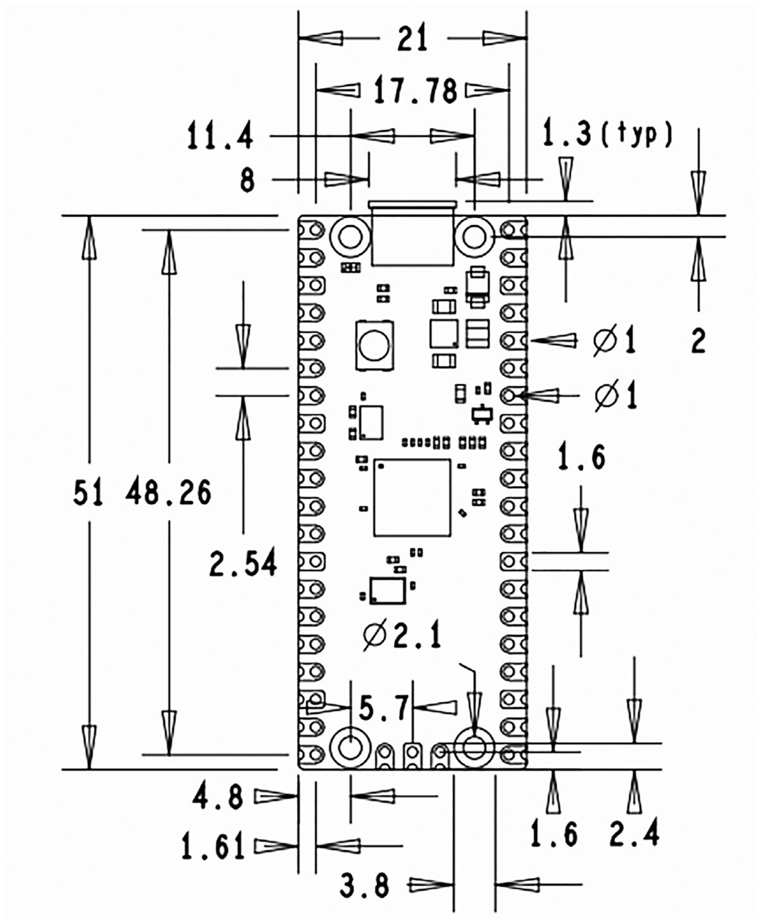
Published November 2021

Philadelphia, Pennsylvania

Version 0.95

Download this zine as a PDF:
https://iffybooks.net/noisemusic

Make noise music with MicroPython and a Pi Pico



a how-to guide from Iffy Books

IFFY BOOKS

319 N. 11th St. #3D
Philadelphia, PA 19107

Join our email list
at iffybooks.net

Follow @iffybooks
on social media

Send corrections to
iffybooks@protonmail.com

The Raspberry Pi Pico is a cheap, versatile microcontroller you can use for DIY projects. MicroPython is a language you can use to program it.

Playing **noise music** at a DIY show (ideally in someone's basement) is a tried-and-true strategy for participating in music culture without really knowing how to play an instrument. With noise music, it's OK if there isn't any discernible key or tempo. You don't need to write lyrics or even make eye contact with the audience! You just need one thing: a weird little sound gadget (other than a computer/phone) that you can connect to your friend's roommate's PA system.

First, this zine will show you how to connect your Raspberry Pi Pico to an audio jack or small speaker using a **GPIO pin** and a **ground pin**. Then you'll use **MicroPython** to generate audio, which will sound grainy like an old video game.

You'll use an application called **Thonny** to write your MicroPython code and program your Pi Pico. Thonny is available for macOS, Windows, and Linux, and it's easy to install. Even if you've never programmed before, you can copy the provided code and start making modifications.

➡️ Get to know your Raspberry Pi Pico

Unlike other Raspberry Pi devices, the Raspberry Pi Pico isn't a complete desktop computer. Instead, it runs the same program (written in MicroPython, C, or C++) every time it turns on. And you won't need a microSD card to use it, since it has 2 MB of internal storage.

The Pi Pico is built around the RP2040, a 133 MHz microcontroller chip designed by Raspberry Pi in the UK. It has 40 input-output pins, which includes 26 multi-function **GPIO** (**general purpose input-output**) pins and 8 **ground** pins. Each pin is labeled on the back of the board.

A GPIO pin works like a tiny switch. At any given time, it can be **on** (high voltage) or **off** (low voltage). Because audio signals require smooth waves, your Rasperry Pi Pico will use a technique called **pulse-width modulation** (**PWM**). Using PWM, your Pi Pico will turn the GPIO pin on and off very quickly, allowing it to simulate a recognizable audio signal.

