Get Started with MicroPython on Raspberry Pi Pico

Errata

Chapters 4–7

We have identified an error in some code examples in the first print run of *Get Started with MicroPython on Raspberry Pi Pico*. We're very sorry for this mistake, and we have corrected it in the PDF edition of the book. We have withdrawn the incorrect print edition from sale and asked our distributors to do the same, and a new, updated, edition is being printed. Customers who have purchased the incorrect edition of the book from the Raspberry Pi Press store or from a Raspberry Pi Approved Reseller will receive a copy of the updated edition free of charge from their retailer.

The error means that the projects in chapters 4 to 7 of the book, which use a digital input device (either a push-button or PIR sensor), will give erratic results if readers copy the code examples as they appear. This is because many of these code examples are missing a necessary parameter. (If you're already familiar with Python, you'll recognise that this parameter is needed to pull down the pin's resistor manually.)

To avoid the problem, you need to add the missing parameter `machine.Pin.PULL_DOWN` to each line of code that sets up the pin input with the code `machine.Pin.IN`. Every time a code example includes `machine.Pin.IN`, you need to add to it so that it reads `machine.Pin.IN, machine.Pin.PULL_DOWN`.

For example, on page 52:

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

…should change to:

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

And on page 82:

```
sensor_pir = machine.Pin(28, machine.Pin.IN)
```

…should change to:

```
sensor_pir = machine.Pin(28, machine.Pin.IN, machine.Pin.PULL_DOWN)
```

Other lines containing 'machine.Pin.IN' should be changed in a similar manner.

This is because Raspberry Pi Pico's GPIO pin resistors aren't automatically pulled down by default, as stated in the book.

Further explanation

When Pico powers up, MicroPython does not touch any GPIO state of the hardware, so the pins should be in whatever state they started in.

If the user constructs a `Pin` object by specifying the mode (eg `IN`, `OUT`), but does not specify any pull, then MicroPython defaults to setting the pin to "no pull" mode (i.e. disables pull-up and pull-down).

If the user constructs a `Pin` object by specifying the mode and the pull, then MicroPython configures the pin as requested.

Page 28

To avoid the infinite loop program being unstoppable, a small time delay should be added to the code using the `utime` library:

```
import utime
print("Loop starting!")
while True:
    print("Loop running!")
    utime.sleep(1)
print("Loop finished!")
```

Page 63

To prevent an intermittent error relating to the button, a small time delay should be added to the loop in the `button_reader_thread` function:

```
def button_reader_thread():
    global button_pressed
    while True:
        if button.value() == 1:
            button_pressed = True
        utime.sleep(0.01)
```

Page 71

For more reliable sensing of a button press, the `button_handler` function should be as follows, here and in subsequent code examples:

```
def button_handler(pin):
    global pressed
    if not pressed:
        pressed=True
        print(pin)
```

Pages 82–83

To prevent the alarm from being triggered by any jitter in the signal from the PIR sensor, the `pir_hander(pin)` function should be as follows, here and where it is used as the basis for subsequent code examples. You will also need to add the line `import utime` right at the start of the program.

```
def pir_handler(pin):
    utime.sleep_ms(100)
    if pin.value():
        print("ALARM! Motion detected!")
```

Pages 97, 103

The pin number for the potentiometer should be 26 (not 28) in the code:

```
potentiometer = machine.ADC(26)
```

Page 102

Instead of attempting to use the `led.duty` method, which no longer exists, you should go straight to using `led.duty_u16`:

```
led.duty_u16(potentiometer.read_u16())
```

Page 113

In the box on this page, Raspberry Pi Pico's file system is said to be 128kB in size; in fact, it is 1.375MiB (1,441,792 bytes). This means it can continue to log data for much longer than stated.

Pages 135–136

To improve the performance of the NeoPixel example, we advise changing the delays from `time.sleep_ms(100)` to `time.sleep_ms(10)`. In addition, the section for the white pixels should be as follows:

```
print("white")
for j in range(0, 255):
    for i in range(NUM_LEDS):
        ar[i] = (j<<16) + (j<<8) + j
    sm.put(ar,8)
    time.sleep_ms(10)
```