Learn CircuitPython using the Adafruit Trinket M0

PRESENTED BY RICHARD GOWEN (@alt_bier)

Created for BSidesDFW 2020 HHV

This Slide Deck Is Available at https://altbier.us/circuitpython/

What is CircuitPython?

- CircuitPython is a programming language designed to simplify experimenting and learning to code on low-cost microcontroller boards.
- CircuitPython is based on the Python programming language.
  - Python is a widely used high level language that is easier to read, write and maintain than low level languages like C. It supports modules and packages, has a built-in interpreter (which means no compiling), and is open source.
  - CircuitPython adds hardware support to all these great features.
- If you already have Python knowledge, you can easily apply that to using CircuitPython. If you don't, it's simple to get started!

What is the Adafruit Trinket M0?

- The Adafruit Trinket M0 is a tiny microcontroller board that has been designed to work with CircuitPython.
- The Trinket is a hardware development board like an Arduino and can even run Arduino code.
- It comes shipped with CicuitPython firmware installed. So when you plug it in it will show up as a very small disk drive with some files including main.py on it.
- Simply edit main.py with python code. No IDE required.

Specs:

- Processor: Atmel ATSAMD21E18 32-bit 48MHz Cortex M0+
- Flash: 256 KB
- RAM: 32 KB
- Native USB Support
- Support for both Arduino IDE and CircuitPython
- 5 GPIO Pins
  - Analog input on 3 pins and true analog output on 1pin
  - PWM output on 3 pins
  - Capacitive touch sensors on 3 pins

Adafruit and CircuitPython Resources

- The Adafruit website https://www.adafruit.com/ is where you will find the documentation and other resources for their products.
- You can purchase a Trinket M0 here: https://www.adafruit.com/product/3500
- Trinket M0 Documentation: https://learn.adafruit.com/adafruit-trinket-m0-circuitpython-arduino/
- The CircuitPython website https://circuitpython.org/ is a great resource for documentation and software and example code.

ELECTRONICS 101

Working with hardware development boards such as the Adafruit Trinket M0 is easier if you have a basic knowledge of electronics concepts.

This presentation will not provide that knowledge. However, I have put together a separate presentation that does.

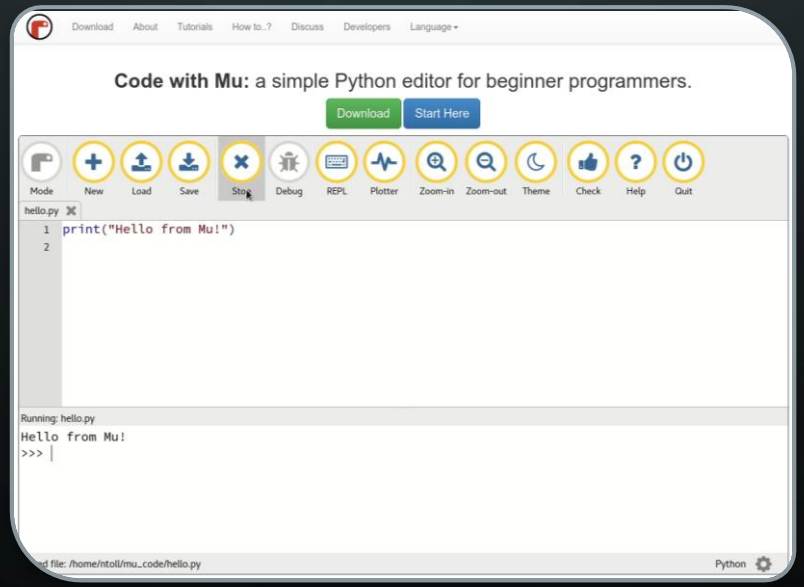
You can find my electronics overview presentation here:

https://altbier.us/electronics/

Mu Editor Software

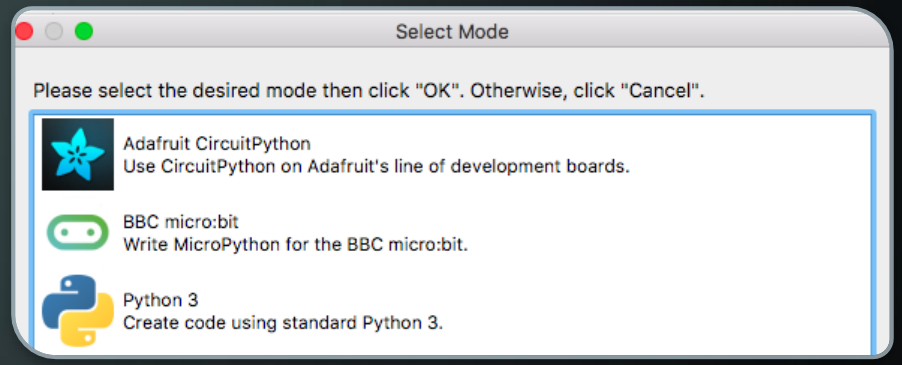
While an Integrated Development Environment is not required to work with CircuitPython (any editor will work), Mu is a simple code editor that works with the Adafruit CircuitPython boards. It has a built-in serial console, so you can get immediate feedback from your board's serial output.

You can download it from `https://codewith.mu/` It is available for the Windows, Linux, MacOS, and Raspberry Pi operating systems.

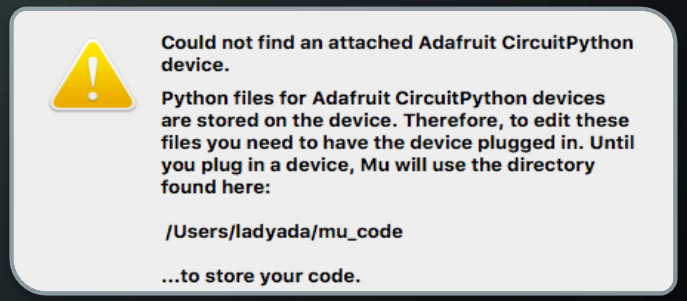


Mu Editor Software

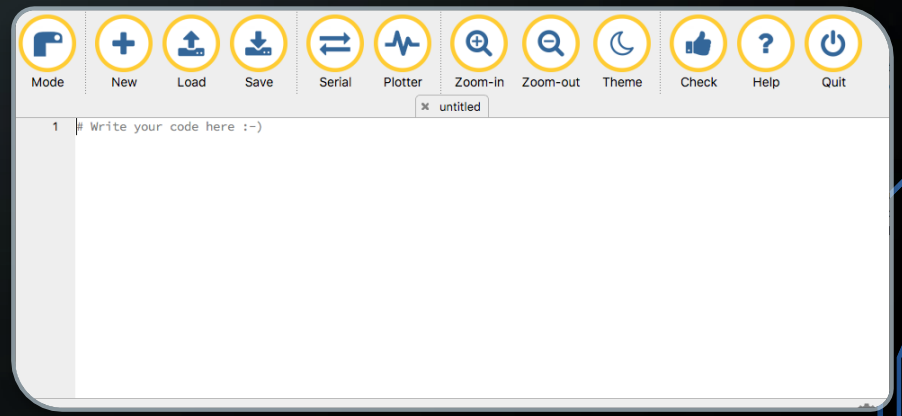
The first time you start Mu, you will be prompted to select your 'mode' - you can change this later. If you are going to use Mu for the Labs presented here, you should select 'Adafruit CircuitPython'.



Mu attempts to auto-detect your board, so plug in your Trinket M0 device via USB and make sure it shows up as a drive named CIRCUITPY before starting Mu.



Once your device is plugged in and Mu is started you are ready to write some code!