Your Raspberry Pi Pico has a micro USB port for power and data connections. The USB port typically supplies 5 volts (V), but you can use any voltage from 1.8V to 5.5V. The board's internal circuitry uses 3.3V.

The Pi Pico has a single button labeled BOOTSEL, which you’ll use when you install MicroPython on the board for the first time. **MicroPython** is a slightly modified version of the **Python** programming language designed for microcontrollers.

➡️ More Raspberry Pi Pico audio projects

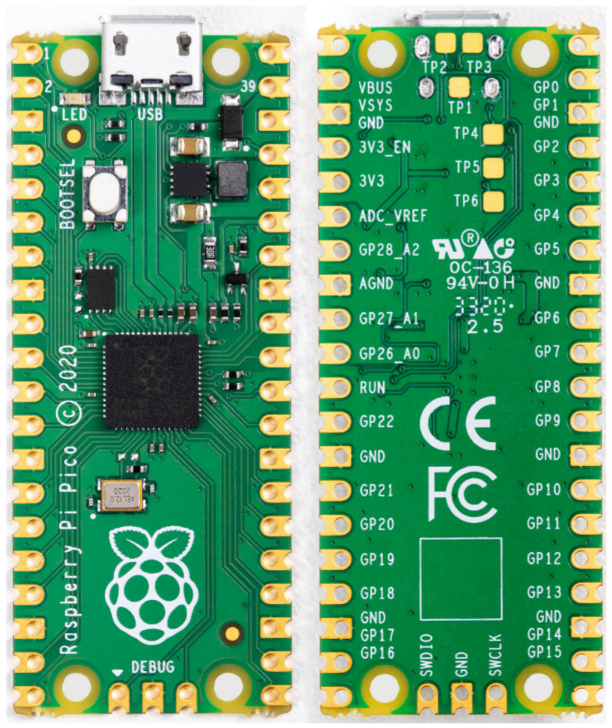
If you want to generate tonal melodies, you may find the example code in this tutorial helpful:

"How to Use a Buzzer to Play Music with Raspberry Pi Pico" by Avram Piltch
https://www.tomshardware.com/how-to/buzzer-music-raspberry-pi-pico

To generate chords and arpeggios, check out the **buzzer_music** library:
https://github.com/james1236/buzzer_music

In the ideal case, you would run your Pi Pico's audio output into a **mixer**, which would then connect to the PA via XLR cables. To connect your Pi Pico to a mixer, you'll likely need a 1/8"-to-RCA adapter like the one below. (You might even be able to plug RCA cables directly into the PA.)

You probably shouldn't connect your Pi Pico directly to a guitar amp, because the higher-than-expected voltage might damage its circuitry. But if you don't have another option, using a lower number in the **buzzer.duty_u16()** function could potentially make it a little safer. If you do go this route, you should plug your Pi Pico into a guitar pedal first to limit the potential damage.



➡️ Install Thonny

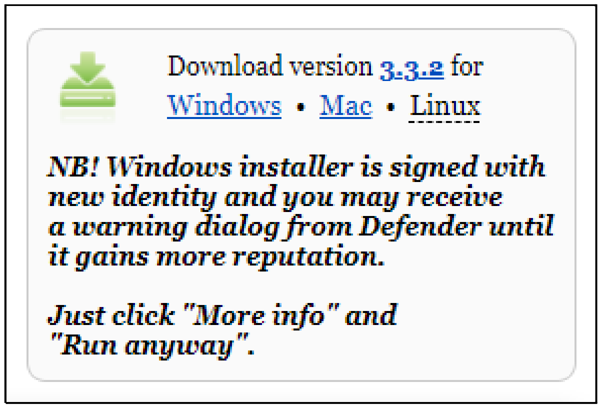
Note: This step and the following one are adapted from the article "Getting started with Raspberry Pi Pico," which is published under a Creative Commons license: https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico

On Windows, macOS, and Linux, you can install the latest Thonny IDE or update an existing version.

In a web browser, navigate to thonny.org.

In the top right-hand corner of the browser window, you will see download links for Windows and macOS, and instructions for Linux.

