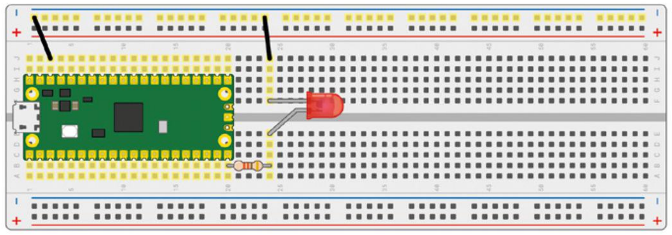
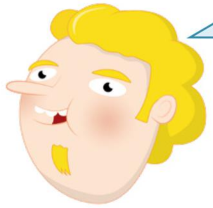

▲ **Figure 4-4:** The finished circuit, with an LED and a resistor

Controlling an external LED in MicroPython is no different to controlling your Pico's internal LED: only the pin number changes. If you closed Thonny, reopen it and load your **Blink.py** program from earlier in the chapter. Find the line:

```
led_onboard = machine.Pin(25, machine.Pin.OUT)
```

Edit the pin number, changing it from 25 – the pin connected to your Pico's internal LED – to 15, the pin to which you connected the external LED. Also edit the name you created: you're not using the on-board LED any more, so have it say `led_external` instead. You'll also have to change the name elsewhere in the program, until it looks like this:

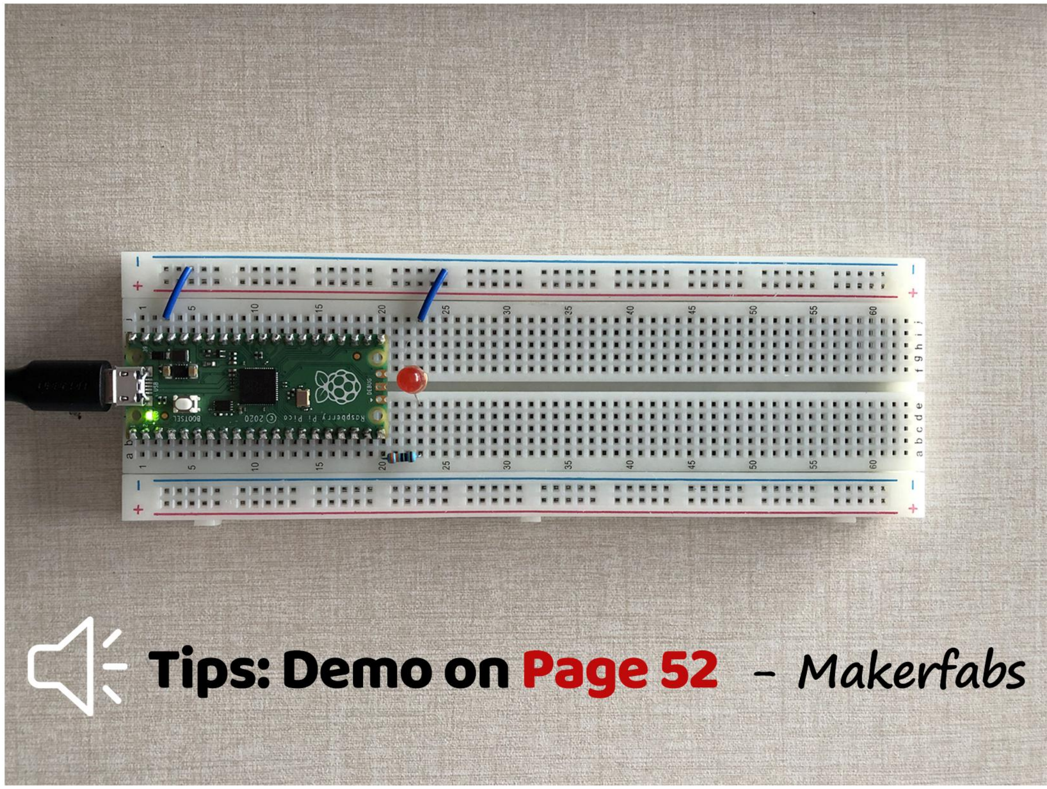
```
import machine
import utime

led_external = machine.Pin(15, machine.Pin.OUT)

while True:
    led_external.toggle()
    utime.sleep(5)
```



NAMING CONVENTIONS

You don't really *need* to change the name in the program: it would run just the same if you'd left it at `led_onboard`, as it's only the pin number which truly matters. When you come back to the program later, though, it would be very confusing to have an object named `led_onboard` which lights up an external LED – so try to get into the habit of making sure your names match their purpose!

RPI Pico: Controlling an LED in MicroPython



CHALLENGE: MULTIPLE LEDS

Can you modify the program to light up both the on-board and external LEDs at the same time? Can you write a program which lights up the on-board LED when the external LED is switched off, and vice versa? Can you extend the circuit to include more than one external LED? Remember, you'll need a current-limiting resistor for every LED you use!

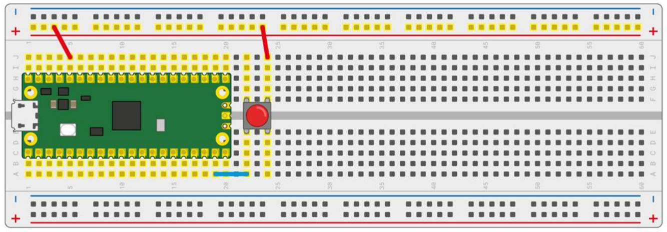
Inputs: reading a button

Outputs like LEDs are one thing, but the 'input/output' part of 'GPIO' means you can use pins as inputs too. For this project, you'll need a breadboard, male-to-male jumper wires, and a push-button switch. If you don't have a breadboard, you can use female-to-female (F2F) jumper wires, but the button will be much harder to press without accidentally breaking the circuit.

Remove any other components from your breadboard except your Pico, and begin by adding the push-button switch. If your push-button has only two legs, make sure they're in different-numbered rows on the breadboard somewhere below your Pico. If it has four legs, turn it so the sides the legs come from are along the breadboard's rows and the flat leg-free sides are at the top and bottom before pushing it home so it straddles the centre divide of the breadboard.

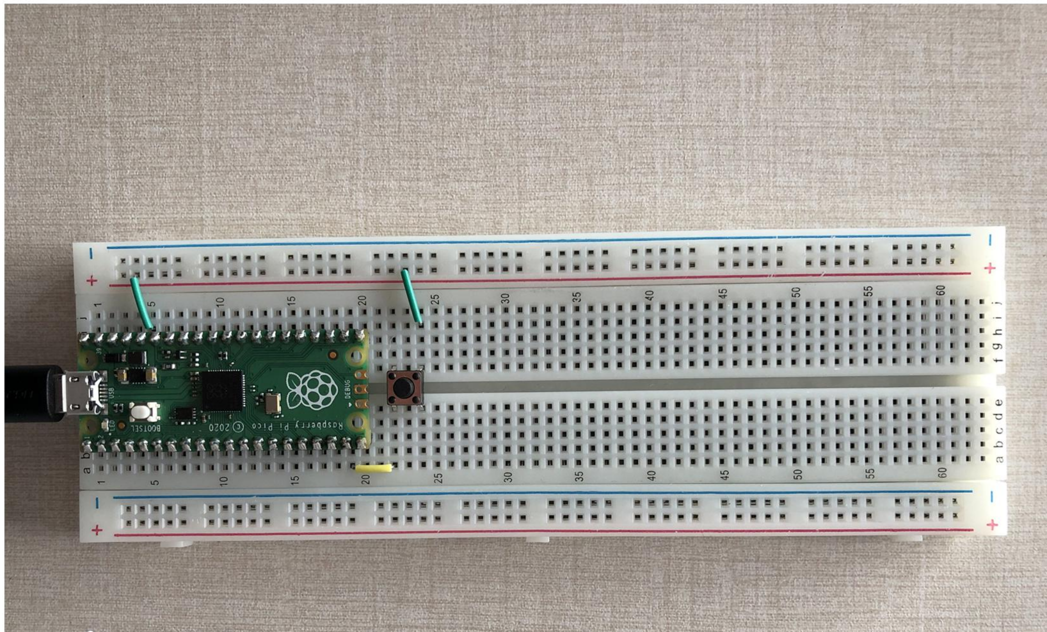
Connect the positive power rail of your breadboard to your Pico's 3V3 pin, and from there to one of the legs of the switch; then connect the other leg to pin GP14 on your Pico – it's the one just above the pin you used for the LED project, and should be in row 19 of your breadboard.

