Projects

Sensory gadget

Create a sensory gadget using your Raspberry Pi Pico

Step 1 You will make

Make a fidget toy or sensory gadget. Your gadget will need to meet the **project brief**.

> A sensory gadget is something that makes you want to keep interacting with it. A fidget toy is a type of sensory gadget that helps the user relieve stress or improve their concentration. An adaptive gadget can be used by people with physical disabilities for communication. A sensory gadget might stimulate all the senses or just focus on one.

You will:

- Use your digital making skills to design and make a gadget for a user
- Use physical inputs such as buttons and potentiometers to control physical outputs such as LEDs and a buzzer
- Let others try our your gadget and improve it based on their feedback

An assistive gadget A user can select an option to let their carer know of their current need. Once they have made a selection, they press another button that alerts their carer.

PROJECT BRIEF: Sensory gadget

Make a sensory gadget that people will want to use.

Your sensory gadget should:

- Have multiple different kinds of input
- Have multiple different outputs
- Be appealing to the user and robust enough to be used

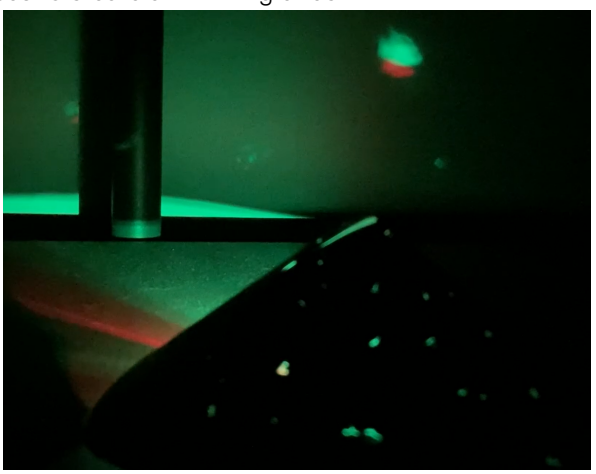
Your sensory gadget could:

- Take ergonomics like user comfort into consideration
- Reset on user input or after a set amount of time
- Connect to a specific theme

> Ergonomics is a science that seeks to overcome problems and improve how humans can interact with their environment. Improving the ergonomics of a gadget will make it easier to use and more comfortable to interact with.

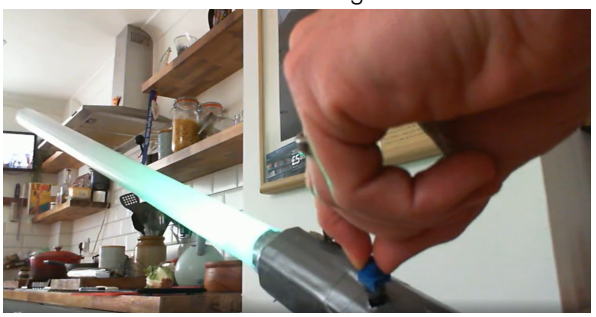
An assistive gadget A user can select an option to let their carer know of their current need. Once they have made a selection, they press another button that alerts their carer.

The night sky Tiny holes have been poked through a piece of black card to make a starry night effect on a ceiling in a dark room. An RGB LED pulses to create a twinkling effect.



The buzzy bee The bee's wings have kitchen foil on the back of them and when pressed down they connect to another piece of foil on the card — this makes a buzzer play a note. Each wing plays a different sound. A potentiometer controls a blue LED on the bee's tail.

Picosaber Pressing the button lights the blade of the saber and starts the buzzers making a humming sound. Turning the potentiometer changes the colour of the blade and pitch of the hum. Turning the potentiometer all the way down plays a 'power-down sound' then switches off the lights and buzzers.



Digital candle The RGB LED is on a loop that appears like a flickering flame. Blowing on the candle causes a foil contact to another contact on the candle and stop the loop. After a while, the loop restarts.

Step 2 Your idea

Use this step to plan your sensory gadget. You can plan by just thinking, tinkering, drawing or writing, or however you like!

Why are you making your sensory gadget?

Think about the purpose of your sensory gadget.

It could be:

- For a younger sibling to learn about sights and sounds
- A way of relieving tension through pressing buttons and hearing sounds
- A communication tool to help people express their needs

Who is it for?

Think about who you will make your sensory gadget for (your **audience**).

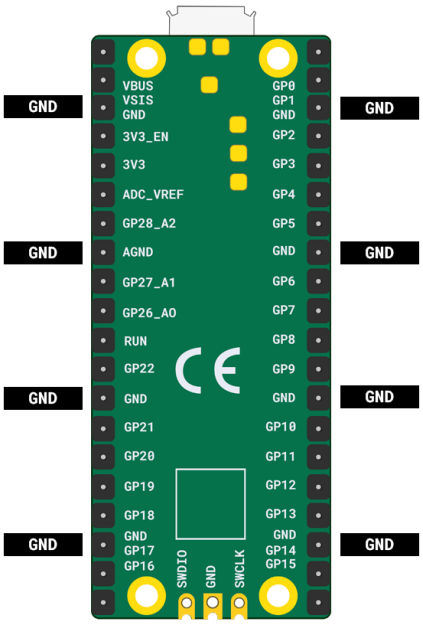
It could be for a friend, for a family member, for a school class, for people who share a hobby, for fans of a TV programme or musician, or just for yourself.

What features will your gadget have?

Think about how many components your gadget will need.

Component limitations of the Raspberry Pi Pico

The Raspberry Pi Pico has eight **GND** pins so when you are using jumper wires, you can only have eight components unless some share a **GND** pin.



Speakers can only play one note at a time so you will need multiple speakers if you want to play multiple sounds at the same time.

There is only one **3V** pin so you can only use one potentiometer. There is also a limit to how much current the Raspberry Pi Pico can supply.

Suggested combinations of input and outputs are:

- 1 potentiometer and 1 buzzer
- 4 buttons and buzzer
- 8 crafted buttons and a buzzer
- 1 potentiometer, 2 buttons and two buzzers
- Multiple buttons and a matching number of buzzers to play chords (multiple notes at the same time)

You **can** use more components than **8** but this will involve sharing a **GND** pin.

Think about the types of inputs and outputs you will have.

You gadget could:

- Have push buttons for inputs
- Use crafted switches
- Use a dial input using a potentiometer
- Play a specific sound
- Play a tune, or several tunes
- Use single-coloured LEDs
- Use an RGB LED

Think about what your sensory gadget will look like.

It could:

- Be based on a sensory gadget that already exists like a fidget cube or popper
- Have a theme that is based on your favourite comic, TV show, or song
- Be a crafted enclosure made from an old cardboard box, a fabric material, or a plastic container

Get started

Gather the components that you will need to make your sensory gadget. You will need inputs, outputs, jumper wires, and your Raspberry Pi Pico.

Test: Connect your Raspberry Pi Pico to your computer and check that it works by blinking the onboard LED.

Here is some example code for blinking the onboard LED:

```
from picozero import pico_led
from time import sleep

pico_led.on()
sleep(1)
pico_led.off()
```

If you have not already prepared your inputs and outputs, and need to remind yourself of how to connect LEDs to resistors and jumper wires, visit our **Introduction to the Pico** (https://projects.raspberrypi.org/en/projects/introduction-to-the-pico) guide.

Step 3 Build and test

Now it's time to make your sensory gadget.

> Your sensory gadget is your own design and may have a **different combination** of components to the ones suggested in the diagrams below. If a pin is already in use, then you will need to select a different pin and ground to use. Make sure that you **note down** which pin is in use for when you are writing your code.