Serial Monitor

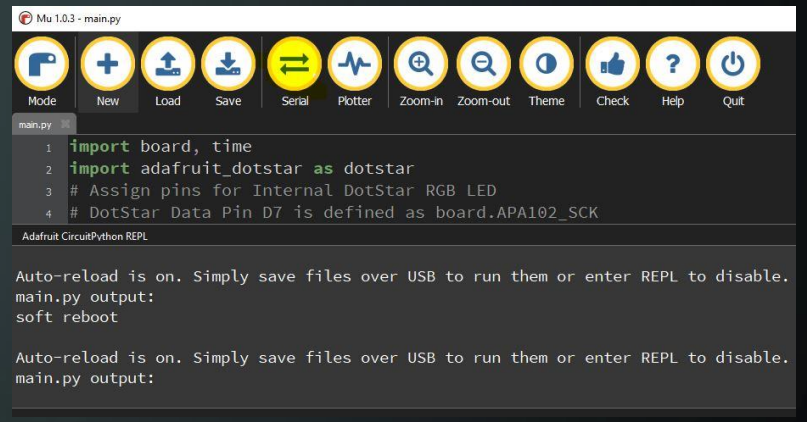
To view the serial output of the Trinket M0 or any CircuitPython device you'll need a serial monitor.

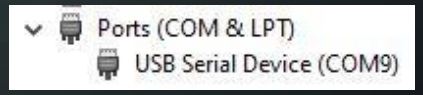
In the Mu editor program there is a serial monitor built in that can be launched by clicking the 'Serial' button.

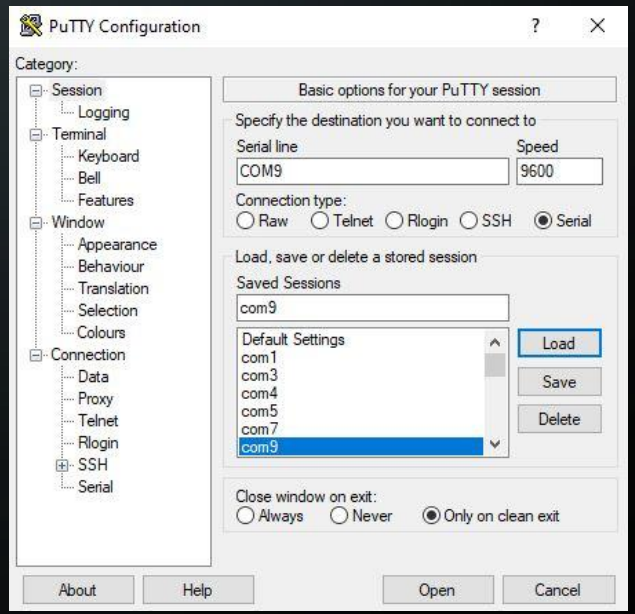
If you're not using that software, there are other ways to monitor the serial output.

Look up the COM port number that your PC assigned to the Trinket when you connected it.

Use a terminal program that supports serial connections (like PuTTY) and configure it to connect to the COM port at 9600 baud.





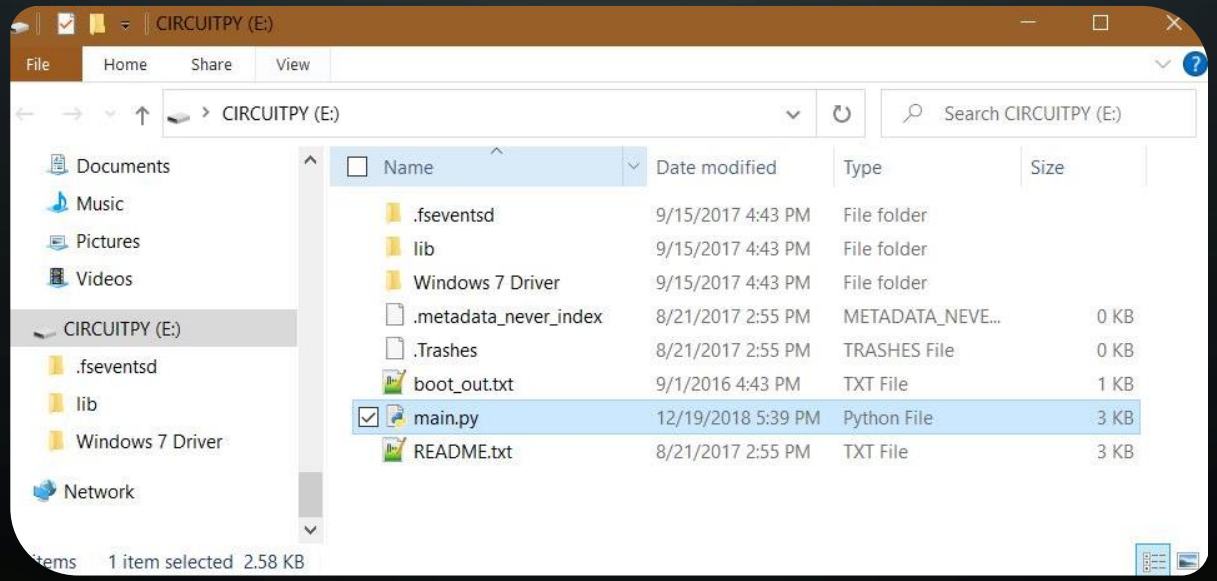


Connecting the Trinket M0

When you connect the Trinket M0 to your machine via USB is should open a small drive named CIRCUITPY.

This drive is where you will place your code and libraries.

It comes installed with a main.py file and some basic library files in a lib directory which are running demo code that will color cycle the on-board DotStar LED.



Verify the Trinket M0 Firmware Version

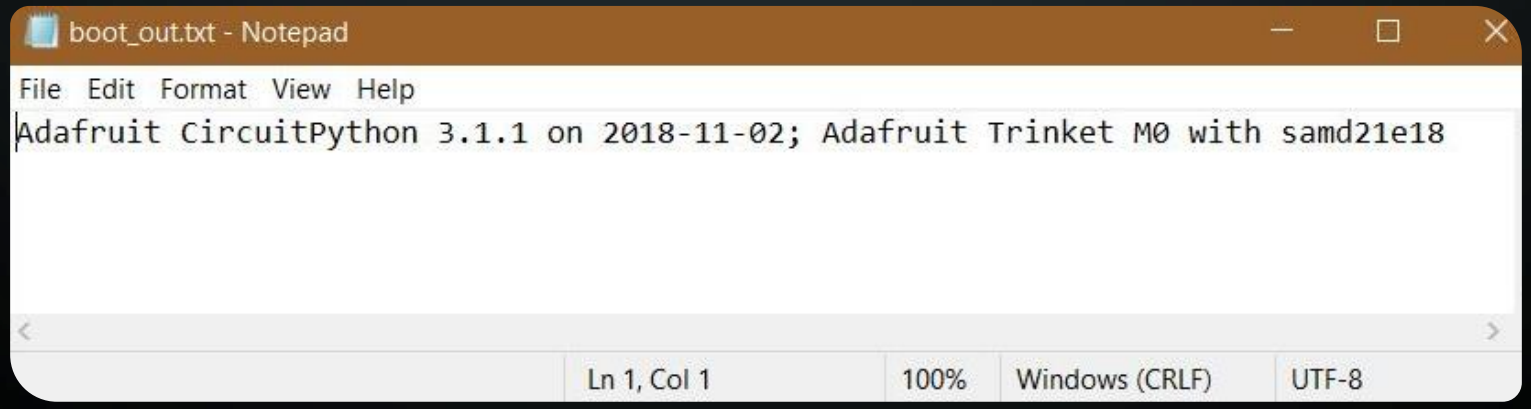
There are several versions of CircuitPython firmware available, and the library files are not compatible between major versions (e.g. 2.x, 3.x, 5.x, 6.x, etc.)

So, you should verify the version of CircuitPython firmware loaded on your Trinket M0 before you start working with it.

This is easily done by connecting it and opening a file named boot_out.txt in the root of the CIRCUITPY drive.

This file will contain a line of text that shows the CircuitPython version.

Note that my Trinket M0 came shipped with version 3.1.1 which I want to upgrade.



Upgrading the Trinket M0 Firmware

To update the firmware is a fairly simple process.

- Download the firmware file from `https://circuitpython.org/board/trinket_m0/`
  - Click the 'Download .UF2 Now' button to download the latest stable firmware version.
  - Note: The latest stable version at the time of this writing is 5.3.1 which is what we will use in these Labs.
- Connect the Trinket M0 which will open a USB drive called CIRCUITPY.
  - Optionally back up any .py files and the lib directory to your machine.
- Click the Reset button on the trinket twice.
  - Not like a mouse Double-click, but more like **Click-pause-Click**.
- You should see the DotStar LED turn green and a new disk drive appear called TRINKETBOOT.
- Copy the .uf2 extension firmware file (e.g. adafruit-circuitpython-trinket_m0-en_US-5.3.1.uf2) to the TRINKETBOOT drive.
- The Red LED will flash then the TRINKETBOOT drive will disappear and the CIRCUITPY drive will reappear.
- The main.py file and library files in lib may be deleted in this process.