If you're using push-button with four legs, your circuit will only work if you use the correct pair of legs: the legs are connected in pairs, so you need to either use the two legs on the same side or (as seen in **Figure 4-5**) diagonally opposite legs.



▲ **Figure 4-5:** Wiring a four-leg push-button switch to GP14

RPI Pico: Reading an Button in MicroPython



Tips: Demo on Page 53 - Makerfabs

input on GP14. Because the input is using a pull-down resistor, this value will be 0 – letting you know the button isn't pushed.

Hold down the button with your finger, and press the Run icon again. This time, you'll see the value 1 printed to the Shell: pushing the button has completed the circuit and changed the value read from the pin.

To read the button continuously, you'll need to add a loop to your program. Edit the program so it reads as below:

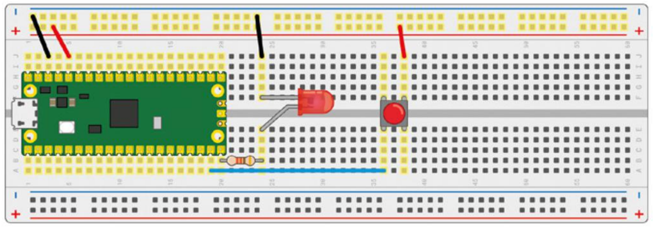
```
import machine
import utime

button = machine.Pin(14, machine.Pin.IN, machine.Pin.PULL_DOWN)

while True:
    if button.value() == 1:
        print("You pressed the button!")
        utime.sleep(2)
```

Click the Run button again. This time, nothing will happen until you press the button; when you do, you'll see a message printed to the Shell area. The delay, meanwhile, is important: remember, your Pico runs a lot faster than you can read, and without the delay even a brief press of the button can print hundreds of messages to the Shell!

You'll see the message print every time you press the button. If you hold the button down for longer than the two-second delay, it will print the message every two seconds until you let go of the button.

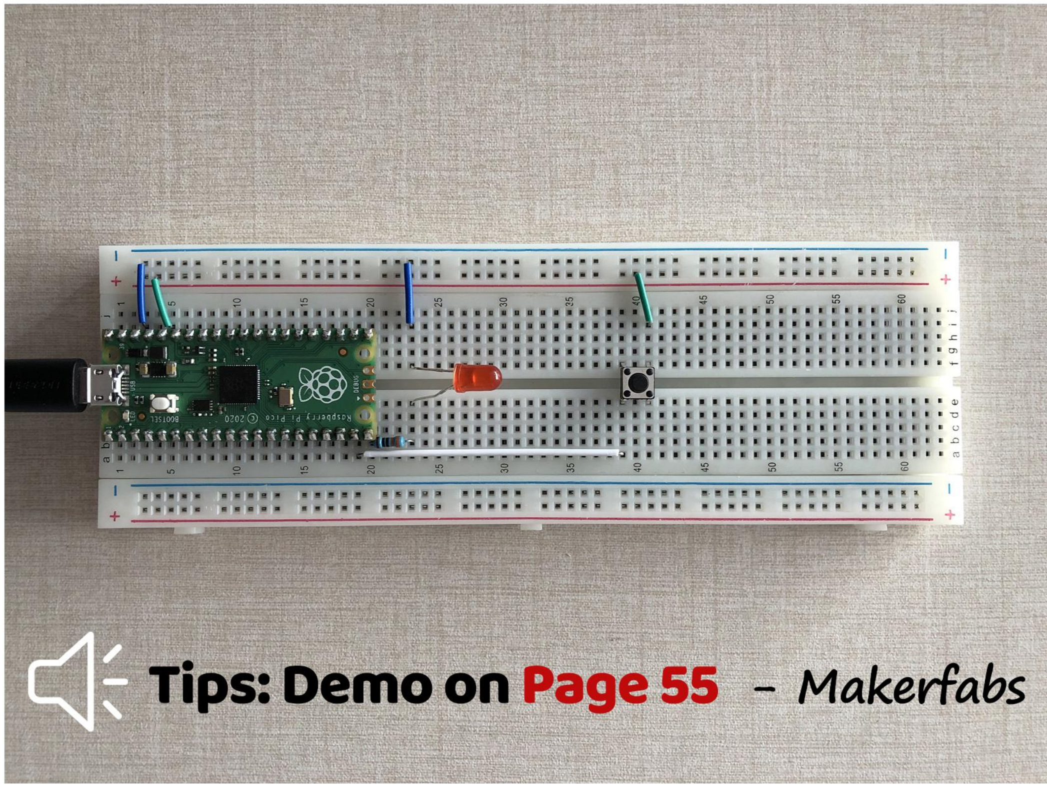
Inputs and outputs: putting it all together

Most circuits have more than one component, which is why your Pico has so many GPIO pins. It's time to put everything you've learned together to build a more complex circuit: a device which switches an LED on and off with a button.



▲ **Figure 4-6:** The finished circuit, with both a button and an LED

RPI Pico: Controlling an LED With a Button



A simple traffic light

Start by building a simple traffic light system, as shown in **Figure 5-1**. Take your red LED and insert it into the breadboard so it straddles the centre divide. Use one of the 330 Ω resistors, and a jumper wire if you need to make a longer connection, to connect the longer leg – the anode – of the LED to the pin at the bottom-left of your Pico as seen from the top with the micro USB cable uppermost, GP15. If you're using a numbered breadboard and have your Pico inserted at the very top, this will be breadboard row 20.

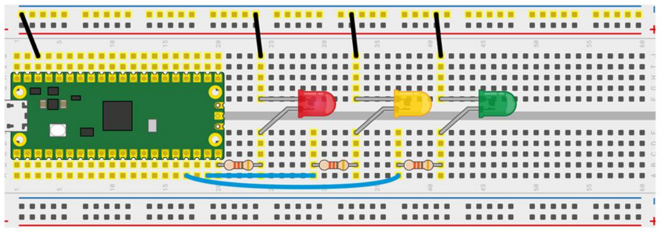
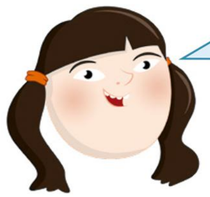


Figure 5-1: A basic three-light traffic light system



WARNING

Always remember that an LED needs a current-limiting resistor before it can be connected to your Pico. If you connect an LED without a current-limiting resistor in place, the best outcome is the LED will burn out and no longer work; the worst outcome is it could do the same to your Pico.

