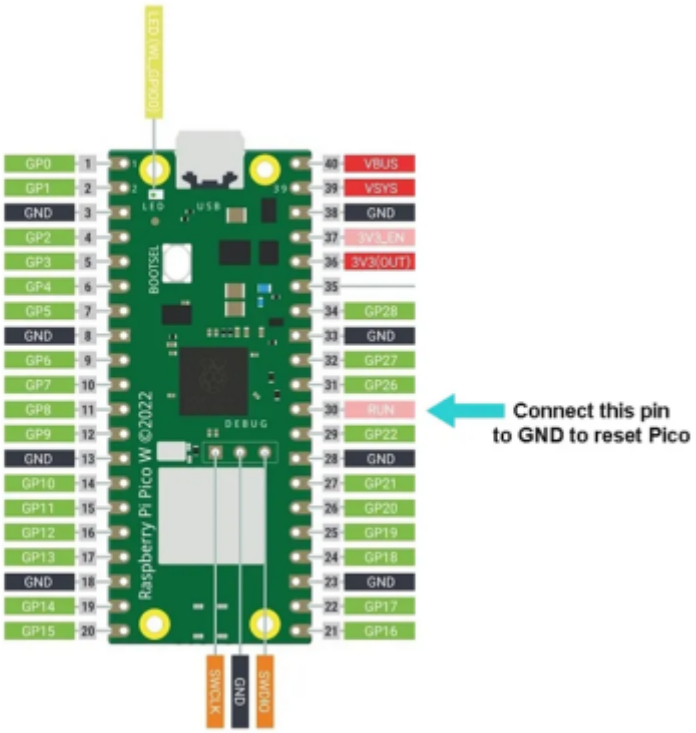
Using the Pico RUN Pin

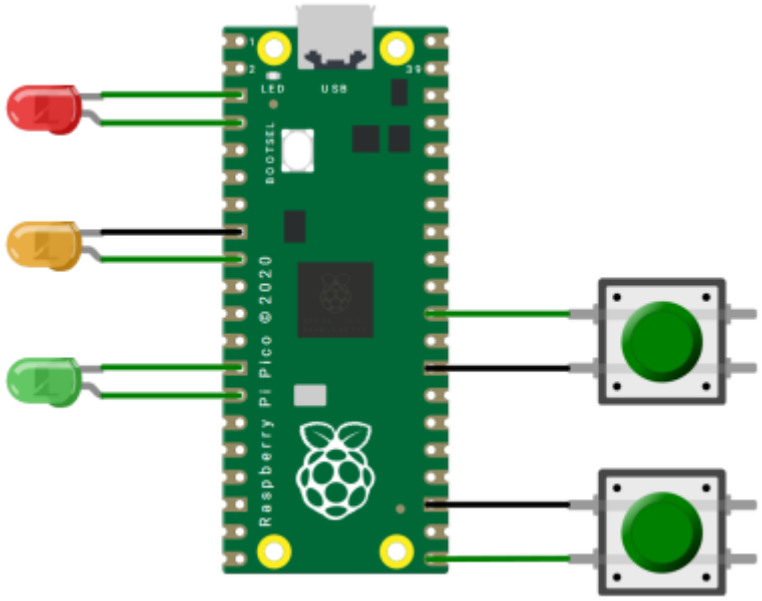
Jul 2026

Introduction

The Raspberry Pi Pico has a RUN pin, see below.



Connecting this Pin to GND starts (or if it is already running restarts) the Raspberry Pi Pico. For this example I am going connect a button from the RUN pin (30) to the GND Pin (28).



The code I am going to use is our traffic light code, with the breakout button. And I am using this code for a very specific reason.

- 1. The code runs as main.py so when the Pico starts (boots) the code runs.
- 2. There is a button to 'breakout' of the code, for a detailed explanation see here.
- 3. Once the breakout button has been used, the cable has to be removed and plugged back in, this causes unnecessary wear.
- By using the RUN pin, I can start/restart the Pico very simply.
- The RUN button operates at a hardware level, it requires no user generated code.

Download the following Micro Python code and save it to your Pico as main.py.

| download

```
#Traffic light LEDs using an array

from machine import Pin
import time

#######################    ADD BREAKOUT BUTTON ##############################
# Configure the button on GPIO16
# PULL_UP means the pin reads HIGH (1) until the button connects it to GND
button = Pin(16, Pin.IN, Pin.PULL_UP)
#######################    ADD BREAKOUT BUTTON ##############################


# Define GPIO pins for the LEDs
RED = Pin(2, Pin.OUT)
AMBER = Pin(6, Pin.OUT)
GREEN = Pin(10, Pin.OUT)

# Put them in an array
lights = [RED, AMBER, GREEN]

# Helper function to turn all lights off
def all_off():
    for light in lights:
        light.off()

# checks to see if the button has been pressed.
def check_breakout():
    if button.value() == 0:   # Button pressed
        print("Button pressed — stopping program.")
        all_off()
        return True
    return False
```

```
# define a function wait_with_break. This function waits 1 second in 0.1s slices. If you pass
# it the value 3, it will wait 3 seconds in 30 x .1s slices. This is so it can check the
# button state every 0.1s rather than waiting 8 whole seconds.
def wait_with_break(seconds):
    for _ in range(int(seconds * 10)):  # 0.1s slices in a loop
        if check_breakout():
            return True
        time.sleep(0.1)
    return False

while True:

    # Red
    all_off()
    RED.on()
    # passes the value 3 to the function above causing a 3s second delay in 0.1s steps
    if wait_with_break(3): break

    # Red + Amber
    AMBER.on()
    # passes the value 1 to the function above causing a 1s second delay in 0.1s steps
    if wait_with_break(1): break

    # Green
    all_off()
    GREEN.on()
    # passes the value 3 to the function above causing a 3s second delay in 0.1s steps
    if wait_with_break(3): break

    # Amber
    all_off()
    AMBER.on()
    # passes the value 1 to the function above causing a 1s second delay in 0.1s steps
    if wait_with_break(1): break

# Code End
```

</code>