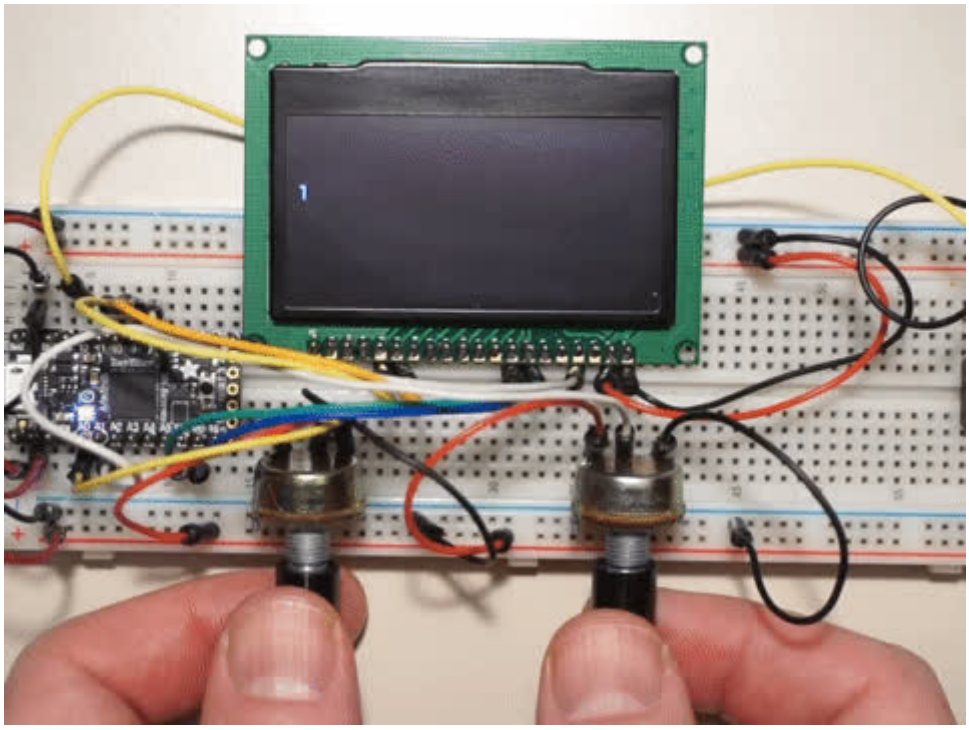
CircuitPython OLED and Dual Knob Sketcher

Created by Carter Nelson

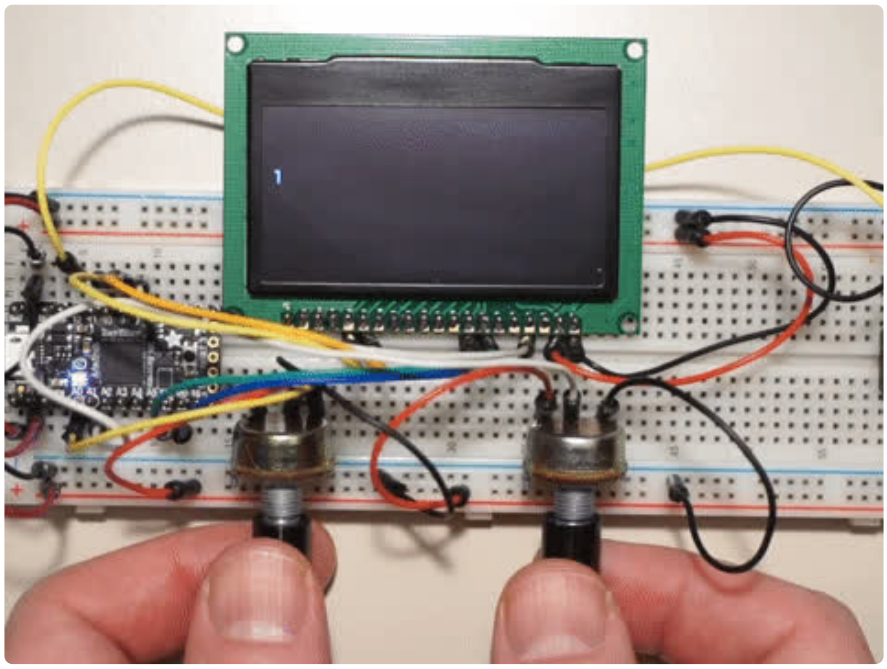


https://learn.adafruit.com/circuitpython-oled-knob-sketcher

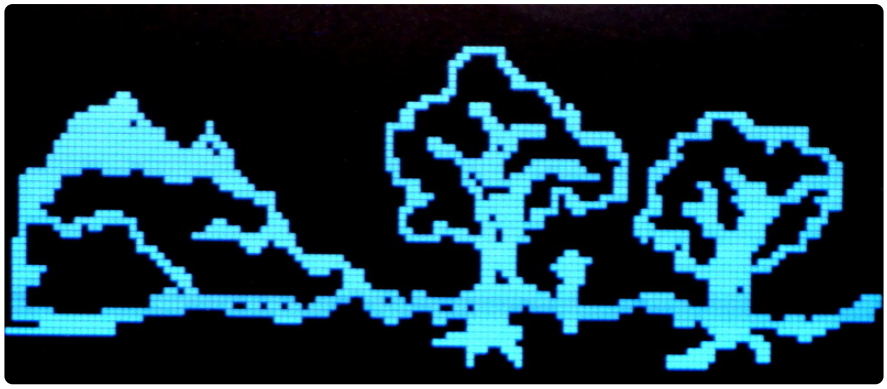
Last updated on 2025-09-02 01:06:09 PM EDT

Table of Contents

Overview

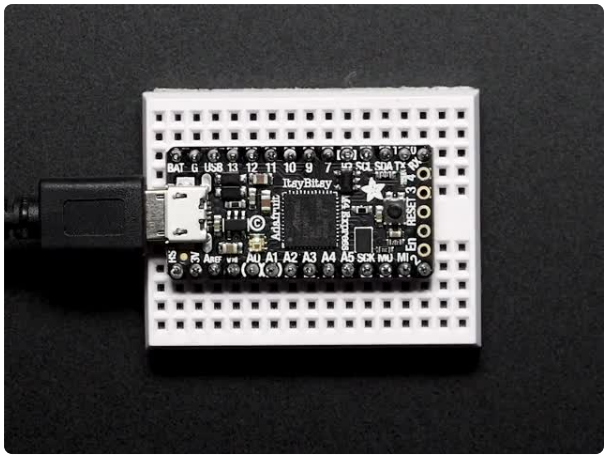


In this guide we will create a simple knob sketcher toy using a graphical display and two potentiometers, also known as trim pots or just pots. We'll go over the basic idea of drawing a pixel to the display as well as how to use the pots to generate a variable voltage. The two concepts will be put together to create a couple different variations of a knob sketcher toy.



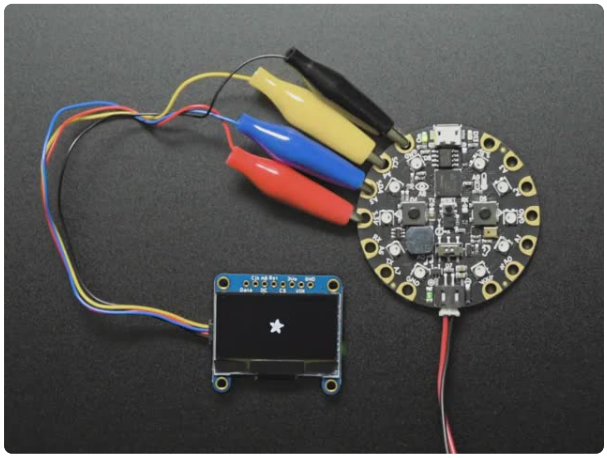
Parts

This guide uses an Adafruit ItsyBitsy M4 Express as the main board. However, you could adapt this to other boards that run CircuitPython and have the available pins. You might need to change the example code to match the pins for your board.