That's it, the trinket is now running the new firmware for the Circuit Python version that you copied to it.

CircuitPython Library Files

Whether you recently upgraded or just want to create a new project with your current version of CircuitPython, you will want to download the library files for the version you are working with.

Given the small amount of storage available it is important to only add the libraries you need to your device.

You'll want the compiled library files available on your PC to allow you to keep file size down and copy only what you need when you need it.

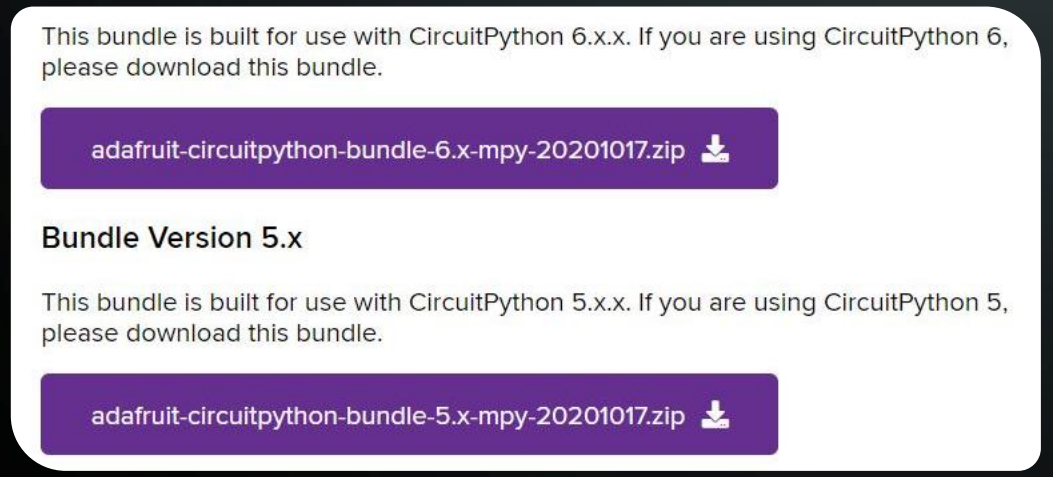
The library file bundle package is also a good source of example code covering various tasks.

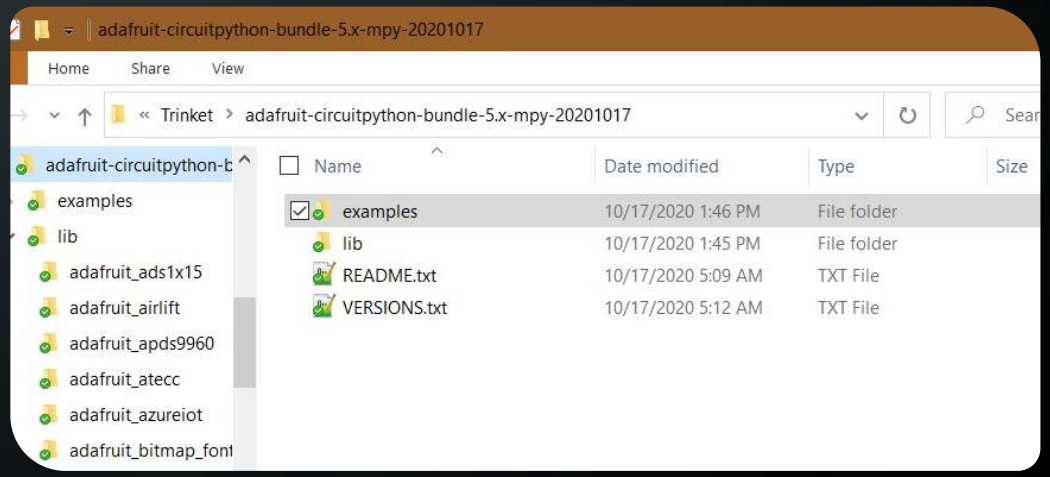
Downloading CircuitPython Library Files

You can find the library file bundle packages for recent versions of CircuitPython here: https://circuitpython.org/libraries

Choose the bundle that corresponds with your version and extract it.

Now just copy any file you need from the lib directory.





Testing Trinket After Firmware Upgrade

When you upgrade the CircuitPython firmware it may delete your python code and library files, or worse leave them in place and non-functional. So, its best to clear any files and load new ones to test.

There are a few empty files that should be left on the CIRCUITPY drive since they are there to prevent your PC from storing hidden files on the tiny drive. These are:

- .metadata_never_index
- .Trashes
- .fseventsd
- .fseventsd/no_log

The lib directory can remain but should be empty.

Testing Trinket After Firmware Upgrade

Create a file named main.py using Mu or your favorite editor.

In the main.py file add the code shown on the right.

This simple code will test that things are working.

This code should blink the small Red LED in the corner of the board and print some text to serial output.

It will also cause the RBG LED in the center of the board to light solid Green indicating that it is running a program without error.

If the Red LED is not blinking or the RBG LED is not solid Green, then there is a problem.

- Check that your indentation is consistent in main.py (Python is strict about indentation)

```
import board
import digitalio
import time
# Assign pin D13 (On-Board Red LED)
rled = digitalio.DigitalInOut(board.D13)
# Set pin IO Direction
rled.direction = digitalio.Direction.OUTPUT
# Main Loop
while True:
    # Serial Output
    print("Hello, CircuitPython!")
    # Set LED state to ON
    rled.value = True
    # Pause for 1 second
    time.sleep(1)
    # Set LED state to OFF
    rled.value = False
    # Pause for 1 second
    time.sleep(1)
```

Troubleshooting Problems

The Trinket M0 and it's CircuitPython firmware will attempt to help you troubleshoot problems.

The DotStar RGB LED will display a status color and flash an error code as shown on the right.

Error messages are sent via serial output. So a serial monitor will allow you to see detailed error messages that will help you correct the problem.

Note: The Mu editor has a built-in serial monitor that can interface with the Trinket M0 and display these messages.

The Trinket M0 uses the DotStar RGB LED on the board to indicate the status of CircuitPython.
Here is how to read it:

- steady GREEN: main.py is running
- pulsing GREEN: main.py has finished or does not exist
- steady YELLOW at start up (4+) Waiting for a reset to indicate that it should start in safe mode
- pulsing YELLOW: In safe mode – (crash & restart)
- steady WHITE: REPL is running
- steady BLUE: boot.py is running

Colors with multiple flashes following indicate a Python exception and then indicate the line number of the error. The color of the first flash indicates the type of error:

- GREEN: IndentationError
- CYAN: SyntaxError
- WHITE: NameError
- ORANGE: OSError
- PURPLE: ValueError
- YELLOW: other error

These are followed by flashes indicating the line number. WHITE flashes are thousands' place, BLUE are hundreds' place, YELLOW are tens' place, and CYAN are one's place.

Integrating Circuits with the Trinket M0

So far, we haven't connected our Trinket to anything. To have it control external circuits it must be integrated into those circuits. To do this we will connect our external circuits to pins on the Trinket.

