real-time, often through online platforms and mobile apps. This accessibility to real-time information allows the public to stay informed about noise pollution in their area.*

***Increased Awareness:*** *By making noise pollution data easily accessible, individuals and communities become more aware of the noise levels in their surroundings. This awareness can be a catalyst for behavior change. When people see the noise levels spiking during certain times or in specific areas, they may be more motivated to take action to reduce noise, such as avoiding noisy activities during late hours or using quieter equipment.*

***Community Engagement:*** *Real-time noise level monitoring systems can encourage community engagement and collaboration. When residents and businesses are aware of the noise they are generating, they may be more inclined to work together to find solutions. For example, a community may collaborate to establish quiet hours or noise reduction initiatives.*

***Regulatory Compliance:*** *Noise monitoring systems can aid in enforcing noise regulations. When local authorities or regulatory agencies have access to real-time noise data, they can more effectively enforce noise ordinances and respond to noise complaints. This can lead to a reduction in noise pollution from businesses and construction activities.*

***Noise Complaint Resolution:*** *For individuals bothered by excessive noise, real-time monitoring systems can provide a streamlined process for reporting noise complaints. This ensures that noise issues are quickly addressed and resolved, contributing to a more peaceful and harmonious community.*

***Data-Driven Decision-Making:*** *Real-time noise data can inform urban planning and development. City planners can use this data to make decisions about zoning, transportation, and infrastructure projects. By considering noise levels, they can design cities and neighborhoods to minimize noise pollution.*

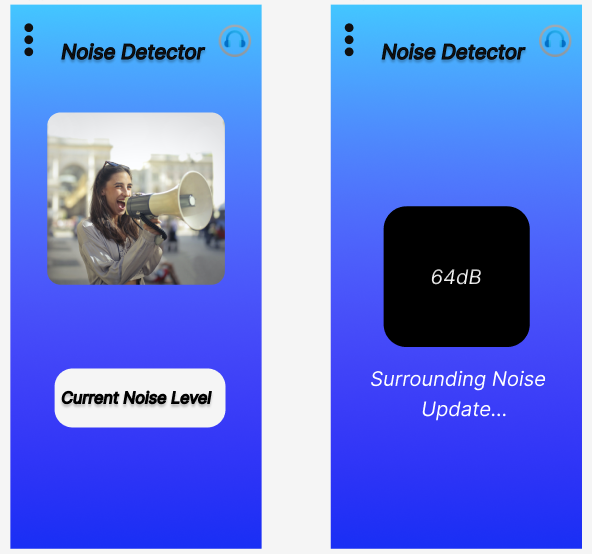
***Behavioral Change:*** *As individuals and businesses become more conscious of their noise impact, they may voluntarily change their behavior to reduce noise pollution. For instance, construction companies may choose quieter equipment, and residents may limit outdoor parties during late hours.*

***Education and Outreach:*** *Real-time noise monitoring systems can be integrated into educational programs and public awareness campaigns. Schools and organizations can use the data to teach students and the public about the importance of noise pollution and how it affects health and well-being.*

***Research and Analysis:*** *The data collected by these systems can also be used for research and analysis. Researchers can study noise patterns, identify hotspots, and develop targeted strategies for noise mitigation.*

*In summary, a real-time noise level monitoring system promotes public awareness by making noise data readily available and encourages behavior change by informing individuals and businesses about their noise impact. It also contributes to noise pollution mitigation by facilitating regulatory compliance, community engagement, and data-driven decision-making, ultimately leading to quieter, healthier, and more pleasant living environments.*

***Mobile Application Development:***

******

***Coding:***

*import random*

*class NoiseMonitorApp(tk.Tk):*

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

*super().\_\_init\_\_()*

*# Window configurations*

*self.title("Noise Level Monitor")*

*self.configure(bg="#F2F2F2")*

*# Header*

*header = tk.Label(self, text="Noise Pollution Detecting", font=("serif", 50, "italic"), bg="#F2F2F2", fg="black")*

*header.pack(pady=20)*

*# Noise level visual*

*self.noise\_level\_frame = tk.Frame(self, bg="#F2F2F2", height=20, width=300)*

*self.noise\_level\_frame.pack\_propagate(False)*

*self.noise\_indicator = tk.Frame(self.noise\_level\_frame, bg="#4CAF50", height=20, width=0)*

*self.noise\_indicator.pack(side="left")*

*self.noise\_level\_frame.pack(pady=50)*

*# Current noise level text*

*noise\_text\_label = tk.Label(self, text="Current Noise Level", font=("Arial", 30), bg="#F2F2F2", fg="black")*

*noise\_text\_label.pack(pady=30)*

*# Display noise in dB (initially set to "0 dB")*

*self.noise\_text = tk.Label(self, text="0 dB", font=("Arial", 20), bg="black", fg="white", padx=30, pady=30)*

*self.noise\_text.pack(pady=10)*

*# Start the noise level updating process*

*self.update\_noise\_level()*

*def update\_noise\_level(self):*

*random\_noise\_level = random.randint(0, 100)*

*self.noise\_indicator["width"] = random\_noise\_level \* 3 # Scale to match the width (300 pixels)*

*self.noise\_text.config(text=f"{random\_noise\_level} dB")*

*self.after(1000, self.update\_noise\_level) # Call this function every second*

*if \_\_name\_\_ == "\_\_main\_\_":*

*app = NoiseMonitorApp()*

*app.mainloop()*