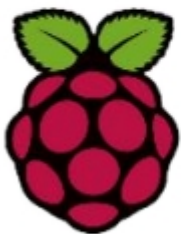


# Raspberry Pi Pico Starter Kit

Prezzo: 45.90 €

Tasse: 10.10 €

Prezzo totale (con tasse): 56.00 €



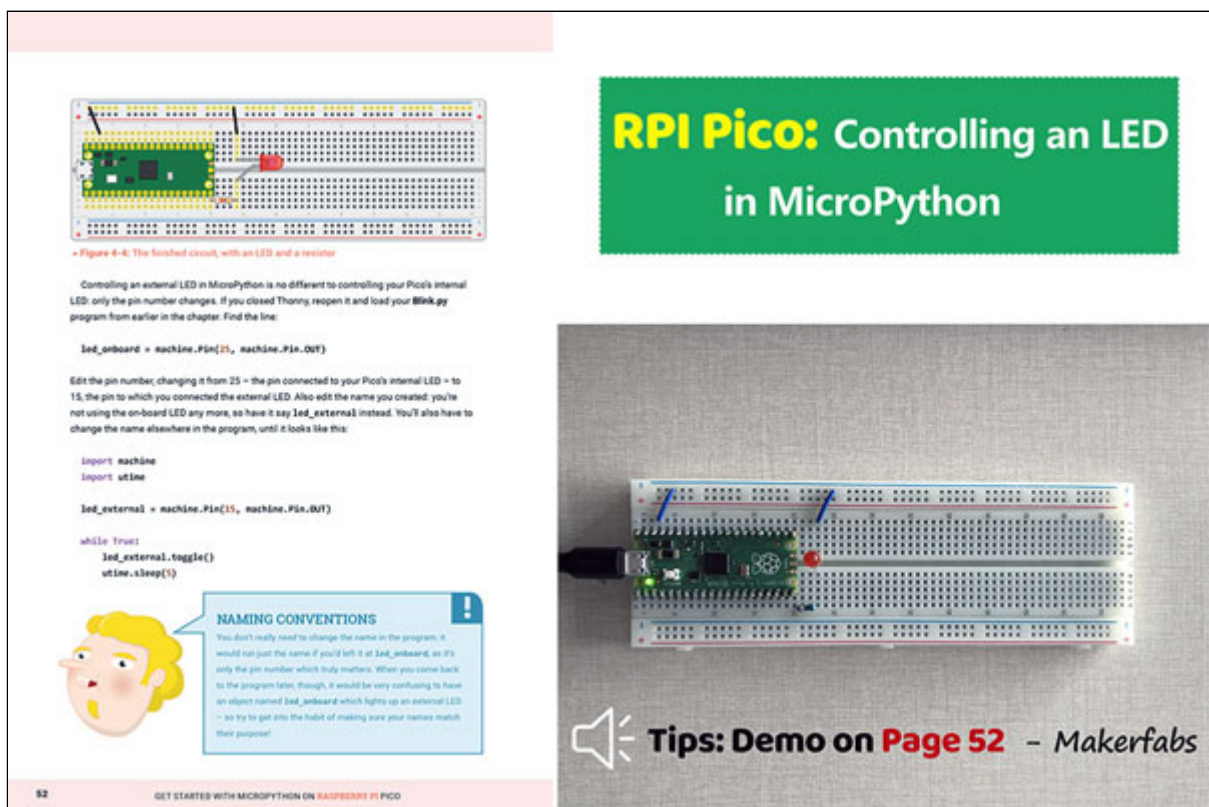
**Raspberry Pi** **Pico**

**Starter Kit**

Starter kit con Raspberry Pi Pico. La confezione comprende una serie di componenti elettronici necessari per realizzare alcuni degli esperimenti descritti nella guida [“Get Started with MicroPython on Raspberry Pi Pico”](#).