You have built up some really useful skills. Here is a reminder to help you make your sensory gadget:

Connect your outputs

Sharing a ground pin

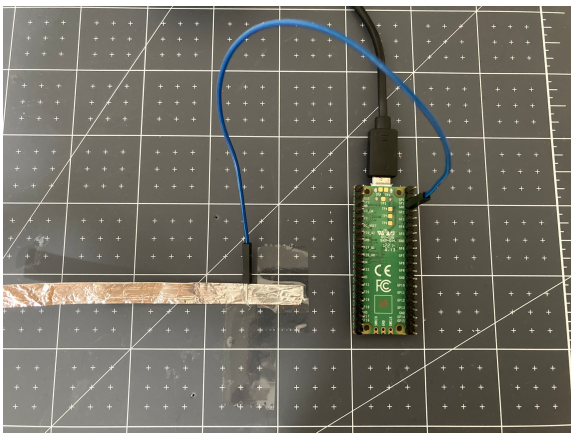
There are only 8 **GND** pins on a Raspberry Pi Pico. This means that if you want to use more than 8 components, you will need to share a **GND**. Here is one method for sharing a ground pin:

You will need:

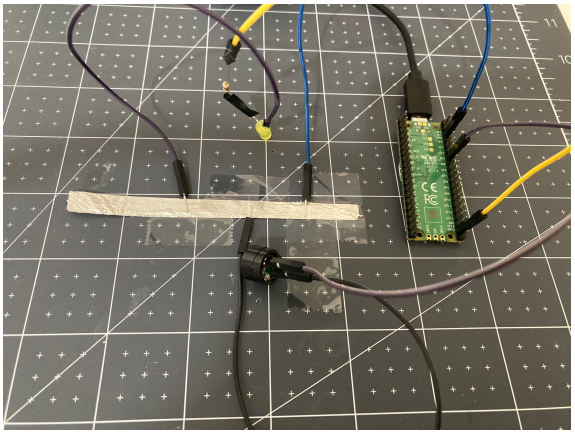
- pin-socket jumper wires
- some kitchen foil, conductive tape or some other conductive material
- sticky tape

Step 1: Create a strip of kitchen foil or use your other conductive material.

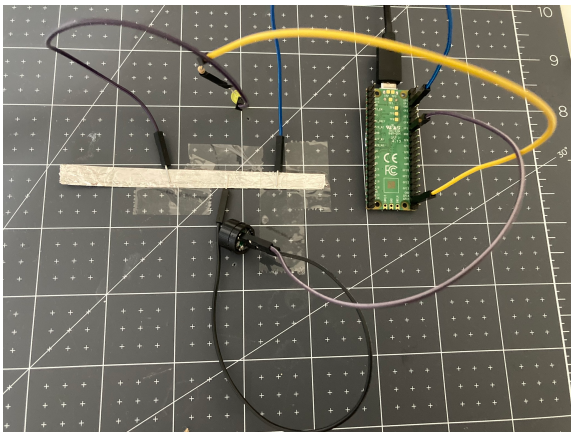
Step 2: Connect a **GND** pin to your kitchen foil (use sticky tape to secure).



Step 3: When adding a component to the Raspberry Pi Pico, connect the **positive** end to a **GP** pin and the **negative** end to the kitchen foil.



Step 4: When adding further components, ensure that the **positive** end is connected to a **GP** pin on the Raspberry Pi Pico and that the **negative** end is stuck to the kitchen foil.



Single-colour LEDs

Wire a single LED

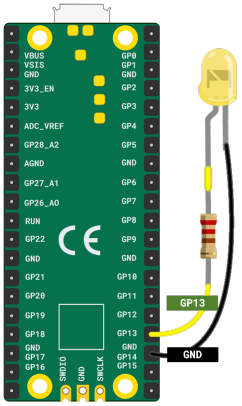
You will need:

- A Raspberry Pi Pico
- An LED in your choice of colour
- A resistor
- 2 x socket-socket jumper wires

Note: The LED should have a resistor attached to the **positive** (long) leg, as well as the two jumper wires.

The diagram below shows how to attach an LED to a Raspberry Pi Pico.

- Attach the positive leg (the one with the resistor) to **GP13**
- Attach the negative leg to the closest **GND** pin



Wire multiple single LEDs

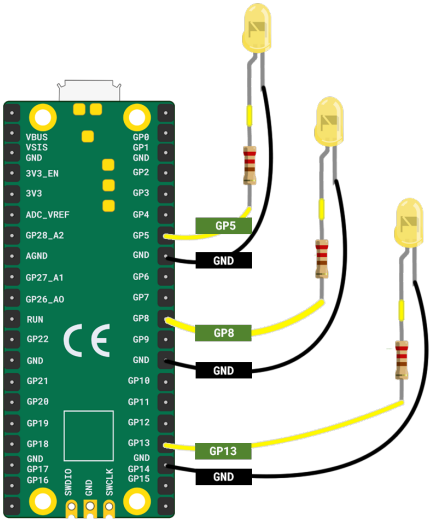
You will need:

- A Raspberry Pi Pico
- Multiple LEDs in your choice of colour
- Enough resistors for each LED
- Enough socket-socket jumper wires for each LED

Note: The **positive** (long) leg of each LED should have a resistor attached. You should also attach jumper wires to both legs.

The diagram below shows **three** single LEDs attached to a Raspberry Pi Pico.

- Attach LED one to **GP13** and the closest **GND** pin
- Attach LED two to **GP8** and the closest **GND** pin
- Attach LED three to **GP5** and the closest **GND** pin



Set the pins for a single LED

To set the pin for a single LED, use the following code:

```
from picozero import LED

led = LED(13)
```

Set the pins for multiple LEDs

Import LED from the picozero library then set the pins for multiple LEDs, use the following code:

```
from picozero import LED

led_1 = LED(13)
led_2 = LED(8)
led_3 = LED(5)
```

Tip: You should give your LED variables a meaningful name. For example, if each LED is a different colour then you could use the following code:

```
red_led = LED(13)
green_led = LED(8)
pink_led = LED(5)
```

Add a function to turn on a single LED

sensory-gadget.py

```
def excited(): # Your mood
    purple.on() # Turn on
```

Add functions to control multiple single LEDs

sensory-gadget.py

```
    def excited(): # Your first mood
        purple.on() # Turn on
        blue.off() # Turn off

    def worried(): # Your second mood
        purple.off() # Turn off
        blue.on() # Turn on
```

RGB LEDs

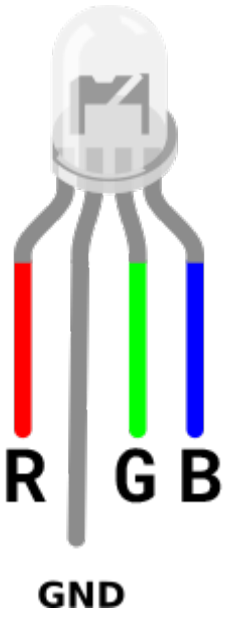
Wire an RGB LED

You will need:

- A Raspberry Pi Pico
- A common cathode RGB LED
- 3 x resistors
- 8 x socket-socket jumper wires

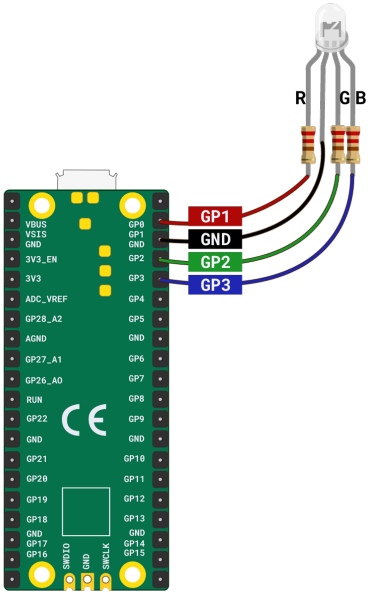
Note: You will need to attach resistors to the three shorter legs of the RGB LED. The longer leg is for **ground** and doesn't require a resistor.

Look: Your RGB LED has four legs. Turn your RGB LED so that the longer **GND** leg is second from the left. Notice how the legs go **R** for **red**, then **GND**, then **G** for **green** and finally **B** for **blue**. This will help you remember what each leg is doing.



Connect: Wire your RGB LED

- Attach the **R** leg to **GP1**
- Attach the **GND** to the **GND** pin
- Attach the **G** leg to the **GP2** pin
- Attach the **B** leg to the **GP3** pin



Set the pins for a RGB LED

Import RGBLED from the picozero library then set the pins for a common cathode RGB LED, use the following code:

```
from picozero import RGBLED

rgb = RGBLED(red = 1, green = 2, blue = 3)
```

Add functions to set an RGB LED colour

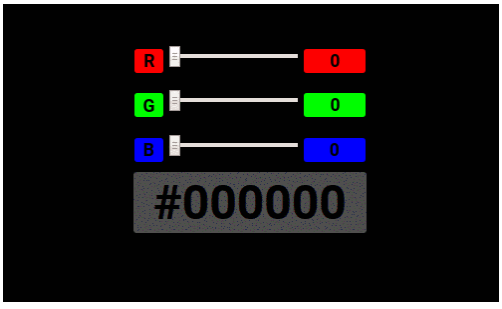
sensory-gadget.py

```
def happy(): # Your first mood
    rgb.color = (0, 255, 0) # Your first colour

def sad(): # Your second mood
    rgb.color = (255, 0, 0) # Your second colour
```

RGB colours

When we want to represent a colour in a computer program, we can do this by defining the amounts of red, blue, and green that are combined to make up that colour. These amounts are stored as a number between 0 and 255.



