Todbot's CircuitPython Tricks

Created by John Park

https://learn.adafruit.com/todbot-circuitpython-tricks

Last updated on 2023-08-29 04:42:27 PM EDT

Table of Contents

Overview

Adafruit's good friend Tod Kurt () has been documenting CircuitPython tips and tricks on his GitHub, and the code snippets have been so helpful we asked him if he wanted to collaborate on turning them into a Learn Guide. Each trick has copy/paste-able code you can use to get things going.

Tod's tricks also form the basis of many of the CircuitPython Parsec () videos, and we're including them along-side the relevant tips and code samples in this guide.

```
print("Todbot's CircuitPython Tricks Rule!")
```

Inputs

Todbot's Inputs Tricks ()

Read a Digital Input Pin as a Button ()

```
import board
from digitalio import DigitalInOut, Pull
button = DigitalInOut(board.D3) # defaults to input
button.pull = Pull.UP # turn on internal pull-up resistor
print(button.value)  # False == pressed
```

Alternate Button Method

```
button = DigitalInOut(board.D3)
button.switch_to_input(Pull.UP)
```

Read a Potentiometer ()

```
import board
import analogio
potknob = analogio.AnalogIn(board.A1)
position = potknob.value  # ranges from 0-65535
pos = potknob.value // 256  # make 0-255 range
```

Read a Touch Pin / Capsense ()

```
import touchio
import board
touch_pin = touchio.TouchIn(board.GP6)
# on Pico / RP2040, need 1M pull-down on each input
if touch_pin.value:
print("touched!")
```

Read a Rotary Encoder ()

```
import board
import rotaryio
encoder = rotaryio.IncrementalEncoder(board.GP0, board.GP1) # must be consecutive
on Pico
print(encoder.position)  # starts at zero, goes neg or pos
```

Debounce a pin / button ()

```
import board
from digitalio import DigitalInOut, Pull
from adafruit_debouncer import Debouncer
button_in = DigitalInOut(board.D3) # defaults to input
button_in.pull = Pull.UP # turn on internal pull-up resistor
button = Debouncer(button_in)
while True:
    button.update()
    if button.fell:
        print("press!")
    if button.rose:
      print("release!")
```

Set up and debounce a list of pins ()

```
import board
from digitalio import DigitalInOut, Pull
from adafruit_debouncer import Debouncer
pins = (board.GP0, board.GP1, board.GP2, board.GP3, board.GP4)
buttons = []   # will hold list of Debouncer objects
for pin in pins:   # set up each pin
    tmp_pin = DigitalInOut(pin) # defaults to input
    tmp_pin.pull = Pull.UP      # turn on internal pull-up resistor
    buttons.append( Debouncer(tmp_pin) )
while True:
    for i in range(len(buttons)):
        buttons[i].update()
        if buttons[i].fell:
            print("button",i,"pressed!")
        if buttons[i].rose:
            print("button",i,"released!")
```

Outputs

Todbot's Outputs Tricks ()

Output HIGH / LOW on a pin (like an LED) ()

```
import board
import digitalio
ledpin = digitalio.DigitalInOut(board.D2)
ledpin.direction = digitalio.Direction.OUTPUT
ledpin.value = True
```

Alternate Method

```
ledpin = digitalio.DigitalInOut(board.D2)
ledpin.switch_to_output(value=True)
```

Output Analog value on a DAC pin ()

Different boards have DAC on different pins

```
import board
import analogio
dac = analogio.AnalogOut(board.A0)  # on Trinket M0 &amp;amp; QT Py
dac.value = 32768   # mid-point of 0-65535
```

Output a "Analog" value on a PWM pin ()

```
import board
import pwmio
out1 = pwmio.PWMOut(board.MOSI, frequency=25000, duty_cycle=0)
out1.duty_cycle = 32768  # mid-point 0-65535 = 50 % duty-cycle
```

Control NeoPixel / WS2812 LEDs ()

```
import neopixel
leds = neopixel.NeoPixel(board.NEOPIXEL, 16, brightness=0.2)
leds[0] = 0xff00ff  # first LED of 16 defined
leds[0] = (255,0,255)  # equivalent
leds.fill( 0x00ff00 )  # set all to green
```