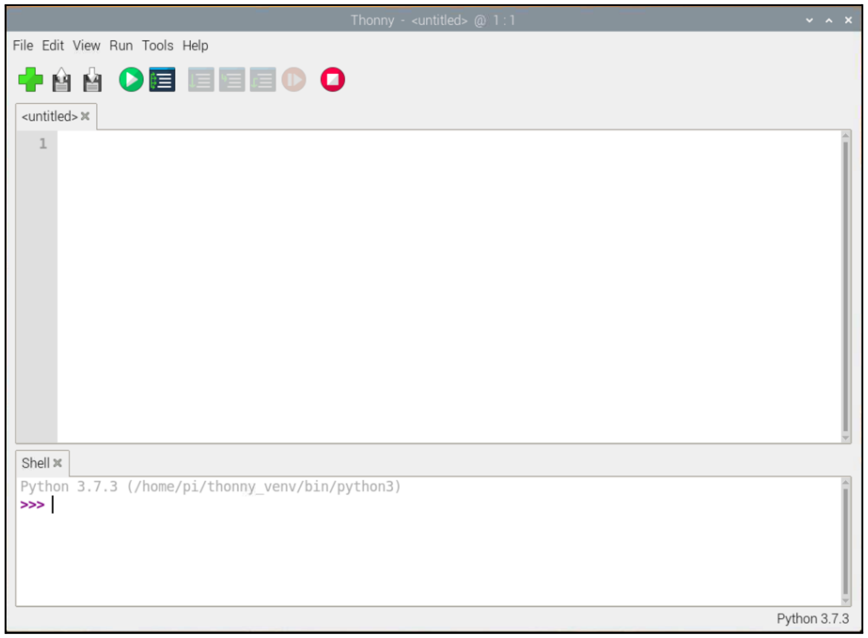
Download the relevant files and run them to install Thonny.


Download version 3.3.2 for
Windows • Mac • Linux

NB! Windows installer is signed with new identity and you may receive a warning dialog from Defender until it gains more reputation.

Just click "More info" and "Run anyway".

Open Thonny from your application launcher. It should look something like this:



```
Python 3.7.3 (/home/pi/thonny_venv/bin/python3)
>>> 
```

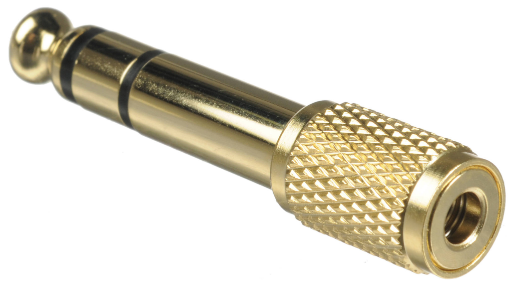
You’re now ready to move on to the next step and connect your Raspberry Pi Pico.

➡️ Connect your Pi Pico to an amplifier

It's important to **turn down the volume** on your audio equipment before testing your Pi Pico noise machine. Start playing some audio on your Pi Pico, then slowly turn up the volume knob on the PA/amplifier/stereo until you get to the right level. It's easy to make a loud noise by mistake and hurt your ears.

Also, you should wear **ear plugs** if you'll be listening to loud music for a long time. 🙂

You'll need some kind of cable to connect your device's 3.5mm audio jack to a PA system. If the PA has 1/4" unbalanced line-level inputs, you can use a 1/8"-to-1/4" adapter cable. Or just use a 1/8" audio cable and an adapter like this:



➡️ Trigger sounds using multiple pins

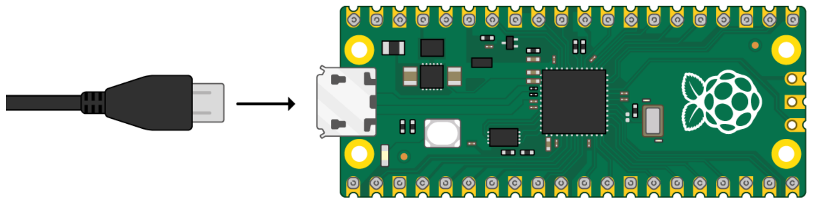
The code below uses **if** and **elif** conditional statements to play different sounds when you touch your wire stylus to the pins GP2, GP3, or GP4.

```
from machine import Pin, PWM
from utime import sleep
import random

# Set GPIO pin for audio output
buzzer = PWM(Pin(15))

def play_tone(frequency):
    # Set maximum volume
    buzzer.duty_u16(1000)
    # Play tone
    buzzer.freq(frequency)

def be_quiet():
    # Set minimum volume
    buzzer.duty_u16(0)

## Set GPIO pins to use for switches
switch_1 = Pin(2, Pin.IN, Pin.PULL_DOWN)
switch_2 = Pin(3, Pin.IN, Pin.PULL_DOWN)
switch_3 = Pin(4, Pin.IN, Pin.PULL_DOWN)

## Infinite loop
while True:
    ## Continuous tone
    if switch_1.value():
        # Set tone frequency
        play_tone(65)

    ## Fast falling tone
    elif switch_2.value():
        # Set a random starting frequency
        start_value = random.randint(150,1450)
        # Play falling frequencies
        for i in range(48):
            play_tone(start_value - i)
            sleep(0.001)

    ## Falling and rising glide
    elif switch_3.value():
        # Set a random starting frequency
        start_value = random.randint(120,220)
        # Play repeatedly falling frequencies
        for i in range(-90,90):
            play_tone(start_value + abs(i))
            sleep(0.008)

    else:
        be_quiet()
        sleep(0.05)
```