Here's a table showing some colour values:

Red	Green	Blue	Colour
255	0	0	Red
0	255	0	Green
0	0	255	Blue
255	255	0	Yellow
255	0	255	Magenta
0	255	255	Cyan

You can find a nice **colour picker to play with at w3schools** (https://www.w3schools.com/colors/colors_rgb.asp).

Speakers and buzzers

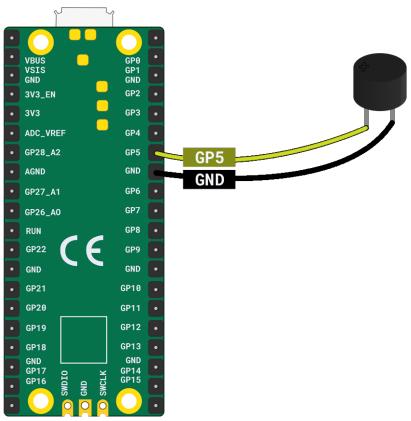
Wire a single buzzer

You will need:

- A Raspberry Pi Pico
- A **passive** tone buzzer
- 2 x socket-socket jumper wires

To wire a single buzzer to a Raspberry Pi Pico, connect the **positive** leg to pin **GP5** and the **negative** leg to the nearest **GND** pin.

Tip: You can identify the positive leg by looking for the longest leg or by finding the side with a **+** sign at the top.



Wire a stereo buzzer

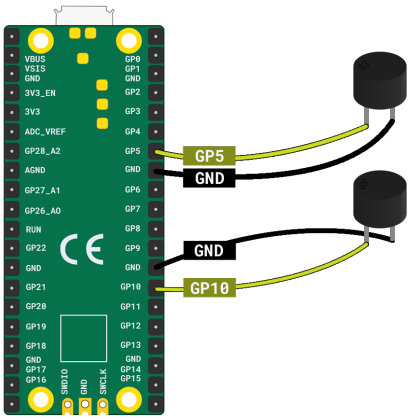
To wire two buzzers to a Raspberry Pi Pico, you will need:

- A Raspberry Pi Pico
- 2 x passive buzzers
- 4 x socket-socket jumper wires

Take the **first buzzer** and connect the **positive** leg to **GP5** and the **negative** leg to the nearest **GND** pin.

Next, take the **second buzzer** and connect the **positive** leg to **GP10** and the **negative** leg to the nearest **GND** pin.

Tip: To identify the positive leg on a buzzer, look for the longest leg. You may also be able to see a **+** symbol on the top of the buzzer.



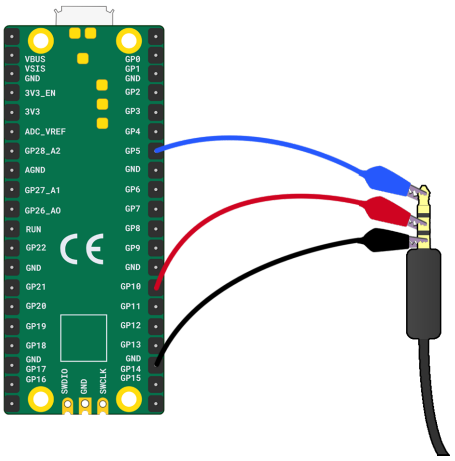
Wire earphones

To connect earphones to your Raspberry Pi Pico, you will need:

- A Raspberry Pi Pico
- A pair of earphones with a 3 or 5mm jack (not a USB end)
- 3 x crocodile clips

Instructions

1. Decide which crocodile clip will be for **ground** and which will be for the **GP** pins.
2. Take your first **GP** pin crocodile clip and attach it to the tip of the jack.
3. Take the other end of your **GP** pin crocodile clip and attach it to the **GP5** pin.
4. Take your second **GP** pin crocodile clip and attach it to the middle of the jack.
5. Take the other end of your **GP** pin crocodile clip and attach it to the **GP10** pin.
6. Take your **ground** crocodile clip and attach one end to the base of the jack.
7. Connect the other end of your **ground** crocodile clip to a **GND** pin.



Set the pins for a single buzzer

Import Speaker from the picozero library then set the pins for a buzzer, use the following code:

```
from picozero import Speaker

speaker = Speaker(5)
```

Set the pins for two buzzers

Import Speaker from the picozero library then set the pins for multiple buzzers, use the following code:

```
from picozero import Speaker

speaker_1 = Speaker(5)
speaker_2 = Speaker(10)
```

Tip: You might want to use variable names that are specific to what the speaker should do. For example, `drum_beat`.

Useful information about sound

Available notes

There are 88 notes available to use with the `picozero` library. They link to western musical notation and are all of the notes that you can find on a full size keyboard.

The notes available are:

- b0
- c1
- c#1
- d1
- d#1
- e1

- f1
- f#1
- g1
- g#1
- a1
- a#1
- b1
- c2
- c#2
- d2
- d#2
- e2
- f2
- f#2
- g2
- g#2
- a2
- a#2
- b2
- c3
- c#3
- d3
- d#3
- e3
- f3
- f#3
- g3
- g#3
- a3
- a#3
- b3
- c4 - This is middle C on a keyboard
- c#4
- d4
- d#4
- e4
- f4
- f#4
- g4
- g#4
- a4
- a#4
- b4
- c5
- c#5
- d5
- d#5
- e5
- f5
- f#5
- g5

- g#5
- a5
- a#5
- b5
- c6
- c#6
- d6
- d#6
- e6
- f6
- f#6
- g6
- g#6
- a6
- a#6
- b6
- c7
- c#7
- d7
- d#7
- e7
- f7
- f#7
- g7
- g#7
- a7
- a#7
- b7
- c8
- c#8
- d8
- d#8

Note length

There are several ways to set the length of a note.

Use a value to set the length of a single note

When you play a single note you can decide how long it will play for by entering a value for the second argument.

The `0.1` in the example code below will play the note for 0.1 seconds.

```
speaker.play(c_note, 0.1) # play the middle c for 0.1 seconds
```

Use a constant for the length of a single note

You can replace the value in the second argument with a constant.

The example below uses the constant `BEAT`. This can then be used to set the beat for any tune in your program.

```
BEAT = 1 # a constant

speaker.play(c_note, BEAT) # play the middle c for 1 second
```

Set the length of each note in a tune using a value

If you want to play a tune then you should use a list to store all of the notes in your tune along with the length of each note.

The example below shows a list of lists. Each list contains a note in quotes `''` and then a value for the length the note should play.

```
my_tune = [ ['d5', 1], ['d#5', 0.5], ['f5', 1.2]] # the notes, along with the length of each note
```

Set the length of each note in a tune using a constant

The example below sets the `BEAT` constant to 0.4 on line 1.

You can then see a list of lists on line 3. Each list contains a note in quotes `''` and then the `BEAT` constant.

```
1 BEAT = 0.4 # The length of a note in a single beat
2
3 my_tune = [ ['d5', BEAT], ['d#5', BEAT / 2], ['f5', BEAT * 1.5]] # the notes, along with the length of each note
```

You can make the note **shorter** by dividing the beat: `BEAT / 2`.

You can make the note **longer** by multiplying the beat: `BEAT * 1.5`

Tip: This option allows you to set the BPM of your tune and adjust it to suit your needs.

Numbers instead of notes

If you don't want to use notes like `c4` in your code then you can use frequency values instead. You can use values between `150` and `10000`.

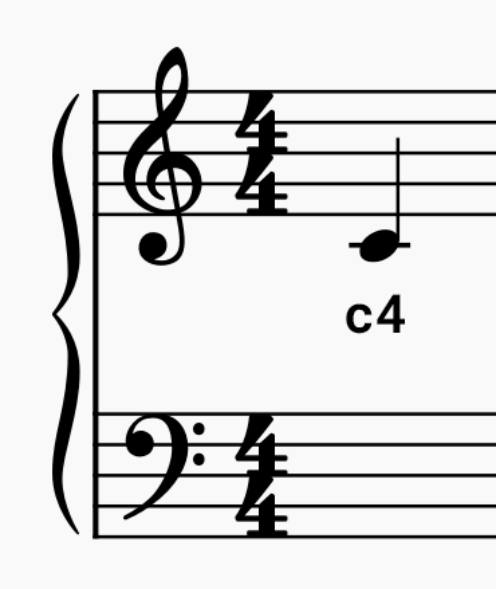
The code below plays a single note with a frequency of `523`.

```
speaker.play(523, 1) # Play the frequency 523 for 1 second
```

Turning sheet music into notes

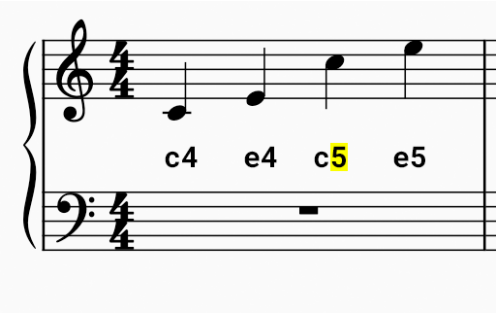
The `picozero` library allows you to enter real musical notes to make tunes to play on your speaker.