()Control a servo, with animation list ()

```
# servo_animation_code.py -- show simple servo animation list
import time, random, board
from pwmio import PWMOut
from adafruit_motor import servo

# your servo will likely have different min_pulse &amp;amp; max_pulse settings
servoA = servo.Servo(PWMOut(board.RX, frequency=50), min_pulse=500, max_pulse=2250)

# the animation to play
animation = (
    # (angle, time to stay at that angle)
    (0, 2.0),
    (90, 2.0),
    (120, 2.0),
    (180, 2.0)
)
ani_pos = 0 # where in list to start our animation

while True:
    angle, secs = animation[ ani_pos ]
    print("servo moving to", angle, secs)
    servoA.angle = angle
    time.sleep( secs )
    ani_pos = (ani_pos + 1) % len(animation) # go to next, loop if at end
```

NeoPixels / Dotstars

Todbot's NeoPixel / DotStar LED Tricks ()

Moving rainbow on built-in `board.NEOPIXEL` ()

In CircuitPython 7, the `rainbowio` module has a `colorwheel()` function. Unfortunately, the `rainbowio` module is not available in all builds. In CircuitPython 6, `colorwheel()` is a built-in function part of `_pixelbuf` or `adafruit_pypixelbuf`.

The `colorwheel()` function takes a single value 0-255 hue and returns an `(R,G,B)` tuple given a single 0-255 hue. It's not a full `HSV_to_RGB()` function but often all you need is "hue to RGB", where you assume saturation=255 and value=255. It can be used with `neopixel`, `adafruit_dotstar`, or any place you need a (R,G,B) 3-byte tuple. Here's one way to use it.

```
# CircuitPython 7 with or without rainbowio module
import time, board, neopixel
try:
    from rainbowio import colorwheel
except:
    def colorwheel(pos):
        if pos &lt; 0 or pos &gt; 255:  return (0, 0, 0)
        if pos &lt; 85: return (255 - pos * 3, pos * 3, 0)
        if pos &lt; 170: pos -= 85; return (0, 255 - pos * 3, pos * 3)
        pos -= 170; return (pos * 3, 0, 255 - pos * 3)

led = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.4)
while True:
    led.fill( colorwheel((time.monotonic()*50)%255) )
    time.sleep(0.05)
```

```
# CircuitPython 6
import time
import board
import neopixel
led = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.4)
while True:
  led.fill( neopixel._pixelbuf.colorwheel((time.monotonic()*50)%255) )
  time.sleep(0.05)
```

Make moving rainbow gradient across an LED strip ()

See a demo of this in this tweet ().

```
import time, random
import board, neopixel, rainbowio
num_leds = 16
```

```
leds = neopixel.NeoPixel(board.D2, num_leds, brightness=0.4, auto_write=False )
delta_hue = 256//num_leds
speed = 10  # higher numbers = faster rainbow spinning
i=0
while True:
  for l in range(len(leds)):
    leds[l] = rainbowio.colorwheel( int(i*speed + l * delta_hue) % 255  )
  leds.show()  # only write to LEDs after updating them all
  i = (i+1) % 255
  time.sleep(0.05)
```

Fade all LEDs by amount for chase effects ()

```
import time, random
import board, neopixel
num_leds = 16
leds = neopixel.NeoPixel(board.D2, num_leds, brightness=0.4, auto_write=False )
my_color = (55,200,230)
dim_by = 20  # dim amount, higher = shorter tails
pos = 0
while True:
  leds[pos] = my_color
  leds[0:] = [[max(i-dim_by,0) for i in l] for l in leds] # dim all by
(dim_by,dim_by,dim_by)
  pos = (pos+1) % num_leds  # move to next position
  leds.show()  # only write to LEDs after updating them all
  time.sleep(0.05)
```

Audio

Todbot's Audio Tricks ()

In CircuitPython, there are three different techniques to output audio:

- DAC using `audioio`
- PWM using `audiopwmio` - requires an external RC filter (at least)
- I2S using `audiobusio` - requires external I2S decoder hardware

CircuitPython support on particular microcontroller may include support for several of these methods (e.g. SAMD51 supports DAC & I2S, just one (e.g. ESP32-S2 supports only I2S) or even none (e.g. Teensy).

Audio out using PWM ()

This uses the `audiopwmio` library, only available for Raspberry Pi Pico (or other RP2040-based boards) and nRF52840-based boards like Adafruit Feather nRF52840

Express. On RP2040-based boards, any pin can be PWM Audio pin. See the audiopwmio Support Matrix () for details.

```
import time,board
from audiocore import WaveFile
from audiopwmio import PWMAudioOut as AudioOut
wave_file = open("laser2.wav", "rb")
wave = WaveFile(wave_file)
audio = AudioOut(board.TX) # must be PWM-capable pin
while True:
    print("audio is playing:",audio.playing)
    if not audio.playing:
      audio.play(wave)
      wave.sample_rate = int(wave.sample_rate * 0.90) # play 10% slower each time
    time.sleep(0.1)
```

Note: Sometimes the `audiopwmio` driver gets confused, particularly if there's other USB access, so you may have to reset the board to get PWM audio to work again.