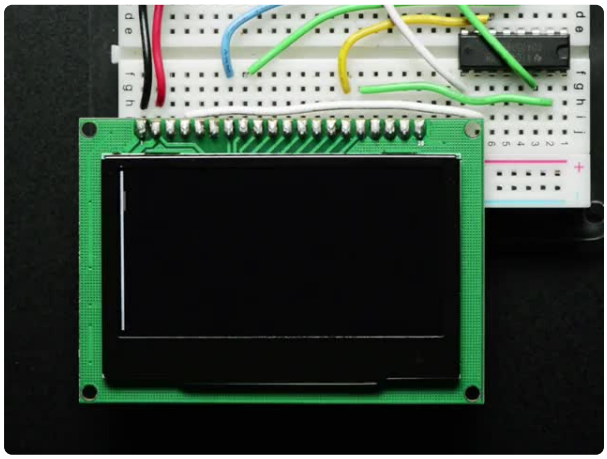


[Adafruit ItsyBitsy M4 Express featuring ATSAMD51]
What's smaller than a Feather but larger than a Trinket? It's an Adafruit ItsyBitsy M4 Express featuring the Microchip ATSAMD51! Small,...
https://www.adafruit.com/product/3800

The other key part for this project is a graphical display. This guide uses the two displays below, but you could also substitute for other similar displays as long as there is a CircuitPython driver available.

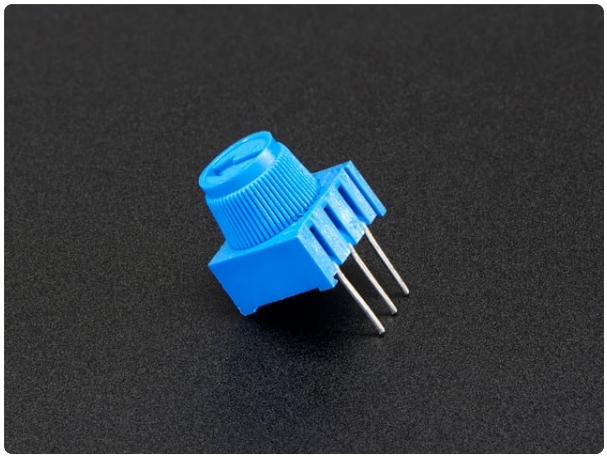


[Monochrome 1.3" 128x64 OLED graphic display - STEMMA QT / Qwiic]
These displays are small, only about 1.3" diagonal, but very readable due to the high contrast of an OLED display. This display is made of 128x64 individual white OLED pixels,...
https://www.adafruit.com/product/938



[Monochrome 2.42" 128x64 OLED Graphic Display Module Kit]
If you've been diggin' our monochrome OLEDs but need something bigger, this display will delight you! These displays...
https://www.adafruit.com/product/2719

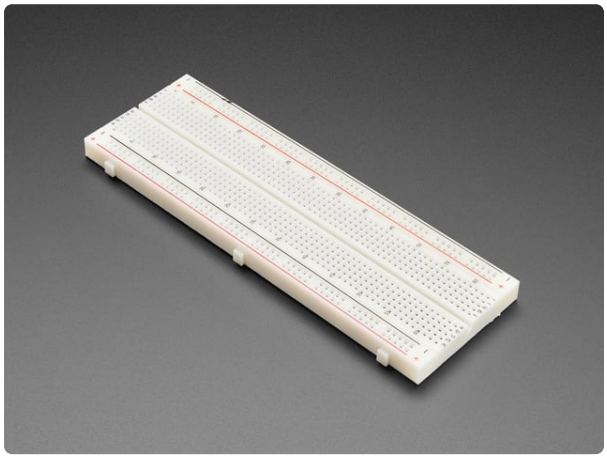
You'll need a pair of potentiometers also. The two options used in this guide are linked below. Pots are also pretty common things, so any similar ones would likely work.



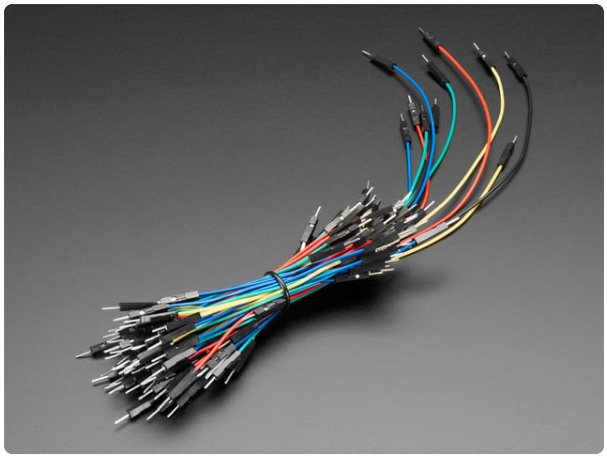
Breadboard trim potentiometer
These are our favorite trim pots, perfect for breadboarding and prototyping. They have a long grippy adjustment knob and with 0.1" spacing, they plug into breadboards or...
https://www.adafruit.com/product/356

Panel Mount 10K potentiometer (Breadboard Friendly)
This potentiometer is a two-in-one, good in a breadboard or with a panel. It's a fairly standard linear taper 10K ohm potentiometer, with a grippy shaft. It's smooth and easy...
https://www.adafruit.com/product/562

In addition, you'll need a breadboard and some hookup wires. A button is used for the second knob sketcher. Details are provided later in the guide.



Full Sized Premium Breadboard - 830 Tie Points
This is a 'full-size' premium quality breadboard, 830 tie points. Good for small and medium projects. It's 2.2" x 7" (5.5 cm x 17 cm) with a standard double-strip...
https://www.adafruit.com/product/239



Breadboarding wire bundle
75 flexible stranded core wires with stiff ends molded on in red, orange, yellow, green, blue, brown, black and white. These are a major improvement over the "box of bent...
https://www.adafruit.com/product/153

Software Setup

The code in this guide is written in CircuitPython. So, before proceeding, make sure you have CircuitPython and the latest libraries installed on your board.

If you are new to CircuitPython, read these guides first. They will explain how to install CircuitPython and the necessary libraries.

- Welcome to CircuitPython (https://adafru.it/BIM)
- CircuitPython Essentials (https://adafru.it/cpy-essentials)

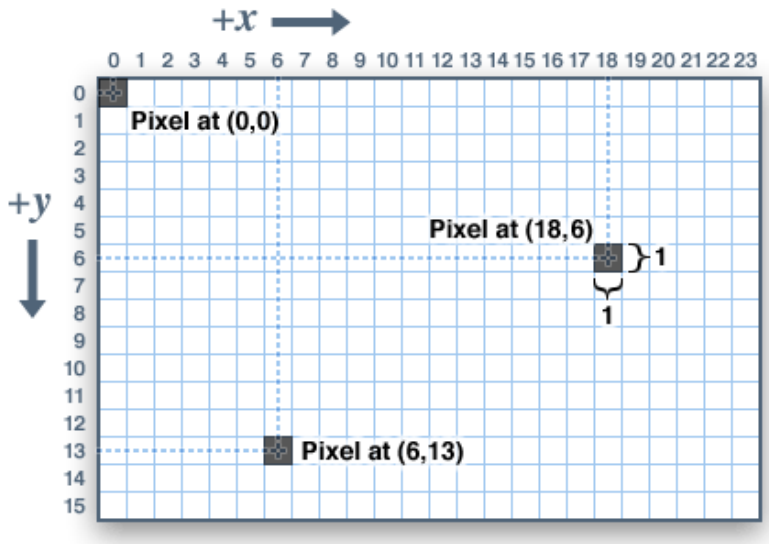
They also cover how to load and run code, access the REPL for troubleshooting, etc.

Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and its installation in this tutorial (https://adafru.it/ANO).