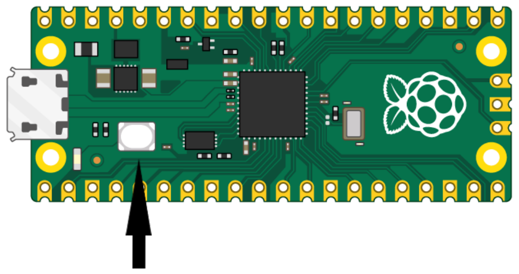
➡️ Add the MicroPython firmware

If you have never used MicroPython on your Raspberry Pi Pico, you will need to add the MicroPython firmware.

Plug your micro USB cable into the port on the board.



Find the BOOTSEL button on your Raspberry Pi Pico.

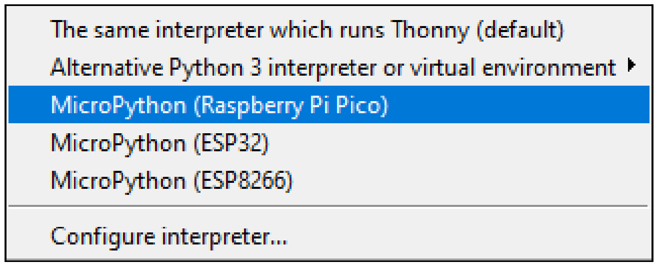


Press the BOOTSEL button and hold it while you connect the other end of the micro USB cable to your computer. A Raspberry Pi is shown in the image below, but the same applies to any computer.

This puts your Raspberry Pi Pico into USB mass storage device mode.

In the bottom right-hand corner of the Thonny window, you will see the version of Python that you are currently using.

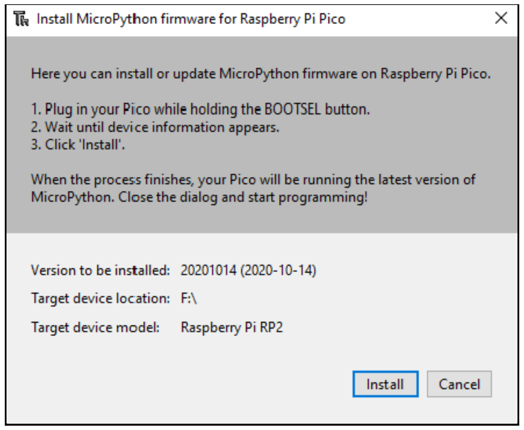
Click on the Python version and choose 'MicroPython (Raspberry Pi Pico)':


The same interpreter which runs Thonny (default)
Alternative Python 3 interpreter or virtual environment
MicroPython (Raspberry Pi Pico)
MicroPython (ESP32)
MicroPython (ESP8266)
Configure interpreter...

If you don't see this option, then check that you have plugged in your Raspberry Pi Pico.

A dialog box will pop up to install the latest version of the MicroPython firmware on your Raspberry Pi Pico.

Click the **Install** button to copy the firmware to your Raspberry Pi Pico.



Install MicroPython firmware for Raspberry Pi Pico

Here you can install or update MicroPython firmware on Raspberry Pi Pico.

1. Plug in your Pico while holding the BOOTSEL button.
2. Wait until device information appears.
3. Click 'Install'.

When the process finishes, your Pico will be running the latest version of MicroPython. Close the dialog and start programming!

Version to be installed: 20201014 (2020-10-14)

Target device location: F:\

Target device model: Raspberry Pi RP2

Wait for the installation to complete and click **Close**.