Note: WAV file whould be "16-bit Unsigned PCM" format. Sample rate can be up to 44.1 kHz, and is parsed by `audiocore.WaveFile`.

Note: PWM output must be filtered and converted to line-level to be usable. Use an RC circuit to accomplish this, see this twitter thread for details ().

Audio out using a DAC ()

Some CircuitPython boards have one or more built-in Digital to Audio Converters (DACs). These are on specific pins. The code is the the same as above, with just the import line changing.

```
import time,random,board
from audiocore import WaveFile
from audioio import AudioOut as AudioOut # only DAC
wave_file = open("laser20.wav", "rb")
wave = WaveFile(wave_file)
audio = AudioOut(board.A0)  # must be DAC-capable pin, A0 on QTPy Haxpress
while True:
  print("audio is playing:",audio.playing)
  if not audio.playing:
    audio.play(wave)
    wave.sample_rate = int(wave.sample_rate * 0.90) # play 10% slower each time
  time.sleep(0.1)
```

Preparing WAV files for CircuitPython ()

Convert files to appropriate WAV format (mono, 22050 Hz, 16-bit signed seem best). There are many software sound editing programs. Here are two solutions.

1) See the Adafruit guide Microcontroller Compatible Audio File Conversion ()

2) Use the command line program sox.

```
sox loop.mp3 -b 16 -c 1 -r 22050 loop.wav
```

To get `sox` on various platforms:

- Linux: `sudo apt install sox libsox-fmt-mp3`
- macOS: `brew install sox`
- Windows: Use installer at http://sox.sourceforge.net/ ()

USB

Todbot's USB Tricks ()

Rename the CIRCUITPY drive to something new ()

For instance, if you have multiple of the same device. The `label` can be up to 11 characters. This goes in boot.py not code.py and you must power cycle the board after the change.

```
# this goes in boot.py not code.py!
new_name = "TRINKEYPY0"
import storage
storage.remount("/", readonly=False)
m = storage.getmount("/")
m.label = new_name
storage.remount("/", readonly=True)
```

()Detect if USB is connected or not ()

```
import supervisor
if supervisor.runtime.usb_connected:
  led.value = True   # USB
else:
  led.value = False  # no USB
```

An older way that tries to mount CIRCUITPY read-write and if it fails, USB is connected:

```
def is_usb_connected():
    import storage
    try:
        storage.remount('/', readonly=False)  # attempt to mount readwrite
        storage.remount('/', readonly=True)  # attempt to mount readonly
```

```
    except RuntimeError as e:
        return True
    return False
is_usb = "USB" if is_usb_connected() else "NO USB"
print("USB:", is_usb)
```

()Get CIRCUITPY disk size and free space ()

```
import os
fs_stat = os.statvfs('/')
print("Disk size in MB", fs_stat[0] * fs_stat[2] / 1024 / 1024)
print("Free space in MB", fs_stat[0] * fs_stat[3] / 1024 / 1024)
```

Programmatically reset to UF2 bootloader ()

```
import microcontroller
microcontroller.on_next_reset(microcontroller.RunMode.BOOTLOADER)
microcontroller.reset()
```

USB Serial

()Print to USB Serial ()

```
print("hello there")  # prints a newline
print("waiting...", end='')   # does not print newline
for i in range(256):  print(i, end=', ')   # comma-separated numbers
```

()Read user input from USB Serial, blocking ()

```
while True:
    print("Type something: ", end='')
    my_str = input()  # type and press ENTER or RETURN
    print("You entered: ", my_str)
```

Read user input from USB Serial, non-blocking (mostly) ()

```
import time
import supervisor
print("Type something when you're ready")
last_time = time.monotonic()
while True:
    if supervisor.runtime.serial_bytes_available:
        my_str = input()
        print("You entered:", my_str)
    if time.monotonic() - last_time > 1:  # every second, print
```

```
        last_time = time.monotonic()
        print(int(last_time),"waiting...")
```

Read keys from USB Serial ()

```
import time, sys, supervisor
print("type characters")
while True:
    n = supervisor.runtime.serial_bytes_available
    if n > 0:  # we read something!
        s = sys.stdin.read(n)  # actually read it in
        # print both text & hex version of recv'd chars (see control chars!)
        print("got:", " ".join("{:s} {:02x}".format(c,ord(c)) for c in s))
    time.sleep(0.01) # do something else
```