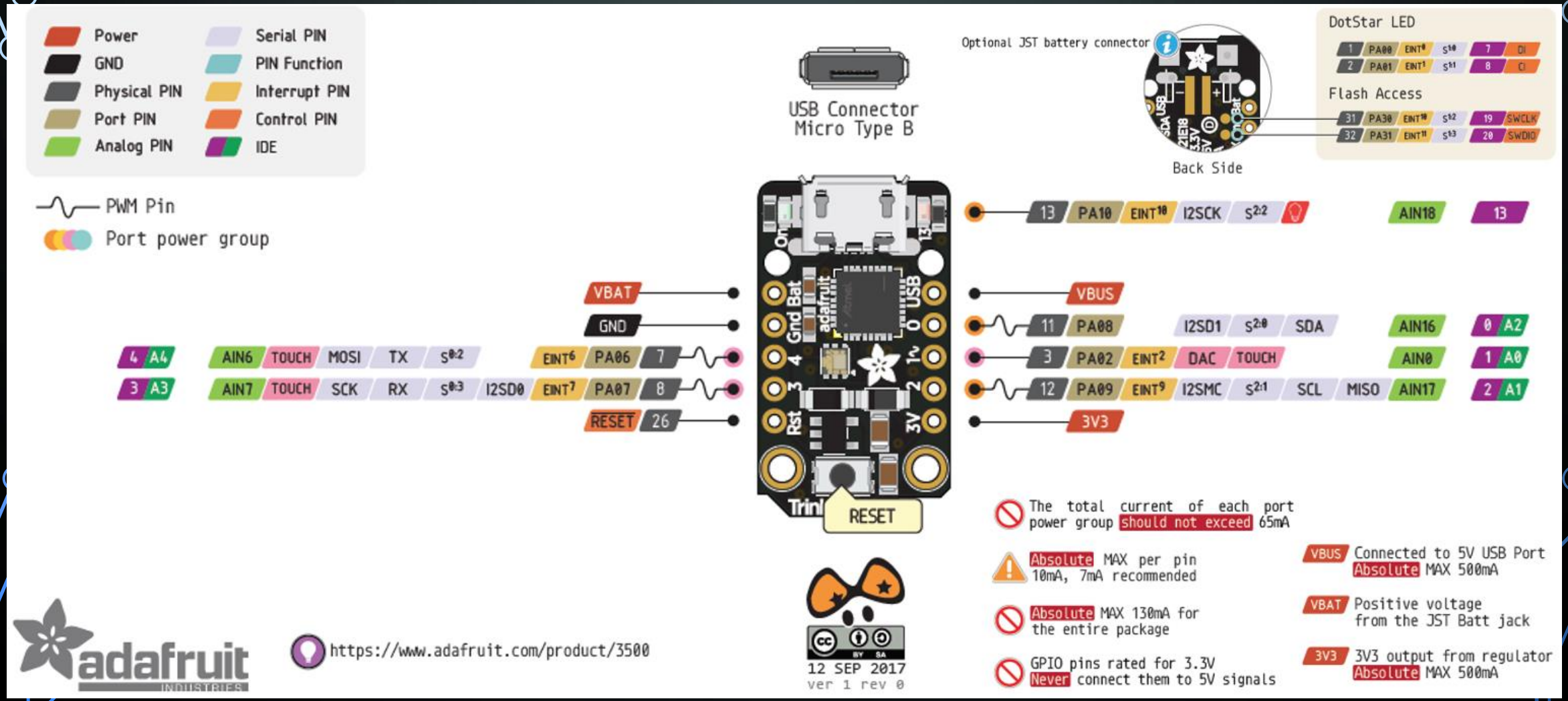
In addition to internal pins (such as those for the onboard LEDs), the Trinket has ten external physical pins that we will use with our external circuits.

There are five general purpose input / output (GPIO) pins labeled 0 thru 4. These each have different features that should be considered (e.g. digital, analog, PWM, touch sensor, etc.)

The other five pins include a voltage input pin labeled ‘Bat’, two voltage output pins labeled ‘USB’ (5V) and ‘3V’, a ground pin labeled ‘Gnd’, and a reset pin labeled ‘Rst’.

Pinout of the Adafruit Trinket M0

This pinout diagram details which pins have which features.



Working with CircuitPython on the Trinket M0

CircuitPython is based on Python 3. So, most things that work with Python 3 will work with CircuitPython.

CircuitPython libraries are separate files designed to work with CircuitPython code. CircuitPython programs require a lot of information to run. CircuitPython is so simple to use because most of this information is processed in the background and stored in libraries. ***Some libraries are built into CircuitPython.*** Others are downloaded and stored on your CIRCUITPY drive in a folder called lib.

CircuitPython looks for a code file on the board to run in the root of the CIRCUITPY drive. There are four options: code.txt, code.py, main.txt and main.py. CircuitPython looks for those files, in that order, and then runs the first one it finds.

Any editor will work to modify the code. Whether you are editing the file directly on the CIRCUITPY drive or copying a code file there, when you save/copy the file it will be immediately run on the board since the board is looking for changes.

Python Quick Reference

Statements Blocks

```
parent statement :
    statement block 1…
    ⋮
    parent statement :
        statement block2…
        ⋮
next statement after block 1
```

indentation !

configure editor to insert 4 spaces in place of an indentation tab.

Variables assignment

=

assignment ⇔ **binding** of a *name* with a *value*

1) evaluation of right side expression value
2) assignment in order with left side names

```
x=1.2+8+sin(y)
a=b=c=0          assignment to same value
y,z,r=9.2,-7.6,0 multiple assignments
a,b=b,a          values swap
a,*b=seq         unpacking of sequence in
*a,b=seq         item and list
x+=3             increment ⇔ x=x+3
x-=2             decrement ⇔ x=x-2
x=None           « undefined » constant value
del x            remove name x
```

and *= /= %= …

Boolean Logic

Comparisons : < > <= >= == != (boolean results) ≤ ≥ = ≠

`a and b`	logical and — *both simulta-neously*
`a or b`	logical or — *one or other or both*

pitfall : `and` *and* `or` *return value of* `a` *or of* `b` *(under shortcut evaluation).*
⇒ *ensure that* `a` *and* `b` *are booleans.*

`not a`	logical not
`True` `False`	True and False constants

Modules/Names Imports

module `truc` ⇔ *file* `truc.py`

```
from monmod import nom1,nom2 as fct
```
→ *direct access to names, renaming with* `as`

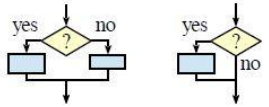
```
import monmod
```
→ *access via* `monmod.nom1` …

modules and packages searched in *python path* (cf `sys.path`)

Conditional Statement

statement block executed only if a condition is true

```
if logical condition :
    statements block
```



Can go with several *elif*, *elif*... and only one final *else*. Only the block of first true condition is executed.

```
if age<=18:
    state="Kid"
elif age>65:
    state="Retired"
else:
    state="Active"
```

with a var `x`:

```
if bool(x)==True:  ⇔ if x:
if bool(x)==False: ⇔ if not x:
```

Function Definition

function name (identifier) — named parameters

```
def fct(x,y,z):
    """documentation"""
    # statements block, res computation, etc.
    return res
```

`fct`

result value of the call, if no computed result to return: `return None`

parameters and all variables of this block exist only *in* the block and *during* the function call (think of a "black box")

Advanced: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`

**args variable positional arguments (→`tuple`), default values,*
***kwargs variable named arguments (→`dict`)*

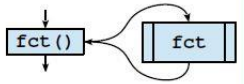
Function Call

```
r = fct(3,i+2,2*i)
```

storage/use of returned value — *one argument per parameter*

this is the use of function name *with parentheses* which does the call

Advanced: **sequence* ***dict*



Base Types

integer, float, boolean, string, bytes

```
int   783   0   -192   0b010   0o642   0xF3
                zero   binary   octal   hexa
float 9.23  0.0  -1.7e-6
                 ×10^-6
bool  True  False
str   "One\nTwo"
        escaped new line
      'I\'m'
        escaped '
bytes b"toto\xfe\775"
        hexadecimal  octal
```

Multiline string:

```
"""X\tY\tZ
1\t2\t3"""
```

escaped tab

immutables

Container Types

- **ordered sequences**, fast index access, repeatable values

```
list  [1,5,9]      ["x",11,8.9]      ["mot"]      []
tuple (1,5,9)      11,"y",7.4        ("mot",)     ()
```

Non modifiable values (immutables) — expression with only comas → `tuple`

```
str bytes  (ordered sequences of chars / bytes)      ""  b""
```

- **key containers**, no *a priori* order, fast key access, each key is unique

```
dictionary  dict {"key":"value"}   dict(a=3,b=4,k="v")   {}
(key/value associations) {1:"one",3:"three",2:"two",3.14:"π"}
collection  set {"key1","key2"}    {1,9,3,0}            set()
```

keys=hashable values (base types, immutables…) — `frozenset` immutable set — *empty*

Conditional Loop Statement

statements block executed ***as long as*** *condition is true*

```
while logical condition :
    statements block
```

beware of infinite loops!

```
s = 0
i = 1
while i <= 100:
    s = s + i**2
    i = i + 1
print("sum:",s)
```

initializations ***before*** *the loop*
condition with a least one variable value (here `i`*)*
make condition variable change !

