Using a Breakout Button

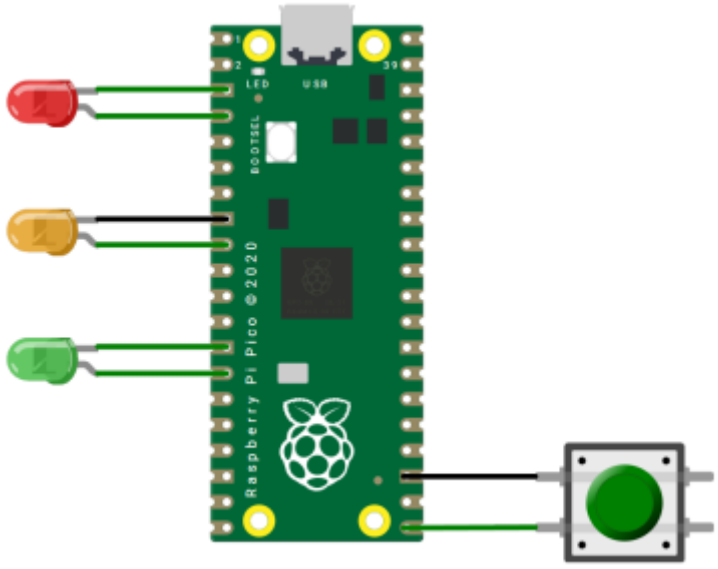
Jul 2026

Introduction

*** Warning *** You can lock yourself out of your Pi Pico if you don't have a method to break the code. Please think about if you want/need to do this!!!

In the previous example, we used some code to cause a 20 second start delay, so you had 20 seconds before the code ran where you could use Thonny to stop the process before it properly started. However, once 20 seconds had elapsed, this would no longer work. So you needed to reboot the Pico and break in before the 20s start delay.

In this example, again we are going to use the traffic light example, we are going to use a button to breakout of our code. This is very simple to do, first lets look at the wiring diagram for our traffic lights.



This is our original traffic light build, but with an additional push button on pins GP16 and GND.

The Code

Below is the code, there are two blocks that have been added. The are:

- Block1 - Sets up GP16 as an input that is pulled up (so we need to set it to GND for it to work)
- Block2 - Monitors the button to see if it is pressed (looking for a zero on Pin16)

- Block1 is at the program start, and is only run once.
- Block2 is in the loop, and is checked every time the code loops.

The Micro Python Code we are using.

| 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()

while True:

    # Red
    all_off()
    RED.on()
    time.sleep(3)

    # Red + Amber
    AMBER.on()
    time.sleep(1)

    # Green
    all_off()
    GREEN.on()
    time.sleep(3)

    # Amber
    all_off()
    AMBER.on()
    time.sleep(1)

###################### CHECK BREAKOUT BUTTON ##########################
    # Read the button
    if button.value() == 0:   # Button pressed (LOW)
        print("Button pressed — stopping program.")
        break                 # Exit the loop and end main.py
###################### CHECK BREAKOUT BUTTON ##########################

# Code End
```

There is on limitation on pressing the button, we need to look at this part of the code to understand it.

| download

```
    # Red
    all_off()
    RED.on()
    time.sleep(3)

    # Red + Amber
    AMBER.on()
    time.sleep(1)
```

```
    # Green
    all_off()
    GREEN.on()
    time.sleep(3)

    # Amber
    all_off()
    AMBER.on()
    time.sleep(1)
```

In our code, to delay the LEDs so they cycle in time like a set of traffic lights would, we have four time.sleep statements, the total delay of these is 8 seconds. This means, when we press the button, depending on where we are in the loop, we might have to hold the button for up to 9 seconds before the button value is read.

However, if you read the code, you can see the button value is read as the Amber LED is turned on after the Green LED>. So it is quite easy to time right if you don't want to hold the button for 8 seconds.