Read user input from USB serial, non-blocking ()

```
class USBSerialReader:
    """ Read a line from USB Serial (up to end_char), non-blocking, with optional
echo """
    def __init__(self):
        self.s = ''
    def read(self,end_char='\n', echo=True):
        import sys, supervisor
        n = supervisor.runtime.serial_bytes_available
        if n > 0:                       # we got bytes!
            s = sys.stdin.read(n)       # actually read it in
            if echo: sys.stdout.write(s)  # echo back to human
            self.s = self.s + s         # keep building the string up
            if s.endswith(end_char): # got our end_char!
                rstr = self.s           # save for return
                self.s = ''             # reset str to beginning
                return rstr
        return None                     # no end_char yet

usb_reader = USBSerialReader()
print("type something and press the end_char")
while True:
    mystr = usb_reader.read()  # read until newline, echo back chars
    #mystr = usb_reader.read(end_char='\t', echo=False) # trigger on tab, no echo
    if mystr:
        print("got:",mystr)
    time.sleep(0.01)  # do something time critical
```

Networking

Todbot's Networking Tricks ()

Scan for WiFi Networks, sorted by signal strength (ESP32-S2) ()

```
import wifi
networks = []
for network in wifi.radio.start_scanning_networks():
    networks.append(network)
wifi.radio.stop_scanning_networks()
networks = sorted(networks, key=lambda net: net.rssi, reverse=True)
for network in networks:
    print("ssid:",network.ssid, "rssi:",network.rssi)
```

()Ping an IP address (ESP32-S2) ()

```
import time
import wifi
import ipaddress
from secrets import secrets
ip_to_ping = "1.1.1.1"

wifi.radio.connect(ssid=secrets['ssid'],password=secrets['password'])

print("my IP addr:", wifi.radio.ipv4_address)
print("pinging ",ip_to_ping)
ip1 = ipaddress.ip_address(ip_to_ping)
while True:
    print("ping:", wifi.radio.ping(ip1))
    time.sleep(1)
```

()Fetch a JSON file (ESP32-S2) ()

```
import time
import wifi
import socketpool
import ssl
import adafruit_requests
from secrets import secrets
wifi.radio.connect(ssid=secrets['ssid'],password=secrets['password'])
print("my IP addr:", wifi.radio.ipv4_address)
pool = socketpool.SocketPool(wifi.radio)
session = adafruit_requests.Session(pool, ssl.create_default_context())
while True:
    response = session.get("https://todbot.com/tst/randcolor.php")
    data = response.json()
    print("data:",data)
    time.sleep(5)
```

What the heck is secrets.py?

It's a config file that lives next to your code.py and is used (invisibly) by many Adafruit WiFi libraries. You can use it too (as in the examples above) without those libraries

It looks like this for basic WiFi connectivity:

```
# secrets.py
secrets = {
  "ssid": "Pretty Fly for a WiFi",
  "password": "donthackme123"
}
# code.py
from secrets import secrets
print("your WiFi password is:", secrets['password'])
```

Other field/value pairs may be in secrets.py for using passwords to Adafruit IO or other applications, to denote a time zone, etc.

Displays (LCD / OLED / E-Ink) and displayio

Todbot's Displays Tricks ()

displayio () is the native system-level driver for displays in CircuitPython. Several CircuitPython boards (FunHouse, MagTag, PyGamer, CLUE) have `displayio`-based displays and a built-in `board.DISPLAY` object that is preconfigured for that display. Or, you can add your own I2C () or SPI () display.

()Get default display and change display rotation ()

Boards like FunHouse, MagTag, PyGamer, CLUE have built-in displays. `display.rotation` works with all displays, not just built-in ones.

```
import board
display = board.DISPLAY
print(display.rotation) # print current rotation
display.rotation = 0    # valid values 0,90,180,270
```

Display an Image ()

The `adafruit_imageload` library makes it easier to load images and display them. The images should be in palettized BMP3 format.

```
import board
import displayio
import adafruit_imageload

img_filename = "bg_jp200.bmp"
display = board.DISPLAY
img, pal = adafruit_imageload.load(img_filename)
group = displayio.Group()
group.append(displayio.TileGrid(img, pixel_shader=pal))
display.show(group)
```

CircuitPython's `displayio` library works like:

- an image `Bitmap` goes inside a `TileGrid`
- a `TileGrid` goes inside a `Group`
- a `Group` is shown on a `Display`.