Drawing Pixels

Many graphics libraries (like this one (https://adafru.it/DtY)) will provide functions for drawing various graphical primitives like lines, circles, squares, etc. But we will only

need one very simple primitive for this project - the lowly pixel. There is a good discussion about the pixel and associate coordinate system in the GFX guide (https://adafru.it/DtY). The excellent figure below is borrowed from that guide. Think of this as looking at the front of your display.



The coordinate system starts in the upper left hand corner of the display. To specify a pixel location, you provide the x value and the y value in the form (x, y). This is another one of those computer things where counting starts at 0. Therefore, the very first pixel in the upper left hand corner is (0, 0). The x value is how far to the right, and the y value is how far down. In the figure above you can see two other pixels for x=6, y=13 => (6, 13) and for x=18, y=6 => (18, 6).

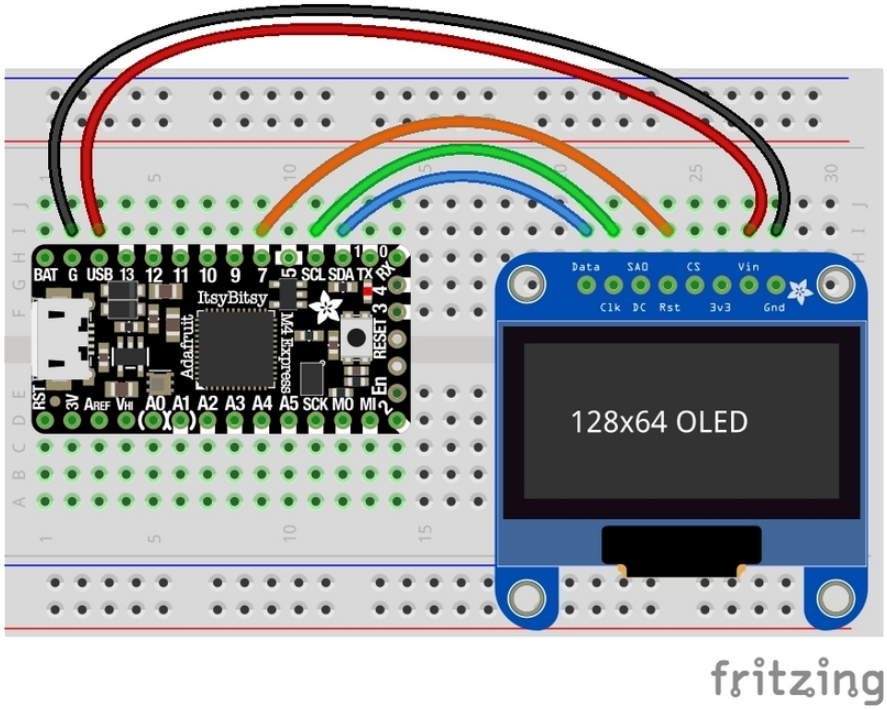
The other thing to note is that every display will have an associate `width` and `height`. This is basically the total number of pixels in each direction. For the example above, the width would be 24 and the height would be 16. Note that the last pixel coordinates are 23 in x and 15 in y. That's due to the counting from 0 thing again. So to draw the very last pixel - all the right, all the way down, the coordinates would be (width -1, height -1).

One Little Pixel

Let's wire up a display and play around with drawing pixels.

The diagram below shows how to wire up a 1.3" OLED (http://adafru.it/938) to an ItsyBitsy M4 Express (http://adafru.it/3800). The connection uses I2C, which is covered in the guide for the OLED, so be sure to read that first (https://adafru.it/DtZ).

Read the OLED guide to make sure the display is configured for I2C.



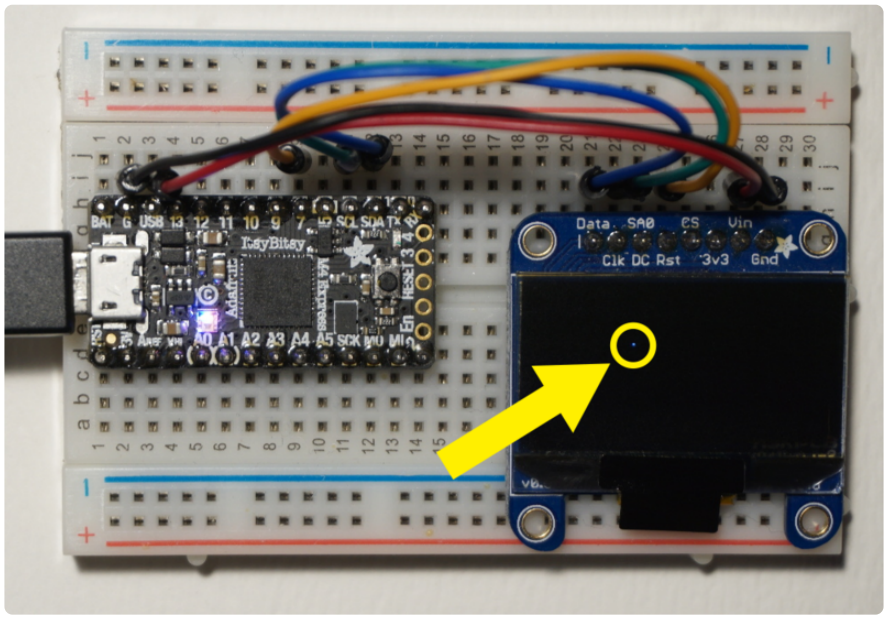
OK, here's the code. You can either save this as **code.py** to have it run automatically, or you can save it to another file name and run it manually.

```
# SPDX-FileCopyrightText: 2018 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Import the needed libraries
import board
import busio
from digitalio import DigitalInOut
import adafruit_ssd1306

# Create I2C bus
i2c = busio.I2C(board.SCL, board.SDA)

# Define display dimensions and I2C address
WIDTH = 128
HEIGHT = 64
ADDR = 0x3d

# Create the digital out used for display reset
rst = DigitalInOut(board.D7)

# Create the display
display = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=ADDR, reset=rst)
display.fill(0)
display.show()

# Define pixel location
```

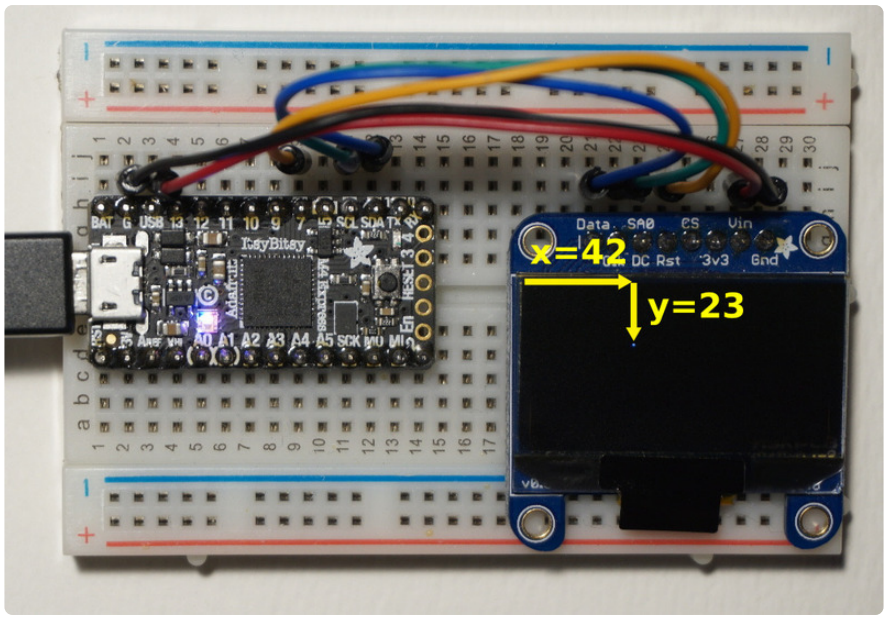
```
x = 42
y = 23
# Draw the pixel
display.pixel(x, y, 1)
display.show()
```

Remember, you can save the code as code.py so it will run automatically.