Algo: $s = \sum_{i=1}^{i=100} i^2$

Loop Control

`break`	*immediate exit*
`continue`	*next iteration*

`else` *block for* ***normal*** *loop exit.*

Iterative Loop Statement

statements block executed ***for each*** *item of a container or iterator*

```
for var in sequence :
    statements block
```

Go over sequence's **values**

```
s = "Some text"
cnt = 0
for c in s:
    if c == "e":
        cnt = cnt + 1
print("found",cnt,"'e'")
```

initializations ***before*** *the loop*
loop variable, ***assignment managed by*** `for` ***statement***

Algo: count number of `e` *in the string.*

loop on dict/set ⇔ loop on keys sequences
use *slices* to loop on a subset of a sequence

Go over sequence's **index**

- modify item at index
- access items around index (before / after)

```
lst = [11,18,9,12,23,4,17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:",lst,"-lost:",lost)
```

Algo: limit values greater than 15, memorizing of lost values.

good habit : don't modify loop variable

Go simultaneously over sequence's **index** and **values**:

```
for idx,val in enumerate(lst):
```

Display

```
print("v=",3,"cm :",x,",",y+4)
```

items to display : literal values, variables, expressions

`print` options:

- `sep=" "` — items separator, default space
- `end="\n"` — end of print, default new line
- `file=sys.stdout` — print to file, default standard output

Input

```
s = input("Instructions:")
```

`input` always returns a **string**, convert it to required type (cf. boxed *Conversions* on the other side).

Maths

floating numbers… approximated values — *angles in radians*

Operators: `+ - * / // % **`
Priority (…) × ÷ ↑ ↑ a^b
integer ÷ ÷ remainder

@ → matrix × *python3.5+*`numpy`

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```

usual order of operations

```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0   ✓
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```

modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, *etc. (cf. doc)*

Integer Sequences

```
range([start,] end [,step])
```

start default 0, *end* not included in sequence, *step* signed, default 1

```
range(5)→0 1 2 3 4
range(3,8)→3 4 5 6 7
range(2,12,3)→2 5 8 11
range(20,5,-5)→20 15 10
range(len(seq))→ sequence of index of values in seq
```

range provides an immutable sequence of int constructed as needed

CIRCUITPYTHON TRINKET PROJECTS

This next section will outline some CircuitPython projects using the Trinket M0.

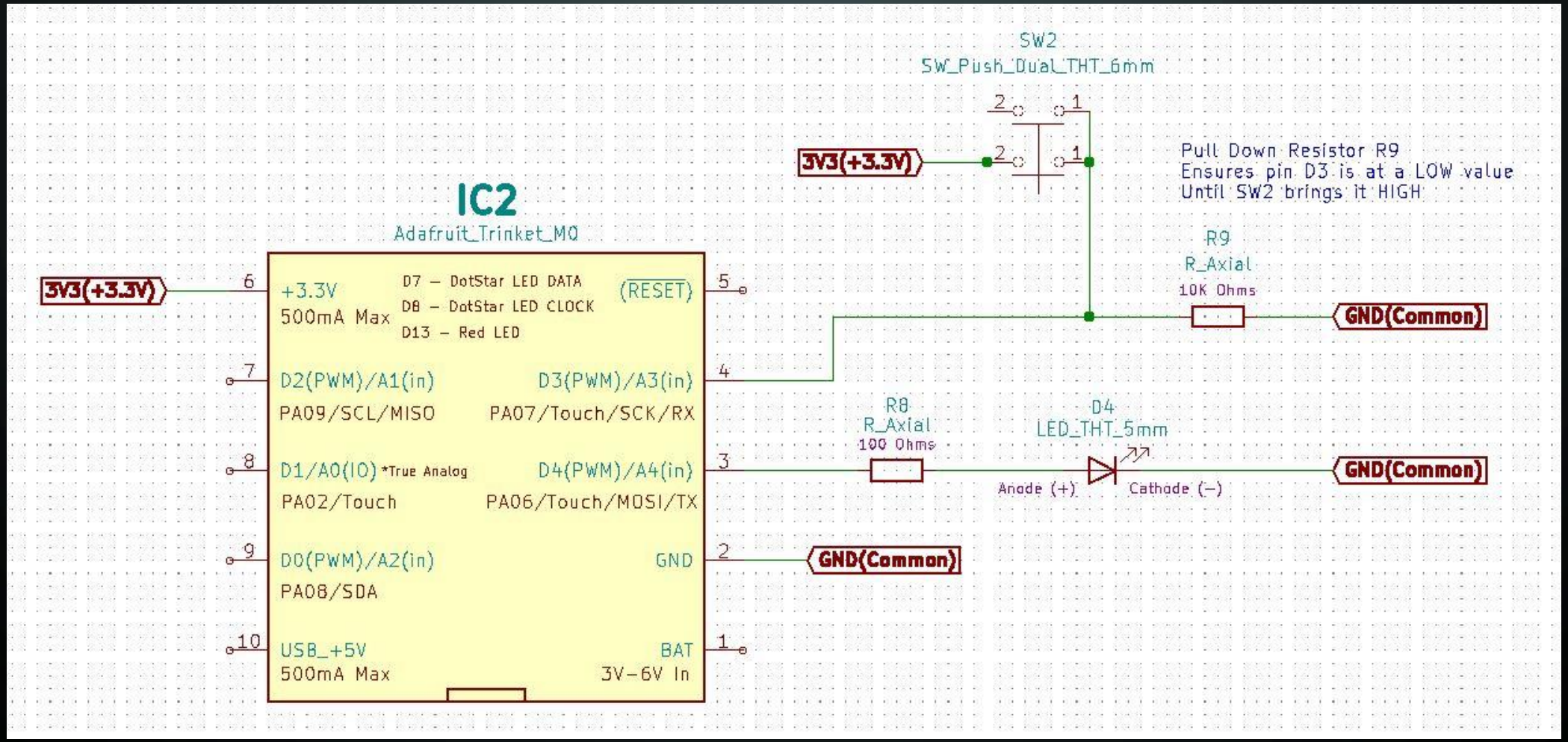
These projects will be centered around two different physical circuit layouts with several CircuitPython code blocks for each.

- Adafruit Trinket LED Control – Lab HHV2020_04
  - Blink the on-board Red LED
  - Blink both external and on-board LEDs
  - Fade external LED on/off using PWM
  - Turn LEDs on/off using a Tactile Switch
- Adafruit DotStar RGB LED and Touch Sensor – Lab HHV2020_05
  - Blink the on-board DotStar RGB LED
  - Color Cycle the on-board DotStar RGB LED
  - Use Touch Sensor to Control an external LED
  - Use Touch Sensor to Color Change the on-board DotStar RGB LED

The Lab reference numbers refer to the BSidesDFW Hardware Hacking Village Videos which can be accessed here: **https://altbier.us/bsidesdfwHHV2020/**

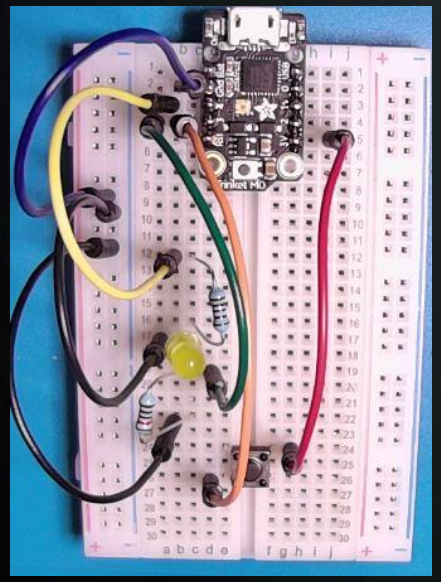
ADAFRUIT TRINKET LED CONTROL

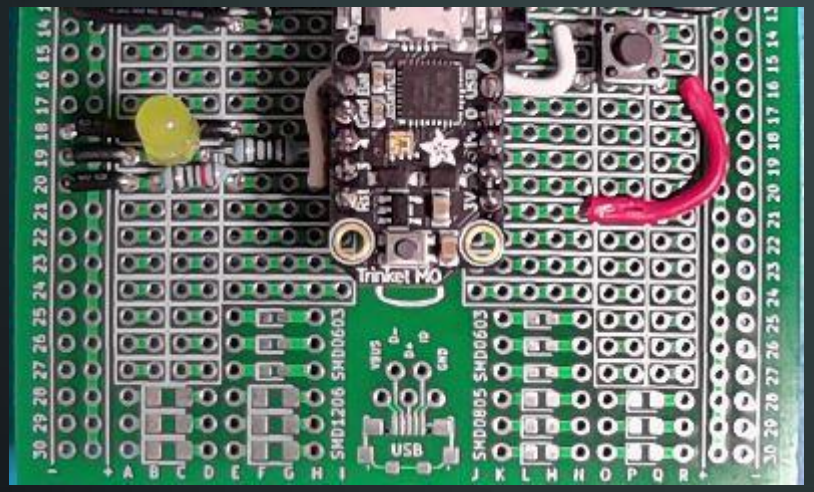
Schematic

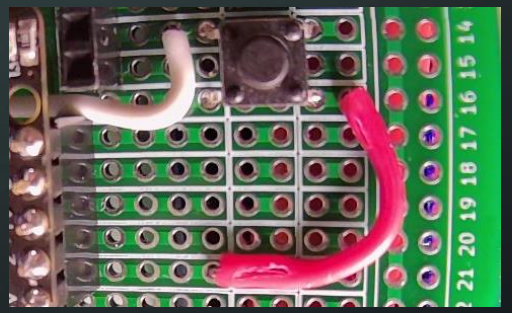


ADAFRUIT TRINKET LED CONTROL

Physical Layout



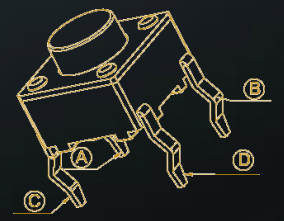




Wire up a circuit as shown in the schematic and physical layout.

Components:

- 1x Resistor 100 Ohm
- 1x Resistor 10K Ohm
- 1x LED 5mm
- 1x Tactile Switch SPST 4pin



Strip Board Connection Details

- Trinket – I17-21 and J17-21
- Resistor 100 Ohm – D19 and H19
- LED
  - Anode – C19
  - Cathode – B19
- Wire – A19 and VCC19
- Wire – E18 and VCC18
- Switch SPST 4 Pin
  - Pin A(1) – N14
  - Pin B(1) – Q14
  - Pin C(2) – N16
  - Pin D(2) – Q16
- Wire – R16 and N21
- Wire – M14 and H20
- Resistor 10K Ohm – E20 and B20
- Wire – A20 and VCC20

ADAFRUIT TRINKET LED CONTROL

Blink the on-board Red LED

This simple bit of code will blink the on-board Red LED. This is the same code shown on the 'Testing Trinket…' slide.

Let's walk through what it is doing:

Lines 1-3 import the library files we will use. These libraries are built into CircuitPython so they will not be in the lib directory.

Line 5 assigns pin Digital-13 to an object named rled. This is the pin associated with the Red LED.

Line 7 sets the IO pin direction to OUTPUT

Line 9 starts the main While loop that will run indefinitely.

Line 11 prints a string to the serial output. You can see this text output with a serial monitor.

Line 13 sets the value of rled to True which turns ON the LED.

Line 15 pauses the program for 1 second

Line 17 sets the value of rled to False which turns OFF the LED.

Line 19 pauses the program for 1 second

main.py