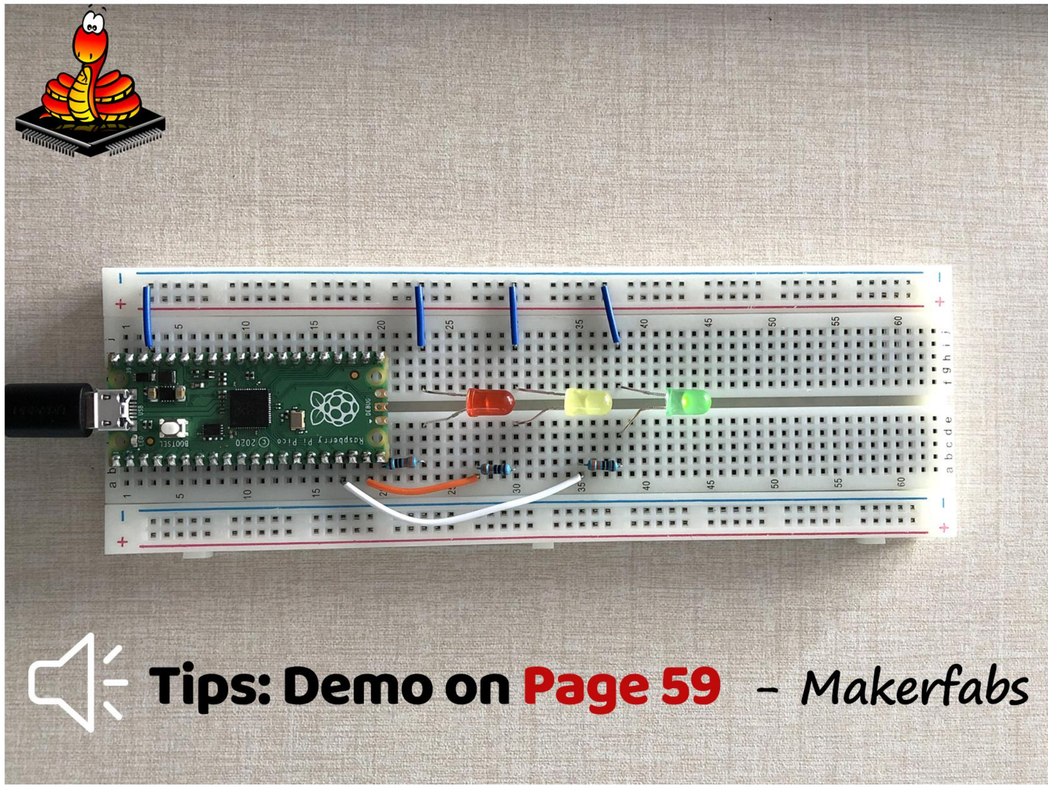
Take a jumper wire and connect the shorter leg – the cathode – of the red LED to your breadboard's ground rail. Take another, and connect the ground rail to one of your Pico's ground (GND) pins – in **Figure 5-1**, we've used the ground pin on row three of the breadboard.

You've now got one LED connected to your Pico, but a real traffic light has at least two more for a total of three: a red light to tell the traffic to stop, an amber or yellow light to tell the traffic the light is about to change, and a green LED to tell the traffic it can go again.

Take your amber or yellow LED and wire it to your Pico in the same way as the red LED, making sure the shorter leg is the one connecting to the ground rail of the breadboard and that you've got the 330 Ω resistor in place to protect it. This time, though, wire the longer leg – via the resistor – to the pin next to the one to which you wired the red LED, GP14.

Finally, take the green LED and wire it up the same way again – remembering the 330 Ω resistor – to pin GP13. This isn't the pin right next to pin GP14, though – that pin is a ground (GND) pin, which you can see if you look closely at your Pico: the ground pins all have a square shape to their pads, while the other pins are round.

RPI Pico: Basic 3-light Traffic Light System



Tips: Demo on Page 59 – Makerfabs

```
    utime.sleep(5)
    led_green.value(0)
    led_amber.value(1)
    utime.sleep(5)
    led_amber.value(0)
```

Click the Run icon and save your program to your Pico as **Traffic_Lights.py.** Watch the LEDs: first the red LED will light up, telling the traffic to stop; next, the amber LED will come on to warn drivers the lights are about to change; next both LEDs switch off and the green LED comes on to let traffic know it can pass; then the green LED goes off and the amber one comes on to warn drivers the lights are about to change again; finally, the amber LED goes off – and the loop restarts from the beginning, with the red LED coming on.

The pattern will loop until you press the Stop button, because it forms an infinite loop. It's based on the traffic light pattern used in real-world traffic control systems in the UK and Ireland, but sped up – giving cars just five seconds to pass through the lights wouldn't let the traffic flow very freely!

Real traffic lights aren't just there for road vehicles, though: they are also there to protect pedestrians, giving them an opportunity to cross a busy road safely. In the UK, the most common type of these lights are known as *pedestrian-operated user-friendly intelligent crossings* or *puffin crossings*.

To turn your traffic lights into a puffin crossing, you'll need two things: a push-button switch, so the pedestrian can ask the lights to let them cross the road; and a buzzer, so the pedestrian knows when it's their turn to cross. Wire those into your breadboard as in **Figure 5-2**, with the switch wired to pin GP16 and the 3V3 rail of your breadboard, and the buzzer wired to pin GP12 and the ground rail of your breadboard.

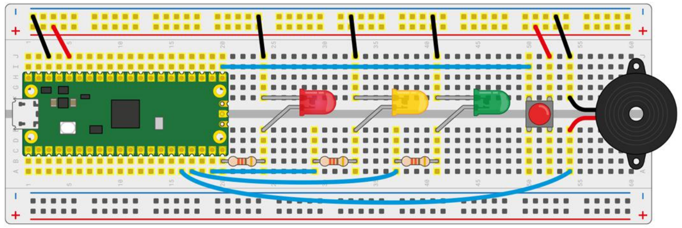
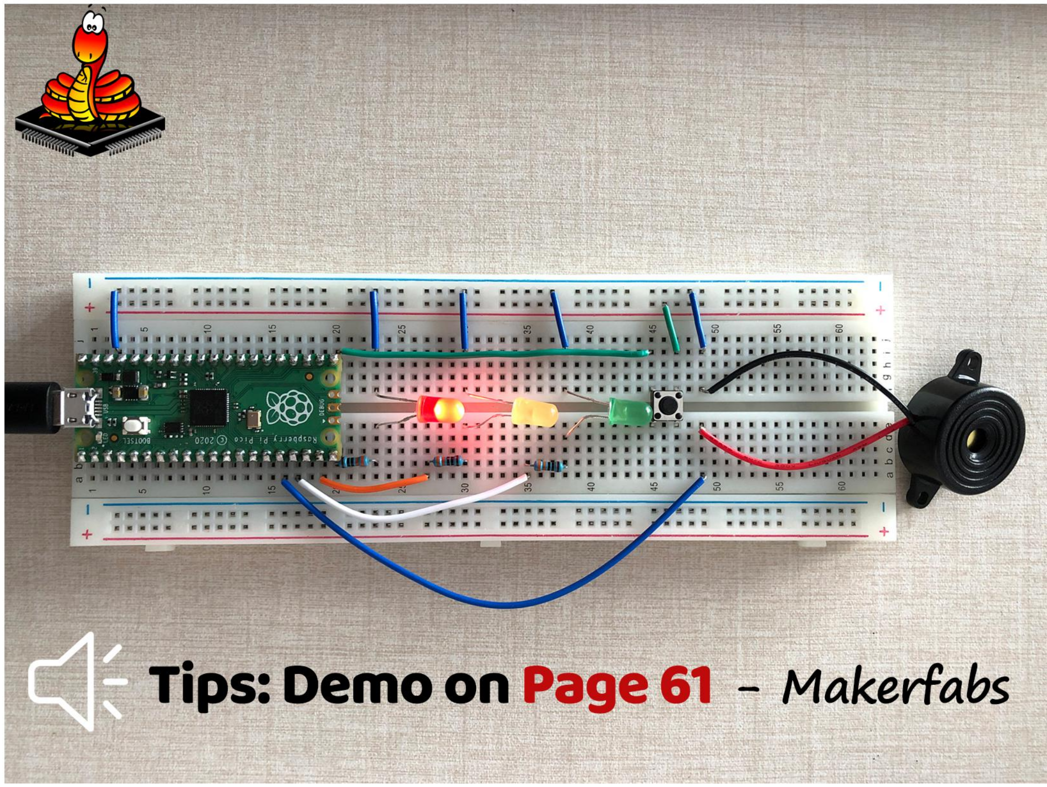


Figure 5-2: A puffin crossing traffic light system

If you run your program again, you'll find the button and buzzer do nothing. That's because you haven't yet told your program how to use them. In Thonny, go back to the lines where you set up your LEDs and add the following two new lines below:

RPI Pico: A Puffin Crossing Traffic Light System



```
    print("Your reaction time was " + str(timer_reaction) +
" milliseconds!")

  led.value(1)
  utime.sleep(urandom.uniform(5, 10))
  led.value(0)
  timer_start = utime.ticks_ms()
  button.irq(trigger=machine.Pin.IRQ_RISING, handler=button_handler)
```

Click the Run button again, wait for the LED to go out, and push the button. This time, instead of a report on the pin which triggered the interrupt, you'll see a line telling you how quickly you pushed the button – a measurement of your reaction time. Click the Run button again and see if you can push the button more quickly this time – in this game, you're trying for as low a score as possible!