Display Background Bitmap ()

Useful for display a solid background color that can be quickly changed.

```
import time, board, displayio
display = board.DISPLAY       # get default display (FunHouse,Pygamer,etc)
maingroup = displayio.Group() # Create a main group to hold everything
display.show(maingroup)       # put it on the display

# make bitmap that spans entire display, with 3 colors
background = displayio.Bitmap(display.width, display.height, 3)

# make a 3 color palette to match
mypal = displayio.Palette(3)
mypal[0] = 0x000000 # set colors (black)
mypal[1] = 0x999900 # dark yellow
mypal[2] = 0x009999 # dark cyan

# Put background into main group, using palette to map palette ids to colors
maingroup.append(displayio.TileGrid(background, pixel_shader=mypal))

time.sleep(2)
background.fill(2)  # change background to dark cyan (mypal[2])
time.sleep(2)
background.fill(1)  # change background to dark yellow (mypal[1])
```

Another way is to use `vectorio` ():

```
import board, displayio, vectorio

display = board.DISPLAY  # built-in display
maingroup = displayio.Group()  # a main group that holds everything
display.show(maingroup)

mypal = displayio.Palette(1)
mypal[0] = 0x999900
background = vectorio.Rectangle(pixel_shader=mypal, width=display.width,
height=display.height, x=0, y=0)
maingroup.append(background)
```

Or can also use `adafruit_display_shapes` ():

```
import board, displayio
from adafruit_display_shapes.rect import Rect

display = board.DISPLAY
maingroup = displayio.Group()  # a main group that holds everything
display.show(maingroup)        # add main group to display

background = Rect(0,0, display.width, display.height, fill=0x000000 ) # background
color
maingroup.append(background)
```

()Image Slideshow

```
import time, board, displayio
import adafruit_imageload

display = board.DISPLAY # get display object (built-in on some boards)
screen = displayio.Group() # main group that holds all on-screen content
display.show(screen)       # add it to display

file_names = [ '/images/cat1.bmp', '/images/cat2.bmp' ]  # list of filenames

screen.append(displayio.Group())  # placeholder, will be replaced w/ screen[0] below
while True:
    for fname in file_names:
        image, palette = adafruit_imageload.load(fname)
        screen[0] = displayio.TileGrid(image, pixel_shader=palette)
        time.sleep(1)
```

Note: Images must be in palettized BMP3 format. For more details, see Preparing images for CircuitPython ()

Dealing with E-Ink "Refresh Too Soon" Error

E-Ink displays are damaged if refreshed too frequently. CircuitPython enforces this, but also provides `display.time_to_refresh`, the number of seconds you need to wait before the display can be refreshed. One solution is to sleep a little longer than that and you'll never get the error. Another would be to wait for `time_to_refresh` to go to zero, as show below.

```
import time, board, displayio, terminalio
from adafruit_display_text import label
mylabel = label.Label(terminalio.FONT, text="demo", x=20,y=20,
                      background_color=0x000000, color=0xffffff )
display = board.DISPLAY  # e.g. for MagTag
display.show(mylabel)
while True:
    if display.time_to_refresh == 0:
        display.refresh()
    mylabel.text = str(time.monotonic())
    time.sleep(0.1)
```

I2C

Todbot's I2C Tricks ()

Scan the I2C bus for devices ()

From: CircuitPython I2C Guide: Find Your Sensor ()

```
import board
i2c = board.I2C() # or busio.I2C(pin_scl,pin_sda)
while not i2c.try_lock():  pass
print("I2C addresses found:", [hex(device_address)
    for device_address in i2c.scan()])
i2c.unlock()
```

()Speed up the I2C bus ()

CircuitPython defaults to 100 kHz I2C bus speed. This will work for all devices, but some devices can go faster. Common faster speeds are 200 kHz and 400 kHz.

```
import board
import busio
# instead of doing
# i2c = board.I2C()
i2c = busio.I2C( board.SCL, board.SDA, frequency=200_000)
# then do something with 'i2c' object as before, like:
oled = adafruit_ssd1306.SSD1306_I2C(width=128, height=32, i2c=i2c)
```

Timing

Todobot's Timing Tricks ()

Measure how long something takes ()

Generally use `time.monotonic()` to get the current "uptime" of a board in fractional seconds. So to measure the duration it takes CircuitPython to do something:

```
import time
start_time = time.monotonic()
# put thing you want to measure here, like:
import neopixel
```

```
stop_time = time.monotonic()
print("elapsed time = ", stop_time - start_time)
```

Note that on the "small" versions of CircuitPython in the QT Py M0, Trinket M0, etc., the floating point value of seconds will become less accurate as uptime increases.