A **letter** is used to represent the musical note and a **number** is used to represent where the note appears on the grand stave (staff).



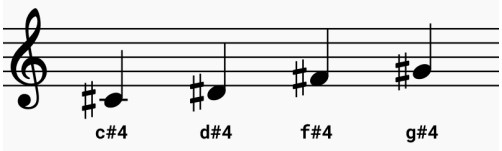
For example, the **middle C** (above) is at the centre of the grand stave and uses `c4`.

When you go up the grand stave, the number increases. When you go down the grand stave, the number decreases.



Representing sharps

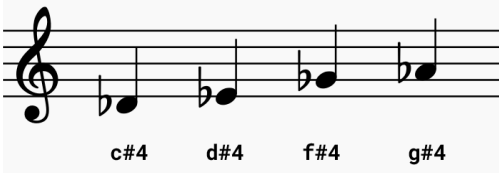
Your musical score might include notes that are **sharps**. These are represented using a `#` symbol. In the example below, the first note is a C sharp. A C sharp is `c#4`.



Representing flats

Your musical score might include notes that are **flats**. These are **also** represented using a `#` because the library doesn't have a specific code for flats. To turn a flat into a sharp you need to go down the scale.

- A **D flat** becomes a **C sharp** or `c#4`
- An **E flat** becomes a **D sharp** or `d#4`
- A **G flat** becomes an **F sharp** or `f#4`
- An **A flat** becomes a **G sharp** or `g#4`



Sound code samples

Play a single note

Play a note and wait for it to finish:

sound_machine.py

```
def c_note():
    speaker.play('c4', 0.5) # play the middle c for half a second
```

Play a tune

Create a list of notes and durations to make a tune.

You can then `play` the tune:

sound_machine.py

```
BEAT = 0.4

liten_mus = [ ['d5', BEAT / 2], ['d#5', BEAT / 2], ['f5', BEAT], ['d6', BEAT], ['a#5', BEAT], ['d5', BEAT],
              ['f5', BEAT], ['d#5', BEAT], ['d#5', BEAT], ['c5', BEAT / 2],['d5', BEAT / 2], ['d#5', BEAT],
              ['c6', BEAT], ['a5', BEAT], ['d5', BEAT], ['g5', BEAT], ['f5', BEAT], ['f5', BEAT], ['d5', BEAT / 2],
              ['d#5', BEAT / 2], ['f5', BEAT], ['g5', BEAT], ['a5', BEAT], ['a#5', BEAT], ['a5', BEAT], ['g5', BEAT],
              ['g5', BEAT], ['', BEAT / 2], ['a#5', BEAT / 2], ['c6', BEAT / 2], ['d6', BEAT / 2], ['c6', BEAT / 2],
              ['a#5', BEAT / 2], ['a5', BEAT / 2], ['g5', BEAT / 2], ['a5', BEAT / 2], ['a#5', BEAT / 2], ['c6', BEAT],
              ['f5', BEAT], ['f5', BEAT], ['f5', BEAT / 2], ['d#5', BEAT / 2], ['d5', BEAT], ['f5', BEAT], ['d6', BEAT],
              ['d6', BEAT / 2], ['c6', BEAT / 2], ['b5', BEAT], ['g5', BEAT], ['g5', BEAT], ['c6', BEAT / 2],
              ['a#5', BEAT / 2], ['a5', BEAT], ['f5', BEAT], ['d6', BEAT], ['a5', BEAT], ['a#5', BEAT * 1.5] ]

def play_liten_mus():
    speaker.play(liten_mus)
```

Make sound effects from frequencies

You can make interesting, and annoying, sound effects by playing short sounds at different frequencies.

We hear different frequencies as different musical notes, tones or pitches.

This example gradually increases in frequency to create a positive sound:

```
def win(): # rising frequency
    for i in range(2000, 5000, 100):
        speaker.play(i, 0.05) # short duration
```

This example decreases the pitch to create a bird chirping sound:

```
def chirp(): # series of high-pitched chirps
    for _ in range(2): # decreasing frequency
        for i in range(5000, 2999, -100):
            speaker.play(i, 0.02) # very short duration
        sleep(0.2)
```

Play around with playing short notes and changing the frequency in a `for` loop. Use frequences between `150` and `10000`.

Whitenoise drum beat effect

sound_machine.py

```
from picozero import Speaker
from time import sleep
from random import randint

speaker = Speaker(5)

for i in range(100):
    speaker.play(randint(500, 5000), duration=None)
    sleep(0.001)
    speaker.stop()
    sleep(0.5)
```

Play notes in a loop

Create a list of notes and durations to make a tune.

You can then use a loop to `play` the tune a note at a time:

sound_machine.py

```
BEAT = 0.4

liten_mus = [ ['d5', BEAT / 2], ['d#5', BEAT / 2], ['f5', BEAT], ['d6', BEAT], ['a#5', BEAT], ['d5', BEAT],
              ['f5', BEAT], ['d#5', BEAT], ['d#5', BEAT], ['c5', BEAT / 2],['d5', BEAT / 2], ['d#5', BEAT],
              ['c6', BEAT], ['a5', BEAT], ['d5', BEAT], ['g5', BEAT], ['f5', BEAT], ['f5', BEAT], ['d5', BEAT / 2],
              ['d#5', BEAT / 2], ['f5', BEAT], ['g5', BEAT], ['a5', BEAT], ['a#5', BEAT], ['a5', BEAT], ['g5', BEAT],
              ['g5', BEAT], ['', BEAT / 2], ['a#5', BEAT / 2], ['c6', BEAT / 2], ['d6', BEAT / 2], ['c6', BEAT / 2],
              ['a#5', BEAT / 2], ['a5', BEAT / 2], ['g5', BEAT / 2], ['a5', BEAT / 2], ['a#5', BEAT / 2], ['c6', BEAT],
              ['f5', BEAT], ['f5', BEAT], ['f5', BEAT / 2], ['d#5', BEAT / 2], ['d5', BEAT], ['f5', BEAT], ['d6', BEAT],
              ['d6', BEAT / 2], ['c6', BEAT / 2], ['b5', BEAT], ['g5', BEAT], ['g5', BEAT], ['c6', BEAT / 2],
              ['a#5', BEAT / 2], ['a5', BEAT], ['f5', BEAT], ['d6', BEAT], ['a5', BEAT], ['a#5', BEAT * 1.5] ]

for note in liten_mus:
    speaker.play(note)
```

Play a tune and allow other actions to take place while it is playing

If you want to play a whole tune and be able to interrupt it before it finishes then you can create a tune as a list of notes and durations and then use `play` with `wait=False`. This will allow you to stop the tune or play another tune by pressing the same or a different button:

sound_machine.py

```
BEAT = 0.4

liten_mus = [ ['d5', BEAT / 2], ['d#5', BEAT / 2], ['f5', BEAT], ['d6', BEAT], ['a#5', BEAT], ['d5', BEAT],
              ['f5', BEAT], ['d#5', BEAT], ['d#5', BEAT], ['c5', BEAT / 2],['d5', BEAT / 2], ['d#5', BEAT],
              ['c6', BEAT], ['a5', BEAT], ['d5', BEAT], ['g5', BEAT], ['f5', BEAT], ['f5', BEAT], ['d5', BEAT / 2],
              ['d#5', BEAT / 2], ['f5', BEAT], ['g5', BEAT], ['a5', BEAT], ['a#5', BEAT], ['a5', BEAT], ['g5', BEAT],
              ['g5', BEAT], ['', BEAT / 2], ['a#5', BEAT / 2], ['c6', BEAT / 2], ['d6', BEAT / 2], ['c6', BEAT / 2],
              ['a#5', BEAT / 2], ['a5', BEAT / 2], ['g5', BEAT / 2], ['a5', BEAT / 2], ['a#5', BEAT / 2], ['c6', BEAT],
              ['f5', BEAT], ['f5', BEAT], ['f5', BEAT / 2], ['d#5', BEAT / 2], ['d5', BEAT], ['f5', BEAT], ['d6', BEAT],
              ['d6', BEAT / 2], ['c6', BEAT / 2], ['b5', BEAT], ['g5', BEAT], ['g5', BEAT], ['c6', BEAT / 2],
              ['a#5', BEAT / 2], ['a5', BEAT], ['f5', BEAT], ['d6', BEAT], ['a5', BEAT], ['a#5', BEAT * 1.5] ]

sound = [ [523, 0.1], [None, 0.1], [523, 0.4] ]

def annoying_sound():
    speaker2.play(sound, wait=False) # don't delay the main code

button.when_pressed = annoying_sound

try:
    speaker.play(liten_mus)
finally:
    speaker.off() # turns speaker off when code is stopped by user
    speaker2.off() # turns speaker2 off when code is stopped by user
```