➡️ Create a falling and rising glide

The code below uses a for loop with the function **range(-90,90)** to loop through each integer from -90 to positive 89. It then uses the function **abs(i)** to convert the current number to a positive value and adds it to the random starting frequency. The **play_tone()** function plays the tone, and the **sleep()** function pauses for 0.00 seconds while it plays.

```
from machine import Pin, PWM
from utime import sleep
import random

# Set GPIO pin for audio output
buzzer = PWM(Pin(15))

def play_tone(frequency):
    # Set maximum volume
    buzzer.duty_u16(1000)
    # Play tone
    buzzer.freq(frequency)

def be_quiet():
    # Set minimum volume
    buzzer.duty_u16(0)

## Set GPIO pins to use for switches
switch = Pin(2, Pin.IN, Pin.PULL_DOWN)

## Infinite loop
while True:
    ## Falling and rising glide
    if switch.value():
        # Set a random starting frequency
        start_value = random.randint(120,220)
        # Play repeatedly falling frequencies
        for i in range(-90,90):
            play_tone(start_value + abs(i))
            sleep(0.008)
    else:
        be_quiet()
```

➡️ Create a fast descending tone

To make a tone that falls in frequency, you can use a **for loop**. The example code below starts with a random frequency value. Then it uses a for loop with the **range()** function to iterate through a series of numbers from zero to 47. The variable **i** represents the current number in the series. For each iteration of the for loop, the code below subtracts the number **i** from the starting frequency and plays a tone at that frequency for 0.001 seconds. Try adjusting the number in the **sleep()** function and see how it changes the sound.

```
from machine import Pin, PWM
from utime import sleep
import random

# Set GPIO pin for audio output
buzzer = PWM(Pin(15))

def play_tone(frequency):
    # Set maximum volume
    buzzer.duty_u16(1000)
    # Play tone
    buzzer.freq(frequency)

def be_quiet():
    # Set minimum volume
    buzzer.duty_u16(0)

## Set GPIO pins to use for switches
switch = Pin(2, Pin.IN, Pin.PULL_DOWN)

## Infinite loop
while True:
    ## Fast descending tone
    if switch.value():
        # Set a random starting frequency
        start_value = random.randint(150,1450)
        # Play falling frequencies
        for i in range(48):
            play_tone(start_value - i)
            sleep(0.001)
    else:
        be_quiet()
```

➡️ Connect an audio jack or speaker to your Pi Pico

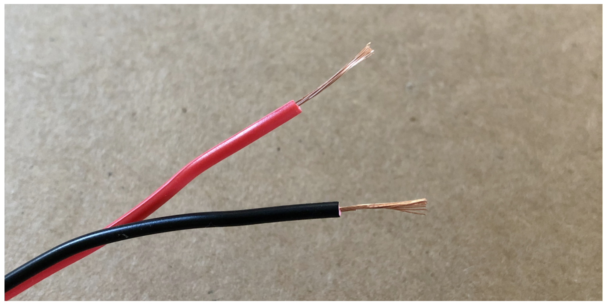
To send audio from your Pi Pico to an audio jack or speaker, you'll need to connect wires to two pins: a GPIO pin and a ground pin (labeled GND). The GPIO pin will provide electrical current with an oscillating voltage level, which your speaker will convert into sound.

Here are some possible ways to connect a speaker to your Pi Pico:

- use a piezo buzzer
- use a quarter-sized ~0.25 Watt speaker
- use a pair of headphones connected with alligator clips
- use an audio jack connected to headphones, a powered speaker, a PA system

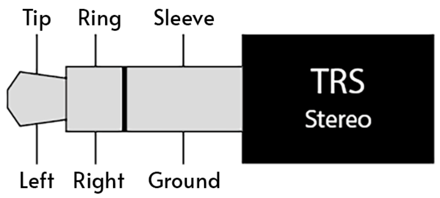
This tutorial will focus on connecting an **audio jack** to your Pi Pico, but you can follow the same steps if you're using a speaker.

First, you'll need to strip the ends of your wires. Use wire strippers (or scissors) to remove the insulation from the end of each wire, leaving about half an inch of exposed wire. Gently twist the strands of each wire together to keep them from fraying.

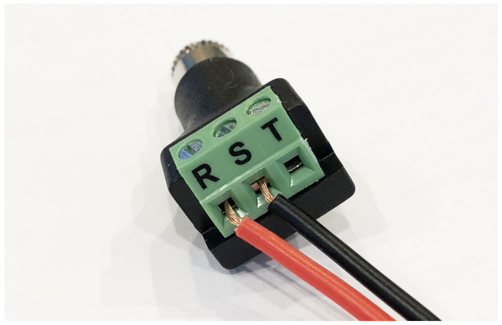


The example project uses red and black wires. The red wire will be the **signal wire**, and the black wire will be the **ground wire**. The signal wire will connect to a GPIO pin on your Pi Pico, and will carry the audio signal to the speaker. The ground wire will connect to a ground pin, letting electricity flow back into the Pi Pico. (It doesn't matter what wire colors you use, but you should decide which is which.)