```
1  import board
2  import digitalio
3  import time
4  # Assign pin D13 (On-Board Red LED)
5  rled = digitalio.DigitalInOut(board.D13)
6  # Set pin IO Direction
7  rled.direction = digitalio.Direction.OUTPUT
8  # Main Loop
9  while True:
10     # Serial Output
11     print("Hello, CircuitPython!")
12     # Set LED state to ON
13     rled.value = True
14     # Pause for 1 second
15     time.sleep(1)
16     # Set LED state to OFF
17     rled.value = False
18     # Pause for 1 second
19     time.sleep(1)
```

This Code Is Available Here: https://github.com/gowenrw/BSidesDFW_2020_HHV/

ADAFRUIT TRINKET LED CONTROL

Blink both external and on-board LEDs

This simple bit of code is like the previous code, adding an external LED. It will blink both the on-board and external LEDs

Let's walk through what it is doing:

Lines 1-3 import the library files we will use.

Line 5 assigns pin D13 (on-board LED) to an object named rled.

Line 7 assigns pin D4 (external LED) to an object named led4.

Line 9-10 sets the IO pin direction to OUTPUT for both pins

Line 12 starts the main While loop that will run indefinitely.

Line 14 sets the value of both LEDs to True which turns them ON.

Line 16 prints strings and values to the serial output.

Line 18 pauses the program for 1 second

Line 20-24 repeats the set/print/pause turning the LEDs OFF.

main.py

```
1  import board
2  import digitalio
3  import time
4  # Assign pin D13 (On-Board Red LED)
5  rled = digitalio.DigitalInOut(board.D13)
6  # Assign pin D4 (External LED)
7  led4 = digitalio.DigitalInOut(board.D4)
8  # Set pin IO Direction
9  rled.direction = digitalio.Direction.OUTPUT
10 led4.direction = digitalio.Direction.OUTPUT
11 # Main Loop
12 while True:
13     # Set LED state to ON
14     rled.value = led4.value = True
15     # Serial Output LED values
16     print("rled Value: ", rled.value, "led4 Value: ", led4.value)
17     # Pause for 1 second
18     time.sleep(1)
19     # Set LED state to OFF
20     rled.value = led4.value = False
21     # Serial Output LED values
22     print("rled Value: ", rled.value, "led4 Value: ", led4.value)
23     # Pause for 1 second
24     time.sleep(1)
```

This Code Is Available Here: https://github.com/gowenrw/BSidesDFW_2020_HHV/

ADAFRUIT TRINKET LED CONTROL

Fade external LED on/off using PWM

In the previous code we defined a digital output pin for our LED.

In this code we define a PWM (Pulse Width Modulation) output pin which mimics analog allowing the LED to be in between ON or OFF.

Let's walk through what's different:

- We import the pulseio instead of the digitalio library.
- We assign the led4 object to a PWMOut which is output only so we don't set a direction.
- We use for loops with a range of 100. The first loop iterates up to fade the LED ON, the second iterates down to fade it OFF.
- Instead of setting a value of True(HIGH) or False(LOW) we are setting the duty_cycle which represents the percentage of time the pin will be in the HIGH state. The duty_cycle is a 16-bit number (0 to 65535).

main.py