Contenuto del kit:

- 01Pz Raspberry Pi Pico
- 01Pz Display LCD seriale
- 01Pz Buzzer
- 01Pz Anello con 12 LED RGB WS2812 e driver integrato LED WS2812B
- 01Pz Breadboard 830 contatti (16,5x5,5 cm)
- 01Pz Set di jumper assortiti per breadboard (140 pezzi)
- 01Pz Potenziometro
- 02Pz Sensore infrarossi HC-SR501
- 02Pz Interruttore a pulsante
- 05Pz Mini pulsante
- 10Pz Resistenze ¼ w 330 ohm
- 10Pz Resistenze ¼ w 10 kohm
- 10Pz Jumper maschio-maschio
- 10Pz Jumper maschio-femmina
- 20Pz LED vari colori
- 01Pz Cavo USB



**RPI Pico: Controlling an LED in MicroPython**

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 to 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(1)
```

**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!

**Tips: Demo on Page 52 - Makerfabs**

### 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.

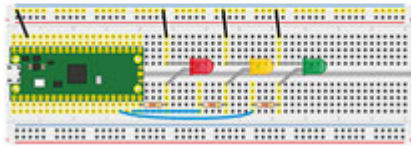


Figure 5-1: A basic three-light traffic light system



#### 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.

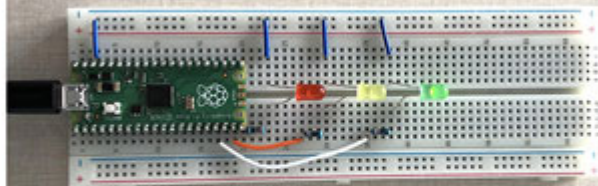
Take a jumper wire and connect the shorter leg – the cathode – of the red LED to your breadboard's ground rail. Take another, and connect the ground rail to one of your Pico's ground (GND) pins – in Figure 5-1, we've used the ground pin on row three of the breadboard.

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

to it. That's because a 10 kΩ resistor isn't strong enough to drop the 3V3 pin's output to 0V. 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 (M/M) 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.

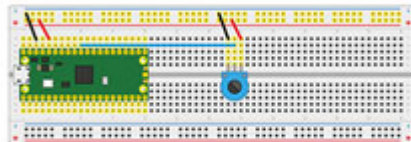


Figure 8-3: Wiring the potentiometer as a voltage divider

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?

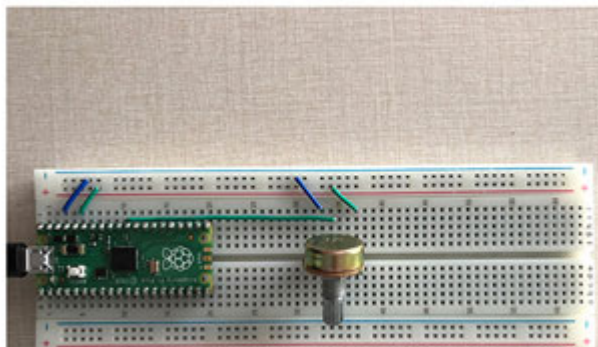
Adding the ground connection to the other end of the potentiometer's conductive strip has created a voltage divider: whereas before the potentiometer was simply acting as a resistor between the 3V3 pin and the analogue input pin, it's now dividing the voltage between the 3.3 V output by the 3V3 pin and 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



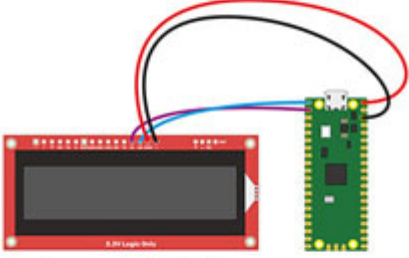
Tips: Demo on Page 96 – Makerfabs

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(0x28, 'a!@')
i2c.writeto(0x28, '1234')
i2c.writeto(0x28, 'hello world!')