Once you have that code loaded and it runs....voila! There's the pixel! Do you see it? It's super tiny on this little display, but there it is. It's so cute. I think I'll call it Dotty. Hi, Dotty!



And Dotty showed up right where we said to, 42 pixels to the right and 23 pixels down.



If you want to have Dotty show up in another location, edit these two lines of code:

```
x = 42
y = 23
```

and run the program again.

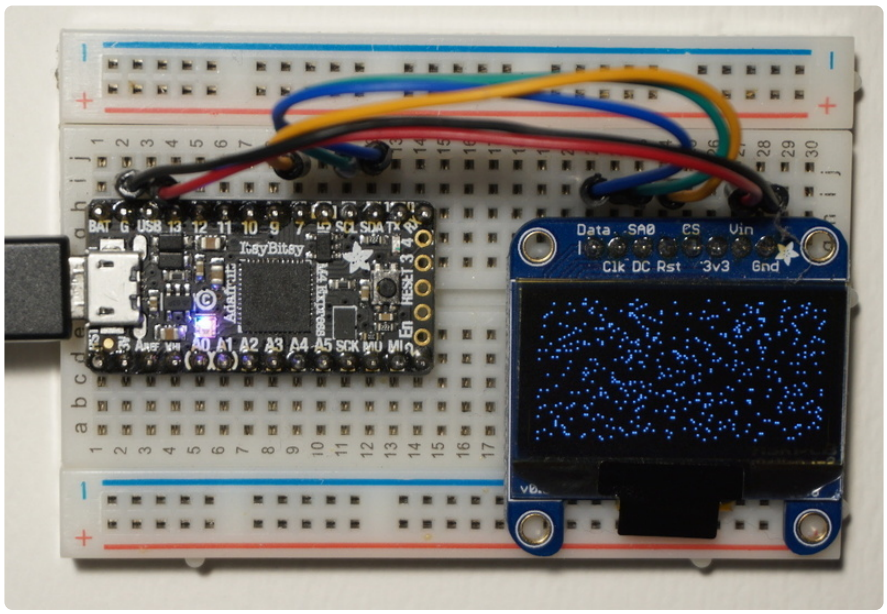
It's Full of Stars

Let's give Dotty the lowly pixels some friends. Lots of friends. To do that, we will draw more pixels. You can call the `display.pixel()` function as many times as you want to draw more than one pixel. Let's have some fun with that. Here's a program that draws 500 pixels in random locations. It does this over and over again, pausing for half a second each time.

```
# SPDX-FileCopyrightText: 2018 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Import the needed libraries
import time
import random
import board
import busio
from digitalio import DigitalInOut
import adafruit_ssd1306

# Create I2c bus
i2c = busio.I2C(board.SCL, board.SDA)

# Define display dimensions and I2C address
WIDTH = 128
HEIGHT = 64
ADDR = 0x3d

# Create the digital out used for display reset
rst = DigitalInOut(board.D7)
```

```
# Create the display
display = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=ADDR, reset=rst)
display.fill(0)
display.show()

# Loop forever drawing random pixels
while True:
    for _ in range(500):
        x = random.randrange(WIDTH)
        y = random.randrange(HEIGHT)
        display.pixel(x, y, 1)
    display.show()
    time.sleep(0.5)
    display.fill(0)
```

Approximately twice a second the display will update with a random pattern of stars. Or is it snow? And where'd Dotty go? Lost in the snow storm I guess. Bye, Dotty!



And that's all there is to draw one or more pixels. Don't forget to call `display.show()` to actually see the results though.

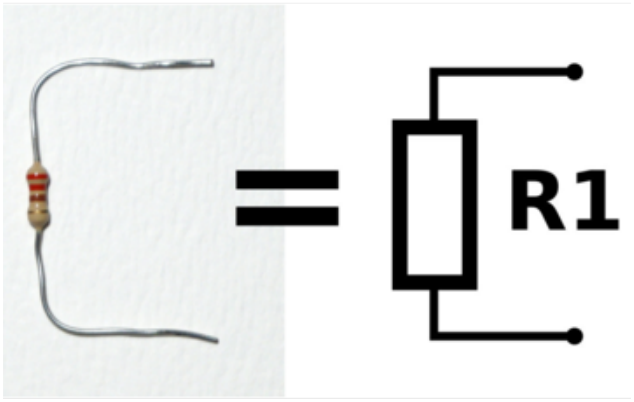
Reading Pots

We want some way to control the location of the pixel using some form of input. For our knob sketcher, we will use a couple of pots. One to control left and right, and one to control up and down.

So what is a pot? A thing you put dirt and plants in? Or the thing you make soup in? Well, yes. But in electronics, it is short for potentiometer. So what is a potentiometer? It's a fancy word for a resistor with a variable value. How do you vary the value of this variable resistor? Easy - with a knob. Just grab it, turn it, change it.

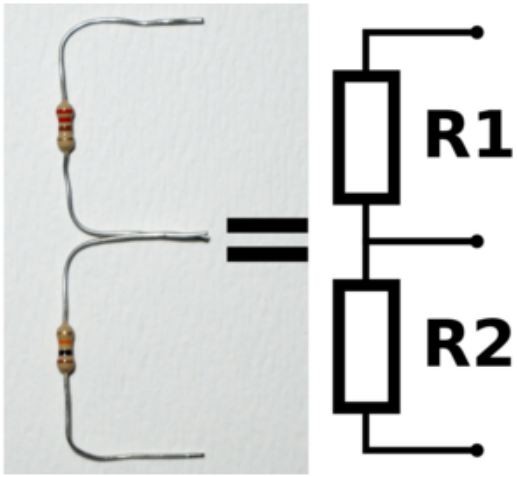
Let's take a closer look.

Pot Internals



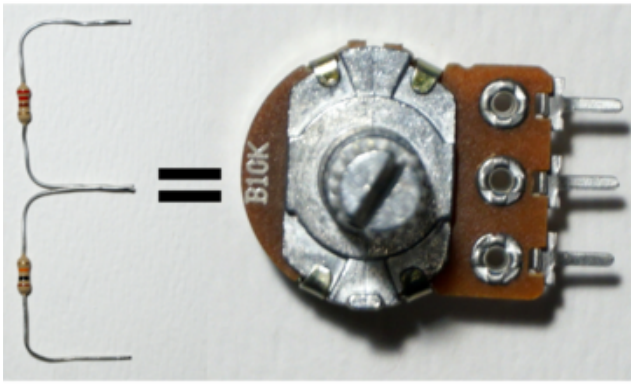
Let's start by looking at a normal resistor with a fixed value. These are super common and look something like the image below. The actual resistor is shown on the left and one way of representing it in drawings is shown on the right.

The color bands can be used to determine what the actual value of the resistors is. But that's not important for what we are doing here. Let's just call it **R1**.

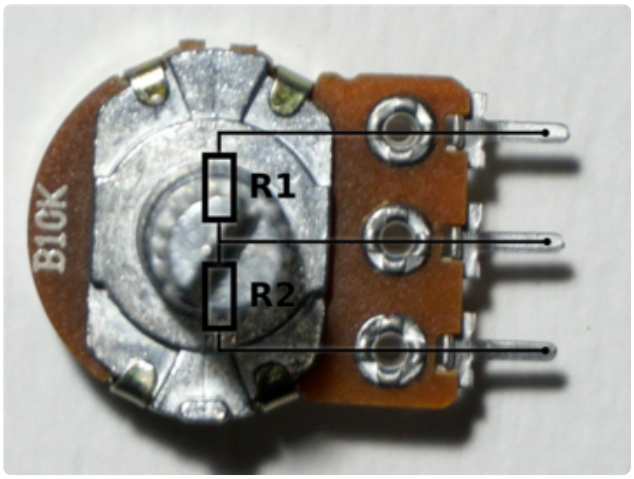


Now consider two resistors put together so they are connected together. Like this!