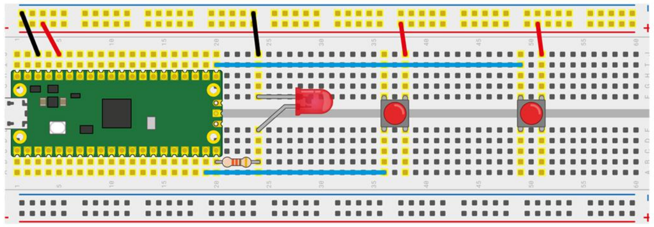
CHALLENGE: CUSTOMISATION

Can you tweak your game so that the LED stays lit for a longer time? What about staying lit for a shorter time? Can you personalise the message that prints to the Shell area, and add a second message congratulating the player?

A two-player game

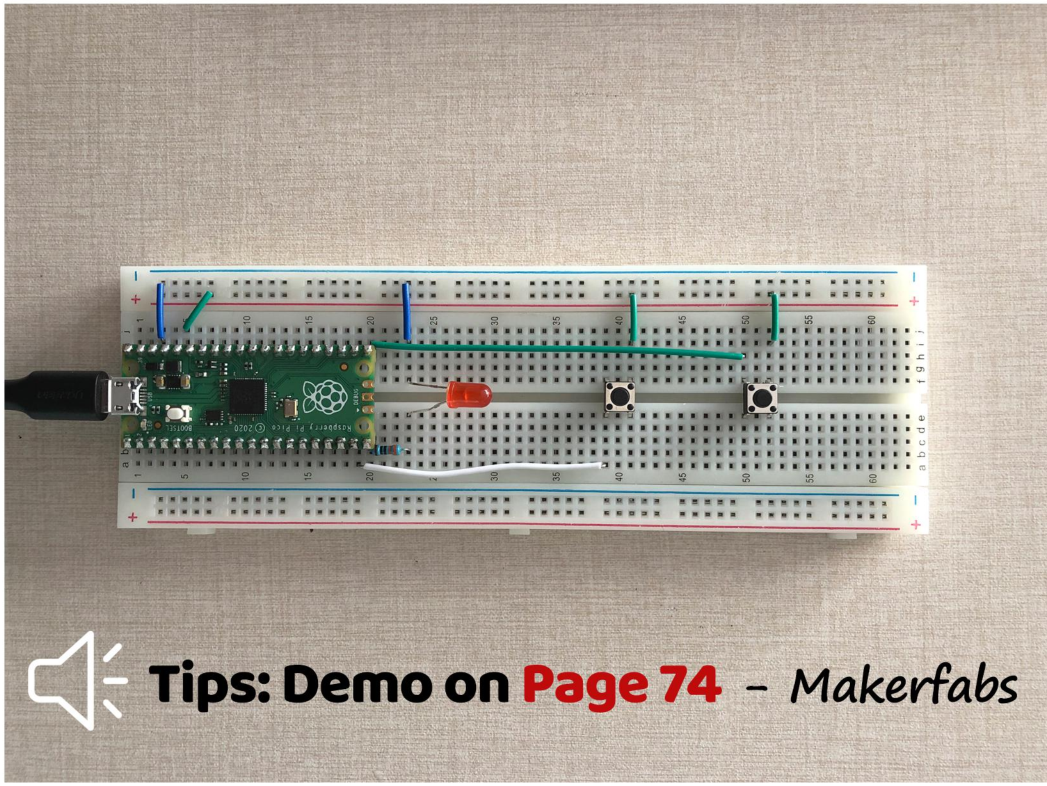
Single-player games are fun, but getting your friends involved is even better. You can start by inviting them to play your game and comparing your high – or, rather, low – scores to see who has the quickest reaction time. Then, you can modify your game to let you go head-to-head!

Start by adding a second button to your circuit. Wire it up the same as the first button, with one leg going to the power rail of your breadboard but with the other going to pin GP16 – the pin across the board from GP14 where the LED is connected, at the opposite corner of your Pico.



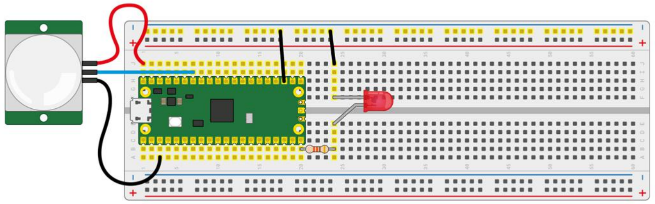
▲ **Figure 6-2:** The circuit for a two-player reaction game

RPI Pico: Two-player Reaction Game



Tips: Demo on Page 74 - Makerfabs

Start by wiring an LED, of any colour, to your Pico as shown in **Figure 7-2**. The longer leg, the anode, needs to connect to pin GP15 via a 330 Ω resistor – remember that without this resistor in place to limit the amount of current passing through the LED, you can damage both the LED and your Pico. The shorter leg, the cathode, needs to be wired to one of your Pico's ground pins – use your breadboard's ground rail and two male-to-male (M2M) jumper wires for this.



▲ **Figure 7-2:** Adding an LED to the burglar alarm

This time, you're going to handle delays in your program rather than relying on the delay built into the PIR sensor. Go to the top of your program and, just below the line **`import machine`**, add the following:

```
import utime
```

Next, add a new line just below where you set up the PIR sensor's pin:

```
led = machine.Pin(15, machine.Pin.OUT)
```

That's enough to configure the LED, but you'll need to make it light up. Add the following new line to your interrupt handler function – remembering that, like all the lines in the function, it will need to be indented by four spaces so MicroPython knows it's part of the nested code:

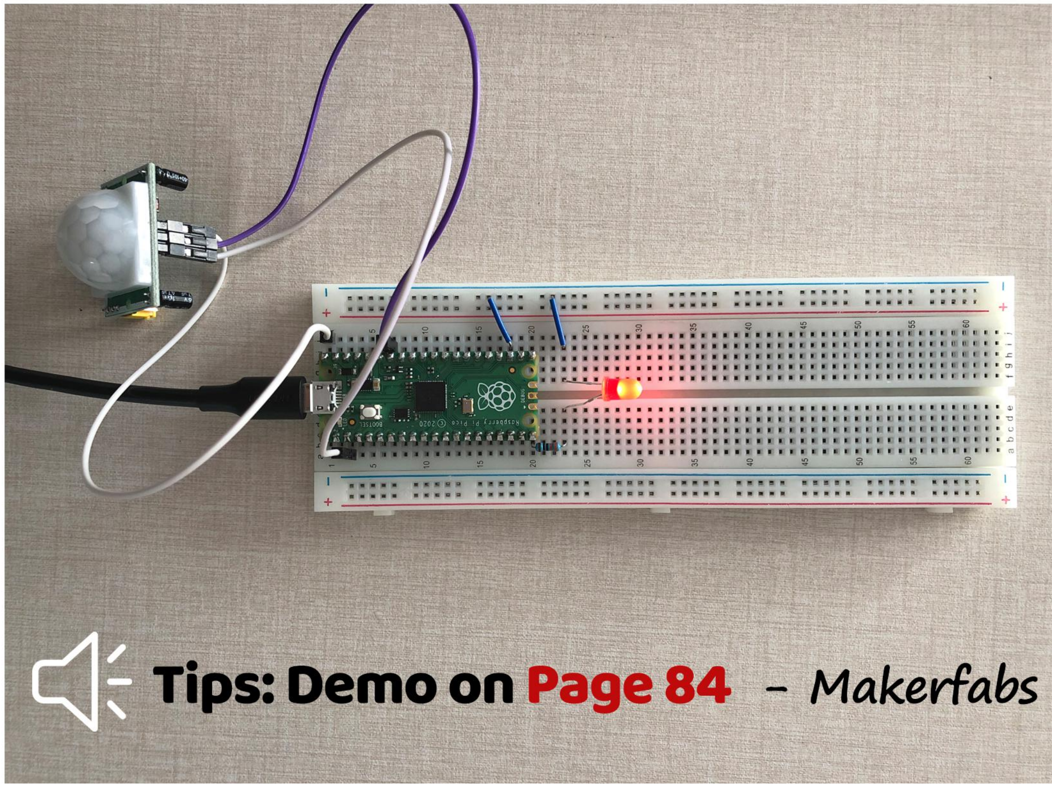
```
for i in range(50):
```

Press **ENTER** at the end of this line and you'll notice Thonny has automatically added another four spaces to make an eight-space indentation. That's because you've just created a finite loop, one which will run 50 times. The letter **`i`** represents an *increment*, a value which goes up each time the loop runs, and which is populated by the instruction **`range(50)`**.

