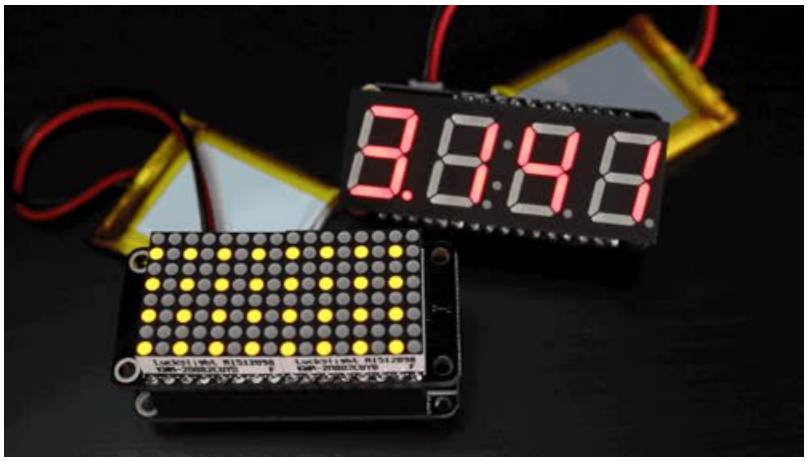
adafruit learning system

CircuitPython Hardware: LED Backpacks & FeatherWings

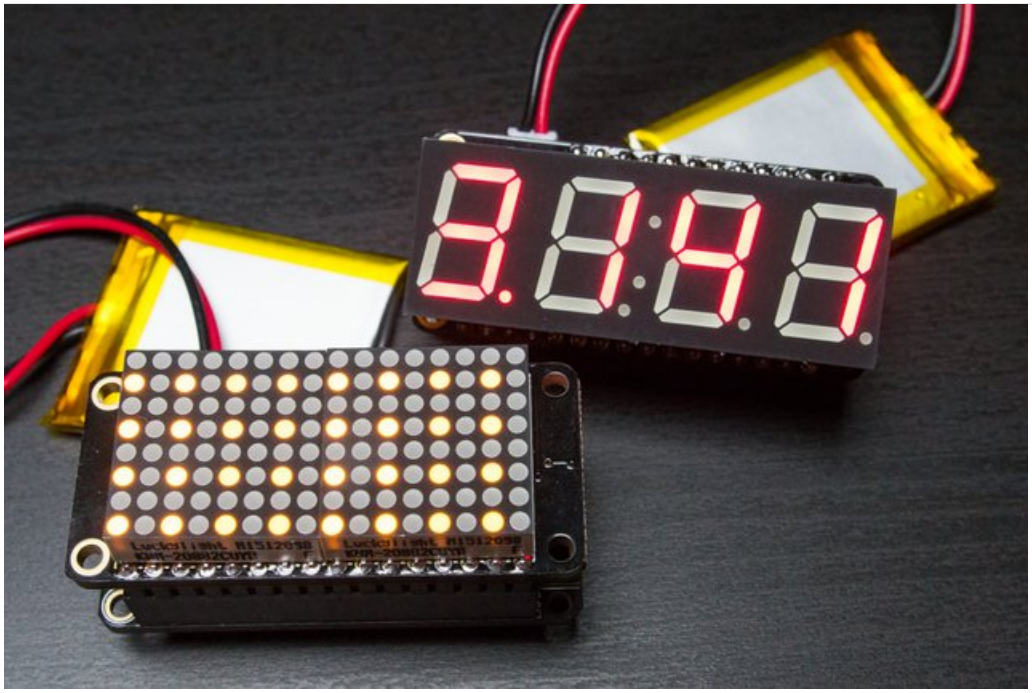
Created by Tony DiCola



Last updated on 2018-08-30 06:13:00 PM UTC

Guide Contents

Overview



Note the video above was made showing the MicroPython version of this library. Follow the guide to see both CircuitPython and MicroPython versions of the LED backpack library.

LED backpacks (https://adafru.it/dEM) are bright and beautiful LED displays with a built-in controller chip that makes them super easy to use. These displays come in all sorts of varieties, like matrices that are a grid of LED pixels or segment displays that can show numbers and characters. Because these displays use LEDs they're bright and easy to read--perfect for showing simple measurements, time, basic graphics, and more.

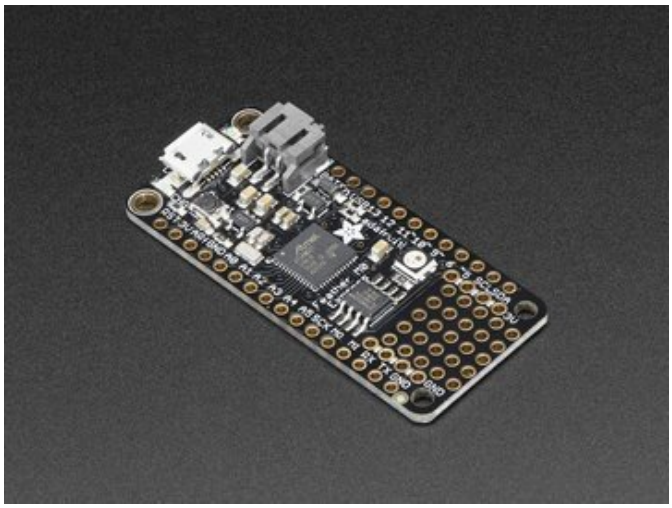
And now you can use LED backpack displays with Python. This guide explores how to use LED matrix, 7-segment, and 14-segment quad alphanumeric displays with CircuitPython and MicroPython!