Connect your inputs

Button

Wire a single button

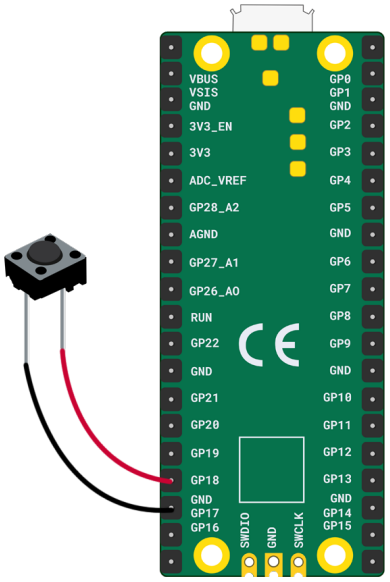
You will need:

- A Raspberry Pi Pico
- 2 x socket-socket jumper wires
- 1 x push button

To connect your push button:

- Attach the two jumper wires to the push button pins (secure with tape if needed)
- Connect one jumper wire to the **GP18** pin
- Connect the other jumper wire to the closest **GND** pin

Note: There are no positive or negative legs here so it doesn't matter which way round you connect the wires.



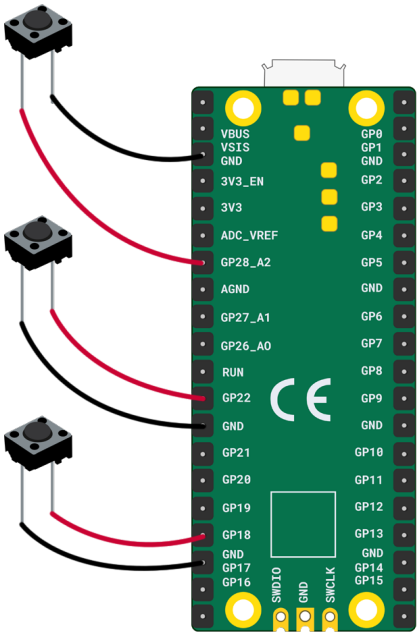
Wire multiple buttons

You will need:

- A Raspberry Pi Pico
- 2 x pin-socket jumper wires for each button
- Your chosen number of buttons

Note: This diagram shows **three** push buttons attached to the Raspberry Pi Pico. If you would like a **fourth** button then you will need to attach it to a **GP** pin and a **GND** on the right hand side.

- Attach two pin-socket jumper wires to **GP18** and the closest **GND**
- Attach two pin-socket jumper wires to **GP22** and the closest **GND**
- Attach two pin-socket jumper wires to **GP28** and the closest **GND**



Import Button

sensory-gadget.py

```
from picozero import Button
```

Set the pins for a single button

Import Button from the picozero library then set the pins for a single button, use the following code:

```
from picozero import Button

button = Button(18)
```

Set the pins for multiple buttons

Import Button from the picozero library then set the pins for multiple buttons, use the following code:

```
from picozero import Button

button_1 = Button(18)
button_2 = Button(22)
button_3 = Button(28)
```

Tip: You might want to give your buttons variable names that relate to what they are doing. For example, `red_button` to turn on a red light.

Call a different function when each button is pressed

You can have multiple buttons that each call a different function when they are pressed.

Make sure you use the function names from your project and just use the name of the function; do not call it by adding brackets.

sensory-gadget.py

```
happy_button.when_pressed = happy
sad_button.when_pressed = sad
angry_button.when_pressed = angry
```

Change to the next function when a single button is pressed

Use an `option` variable to keep track of the current mood so that you can decide which function to call next.

Make sure the function names match the mood functions you defined in the previous step.

sensory-gadget.py

```
option = 0 # Store the current option

def choice(): # Call the next function and update the option
    global option
    if option == 0:
        energised() # Your first mood
    elif option == 1:
        calm()      # Your second mood
    elif option == 2:
        focused()   # Your third mood
    elif option == 3:
        rgb.off()

    # Move to the next option
    if option == 3:
        option = 0
    else:
        option = option + 1

button.when_pressed = choice # Call the choice function when the button is pressed
```

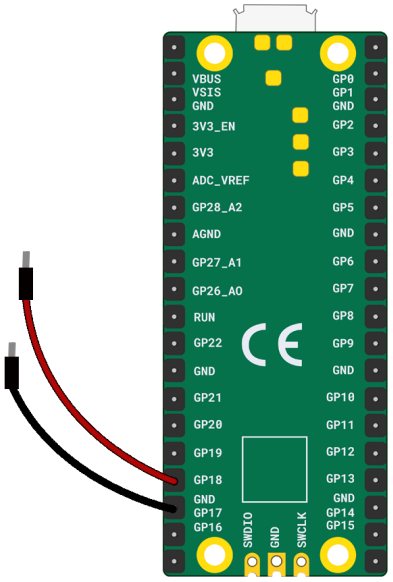
Switch

Wire a crafted button or switch

You will need:

- A Raspberry Pi Pico
- Two pin-socket jumper wires

Attach one pin-socket jumper wire to **GP18** and attach the other pin-socket jumper wire to the **GND** closest to it.



Tip: When you craft a switch you will stick the pin end of the jumper wire to a conductive surface, like copper taper or kitchen foil.

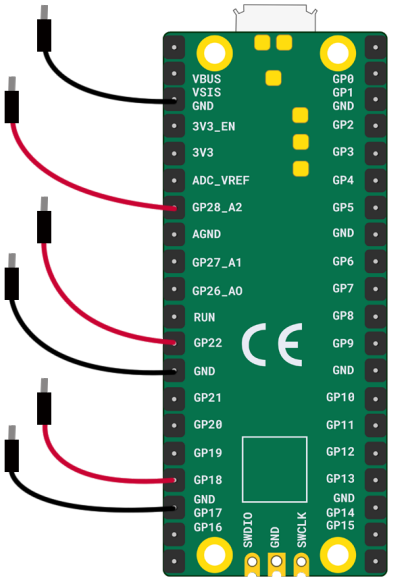
Wire multiple crafted switches or buttons

You will need:

- A Raspberry Pi Pico
- 2 x pin-socket jumper wires for each crafted switch

Note: This diagram shows **six** jumper wires attached to the Raspberry Pi Pico. This is enough for **three** crafted buttons or switches. If you would like a **fourth** switch then you will need to attach it to a **GP** pin and a **GND** on the right hand side.

- Attach two pin-socket jumper wires to **GP18** and the closest **GND**
- Attach two pin-socket jumper wires to **GP22** and the closest **GND**
- Attach two pin-socket jumper wires to **GP28** and the closest **GND**



Import Switch

sensory-gadget.py

```
from picozero import Switch
```

Set the pins for a single switch

Import Switch from the picozero library then set the pins for a single switch, use the following code:

```
from picozero import Switch

switch = Switch(18)
```

Set the pins for multiple switches

Import Switch from the picozero library then set the pins for multiple switches, use the following code:

```
from picozero import Switch

switch_1 = Switch(18)
switch_2 = Switch(22)
switch_3 = Switch(28)
```

Tip: You might want to give your switches variable names that relate to what they are doing. For example, `red_switch` to turn on a red light.