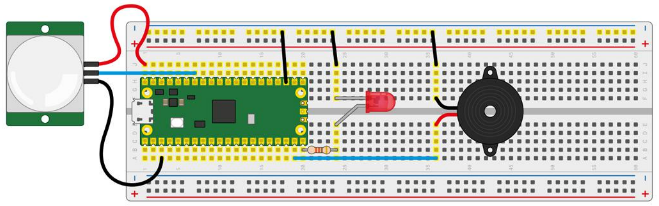
Give your new loop something to do, remembering that these lines will need to be indented by eight spaces – which Thonny will have done automatically – as they form both part of the loop you just opened and the interrupt handler function:

```
led.toggle()
```

RPI Pico: Adding an LED to the Burglar Alarm



Tips: Demo on Page 84 – Makerfabs



▲ **Figure 7-3:** Wiring a two-wire piezoelectric buzzer

If your buzzer has three pins, connect the leg marked with a minus symbol (-) or the letters GND to the ground rail of your breadboard, the pin marked with S or SIGNAL to pin GP14 on your Pico, and the remaining leg – which is usually the middle leg – to the 3V3 pin on your Pico.

If you run your program now, nothing will change: the buzzer will only make a sound when it receives power from your Pico's GPIO pins. Go back to the top of your program and set the buzzer up just below where you set the LED up:

```
buzzer = machine.Pin(14, machine.Pin.OUT)
```

Next, change your interrupt handler to add a new line below **led.toggle()** – remembering that, as it's part of both the loop and the handler function, it will need to be indented by eight spaces:

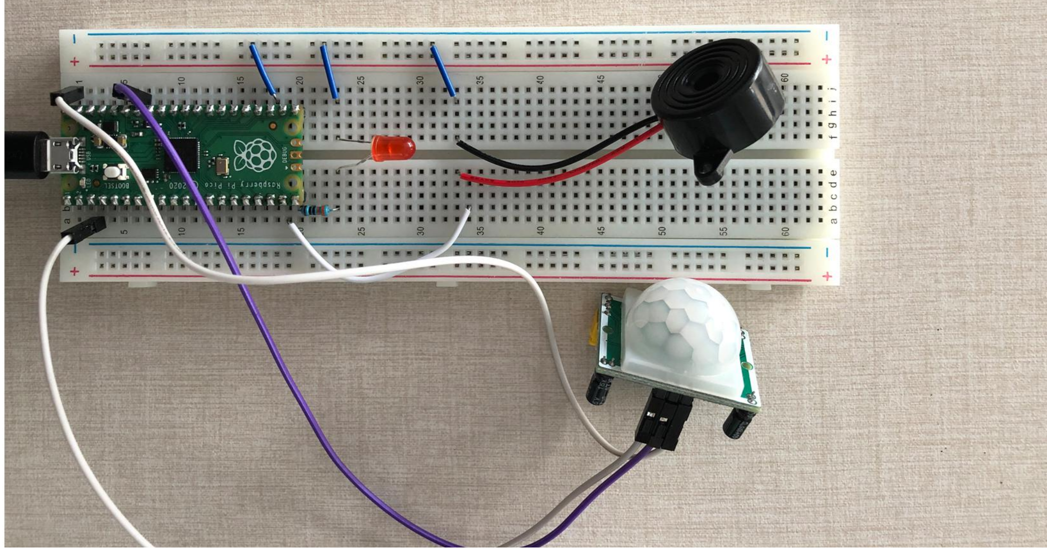
```
buzzer.toggle()
```

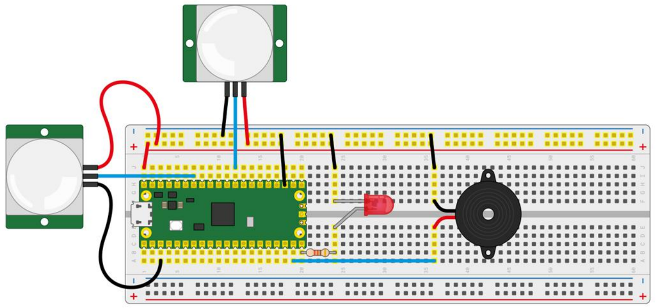
Your program will now look like this:

```
import machine
import utime

sensor_pir = machine.Pin(28, machine.Pin.IN, machine.Pin.PULL_DOWN)
led = machine.Pin(15, machine.Pin.OUT)
buzzer = machine.Pin(14, machine.Pin.OUT)

def pir_handler(pin):
    print("ALARM! Motion detected!")
    for i in range(50):
        led.toggle()
        buzzer.toggle()
        utime.sleep_ms(100)

sensor_pir.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_handler)
```

RPI Pico: Wiring a 2-wire Piezoelectric Buzzer

Tips: Demo on Page 87 – Makerfabs





▲ **Figure 7-4:** Adding a second PIR sensor to cover another room

```
sensor_pir2.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_handler)
```

Remember that you can have multiple interrupts with a single handler, so there's no need to change that part of your program.

Click Run, and wave your hand over the first PIR sensor: you'll see the alert message, the LED flash, and the buzzer sound as normal. Wait for them to finish, then wave your hand over the second PIR sensor: you'll see your burglar alarm respond in exactly the same way.

To make your alarm really smart, you can customise the message depending on which pin was responsible for the interrupt – and it works exactly the same way as in the two-player reaction game you wrote earlier.

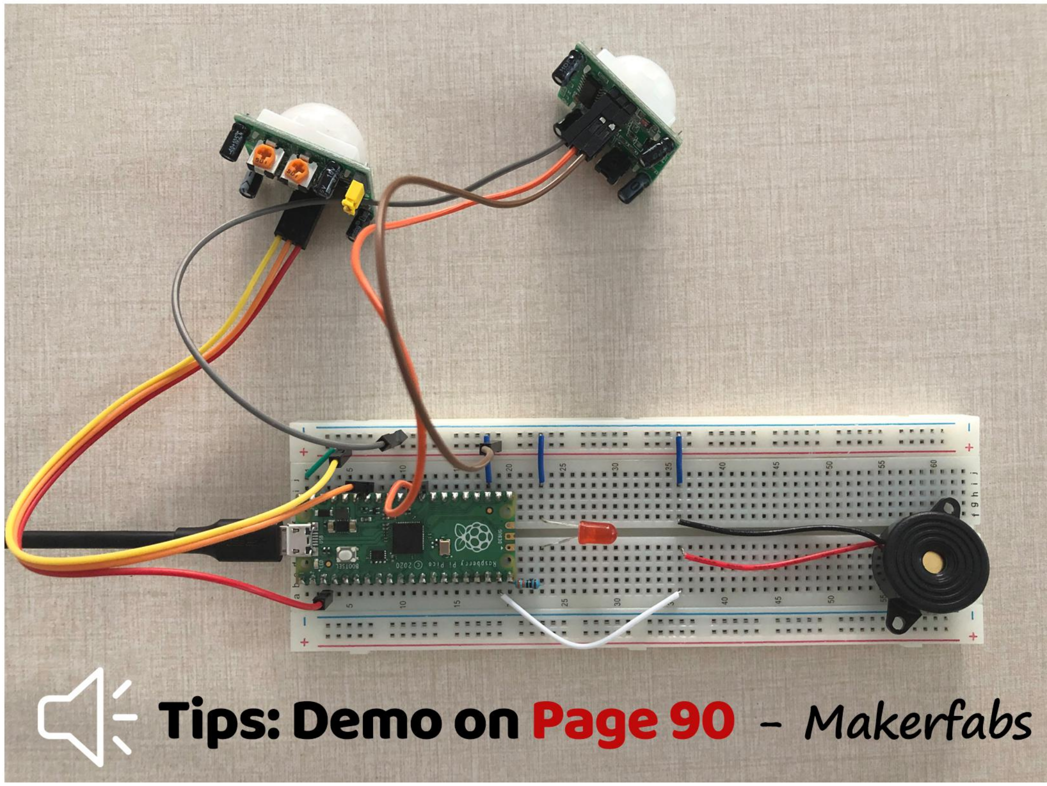
Go back to your interrupt handler and modify it so it looks like:

```
def pir_handler(pin):
    if pin is sensor_pir:
        print("ALARM! Motion detected in bedroom!")
    elif pin is sensor_pir2:
        print("ALARM! Motion detected in living room!")
    for i in range(50):
        led.toggle()
        buzzer.toggle()
        utime.sleep_ms(100)
```

Just as in the reaction game project in **Chapter 6,** this code uses the fact that an interrupt reports which pin it was triggered by: if the PIR sensor attached to pin GP28 is responsible, it will print one message; if it was the PIR sensor attached to pin GP22, it will print another.