Now there are three leads as shown in the drawing on the right. Again, we are not concerned about the actual value of the second resistor, so we are just calling it **R2**.



This is basically what the inside of a potentiometer has. So those two resistors become our pot.



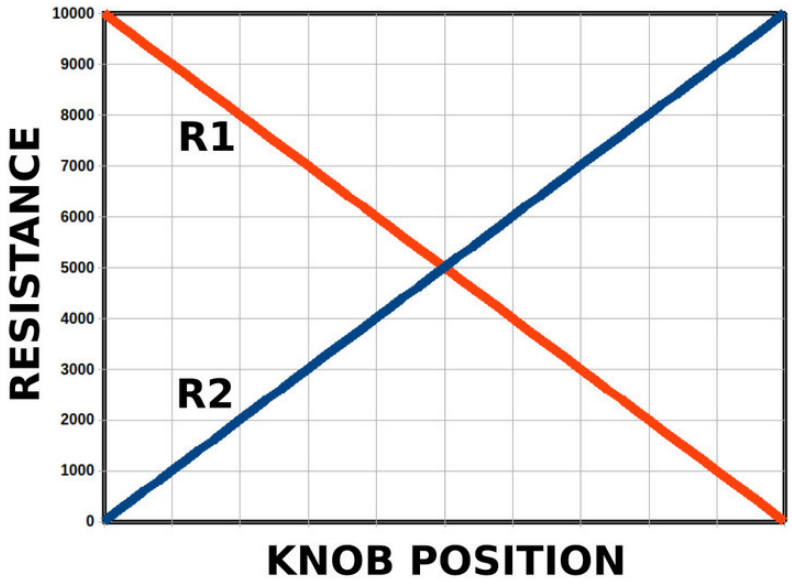
And to better illustrate what the three tabs are doing.

But where is the variability? Are **R1** and **R2** fixed values and that's all there is to it? No. The way a pot works is you can change the values of **R1** and **R2** by turning the knob. They don't change separately. The total resistance obtained by adding **R1** and **R2** together will always be the same - the overall value for the pot.

The pot shown above is a fairly common 10k pot. That means its total resistance is 10,000 ohms. So:

$$R1 + R2 = 10000$$

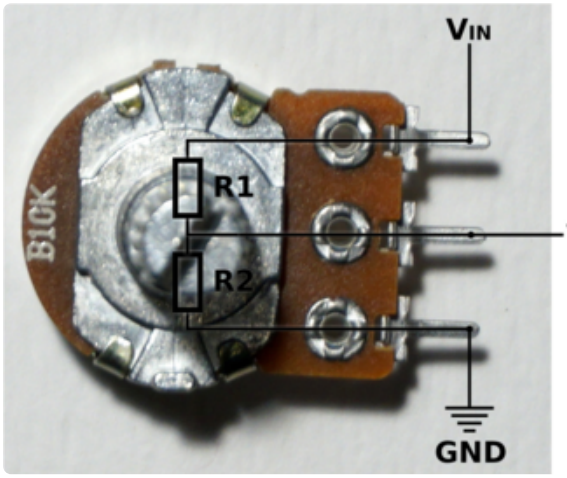
The knob will change the values of **R1** and **R2**, anywhere from 0 up to 10000. But the above relation will always hold. For this style pot (linear), the variation would look something like this:



If you just wanted to use a pot as a single variable resistor, you could. You would just use the middle connector and only one of the outer connectors. Then you would have a resistor that would vary from 0 to 10000 as the knob is turned.

However, we are going to use all three pins in an arrangement known as a voltage divider. So let's look at that in a little more detail.

Variable Voltage Divider



Here is how to wire up the pot to create a voltage divider.

You could use Ohm's Law (https://adafru.it/Dt-) on this to figure out how it works, but we'll just give you the answer. Here it is:

$$V_{OUT} = V_{IN} \bullet \frac{R2}{R2 + R1}$$

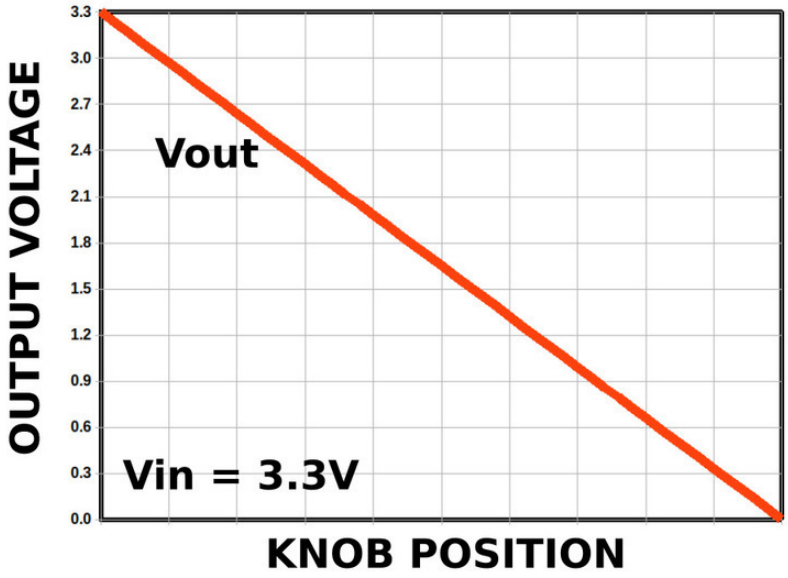
In our case, we will have **Vin** be a constant value (3.3V). As we turn the knob on the pot, we will change **R1** and **R2** as described previously and therefore get a varying **Vout**. Consider the limiting cases of turning the knob all the way in either direction.

All the way in one direction (not the boy band!) will produce **R1** = **0, R2** = **10000**. The fraction then becomes 1 and we get **Vout** = **Vin**.

All the way in the other direction (is that a boy band?) will produce **R1** = **10000, R2** = **0**. The fraction then becomes 0 and we get **Vout** = **0**.

Knob positions in between these two limits will produce different values, and so will "divide" the input voltage down to some value.

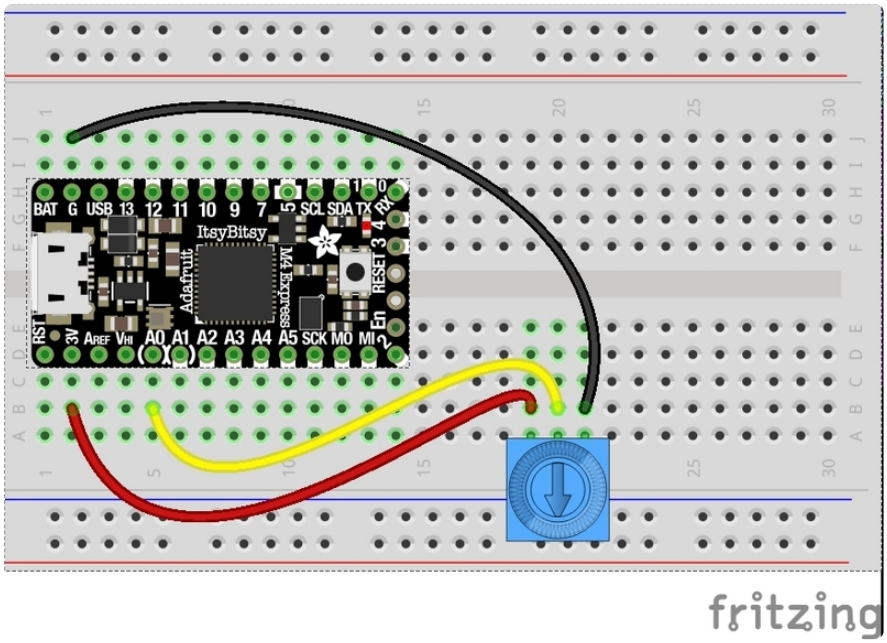
The result looks something like this:



Let's wire this up and see it in action.