More accurate timing with `ticks_ms()`, like Arduino ()`millis()` ()

If you want something more like Arduino's `millis()` function, the `supervisor.ticks_ms()` function returns an integer, not a floating point value. It is more useful for sub-second timing tasks and you can still convert it to floating-point seconds for human consumption.

```
import supervisor
start_msecs = supervisor.ticks_ms()
import neopixel
stop_msecs = supervisor.ticks_ms()
print("elapsed time = ", (stop_msecs - start_msecs)/1000)
```

Board Info

Todbot's Board Info Tricks ()

Display the amount of free RAM ()

From: https://learn.adafruit.com/welcome-to-circuitpython/frequently-asked-questions ()

```
import gc
print( gc.mem_free() )
```

Show microcontroller.pin to board mappings ()

From https://gist.github.com/anecdata/1c345cb2d137776d76b97a5d5678dc97 ()

```
import microcontroller
import board

for pin in dir(microcontroller.pin):
    if isinstance(getattr(microcontroller.pin, pin), microcontroller.Pin):
        print("".join(("microcontroller.pin.", pin, "\t")), end=" ")
        for alias in dir(board):
            if getattr(board, alias) is getattr(microcontroller.pin, pin):
```

```
                print("".join(("", "board.", alias)), end=" ")
    print()
```

()Determine which board you're on ()

```
import os
print(os.uname().machine)
'Adafruit ItsyBitsy M4 Express with samd51g19'
```

To get the chip family

```
import os
print(os.uname().sysname)
'ESP32S2'
```

()Support multiple boards with one code.py ()

```
import os
board_type = os.uname().machine
if 'QT Py M0' in board_type:
  tft_clk  = board.SCK
  tft_mosi = board.MOSI
  spi = busio.SPI(clock=tft_clk, MOSI=tft_mosi)
elif 'ItsyBitsy M4' in board_type:
  tft_clk  = board.SCK
  tft_mosi = board.MOSI
  spi = busio.SPI(clock=tft_clk, MOSI=tft_mosi)
elif 'Pico' in board_type:
  tft_clk = board.GP10 # must be a SPI CLK
  tft_mosi= board.GP11 # must be a SPI TX
  spi = busio.SPI(clock=tft_clk, MOSI=tft_mosi)
else:
  print("supported board", board_type)
```

Computery Tasks

Todbot's Computery Tasks Tricks ()

Formatting strings ()

```
name = "John"
fav_color = 0x003366
body_temp = 98.65
fav_number = 123
print("name:%s color:%06x temp:%2.1f num:%d" %
(name,fav_color,body_temp,fav_number))
# 'name:John color:ff3366 temp:98.6 num:123'
```

()Formatting strings with f-strings ()

(This doesn't work on 'small' CircuitPythons like QTPy M0 due to the small amounts of flash memory on the board.)

```
name = "John"
fav_color = 0x003366
body_temp = 98.65
print(f"name:{name} color:{color:06x} temp:{body_temp:2.1f} num:{fav_number:%d}")
# 'name:John color:ff3366 temp:98.6 num:123'
```

()Make and use a config file ()

```
# my_config.py
config = {
    "username": "Grogu Djarin",
    "password": "ig88rules",
    "secret_key": "3a3d9bfaf05835df69713c470427fe35"
}

# code.py
from my_config import config
print("secret:", config['secret_key'])
# 'secret: 3a3d9bfaf05835df69713c470427fe35'
```

Run different code.py on startup ()

Use `microcontroller.nvm` to store persistent state across resets or between boot.py and code.py, and declare that the first byte of `nvm` will be the `startup_mode`. Now if you create multiple code.py files (say) code1.py, code2.py, etc. you can switch between them based on `startup_mode`.

```
import time
import microcontroller
startup_mode = microcontroller.nvm[0]
if startup_mode == 1:
    import code1      # runs code in `code1.py`
if startup_mode == 2:
    import code2      # runs code in `code2.py`
# otherwise runs 'code.py`
while True:
    print("main code.py")
    time.sleep(1)
```

Note: in CircuitPyton 7+ you can use `supervisor.set_next_code_file()` () to change which .py file is run on startup. This changes only what happens on reload, not hardware reset or powerup. Using it would look like:

```
import supervisor
supervisor.set_next_code_file('code_awesome.py')
```

```
# and then if you want to run it now, trigger a reload
supervisor.reload()
```

More Esoteric Tasks

Todbot's More Esoteric Tasks ()

Map an input range to an output range ()

```
# simple range mapper, like Arduino map()
def map_range(s, a1, a2, b1, b2):
    return  b1 + ((s - a1) * (b2 - b1) / (a2 - a1))

# example: map 0-0123 value to 0.0-1.0 value
val = 768
outval = map_range( val, 0,1023, 0.0,1.0 )
# outval = 0.75
```