Your finished program will look like this:

RPI Pico: Adding 2 PIR + 1 LED + 1 Buzzer



to it. That's because a 10 kΩ resistor isn't strong enough to drop the 3V3 pin's output to 0 V. You could look for a bigger potentiometer with a higher maximum resistance, or you could simply wire your existing potentiometer up as a *voltage divider*.

A potentiometer as a voltage divider

The unused pin on your potentiometer isn't there for show: adding a connection to that pin to your circuit completely changes how the potentiometer works. Click the Stop icon to stop your program, and grab two male-to-male (M2M) jumper wires. Use one to connect the unused pin of your potentiometer to your breadboard's ground rail as shown in **Figure 8-3**. Take the other and connect the ground rail to a GND pin on your Pico.

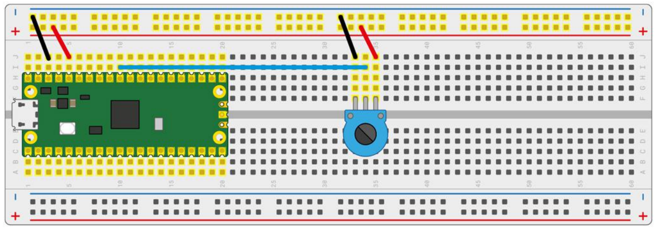


Figure 8-3: Wiring the potentiometer as a voltage divider

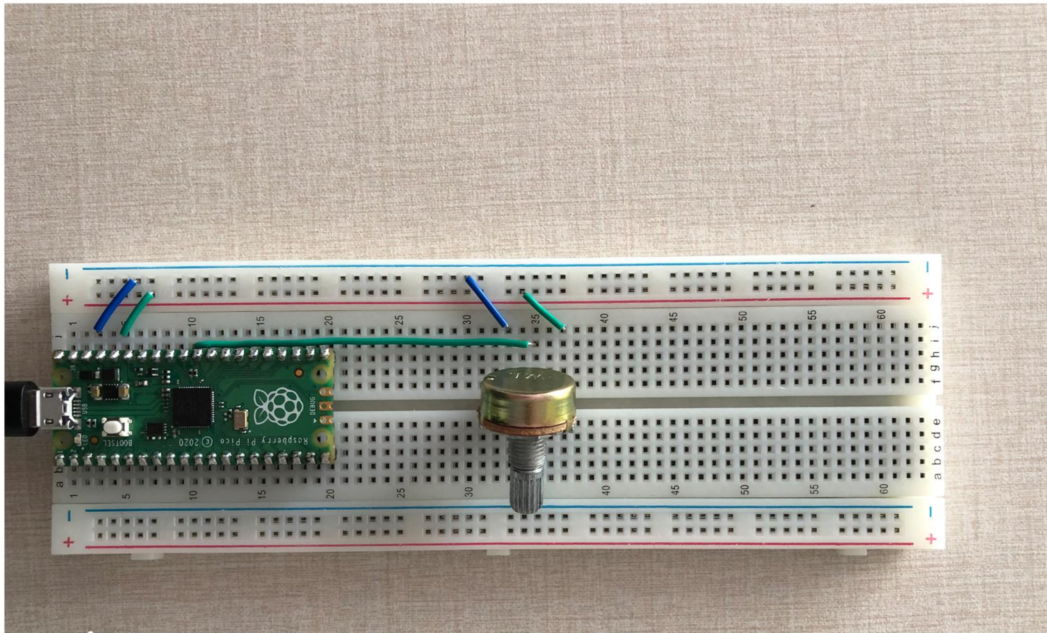
Click the Run icon to restart your program. Turn the potentiometer knob again, all the way one direction then all the way the other. Watch the values that are printed to the Shell area: unlike before, they're now going from near-zero to nearly a full 65,535 – but why?

Adding the ground connection to the other end of the potentiometer's conductive strip has created a voltage divider: whereas before the potentiometer was simply acting as a resistor between the 3V3 pin and the analogue input pin, it's now dividing the voltage between the 3.3 V output by the 3V3 pin and the 0 V of the GND pin. Turn the knob fully one direction, you'll get 100 percent of the 3.3V; turn it fully the other way, 0 percent.

ZERO'S THE HARDEST NUMBER

If you can't get your Pico's analogue input to read exactly zero or exactly 65,535, don't worry – you haven't done anything wrong! All electronic components are built with a *tolerance*, which means any claimed value isn't going to be precise. In the case of the potentiometer, it will likely never reach exactly 0 or 100 percent of its input – but it will get you very close!

RPI Pico: A Potentiometer as a Voltage Divider



Tips: Demo on Page 96 – Makerfabs

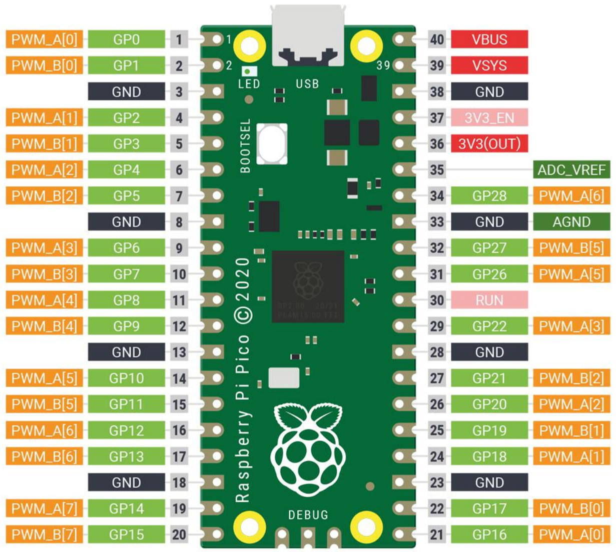


Figure 8-4: The pulse-width modulation pins

If that sounds confusing, don't worry: all it means is that you need to make sure you keep track of the PWM slices and outputs you're using, making sure to only connect to pins with a letter and number combination you haven't already used. If you're using PWM_A[0] on pin GP0 and PWM_B[0] on pin GP1, things will work fine, and will continue to work if you add PWM_A[1] on pin GP2; if you try to use the PWM channel on pin GP0 and pin GP16, though, you'd run into problems as they're both connected to PWM_A[0].