Try It Yourself

The variable voltage divider setup is shown below. **Vin** (red wire) is wired to 3.3V (**3V**) and **Vout** (yellow wire) is wired to one of the analog inputs (**A0**) so we can measure the resulting voltage. The other wire (black wire) goes to GND (**G**).

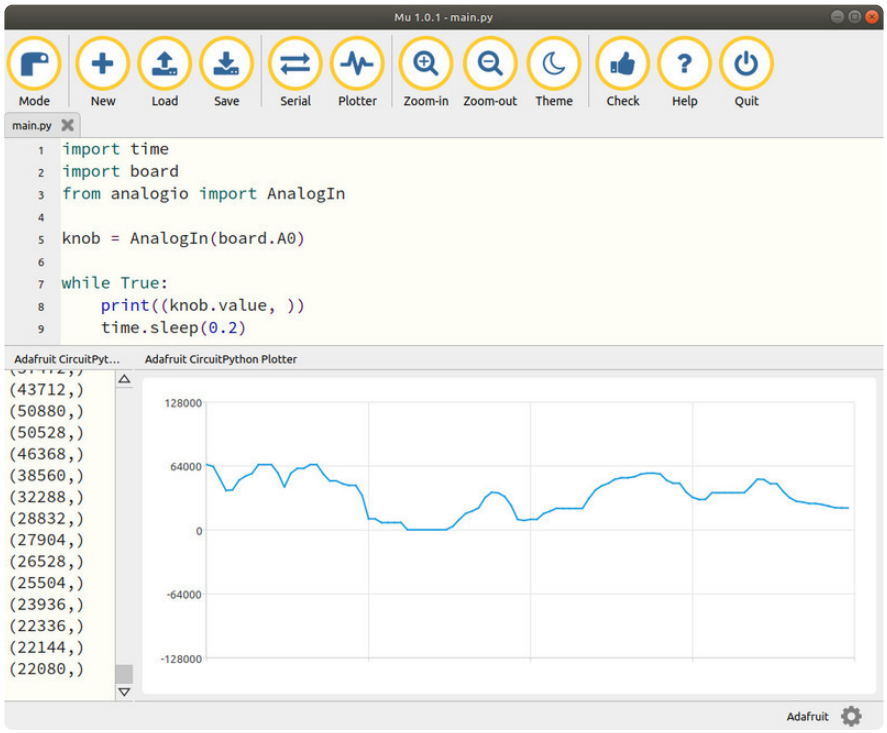


Here is a short CircuitPython program that simply reads the analog input value over and over in a loop and prints it out.

```
import time
import board
from analogio import AnalogIn

knob = AnalogIn(board.A0)
```

```
while True:
    print((knob.value, ))
    time.sleep(0.2)
```

We aren't worried about computing actual volts, so we just look at how the raw reading varies as we turn the knob. If you have Mu installed, this a good use for its builtin plotter (https://adafru.it/Du0).

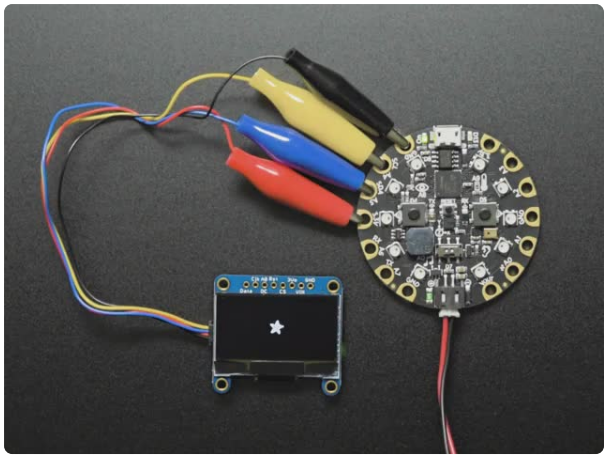


With that code running you can turn the knob and watch the value change. Since we are not reading volts, it will vary in terms of raw values, which will be 0 to 65535 (more info (https://adafru.it/Du1)). You may not get all the way to these values since the pot isn't perfect.

Tiny Sketcher

Let's put these two things together - drawing pixels + reading pots, to create our knob sketcher toy. We'll start with a minimalistic version called the Tiny Sketcher.

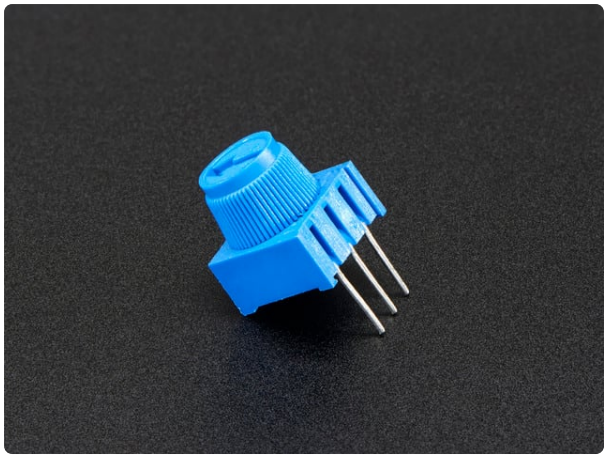
In addition to the ItsyBitsy M4, the Tiny Sketcher uses these parts.



Monochrome 1.3" 128x64 OLED graphic display - STEMMA QT / Qwiic

These displays are small, only about 1.3" diagonal, but very readable due to the high contrast of an OLED display. This display is made of 128x64 individual white OLED pixels,...

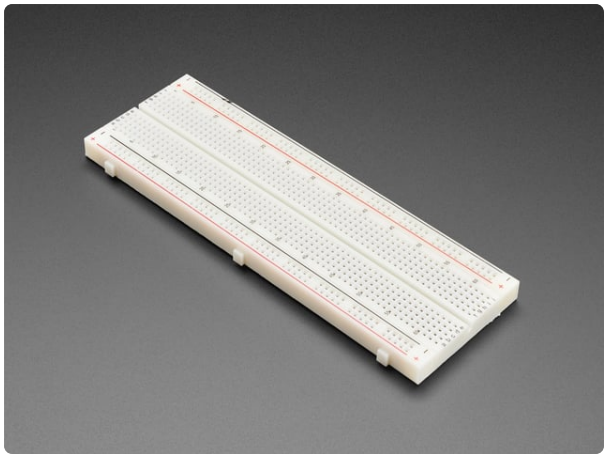
https://www.adafruit.com/product/938



Breadboard trim potentiometer

These are our favorite trim pots, perfect for breadboarding and prototyping. They have a long grippy adjustment knob and with 0.1" spacing, they plug into breadboards or...