Constrain an input to a min/max ()

The Python built-in `min()` and `max()` functions can be used together to make something like Arduino's `constrain()`.

```
# constrain a value to be 0-255
outval = min(max(val, 0), 255)
# constrain a value to be 0-255 integer
outval = int(min(max(val, 0), 255))
# constrain a value to be -1 to +1
outval = min(max(val, -1), 1)
```

Preventing Ctrl-C from stopping the program ()

Put a `try`/`except KeyboardInterrupt` to catch the Ctrl-C on the inside of your main loop.

```
while True:
  try:
    print("Doing something important...")
    time.sleep(0.1)
  except KeyboardInterrupt:
    print("Nice try, human! Not quitting.")
```

Also useful for graceful shutdown (turning off neopixels, say) on Ctrl-C.

```
import time, random
import board, neopixel, rainbowio
```

```
leds = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.4 )
while True:
  try:
    rgb = rainbowio.colorwheel(int(time.monotonic()*75) % 255)
    leds.fill(rgb)
    time.sleep(0.05)
  except KeyboardInterrupt:
    print("shutting down nicely...")
    leds.fill(0)
    break  # gets us out of the while True
```

Prevent auto-reload when CIRCUITPY is touched ()

Normally, CircuitPython restarts anytime the CIRCUITPY drive is written to. This is great normally, but is frustrating if you want your code to keep running, and you want to control exactly when a restart happens.

```
import supervisor
supervisor.disable_autoreload()
```

To trigger a reload, do a Ctrl-C + Ctrl-D in the REPL or reset your board.

Raspberry Pi Pico boot.py Protection ()

Also works on other RP2040-based boards like QTPy RP2040. From https://gist.github.com/Neradoc/8056725be1c209475fd09ffc37c9fad4 ().

```
# Copy this as 'boot.py' in your Pico's CIRCUITPY drive
# Useful in case Pico locks up (which it's done a few times on me)
import board
import time
from digitalio import DigitalInOut,Pull

led = DigitalInOut(board.LED)
led.switch_to_output()

safe = DigitalInOut(board.GP14)  # <-- choose your button pin
safe.switch_to_input(Pull.UP)

def reset_on_pin():
    if safe.value is False:
        import microcontroller
        microcontroller.on_next_reset(microcontroller.RunMode.SAFE_MODE)
        microcontroller.reset()

led.value = False
for x in range(16):
    reset_on_pin()
    led.value = not led.value  # toggle LED on/off as notice
    time.sleep(0.1)
```

Hacks

Todbot's Hacks Tricks ()

Using the REPL ()

Display built-in modules / libraries

```
Adafruit CircuitPython 6.2.0-beta.2 on 2021-02-11; Adafruit Trinket M0 with
samd21e18
>>> help("modules")
__main__          digitalio         pulseio           supervisor
analogio          gc                pwmio             sys
array             math              random            time
board             microcontroller   rotaryio          touchio
builtins          micropython       rtc               usb_hid
busio             neopixel_write    storage           usb_midi
collections       os                struct
Plus any modules on the filesystem
```

Use REPL fast with copy-paste multi-one-liners ()

(yes, semicolons are legal in Python)

```
# load most common libraries
import time; import board; from digitalio import DigitalInOut,Pull; import
analogio; import touchio

# print out board pins and objects (like 'I2C' and 'display')
import board; dir(board)

# print out microcontroller pins (chip pins, not the same as board pins)
import microcontroller; dir(microcontroller.pin)

# release configured / built-in display
import displayio; displayio.release_displays()

# turn off auto-reload when CIRCUITPY drive is touched
import supervisor; supervisor.disable_autoreload()

# make all neopixels purple
import board; import neopixel; leds = neopixel.NeoPixel(board.D3, 8,
brightness=0.2); leds.fill(0xff00ff)
```

General Python

Todbot's General Python Tricks ()

These are general Python tips that may be useful in CircuitPython.

Create list with elements all the same value ()

```
blank_array = [0] * 50   # creats 50-element list of zeros
```

Storing multiple values per list entry ()

Create simple data structures as config to control your program. Unlike Arduino, you can store multiple values per list/array entry.

```
mycolors = (
    # color val, name
    (0x0000FF, "blue"),
    (0x00FFFF, "cyan"),
    (0xFF00FF, "purple"),
)
for i in range(len(mycolors)):
    (val, name) = mycolors[i]
    print("my color ", name, "has the value", val)
```

Display which (not built-in) libraries have been imported ()

```
import sys
print(sys.modules.keys())
# 'dict_keys([])'
import board
import neopixel
import adafruit_dotstar
print(sys.modules.keys())
prints "dict_keys(['neopixel', 'adafruit_dotstar'])"
```