Potentiometer

Wire a potentiometer

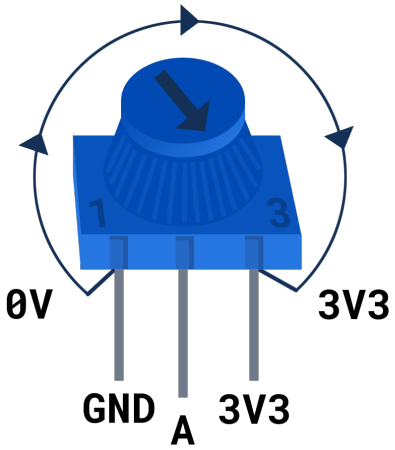
You will need:

- A Raspberry Pi Pico
- A potentiometer
- 3 x socket-socket jumper wires

A **potentiometer** has three pins:

- Ground
- Analogue signal
- 3V3 (power)

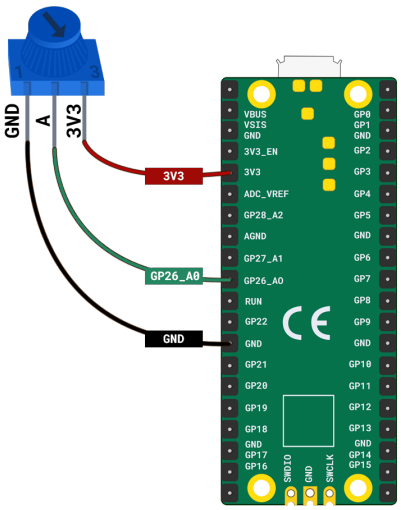
When the potentiometer is turned all the way to the **left** the arrow points to the **GND** pin, when it is turned all the way to the **right**, the arrow points to the **3V3** pin. The middle pin is the pin that the Raspberry Pi Pico reads a value from.



Connect: Find three socket-socket jumper wires and attach one to each leg of the potentiometer. You may wish to secure the legs with some electrical tape if they feel loose.

Connect the other end of each jumper to the Raspberry Pi Pico:

- Connect the labelled with a small 1 to the **GND** pin between **GP21** and **GP22**.
- Connect the middle pin the **GP26_A0**. This is an analogue pin.
- Connect pin labelled with a small 3 to the **3V3** pin.



Set the pins for a potentiometer

Import Pot from the picozero library then set the pins for a potentiometer, use the following code:

```
from picozero import Pot

dial = Pot(0) # Connected to pin A0 (GP_26)
```

Import Potentiometer

sensory-gadget.py

```
from picozero import Pot
```

Call a function based on the value of the potentiometer

If you are using a potentiometer to control outputs, then you will need to divide up the dial into equal sections.

You can use `dial.value` to get a value between 0 and 1 from the potentiometer.

Tip: You can multiply the value by 100 to get a percentage. If you have five moods, then you can check whether the value is less than 20, 40, 60, 80, or 100. If you have three moods, then you can check whether the value is less than 33, 66, or 100.

sensory-gadget.py

```
while True:
    mood = dial.value * 100 # turn to a percentage
    print(mood)
    if mood < 20:
        happy()
    elif mood < 40:
        good()
    elif mood < 60:
        okay()
    elif mood < 80:
        unsure()
    else:
        unhappy()
    sleep(0.1)
```

Test: Test your inputs and outputs to make sure everything works as expected.

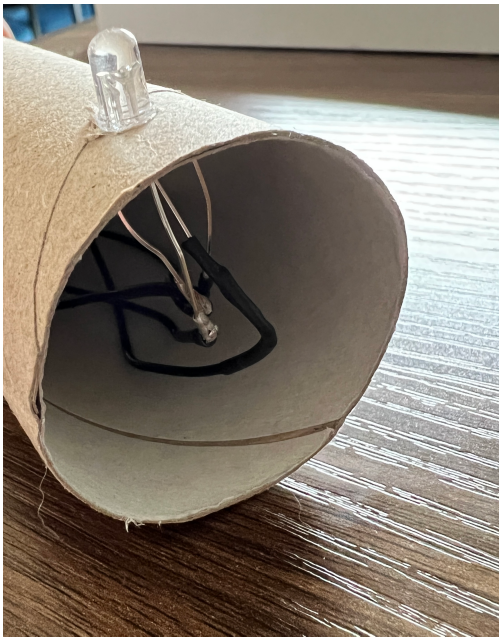
Use the **debug** task at the bottom of this step if you come across any problems.

Craft your gadget

Mount components in card or plastic

You may want to mount LEDs, buttons, buzzers and potentiometers in card or plastic.

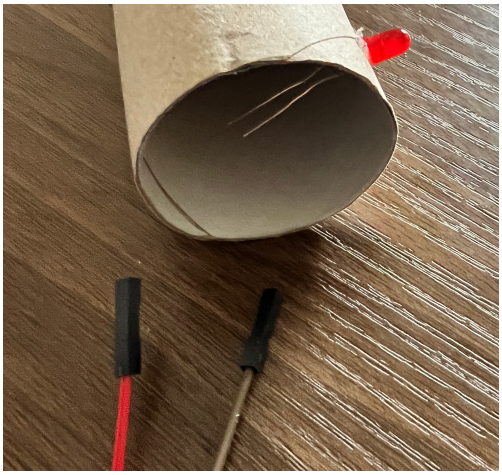
If your components have soldered jumper wires, make a hole in the card or plastic and then push the component through from the back.



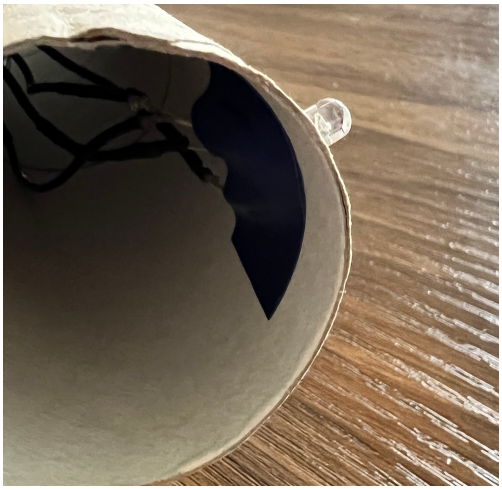
If you are using components with socket-socket jumper wires then remove the jumper wires and push the legs of the component through card. For plastic you will need make holes first by carefully using a tool with a sharp point.

Tip: Remember which leg connects to which jumper wire.

Then reconnect the jumper wires on the back of the card or plastic.



If necessary you can use sticky tape or electrical tape to keep your components in place.



Diffuse lights from LEDs

You often get a better visual effect by diffusing (spreading out) the light from an LED, especially if you are using clear LEDs or RGB LEDs that mix multiple colours.

Placing an LED behind a material such as thin paper, tracing paper or invisible sticky tape will spread the light over a larger area and create a softer effect that is nicer to look at.

You can also use a white material, such as paper to direct light in the direction you want it to be seen from so that you make the most of the light from the LED.

Use tape to hold jumper wires in place

Use sticky tape or electrical tape to hold the jumper wires in place so that your device stays together.

You can remove the tape later if you want to reuse the components.

Safe use of a craft knife

Craft and utility knives are very useful when making models, but you must be very careful when using them, as they are extremely sharp and can easily cause an injury.

If you are using a craft or utility knife, make sure you have a responsible adult with you, or ask them to do the cutting for you if you prefer.

It's also a good idea to use a cutting mat to protect the surface you are working on. If you don't have a cutting mat, a kitchen chopping board is a great alternative but ask permission before you use something that isn't yours.

Joining jumper wires to extend them

You might need extra-long wires to attach your component (for example LED, button, or buzzer) to your Raspberry Pi pins. You can do this by 'daisy chaining' wires together. For instance, to make an extra-long socket-socket wire, you can attach an pin-socket wire to a socket-socket wire.

Tip: keep the colour of the wire the same to make it easier to follow fro the component to the Raspberry Pi pin.

A pin-socket wire from the Raspberry Pi and a socket-socket wire from the component.

A pin-socket wire attached to a socket-socket wire connecting the Raspberry Pi and the component.

The problem with this method is that often the wires will become detached from each other. You can use a small piece of tape to secure the connection.

A pin-socket wire taped to a socket-socket wire.

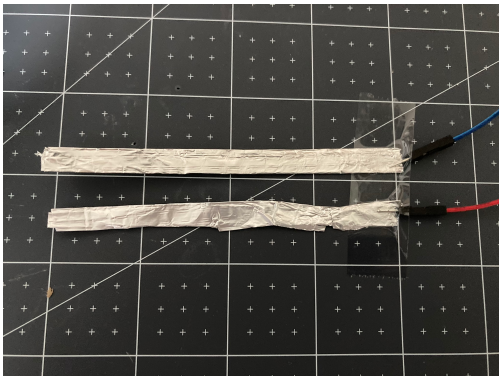
Craft a drop switch

To craft a drop switch for your Raspberry Pi Pico, you will need:

- A Raspberry Pi Pico
- 2 x socket-pin jumper wires
- Sticky tape
- Kitchen foil or conductive tape

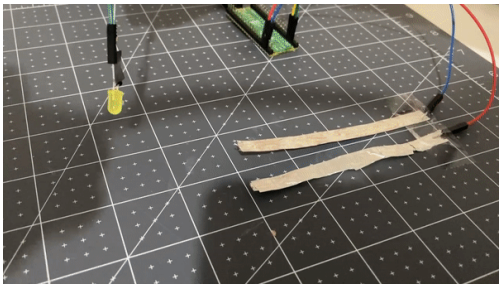
Instructions

Step 1: Create two conductive strips either with kitchen foil or conductive tape.
Step 2: Stick the pin ends of the jumper wire to each strip.