https://www.adafruit.com/product/356

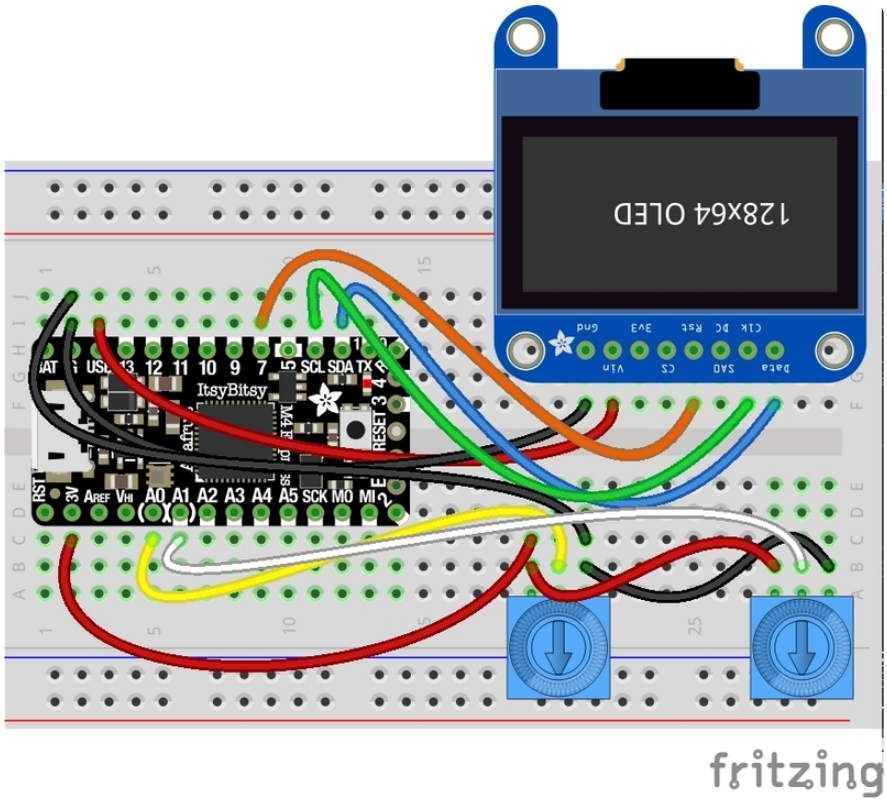


Full Sized Premium Breadboard - 830 Tie Points

This is a 'full-size' premium quality breadboard, 830 tie points. Good for small and medium projects. It's 2.2" x 7" (5.5 cm x 17 cm) with a standard double-strip...

https://www.adafruit.com/product/239

The breadboard setup is shown below. The display has been rotated, but is otherwise wired the same as before. A second pot has been added, but they are both being used in the same way - as variable voltage dividers. The outputs are sent to different analog inputs.



And here is the CircuitPython code to create the sketcher.

```
# SPDX-FileCopyrightText: 2018 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import board
import busio
import adafruit_ssd1306
from simpleio import map_range
from analogio import AnalogIn
from digitalio import DigitalInOut

# Create the I2C bus
i2c = busio.I2C(board.SCL, board.SDA)

# Define display dimensions and I2C address
WIDTH = 128
HEIGHT = 64
ADDR = 0x3d

# Create the digital out used for display reset
rst = DigitalInOut(board.D7)

# Create the display
display = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=ADDR, reset=rst)
display.fill(0)
display.show()

# Create the knobs
x_knob = AnalogIn(board.A0)
y_knob = AnalogIn(board.A1)

while True:
    x = map_range(x_knob.value, 0, 65535, WIDTH - 1, 0)
    y = map_range(y_knob.value, 0, 65535, 0, HEIGHT - 1)
```

```
    display.pixel(int(x), int(y), 1)
    display.show()
```

> Remember, you can save the code as code.py so it will run automatically.

Most of this code is just setup. All the work to create the sketcher is done in the last 5 lines.

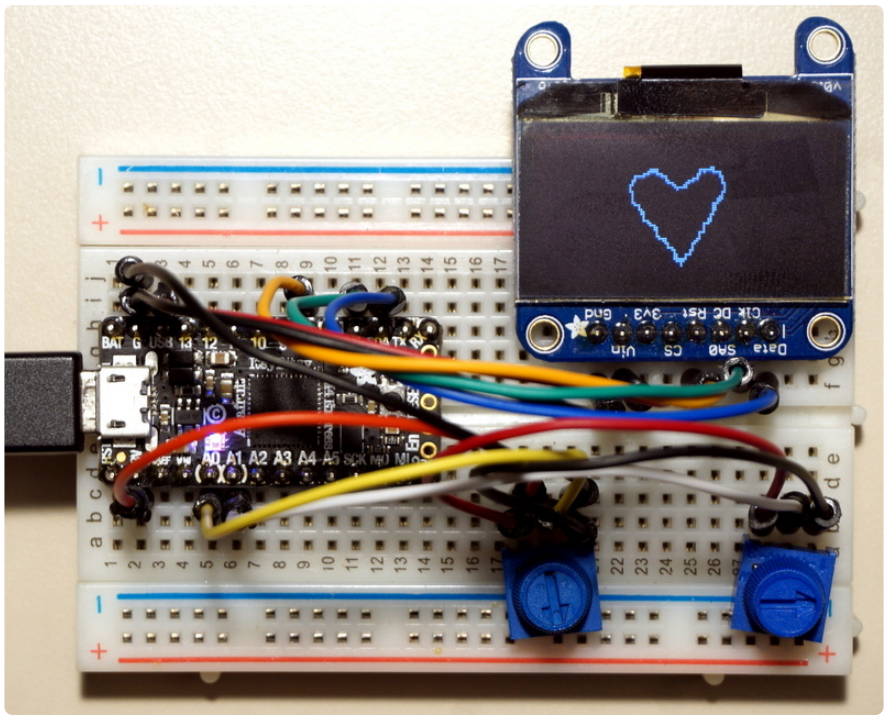
```
while True:
    x = map_range(x_knob.value, 0, 65535, WIDTH - 1, 0)
    y = map_range(y_knob.value, 0, 65535, 0, HEIGHT - 1)
    display.pixel(int(x), int(y), 1)
    display.show()
```

We read each of the knob values using the `value` property. To turn those values into a pixel location, we use the very handy function `map_range`. This function is so handy, it's a wonder why it wasn't built into the Python core. Instead, it is provided by the **simpleio** library - more info here (https://adafru.it/Du2). It basically converts one range of values into another range of values. In this case, from the range of analog read values, 0 to 65535, to the range of pixel locations, 0 to WIDTH or HEIGHT. (the -1 is because of 0 indexing)

Note that the directionality of the knobs can be changed by swapping the order of the last two parameters. If you don't like the direction the pixel moves when you turn the knob, just swap these two and it will reverse it.

Once we have converted the knob values to pixel location, we draw the pixel. The `int()` is needed since `map_range` returns a float and `pixel()` expects integers for the pixel location.

And then we just do that over and over again...forever!



Run the code and turn the knobs. It should be obvious how it works. Sketch away!

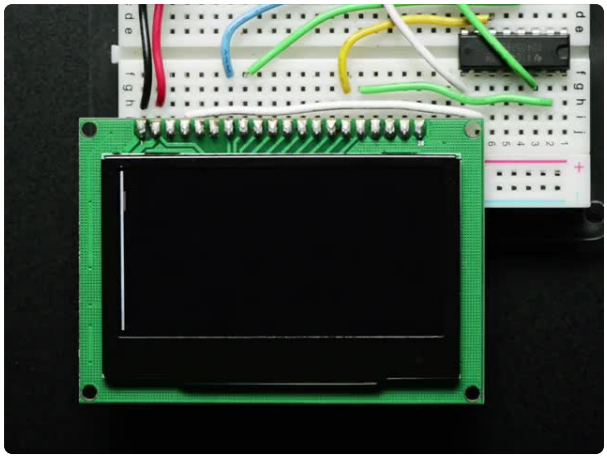
> Turn the knobs slowly to prevent skipping pixels. This version isn't super fast.

Bigger Sketcher

Tiny Sketcher works just fine and demonstrates the basic idea of how the sketcher works. However, that tiny screen and those tiny knobs can be a little tricky to work with. And the only way to start a new sketch is to restart the program.

The Bigger Sketcher improves on Tiny Sketcher by using a larger screen, larger pot knobs, and adds a button that will erase the screen and start over.