```
import board
import pulseio
import time
# Assign pin D4 (External LED)
led4 = pulseio.PWMOut(board.D4)
# Note: PWMOut.duty_cycle is a 16 bit value between 0 and 65535
# Main Loop
while True:
    # Fade LED from 0% to 100% - OFF to ON
    for i in range(100):
        # Set LED Value
        led4.duty_cycle = int(i / 100 * 65535)
        # Serial Output LED value
        print("led4 Value: ", led4.duty_cycle)
        # Pause for 0.01 second
        time.sleep(0.01)
    # Fade LED from 100% to 0% - ON to OFF
    for i in range(100, -1, -1):
        # Set LED Value
        led4.duty_cycle = int(i / 100 * 65535)
        # Serial Output LED value
        print("led4 Value: ", led4.duty_cycle)
        # Pause for 0.01 second
        time.sleep(0.01)
```

This Code Is Available Here: https://github.com/gowenrw/BSidesDFW_2020_HHV/

ADAFRUIT TRINKET LED CONTROL

Turn LEDs on/off using a Tactile Switch

This code will read the state of a tactile switch (button) and use that to control the LEDs turning them on when pressed.

We will use the button value in two ways, as a conditional check and as a raw value to send to an object (an LED).

Let's look at the key items in this code:

- Assign **rled** and **led4** pins as **digital output**
- Assign **sw1** pin as **digital input**
- Read in the **sw1** value and print it to serial
- Use an **if** conditional statement to check if sw1 is True and set rled to True (LED ON) if it is, **else** set it to False (LED OFF)
- Set **led4** to be equal to sw1 value (True or False, ON or OFF)

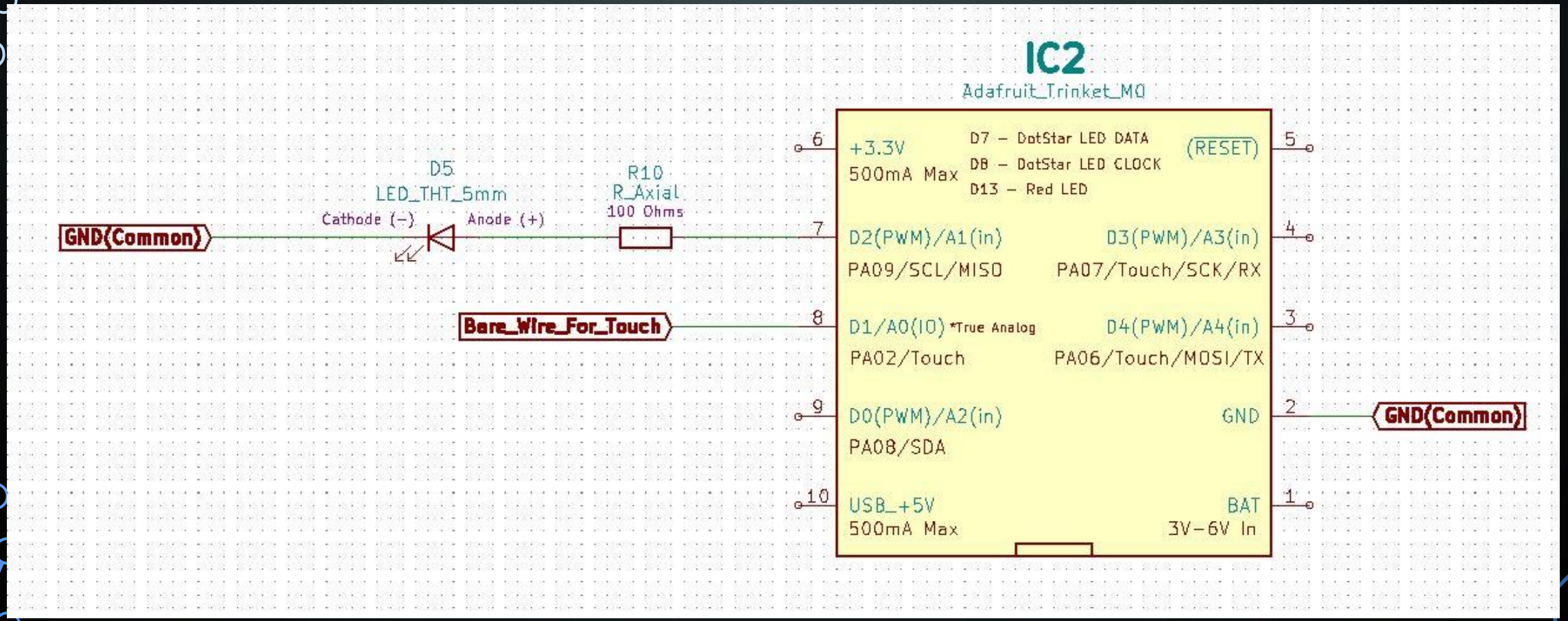
main.py

```
import board
import digitalio
import time
# Assign pin D13 (On-Board Red LED)
rled = digitalio.DigitalInOut(board.D13)
# Assign pin D4 (External LED)
led4 = digitalio.DigitalInOut(board.D4)
# Assign pin D3 (External Switch Input)
sw1 = digitalio.DigitalInOut(board.D3)
# Set IO Direction
rled.direction = digitalio.Direction.OUTPUT
led4.direction = digitalio.Direction.OUTPUT
sw1.direction = digitalio.Direction.INPUT
# Main Loop
while True:
    # Read Button Value
    Button = sw1.value
    # Serial Output Button Value
    print("Button Value:", Button)
    # Turn rled ON/OFF based on conditional Button Value
    if (Button == True):
        # Set LED state to ON
        rled.value = True
    else:
        # Set LED state to OFF
        rled.value = False
    # Set led4 to Button Value
    led4.value = Button
    # Pause for 0.2 second
    time.sleep(0.2)
```

This Code Is Available Here: https://github.com/gowenrw/BSidesDFW_2020_HHV/

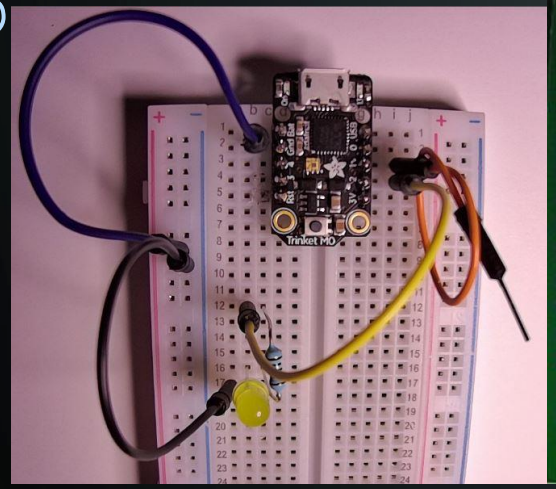
ADAFRUIT DOTSTAR RGB LED AND TOUCH SENSOR

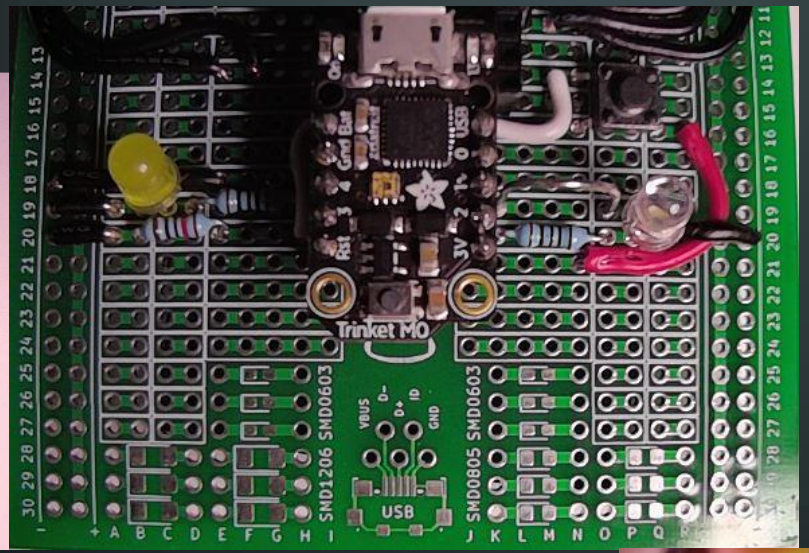
Schematic



ADAFRUIT DOTSTAR RGB LED AND TOUCH SENSOR

Physical Layout





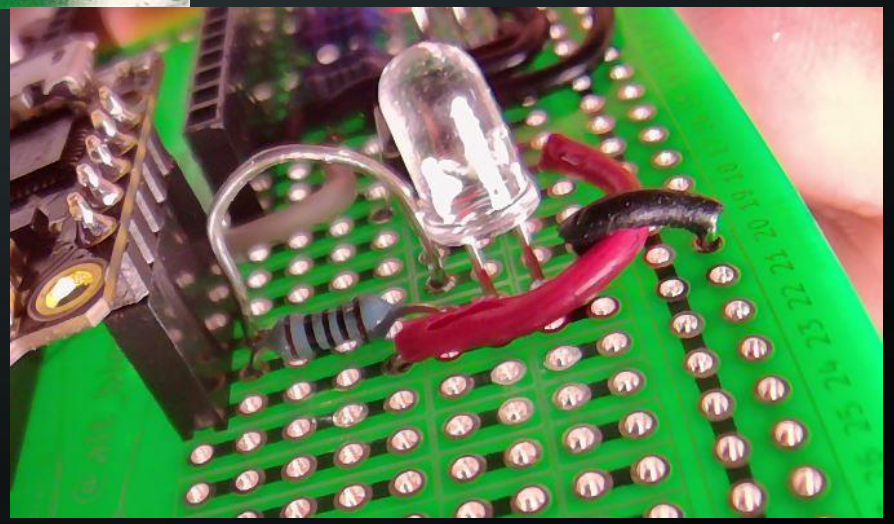
Strip Board Connection Details

- Trinket – I17-21 and J17-21
- Resistor 100 Ohm – K20 and O20
- LED
  - Anode – P20
  - Cathode – Q20
- Wire – R20 and VCC20
- Bare Wire – K19 and O19

Wire up a circuit as shown in the schematic and physical layout.

Components:

- 1x Resistor 100 Ohm
- 1x LED 5mm
- 1x Bare Wire (for touch)



ADAFRUIT DOTSTAR RGB LED AND TOUCH SENSOR

CircuitPython Library Files Needed

The code used in the following examples will require some library files that are not built into CircuitPython.

You can find the library file bundle packages for recent versions of CircuitPython here: `https://circuitpython.org/libraries`

Download the bundle for the version you are working with (5.x) and extract it in a directory on your PC.

Locate these two library files on your PC and copy them to the lib directory on the CIRCUITPY drive:

- **adafruit_dotstar.mpy**
- **adafruit_pypixelbuf.mpy**

ADAFRUIT DOTSTAR RGB LED AND TOUCH SENSOR

Blink the on-board DotStar RGB LED

This code will blink the on-board DotStar RGB LED. This uses some library files not included in CircuitPython.

Let's look at the key items in this code:

- Import adafruit_dotstar library
  - It calls the library adafruit_pypixelbuf
- Assign **dot** object data pin APA102_SCK, clock pin APA102_MOSI, pixel_num=1, and brightness=0.2
  - Note: The board lib defines pins D7 & D8 as different names
- Set the Value of dot[0] to Red, Green, Blue, then OFF using a value list formatted (R,G,B) with values 0-255
- Pause briefly between color changes.

main.py