Here's what a **tip ring sleeve** (**TRS**) stereo audio plug looks like. It has three metal contacts: The tip carries the audio signal for the left channel, and the ring carries the signal for the right channel. The sleeve is the ground.



If you’re using an audio jack like the one below, connect your ground wire to the **sleeve** (**S**) terminal and **tighten the screw** to secure the wire. If you're using an audio jack without screws, use the terminal that will connect to an audio plug’s sleeve (the metal contact closest to the base of the plug).



```
## Infinite loop
while True:
  ## Continuous tone
  if switch_1.value():
    # Set tone frequency
    play_tone(65)
  else:
     be_quiet()
```

Here's what your updated script will look like (new code in bold):

```
from machine import Pin, PWM
from utime import sleep
import random

# Set GPIO pin for audio output
buzzer = PWM(Pin(15))

def play_tone(frequency):
    # Set maximum volume
    buzzer.duty_u16(1000)
    # Play tone
    buzzer.freq(frequency)

def be_quiet():
    # Set minimum volume
    buzzer.duty_u16(0)

## Set GPIO pin to use for a switch
switch = Pin(2, Pin.IN, Pin.PULL_DOWN)

## Infinite loop
while True:
    ## Continuous tone
    if switch.value():
        # Set tone frequency
        play_tone(65)
    else:
        be_quiet()
```

```
## Infinite loop
while True:
    frequency_value = random.randint(50,3000)
    play_tone(frequency_value)
    sleep(0.75)
    be_quiet()
    sleep(0.75)
```

➡️ Use a wire as a stylus to trigger a tone

Strip both ends of a piece of wire. Insert one end into the **3v3** pin hole and secure it in place with a toothpick, or solder it if you wish. To trigger a tone, you'll touch the other end of the wire to a GPIO pin of your choice.

Before the while loop in your program, add the following line of code to set up the GPIO pin you'll use for your switch. The example uses pin 2 (labeled **GP2** on your board).

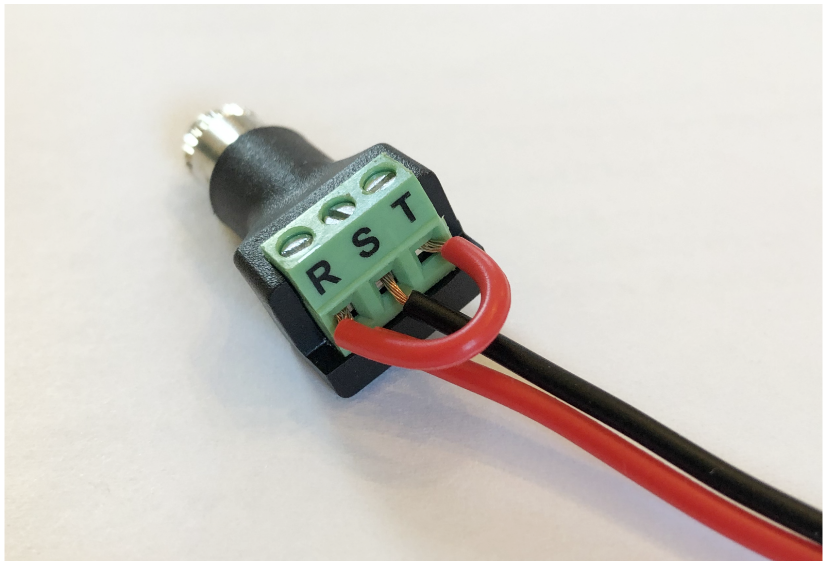
```
## Set GPIO pin to use for switches
switch_1 = Pin(2, Pin.IN, Pin.PULL_DOWN)
```

Next you'll use an **if ... then** statement to check whether the jumper cable is connected to the chosen GPIO pin. When your jumper cable is connected, the method **switch_1.value()** will return **True**. Otherwise it will return **False** and the **be_quiet()** function will run instead.

Update your while loop to match the example below. When the jumper wire is connected to GP2, the **play_tone(65)** function will play a 65 Hz tone. When the jumper cable is disconnected, the **be_quiet()** function will stop the sound.