Step 3: Stick some kitchen foil or conductive tape to the bottom of an object. The base must be large enough to create a connection between the two wires.

Step 4: Attach the socket ends of the jumper wires to your Raspberry Pi Pico and drop your switch!



Craft a pull switch

To create a pull switch, you will need:

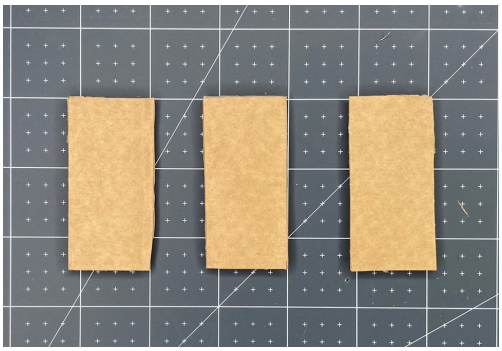
- A pair of scissors
- Corrugated card
- Aluminium foil
- A glue stick
- Some sticky tape

Optional:

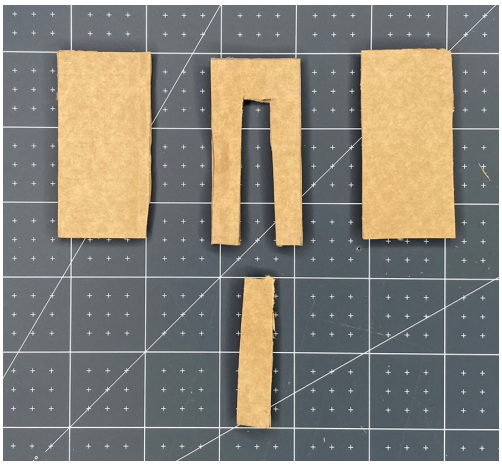
- A pencil and a ruler (if you want to be more precise with your make)
- Some nice ribbon OR string OR coloured paper/card OR plain paper that you have coloured in

Instructions

Step 1: Cut the **corrugated card** into three rectangles that are the same size.

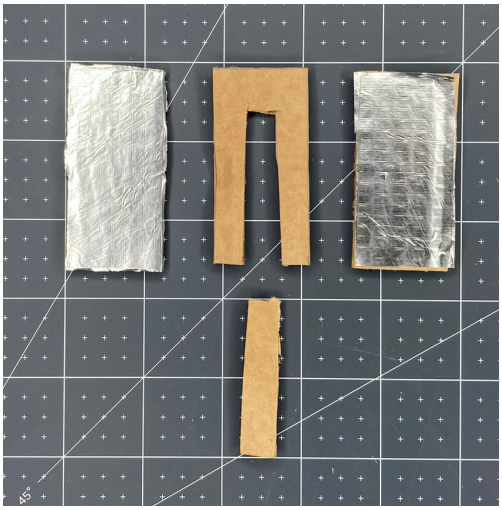


Step 2: Cut a section out of the centre of one of your rectangles. Keep the piece of card that you have cut out as this will be used later.



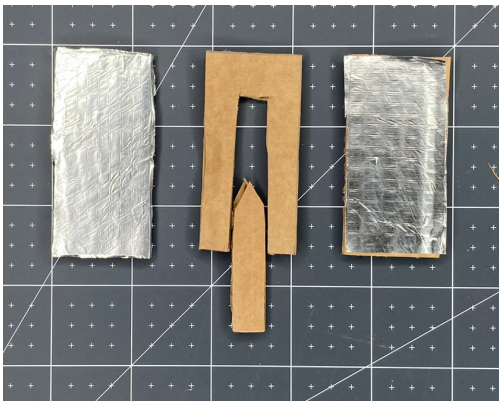
Step 3: Take the **aluminium foil** and cut it to the same size as the un-cut rectangles.

Step 4: Glue the cardboard and attach the foil. Make sure you don't get too much glue on the outside of the foil, or it will affect the contacts of the switch.

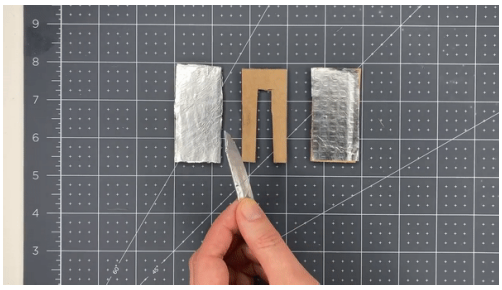


Step 5: Take the piece of card that you removed from the centre rectangle and cut a V shape out of the top to make it easier to place it inside your switch.

Step 6: Trim the sides by a few millimetres to make sure that it will easily fit into your popper.



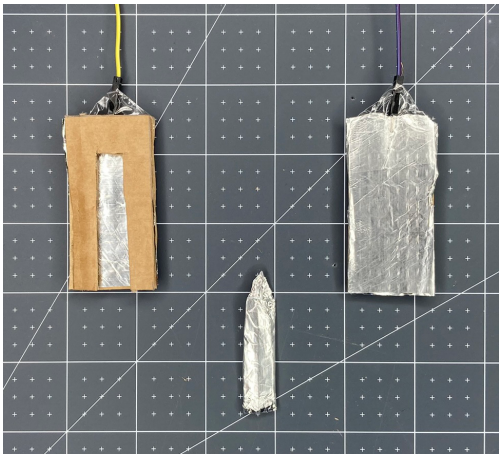
Step 7: Cover the removed piece in **aluminium foil**. It is very important that you use one piece of foil and that it wraps all the way around. This is what will make the switch close and allow the current to flow.



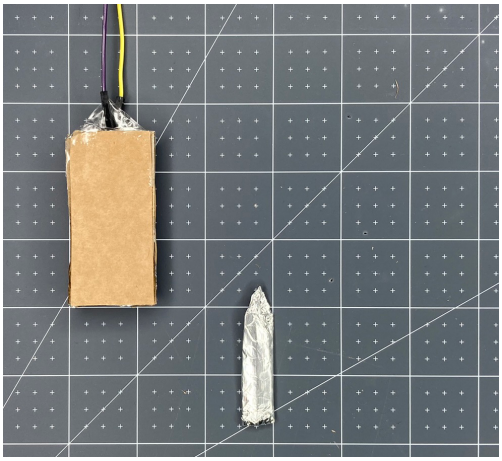
Step 8: Next, use some sticky tape to secure the pin end of two jumper wires to the top of each rectangle. It is important that the pins make a secure contact with the aluminium foil. Make sure that each pin is lying flat against the foil with the plastic part of the jumper wire against the edge of the cardboard.

Step 9: Add more tape to secure the jumper wire and stop it from accidentally coming loose.

Step 10: Add glue to one side of the middle piece and stick it to the left rectangle. This will create a barrier between the two pieces of foil and allow space for your centre piece to be placed inside.



Step 11: Add glue to the other side of the middle piece and stick the foil face of your other rectangle on top. Make sure that the two pieces of foil **are not** touching. You may need to trim your foil if it is overlapping.

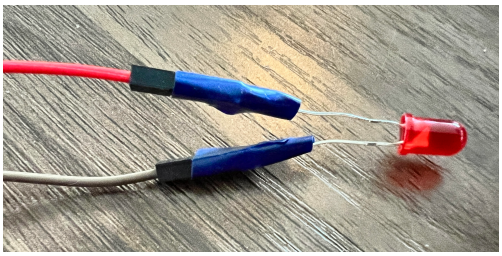


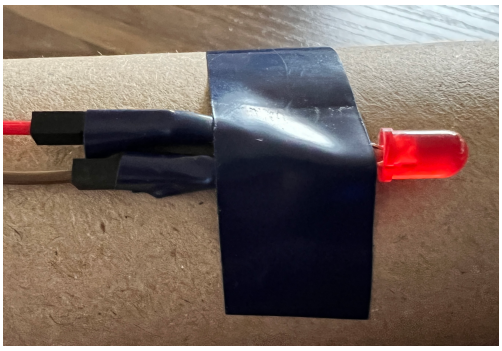
Now use your switch with your project!