()List names of all global variables ()

```
a = 123
b = 'hello there'
my_globals = sorted(dir)
print(my_globals)
# prints "['__name__', 'a', 'b']"
if 'a' in my_globals:
  print("you have a variable named 'a'!")
```

```
if 'c' in my_globals:
  print("you have a variable named 'c'!")
```

Display the running CircuitPython release ()

With an established serial connection, press `Ctrl+c`:

```
Adafruit CircuitPython 7.1.1 on 2022-01-14; S2Pico with ESP32S2-S2FN4R2
>>>
```

Without connection or code running, check the boot_out.txt file in your CIRCUITPY drive.

```
import os
print(os.uname().release)
'7.1.1'
print(os.uname().version)
'7.1.1 on 2022-01-14'
```

Host-side tasks

Todbot's Host-side tasks tricks ()

Things you might need to do on your computer when using CircuitPython.

Installing CircuitPython libraries ()

The below examples are for MacOS / Linux. Similar commands are used for Windows

Installing libraries with `circup`

`circup` can be used to easily install and update modules

```
$ pip3 install --user circup
$ circup install adafruit_midi
$ circup update   # updates all modules
```

Freshly update all modules to latest version (e.g. when going from CP 6 -> CP 7) (This is needed because `circup update` doesn't actually seem to work reliably)

```
circup freeze > mymodules.txt
rm -rf /Volumes/CIRCUITPY/lib/*
circup install -r mymodules.txt
```

And updating circup when a new version of CircuitPython comes out:

```
$ pip3 install --upgrade circup
```

()Copying libraries by hand with `cp`

To install libraries by hand from the CircuitPython Library Bundle () or from the CircuitPython Community Bundle () (which circup doesn't support), get the bundle, unzip it and then use `cp -rX`.

```
cp -rX bundle_folder/lib/adafruit_midi /Volumes/CIRCUITPY/lib
```

Note: on limited-memory boards like Trinkey, Trinket, QTPy, you must use the `-X` option on MacOS to save space. You may also need to omit unused parts of some libraries (e.g. `adafruit_midi/system_exclusive` is not needed if just sending MIDI notes)

Preparing images for CircuitPython ()

CircuitPython requires "indexed" (aka "palette") BMP3 images. If using `adafruit_imageload` the images can have RLE compression, but if using `displayio.OnDiskBitmap()`, then make sure no compression is used.

To make images load faster, you can reduce the number of colors in the image. The maximum number of colors is 256, but try reducing colors to 64 or even 2 if it's a black-n-white image.

Some existing Learn Guides:

- Creating Your First Tilemap Game with CircuitPython ()
- Preparing Graphics for E-Ink Displays ()

And here's some ways to do the conversions.

Command-line: using ImageMagick

ImageMagick () is a command-line image manipulation tool. With it, you can convert any image format to BMP3 format. The main ImageMagick CLI command is `convert`.

```
convert myimage.jpg -resize 240x240 -colors 64 -type palette -compress None
BMP3:myimage_for_cirpy.bmp
```

Command-line: using GraphicsMagick

GraphicsMagick () is a slimmer, lower-requirement clone of ImageMagick. All GraphicsMagick commands are accessed via the `gm` CLI command.

```
gm convert myimage.jpg -resize 240x240 -colors 64 -type palette -compress None
BMP3:myimage_for_cirpy.bmp
```

Making images smaller or for E-Ink displays

To make images smaller (and load faster), reduce number of colors from 256. If your image is a monochrome (or for use with E-Ink displays like MagTag), use 2 colors. The "-dither" options () are really helpful for monochrome:

```
convert cat.jpg -dither FloydSteinberg -colors 2 -type palette BMP3:cat.bmp
```

NodeJs: using gm

There is a nice wrapper around GraphicsMagick / Imagemagick with the `gm library` (). A small NodeJs program to convert images could look like this:

```
var gm = require('gm');
gm('myimage.jpg')
    .resize(240, 240)
    .colors(64)
    .type("palette")
    .compress("None")
    .write('myimage_for_cirpy.bmp', function (err) {
        if (!err) console.log('done1');
    });
```

()Python: using PIL / pillow

The Python Image Library (PIL) fork `pillow` () seems to work the best. It's unclear how to toggle compression.

```
from PIL import Image
size = (240, 240)
num_colors = 64
```

```
img = Image.open('myimage.jpg')
img = img.resize(size)
newimg = img.convert(mode='P', colors=num_colors)
newimg.save('myimage_for_cirpy.bmp')
```