```
import board, time
import adafruit_dotstar as dotstar
# Assign pins for Internal DotStar RGB LED
# DotStar Data Pin D7 is defined as board.APA102_SCK
# DotStar Clock Pin D8 is defined as board.APA102_MOSI
# DotStar parameters (DATA, CLOCK, num_pixels, brightness)
dot = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
### MAIN LOOP ###
while True:
    # Blink internal DotStar LED
    # DotStar value format (R,G,B) with each value range 0-255
    # Blink Red
    dot[0] = (255, 0, 0) # Red
    time.sleep(0.5)
    # Blink Green
    dot[0] = (0, 255, 0) # Green
    time.sleep(0.5)
    # Blink Blue
    dot[0] = (0, 0, 255) # Blue
    time.sleep(0.5)
    # Blink OFF
    dot[0] = (0, 0, 0) # OFF
    time.sleep(1)
```

This Code Is Available Here: https://github.com/gowenrw/BSidesDFW_2020_HHV/

ADAFRUIT DOTSTAR RGB LED AND TOUCH SENSOR

Color Cycle the on-board DotStar RGB LED

This code will Color Cycle the on-board DotStar RGB LED.

Let's look at the key items in this code:

- Import adafruit_dotstar library
- Assign **dot** object data pin APA102_SCK, clock pin APA102_MOSI, pixel_num=1, and brightness=0.2
- Define a function wheel(pos) that will take an integer and return a list formatted (R,G,B) with values 0-255
- Set dot[0] to the value that wheel(**i**) returns
- Increment the var **i** keeping it between 0-255
- Pause briefly between color changes.

main.py

```
import board, time
import adafruit_dotstar as dotstar
# Assign pins for Internal DotStar RGB LED
# DotStar Data Pin D7 is defined as board.APA102_SCK
# DotStar Clock Pin D8 is defined as board.APA102_MOSI
# DotStar parameters (DATA, CLOCK, num_pixels, brightness)
dot = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
# Function to produce a nice color swirl
def wheel(pos):
    if (pos < 0) or (pos > 255):
        return (0, 0, 0)
    if (pos < 85):
        return (int(pos * 3), int(255 - (pos*3)), 0)
    elif (pos < 170):
        pos -= 85
        return (int(255 - pos*3), 0, int(pos*3))
    else:
        pos -= 170
        return (0, int(pos*3), int(255 - pos*3))
# Iteration Var
i = 0
### MAIN LOOP ###
while True:
    # spin internal DotStar LED around!
    dot[0] = wheel(i & 255)
    i = (i+1) % 256  # run from 0 to 255
    time.sleep(0.05)
```

This Code Is Available Here: https://github.com/gowenrw/BSidesDFW_2020_HHV/

ADAFRUIT DOTSTAR RGB LED AND TOUCH SENSOR

Use Touch Sensor to Control an external LED

This code will use a Capacitive Touch Sensor to control and external LED turning it on when touched.

Let's look at the key items in this code:

- Import **touchio** library
- Assign **touch** object to **TouchIn** analog pin **A0**
- Read the value of **touch**
- Use an **if** conditional statement to test for a touch.
  - Given that the **touch** pin is analog it will register values all the time and we must set our test value higher than ambient values
- If a touch is registered set **led5** to **True** turning it ON
- Pause very briefly between loops.

main.py

```
import board, time
import digitalio
import touchio
# Assign pin D4 (External LED)
led5 = digitalio.DigitalInOut(board.D2)
# Assign Capacitive touch on A0 (Using Analog for raw_value)
touch = touchio.TouchIn(board.A0)
# Set IO Direction
led5.direction = digitalio.Direction.OUTPUT
# Main Loop
while True:
    # Read Touch Raw Value
    tval = touch.raw_value
    # Serial Output Touch Values
    print("Touch Value:", tval)
    if tval > 3500:
        print("Touched! LED ON")
        led5.value = True
    else:
        led5.value = False
    # Pause for 0.2 second
    time.sleep(0.2)

```

This Code Is Available Here: https://github.com/gowenrw/BSidesDFW_2020_HHV/

ADAFRUIT DOTSTAR RGB LED AND TOUCH SENSOR

Use Touch Sensor to Color Change the DotStar RGB LED

This is the same code we used earlier to Color Cycle the DotStar RGB LED with a slight modification to add touch.

The color will now only cycle when touched staying steady at its current color when not touched.

Let's look at the key items added to the existing code:

- Added the **touchio** library import
- Assign **touch** object to **TouchIn** analog pin **A0**
- Read the value of **touch**
- Use an **if** conditional statement to test for a touch and increment the iteration variable **i** only if touched
- Pause very briefly between loops.

main.py

```
import board, time
import adafruit_dotstar as dotstar
import touchio
# Assign pins for Internal DotStar RGB LED
# DotStar Data Pin D7 is defined as board.APA102_SCK
# DotStar Clock Pin D8 is defined as board.APA102_MOSI
# DotStar parameters (DATA, CLOCK, num_pixels, brightness)
dot = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
# Assign Capacitive touch on A0 (Using Analog for raw_value)
touch = touchio.TouchIn(board.A0)
# Function to produce a nice color swirl
def wheel(pos):
    if (pos < 0) or (pos > 255):
        return (0, 0, 0)
    if (pos < 85):
        return (int(pos * 3), int(255 - (pos*3)), 0)
    elif (pos < 170):
        pos -= 85
        return (int(255 - pos*3), 0, int(pos*3))
    else:
        pos -= 170
        return (0, int(pos*3), int(255 - pos*3))
# Iteration Var
i = 0
### MAIN LOOP ###
while True:
    # Read Touch Raw Value
    tval = touch.raw_value
    print("Touch Value:", tval)
    # spin internal DotStar LED around!
    dot[0] = wheel(i & 255)
    if tval > 3500:
        print("Touched! Cycle LED Color")
        i = (i+1) % 256  # run from 0 to 255
    time.sleep(0.05)
```

This Code Is Available Here: https://github.com/gowenrw/BSidesDFW_2020_HHV/

THANK YOU

I hope you enjoyed this presentation and learned something from it.

-- @alt_bier

This Slide Deck – https://altbier.us/circuitpython/

Code – https://github.com/gowenrw/BSidesDFW_2020_HHV/