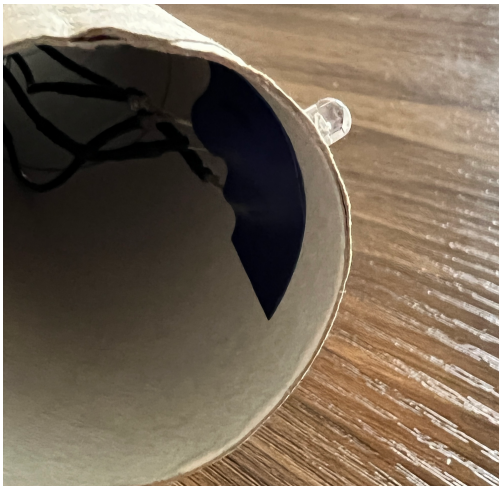
Secure wires and components using tape

Use sticky tape or electrical tape to secure jumper wires to components or hold components in place so that your device stays together.

You can remove the tape later if you want to reuse the components.







Test: Show someone else your project and get their feedback. Do you want make any changes to your gadget?

Debug: You might find some bugs in your project that you need to fix. Here are some common bugs.

Debug Pico code

- Check the Shell in Thonny for error messages.
- Check carefully for mistakes in your code such as a missing ':' or incorrect indentation.
- Make sure you have imported the components that you are using such as 'Button' or 'RGBLED'
- Add `print` statements to help you understand what is happening such as `print('Starting')` or `print('Setting colour to green')`.

My inputs or outputs don't work as expected

Check that all you inputs, outputs, and wires have a good connection. Even some that were working originally may have come loose as you crafted your device.

Check that the pins in your code match the pins your LED is connected to. If you removed your pins to craft your device you need to check they have been reconnected as they were originally.

If your component has polarity, positive (+) and negative (-) sides, make sure they are connected the right way round.

If you have checked everything and you are still not seeing or hearing the result you expect then try using a new electronic component if you have some spare. This will ensure that your current electronic component is not broken.

Use the Pico LED for debugging

Turn the LED on the Raspberry Pi Pico board on and off to check that your device is working and your code is running.

Add this code to the top of your script:

main.py

```
from picozero import pico_led
from time import sleep

pico_led.on()
sleep(2)
pico_led.off()
```

Code runs, but nothing happens:

- Check that your inputs are connected correctly and that you used the correct pin in your code.
- Check the Thonny shell for any messages about variables or functions not being defined; you might have forgotten to change the examples to match your code.
- Check your code carefully. You could add `print` statements to help you understand what is happening.
- Check that you have called your functions.

Call a function

sensory-gadget.py

```
def happy(): # Your first mood
    rgb.color = (0, 255, 0) # Your first colour

def sad(): # Your second mood
    rgb.color = (255, 0, 0) # Your second colour

happy()
```

My LED doesn't light when I call my function

Check that the pins in your code match the pins your LED is connected to.

My RGB LED show the wrong colour

Check your code to make sure that your colour values are in the right order. Use the **'RGB colour guide'** (https://www.w3schools.com/colors/colors_rgb.asp) to check your code matches the colour you expect.

My sounds are not playing, or not playing as I expected

Your code was working before you assembled your sensory gadget. It is unlikely that your code will be broken at this stage. The majority of bugs will be from wiring and components.

- Check that your components have been wired to the correct pins (you should have noted these down earlier, they are displayed at the top of your code)
- Look for any loose connections and secure with tape
- Check that you haven't covered any conductive elements of your circuit with sticky tape or glue

The main tune delays when I press a button

When you use an event such as `when_pressed` to run a function, that function will run until it is finished and it will stop other code from running.

If you want to start a tune from an event then you can use `play` with `wait=False`. The function will finish and the tune will continue playing without delaying the code running in your main code.

```
sound = [ [523, 0.1], [None, 0.1], [523, 0.4] ]

def annoying_sound():
    speaker.play(sound, wait=False) # Don't delay the main code

button.when_pressed = annoying_sound
```

My wires aren't long enough now that they are in my sensory gadget

Now that you have crafted your sensory gadget, you might need extra-long wires to attach your component to your Raspberry Pi pins. Look at the instructions above to 'join jumper wires to extend them'.

My wires or components won't stay in place

Some connections are stronger than others so you might find that you need to use tape to keep your wires connected to your components or to hold your components in place on your gadget. Look at the instructions above to 'secure wires and components using tape'.

You might find a bug not listed here. Can you figure out how to fix it?

We love hearing about your bugs and how you fixed them. Use the feedback button at the bottom of this page if you found a different bug in your project.

Step 4 Checklist

Did you meet the **project brief**? Think about your project and go through the checklist below and check off the ones that apply to your project.

Does your sensory gadget:

- [ ] Have multiple different kinds of input?
- [ ] Have multiple different outputs?
- [ ] Appeal to the user and is it robust enough to be used?

Your sensory gadget could also:

- [ ] Take ergonomics such as user comfort into consideration
- [ ] Reset after user input or after a set amount of time
- [ ] Connect to a specific theme

Reflecting about how you made your sensory gadget will help you in your future projects.

Answer the questions below by drawing, writing, typing in a document, talking to someone, or in your own creative way that works best for you.

- [ ] How did you get your ideas?
- [ ] What cool new thing(s) did you learn?
- [ ] Mistakes are powerful. What mistakes did you make and how did they help you create your sensory gadget?

Now you are the creator of your own sensory gadget!

Take a moment to celebrate what you have made.

Where will you take your new powers? What will you make next?

Upgrade

If you have time, you can upgrade your project.

You can look inside each of the examples in the **Introduction** (.) to see the how the code works and get more inspiration.

Here are some other ideas you could try:

- Do some testing with your audience then improve usability
- Add more input or output components
- Continue to craft your device to improve the structure or design

The Night Sky

The original **Night Sky** project used a single RGBLED and a piece of card with holes poked through it to simulate a starry sky using projected light, which could be turned on and off by a button.

This upgraded version of the project adds another RGBLED to increase the brightness, and adds a diffuser for the LEDs to spread the light around more widely.

The card with holes has been replaced by a lasercut dodecahedron, with the logos of popular superheroes used as cutouts to project the light into the room.

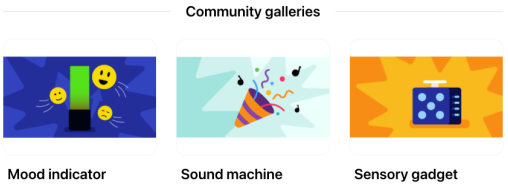
Take a look at our
Sensory gadget - Community (https://wke.lt/w/s/qX5TaK) gallery to see projects created by our community members.

Share

If you are in a club, why not share your project with friends? You could also show your family how your project works.

Inspire the Raspberry Pi Foundation community with your project!

To submit a video of your gadget to our **Sensory gadget - Community** (https://wke.lt/w/s/qX5TaK) gallery, please complete **this form** (https://form.raspberrypi.org/f/community-project-submissions). Remember to protect your identity by making sure that people do not appear in your video.



The Night Sky

The original **Night Sky** project used a single RGBLED and a piece of card with holes poked through it to simulate a starry sky using projected light, which could be turned on and off by a button.

This upgraded version of the project adds another RGBLED to increase the brightness, and adds a diffuser for the LEDs to spread the light around more widely.

The card with holes has been replaced by a lasercut dodecahedron, with the logos of popular superheroes used as cutouts to project the light into the room.

Take a look at our
Sensory gadget - Community (https://wke.lt/w/s/qX5TaK) gallery to see projects created by our community members.

What next?

You have reached the end of the **Introduction to the Raspberry Pi Pico** (https://projects.raspberrypi.org/en/pathways/pico-intro) path!

Now you can bring all your ideas to life and use the Raspberry Pi Pico to make more cool stuff. If you need a reminder of what you have learnt, you can go to the **Introduction to the Pico guide** (https://projects.raspberrypi.org/en/projects/introduction-to-the-pico).

Enter Coolest Projects

Check out **Coolest Projects** (https://coolestprojects.org/), the world's leading technology showcase for young people! On the **Coolest Projects website** (https://coolestprojects.org/), you can find out more about the project categories and see when project registration is open, then get ready to register your project!

Your project doesn't have to be finished — prototypes and works in progress are welcome too! When you have entered your project, your creation will be showcased in the Coolest Projects online gallery, for people all over the world to see! Join other young people in celebrating and recognising each other's achievements as a community.

You can use the **Introduction to the Raspberry Pi Pico** (https://projects.raspberrypi.org/en/pathways/pico-intro) path and the Coolest Projects **How to make a project** (https://coolestprojects.org/2020/03/31/how-to-make-a-project-workbook-and-additional-resources/) resources to help you plan and build your original project.

Or, you can explore our **other Raspberry Pi Pico projects** (https://projects.raspberrypi.org/en/projects?hardware%5B%5D=pico) and try them out.