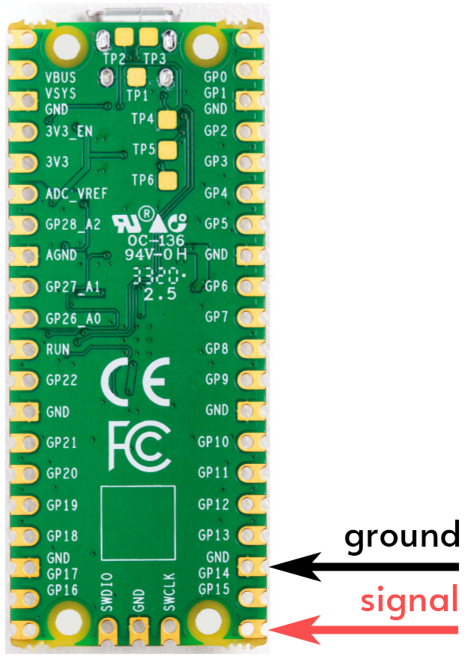
If you're using a stereo audio jack, you can connect the signal wire to the **tip** (**T**) terminal to play audio only in the left channel. Or you can connect it to the **ring** (**R**) terminal to play through the right channel.

To play your audio through left and right channels simultaneously, you can connect the tip and ring terminals with a piece of wire.



Tighten the remaining terminal screws to secure the wires in place.

Next you'll connect the **signal wire** (red) from your audio jack or speaker to the pin labeled **GP15**. And you'll connect your ground wire to a pin labeled **GND** for ground.



This tutorial uses **GPIO pin 15** for the **audio signal**. If you're looking at the back of your Pi Pico, **GP15** is at the bottom right corner. There's a ground pin two spaces away. (Note that all the ground pins have square outlines, while the rest are round.)

> *Note:* You can use any GPIO pin you want, as long as the pin number matches your code. It never matters which ground pin you use.

Insert your **ground wire** into a ground pin hole (marked **GND**). To make sure the wire stays connected, try wedging a round toothpick into the hole alongside the wire. You can also use masking tape to keep it in place

In the Thonny menu bar, select **File > Save**. Click **Save to Raspberry Pi Pico**, then save your file with the name **main.py**. If you use a different filename, your program won't run when you plug in your Pi Pico.

Unplug your Pi Pico from your computer and then plug it back in (or try using a different power source). As soon as you plug it in, it should run your program and start beeping.

➡️ Make your Pi Pico beep at random frequencies

To make your program beep at random frequencies,

Add the line **import random** to the beginning of your program to import the **random** module. You can then use the **random.randint()** function to select random frequency values.

The sample code below uses **random.randint(50,3000)** to select a random frequency from 50 Hz to 3000 Hz.

```
from machine import Pin, PWM
from utime import sleep
import random

# Set GPIO pin for audio output
buzzer = PWM(Pin(15))

def play_tone(frequency):
   # Set maximum volume
   buzzer.duty_u16(1000)
   # Play tone
   buzzer.freq(frequency)

def be_quiet():
   # Set minimum volume
   buzzer.duty_u16(0)

# Continued on the next page ...
```

```
## Infinite loop
while True:
    play_tone(65)
    sleep(0.75)
    be_quiet()
    sleep(0.75)
```

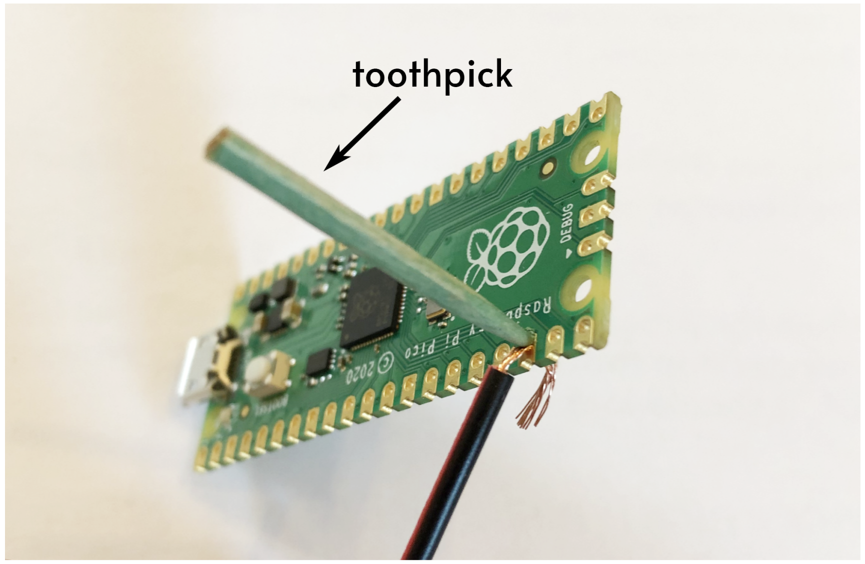
A tone with a frequency of 65 Hz contains 65 sound pressure oscillations per second. For reference, keys on a piano range from ~28 Hz to ~4186 Hz, with middle C at ~262 Hz.