```

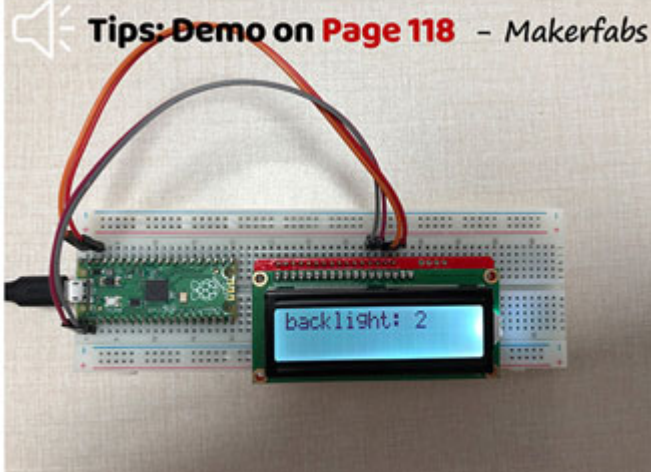
118 GET STARTED WITH MICROPYTHON ON RASPBERRY PI PICO

## RPI Pico: Wiring up a SerLCD Module for I2C





**Tips: Demo on Page 118** - Makerfabs

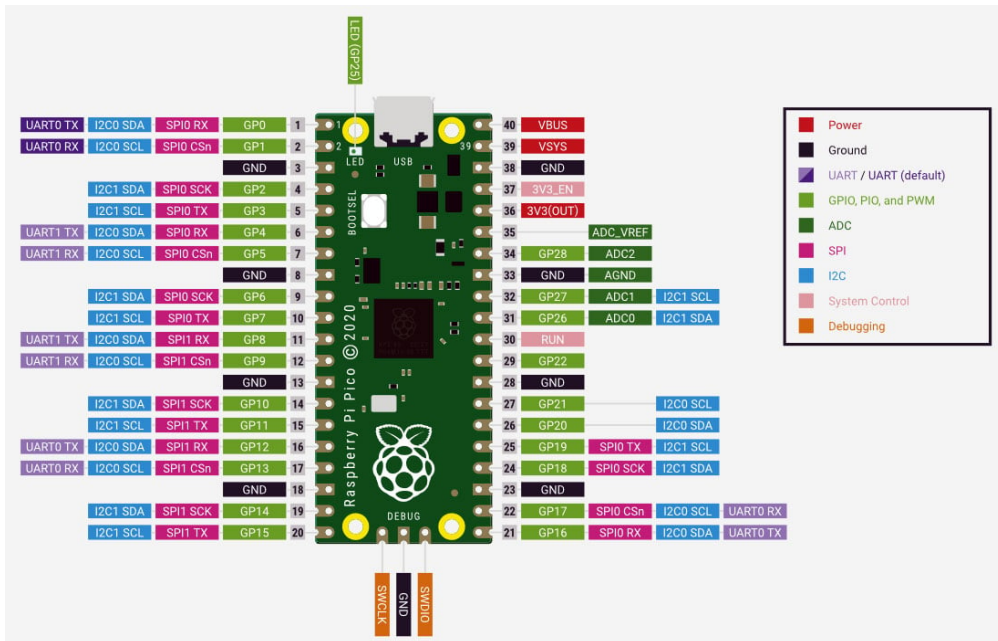


## Raspberry Pi Pico

Raspberry Pi Pico è una piccola scheda, veloce e versatile costruita utilizzando il chip RP2040, un nuovissimo chip progettato dalla Raspberry Pi Foundation nel Regno Unito. L'RP2040 è dotato di un processore Arm Cortex-M0 + dual-core con 264 KB di RAM interna e supporta fino a 16 MB di Flash. Dispone di diversio GPIO che includono I2C, SPI e altre funzioni speciali. Programmabile in C e MicroPython può essere utilizzata in varie applicazioni, dal controllo di un elettrodomestico al funzionamento di un display luminoso.

## Specifiche tecniche di Raspberry Pi Pico

- Chip microcontrollore RP2040 progettato da Raspberry Pi Foundation (nel Regno Unito)
- Processore: ARM Cortex M0+ dual-core, clock fino a 133 MHz
- 264 kB di SRAM e 2 MB di memoria Flash integrata
- Supporto per host e dispositivi USB 1.1
- Modalità Low-power sleep e dormant
- Programmazione drag & drop
- 26 pin GPIO multifunzione
- 2xSPI, 2xI2C, 2xUART, 3x12 bit ADC, 16 canali PWM controllabili
- Orologio e timer precisi
- Sensore di temperatura
- Accelerated floating point libraries on-chip



clicca sull'immagine per ingrandire

## Documentazione e link utili

- [Raspberry Pi Pico Datasheet](#)