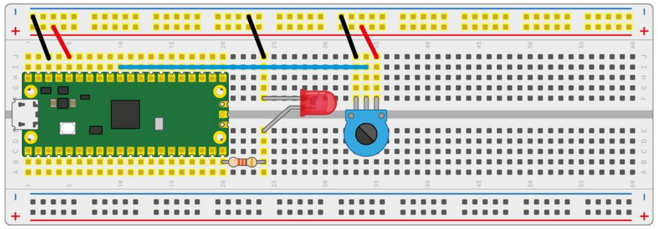
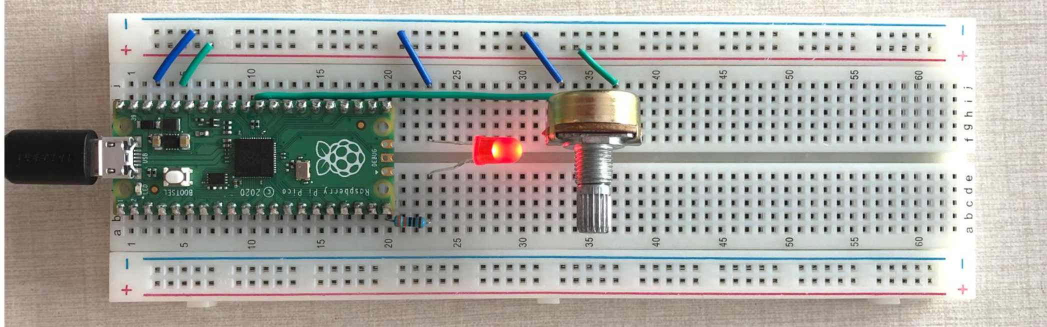


Figure 8-5: Adding an LED

RPI Pico: Fading an LED with PWM



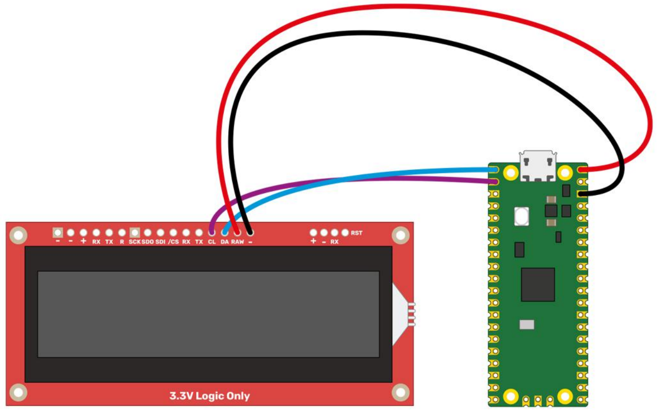
Tips: Demo on Page 101 – Makerfabs

These have to connect to specific pins on the Pico. There are a few choices; take a look at the pinout diagram for the options (**Figure 10-1**). There are two I2C buses (I2C0 and I2C1), and you can use either or both. In our example, we'll use I2C0 – with GP0 for SDA, and GP1 for SCL.

To demonstrate the protocols, we'll use a SerLCD module from SparkFun. This has the advantage that it has both I2C and SPI interfaces, so we can see the differences between the two methods with the same hardware.

This LCD can display two lines, each with up to 16 characters. It's a useful device for outputting bits of information about our system. Let's take a look at how to use it.

Wiring I2C is just a case of connecting the SDA pin on the Pico with the SDA pin on the LCD and the same for the SCL. Because of the way I2C handles communication, there also needs to be a resistor connecting SDA to 3.3 V and SCL to 3.3 V. Typically these are about 4.7 kΩ. However, with our device, these resistors are already included, so we don't need to add any extra ones.

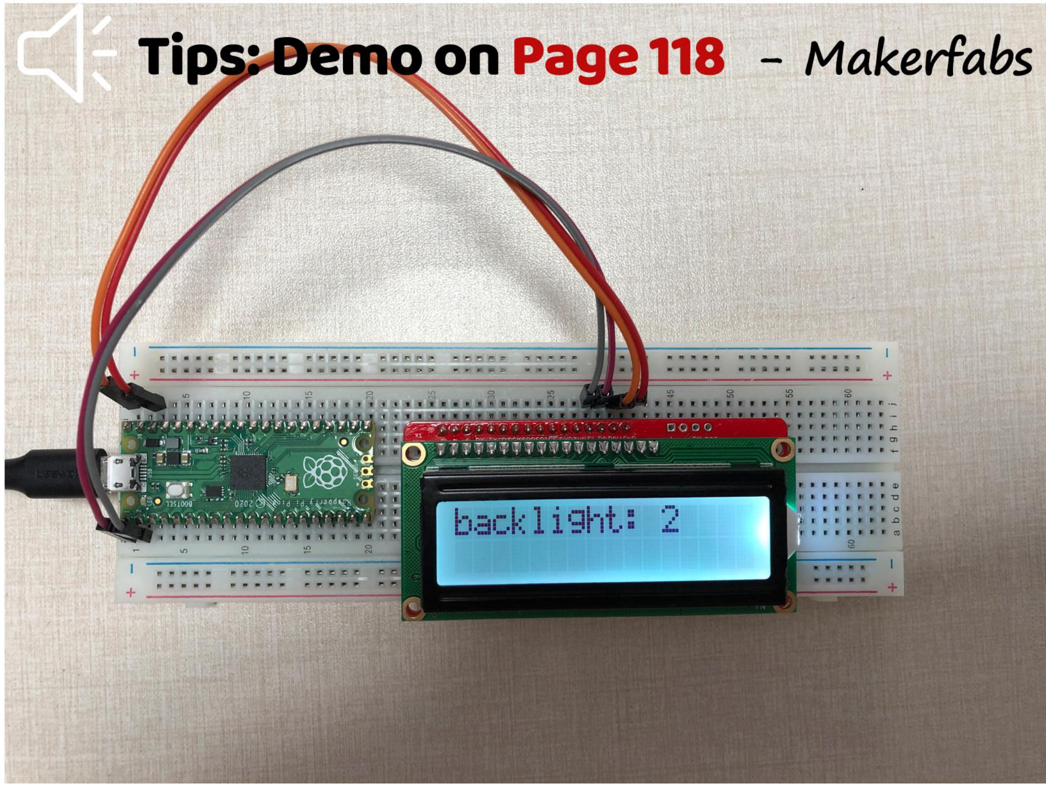


▲ **Figure 10-2:** Wiring up a SerLCD module for I2C

With this wired up (see **Figure 10-2**), displaying information on the screen is as simple as:

```
import machine
sda=machine.Pin(0)
scl=machine.Pin(1)
i2c=machine.I2C(0,sda=sda, scl=scl, freq=400000)
i2c.writeto(114, '\x7C')
i2c.writeto(114, '\x2D')
i2c.writeto(114, "hello world")
```

RPI Pico: Wiring up a SerLCD Moule for I2C



CS line to enable an SPI peripheral and pull it low to disable it. To confuse things slightly, this particular device doesn't have CS, but /CS which stands for NOT CS – in other words, it's the opposite of CS, so you bring it low to enable the LCD and high to disable it. You could connect the CS to a GPIO pin and toggle this on and off to enable and disable the display, but since we only have one device, we can simply connect it to ground to keep it enabled (**Figure 10-3**).

So, with the SerLCD's power lines connected to VBUS and GND, we just need to connect its SDO to Pico's MISO (GP4 / SPI0 RX), SDI to MOSI (GP3 / SPI0 TX), SCK to SCLK (GP2 / SPI0 SCK), and /CS to GND.

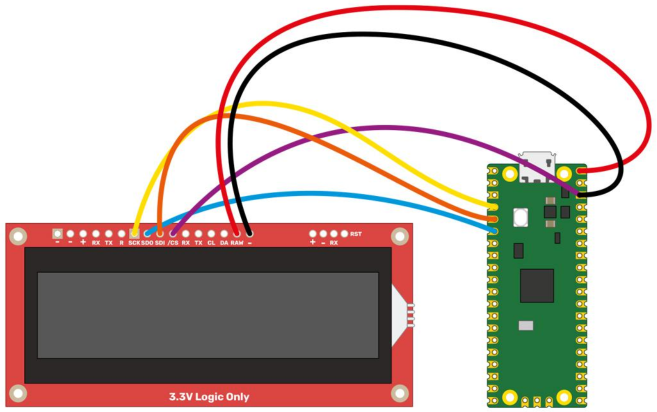
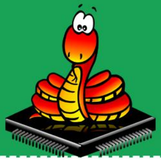