Here's what your script will look like when you're done:

```
from machine import Pin, PWM
from utime import sleep

# Set GPIO pin for audio output
buzzer = PWM(Pin(15))

def play_tone(frequency):
    # Set maximum volume
    buzzer.duty_u16(1000)
    # Play tone
    buzzer.freq(frequency)

def be_quiet():
    # Set minimum volume
    buzzer.duty_u16(0)

## Infinite loop
while True:
    play_tone(65)
    sleep(0.75)
    be_quiet()
    sleep(0.75)
```

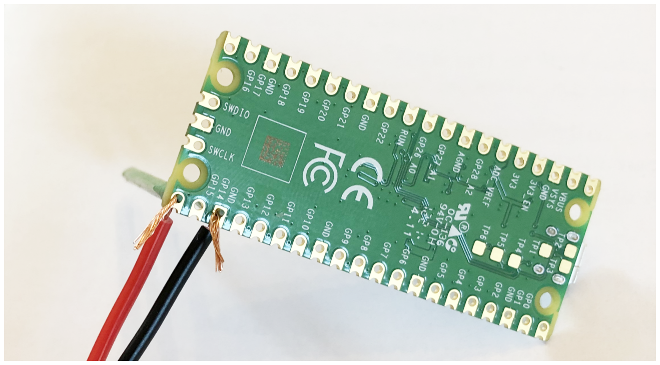
Click the **Play** icon at the top of the Thonny window to run your program on your Pi Pico. You should hear beeping!

(not very reliable), or use solder if you want the connection to be permanent.



Note: If you're connecting a speaker directly to your Pi Pico and you aren't sure which wire is which, don't stress too much. It will probably work either way.

Once your ground wire is connected, you'll connect the **signal wire** (red in the example) to **GP15**. Insert the wire in the hole and wedge it in with another toothpick. When you're done, your board should look like this:



Make sure the ends of your wires aren't touching each other, or any other pins.

Now you can use a 1/8" (a.k.a. 3.5mm) audio cable to plug your Pi Pico into headphones, a powered speaker, or a PA system

➡️ Make your Pi Pico beep like an alarm

> ***Important note:*** If you're using a powered speaker, start with the volume low and turn it up slowly while your Pi Pico is playing sound. If you're using headphones, test the volume away from your ears first.

Plug your Raspberry Pi Pico into your computer with a micro USB cable.

Open **Thonny** and you'll see a blank file where you'll write your code. You can transcribe the code examples by hand if you want, or find the code at the following URL:

https://iffybooks/noisemusic

At the beginning of your program, you'll import the **Pin** and **PWM** functions from the **machine** module. Then you'll import the **sleep** function from **utime**.

```
from machine import Pin, PWM
from utime import sleep
```

Then you'll create a variable called **buzzer** that you'll use to control your audio output. The code below specifies GPIO pin 15 for audio output.

```
# Set GPIO pin for audio output
buzzer = PWM(Pin(15))
```

> *Note:* In MicroPython, lines that begin with the pound sign (#) are comments, not code. You can safely omit those lines, or write your own comments.

Next you'll define two functions: one for noise and one for silence. The first is called **play_tone()**, which takes a value called **frequency** as input. First, the **buzzer.duty_u16(1000)** function sets the volume to the maximum level. Then the **buzzer.freq(frequency)** function sets the frequency of the tone in **hertz** (**Hz**).

```
def play_tone(frequency):
    # Set maximum volume
    buzzer.duty_u16(1000)
    # Play tone
    buzzer.freq(frequency)
```

> *Note:* When you're writing MicroPython code, indentations matter. Use spaces or tabs (either are fine) to indent each line of code to match the examples.

Your next function, called **be_quiet()**, plays silence. It simply uses the function **buzzer.duty_u16(0)** to set the volume to the minimum level.

```
def be_quiet():
    # Set minimum volume
    buzzer.duty_u16(0)
```

Next you'll create an infinite **while loop**, which will repeatedly run the block of code beneath. In the example code, the **play_tone(65)** function will play a 65 Hz tone. The function **sleep(0.75)** will pause for 0.75 seconds while the tone plays. Then the **be_quiet()** function will silence the tone, and the **sleep(0.75)** function will pause for another 0.75 seconds.