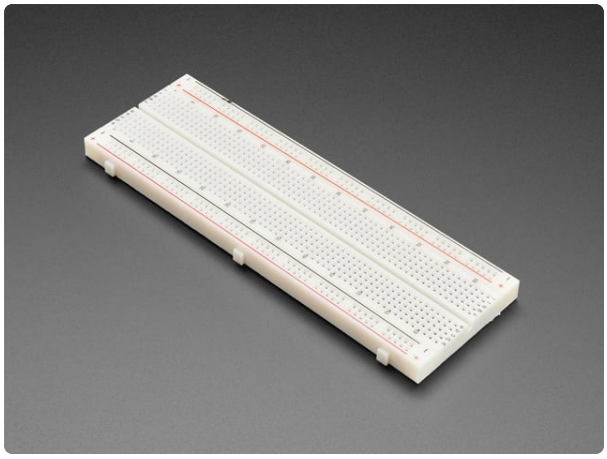
In addition the the Itsy Bitsy M4, the Bigger Sketcher uses these parts.



[Monochrome 2.42" 128x64 OLED Graphic Display Module Kit]
If you've been diggin' our monochrome OLEDs but need something bigger, this display will delight you! These displays...
https://www.adafruit.com/product/2719

[Panel Mount 10K potentiometer (Breadboard Friendly)]
This potentiometer is a two-in-one, good in a breadboard or with a panel. It's a fairly standard linear taper 10K ohm potentiometer, with a grippy shaft. It's smooth and easy...
https://www.adafruit.com/product/562

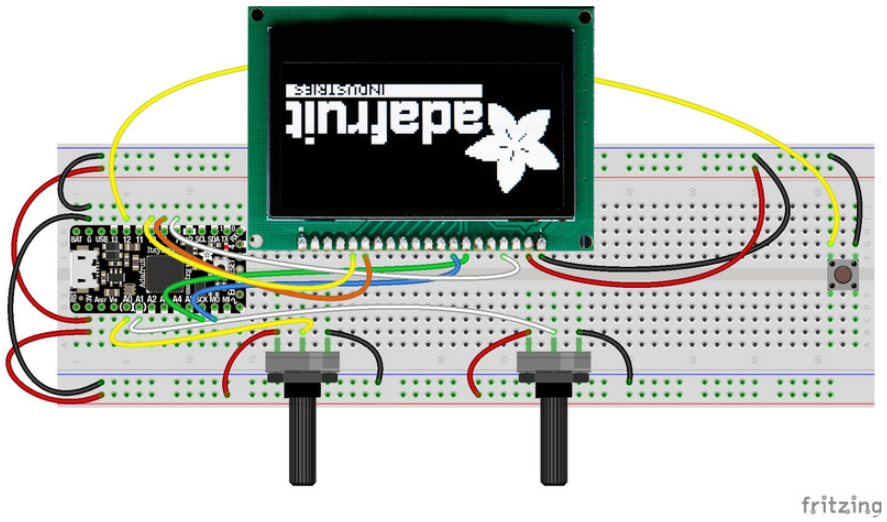
[Colorful Round Tactile Button Switch Assortment - 15 pack]
Little clicky switches are standard input "buttons" on electronic projects. These work best in a PCB but can be...
https://www.adafruit.com/product/1009



[Full Sized Premium Breadboard - 830 Tie Points]
This is a 'full-size' premium quality breadboard, 830 tie points. Good for small and medium projects. It's 2.2" x 7" (5.5 cm x 17 cm) with a standard double-strip...
https://www.adafruit.com/product/239

The connection uses SPI, which is covered in the guide for the OLED, so be sure to read that first (https://adafru.it/Du3).

Read the OLED guide to make sure the display is configured for SPI.



And here's the code for the Bigger Sketcher.

```
# SPDX-FileCopyrightText: 2018 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import board
import busio
import adafruit_ssd1306
from simpleio import map_range
from analogio import AnalogIn
from digitalio import DigitalInOut, Direction, Pull

# Create SPI bus
spi = busio.SPI(board.SCK, board.MOSI)

# Create the display
WIDTH = 128
HEIGHT = 64
DC = DigitalInOut(board.D7)
CS = DigitalInOut(board.D9)
RST = DigitalInOut(board.D10)
display = adafruit_ssd1306.SSD1306_SPI(WIDTH, HEIGHT, spi, DC, RST, CS)
display.fill(0)
display.show()

# Create the knobs
READS = 5
x_knob = AnalogIn(board.A0)
y_knob = AnalogIn(board.A1)

# Create the clear button
clear_button = DigitalInOut(board.D12)
```

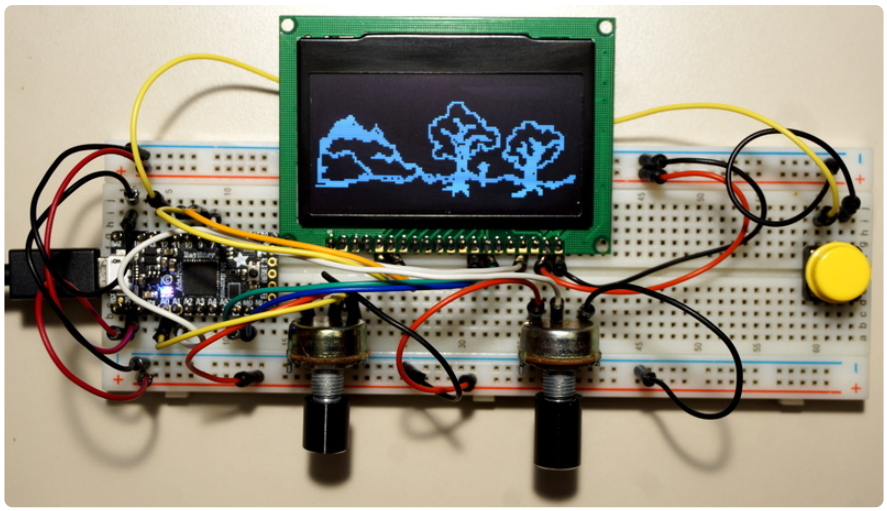
```
clear_button.direction = Direction.INPUT
clear_button.pull = Pull.UP

def read_knobs(reads):
    avg_x = avg_y = 0
    for _ in range(reads):
        avg_x += x_knob.value
        avg_y += y_knob.value
    avg_x /= reads
    avg_y /= reads
    x_screen = map_range(avg_x, 0, 65535, 0, WIDTH - 1)
    y_screen = map_range(avg_y, 0, 65535, 0, HEIGHT - 1)
    return int(x_screen), int(y_screen)

while True:
    while clear_button.value:
        x, y = read_knobs(READS)
        display.pixel(x, y, 1)
        display.show()
    display.fill(0)
    display.show()
```

As before, this code is largely setup. One big difference from Tiny Sketcher is how the knob values are determined. In order to smooth out possible noise in the analog readings, several readings are made and averaged together. This is all taken care of in the new function `read_knobs()`.

The other change is the addition of the button used to clear the display. This is handled with a simple modification of the previous loop.

```
while True:
    while clear_button.value:
        x, y = read_knobs(READS)
        display.pixel(x, y, 1)
        display.show()
    display.fill(0)
    display.show()
```

There is still the outer loop that runs forever - `while True:`. But now the pixel drawing is done in a new nested loop - `while clear_button.value:` . The way the button is wired, it will read `True` when it is **not** pressed. So this loop will keep running as long as the button is left alone. This is when you are drawing. Anytime you want to clear the current display, just hit the button. `clear_button.value` will return `False`, and the drawing loop will exit. The display is cleared by the last two lines, but then it starts all over again and you are back to drawing.



Be careful not to accidentally hit the clear button. There's no UNDO.

Going Further

The two example knob sketcher toys in this guide work well enough to demonstrate the basic idea and have some fun. However, there are lots of other "features" that cold be added. Here are some ideas in no particular order.

- A pen up / pen down capability
- Add an eraser capability
- `COLOR` display and control
- Save / load display to file
- Different brushes
- Connect skipped pixels with a line
- Accelerometer to erase on shake (for old skool feel)

Try adding one or more of these features using the provided code as a starting point. Or add some other cool feature. Have fun!

We don't make mistakes, just happy little accidents.