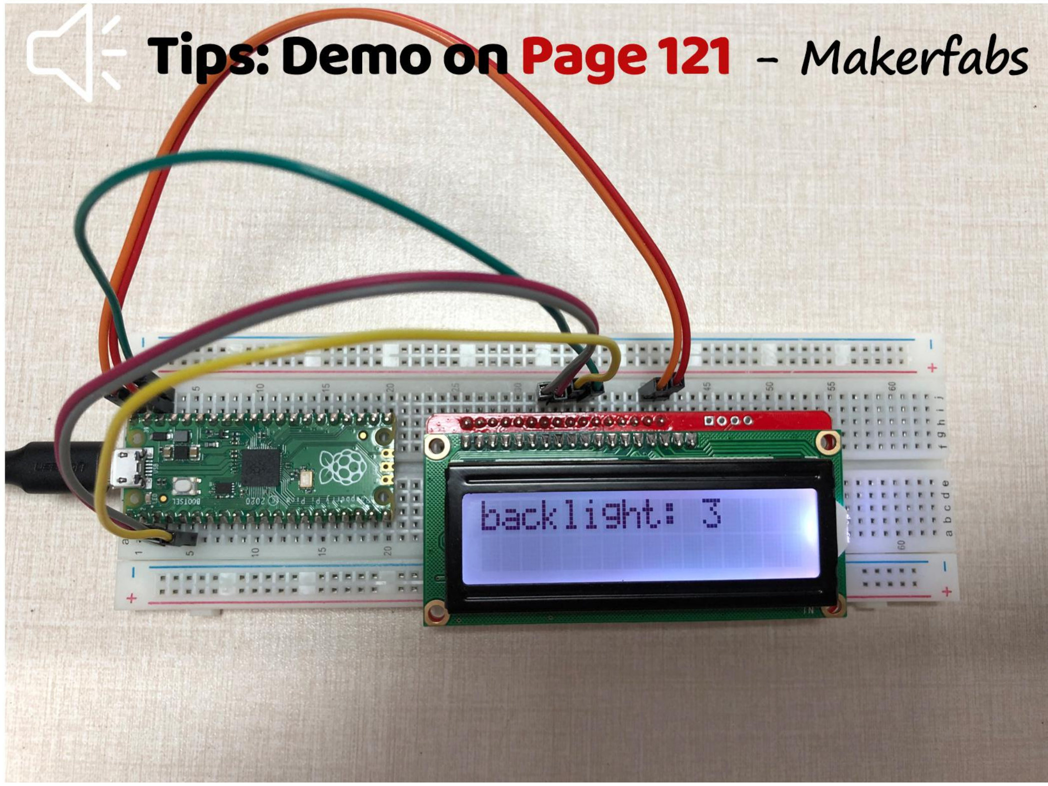
Figure 10-3: Wiring up a SerLCD module for SPI

SPI TERMINOLOGY

SPI requires four connections: one that takes data from the master device to the slave device, another that takes data in the opposite direction, plus power and ground. Two data wires mean that data can travel in both directions at the same time. These are usually called Master Out Slave In (MOSI) and Master In Slave Out (MISO). However, you will come across them with different names. If you look at the Raspberry Pi Pico pinout (Appendix B), they're referred to as SPI TX (Transmit) and SPI RX (Receive). This is because Pico can be either a master or slave device, so whether these connections are MOSI or MISO depends on the current function of Pico. On the LCD we're using, they're labelled SDI (Serial Data In) and SDO (Serial Data Out).

RPI Pico: Wiring Up a SerLCD for SPI





```
sm1 = StateMachine(1, led_quarter_brightness, freq=10000, set_base=Pin(25))
```

The parameters here are:

- The state machine number
- The PIO program to load
- The frequency (which must be between 2000 and 125000000)
- The GPIO pin that the state machine manipulates

There are some additional parameters that you'll see in other programs that we don't need here.
Once you've created your state machine, you can start and stop it using the **active** method with 1 (to start) or 0 (to stop). In our loop, we cycle through the three different state machines.

A real example

The previous example was a little contrived, so let's take a look at a way of using PIO with a real example. WS2812B LEDs (sometimes known as NeoPixels) are a type of light that contains three LEDs (one red, one green, and one blue) and a small microcontroller. They're controlled by a single data wire with a timing-dependent protocol that's hard to bit-bang.

Wiring your LED strip is simple, as shown in **Figure C-1**. Depending on the manufacturer of your LED strip, you may have the wires already connected, you may have a socket that you can push header wires in, or you may need to solder them on yourself.

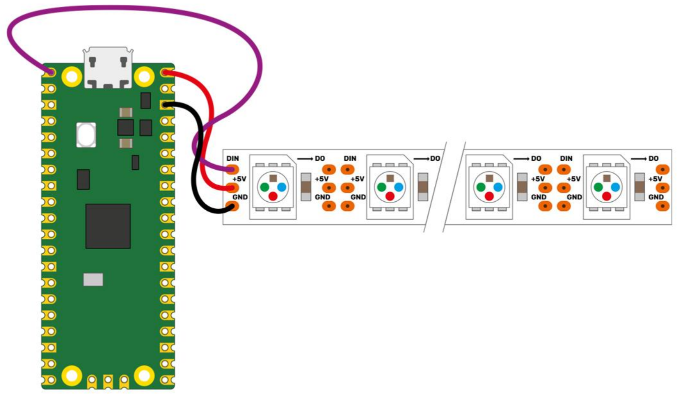
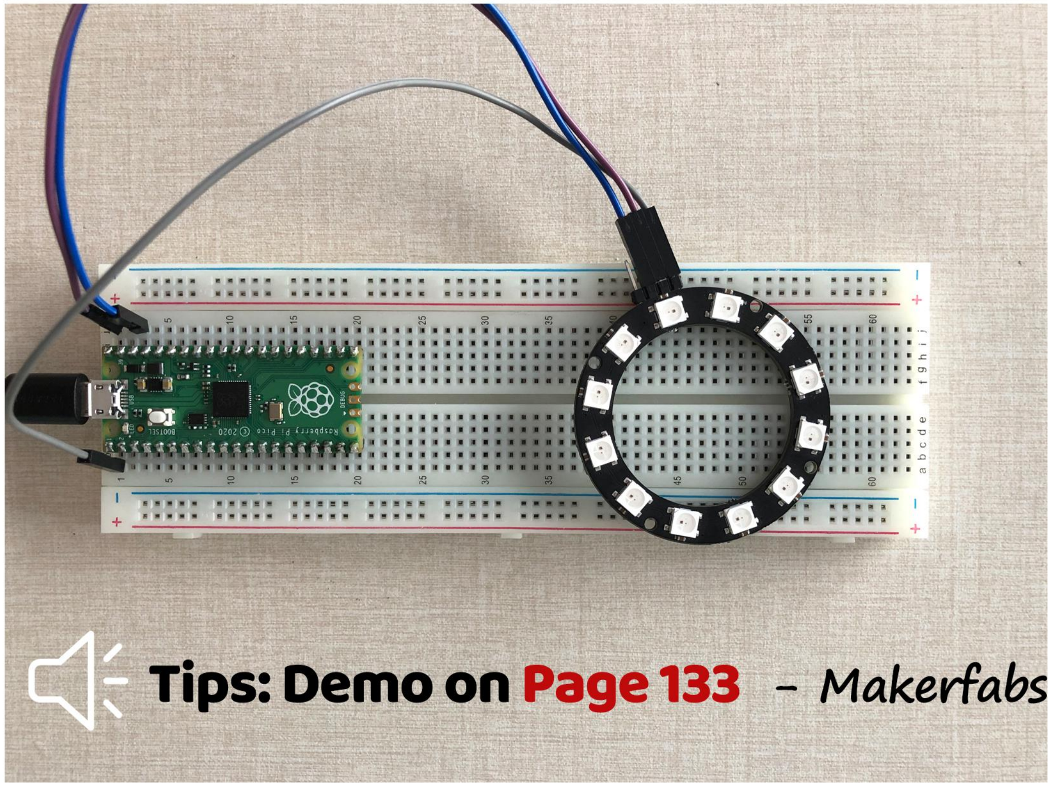


Figure C-1: Connecting an LED strip

One thing you need to be aware of is the potential current draw. While you can add an almost endless series of NeoPixels to your Pico, there's a limit to how much power you can get out

RPI Pico: Connecting an LED Strip



Tips: Demo on Page 133 – Makerfabs