Be sure to check out the Adafruit LED Backpack guide (https://adafru.it/dEM) for details on all of the LED backpack modules.

Hardware

Parts

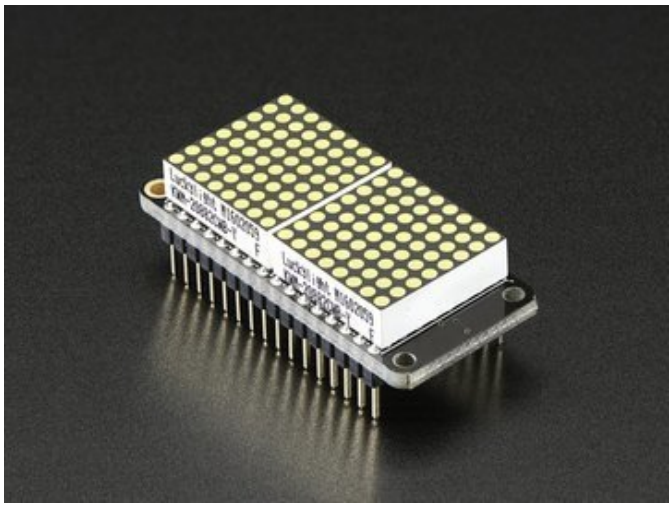
You'll need the following parts to follow this guide:



CircuitPython board. This guide focuses on the ESP8266 (https://adafru.it/n6A) and Feather M0/SAMD21-based boards (http://adafru.it/2772), but any CircuitPython board that supports I2C should work.

If your board doesn't come with CircuitPython running on it already then check out your board's guide for how to load CircuitPython firmware. For example the Feather M0 express guide (https://adafru.it/wbv) is a good reference.

If you're using a Feather board and FeatherWing you probably want a Feather female header set (http://adafru.it/2886) or Feather stacking female header set (http://adafru.it/2830).



LED Matrix, 7-segment, or 14-segment FeatherWing or Backpack Display. If you're using a Feather the 8x16 Matrix FeatherWing (http://adafru.it/3155), 14-segment Quad Alphanumeric FeatherWing (http://adafru.it/3139), or 7-segment FeatherWing (http://adafru.it/3140) are perfect options that easily connect to the Feather board. For other boards check out the large assortment of LED backpacks (https://adafru.it/dEM)--the matrix (both single and bi-color), 14-segment quad alphanumeric, and 7-segment numeric backpacks can be used with the CircuitPython library.

Make sure to check if your backpack requires 5V or 3.3V of power. Larger displays like the 1.2" 7-segment display need 5V of power, but boards like Feathers only provide 5V power when plugged in to USB ports. Read the LED backpack guide (https://adafru.it/dEM) for details on the power needed for each type of backpack!

Breadboard (http://adafru.it/64)and **jumper wires (http://adafru.it/153)**. If you aren't using a Feather and FeatherWing you'll need a breadboard and jumper wires to connect the components.

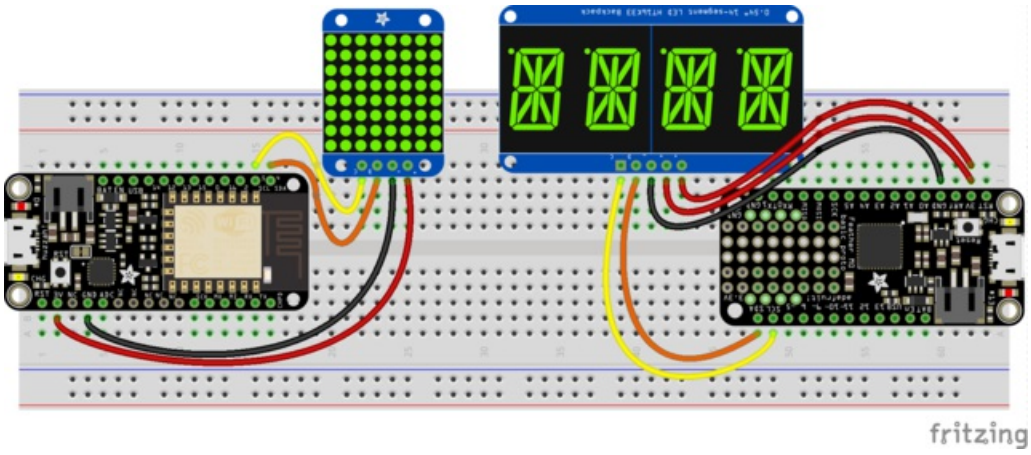
Soldering tools (http://adafru.it/136). You'll need to solder headers to the boards Check out the guide to excellent soldering (https://adafru.it/dxy) if you're new to soldering.

Make sure to follow the board and LED FeatherWing or backpack product guides to assemble and verify they work before continuing.

Wiring

If you're using a FeatherWing and Feather just slide the wing onto the Feather board and you're all set! The FeatherWing will automatically be connected to the board using its I2C connection. Skip to the next page to learn about the software to control the display.

If you're using a LED backpack breakout you'll need to connect its power, ground, and I2C connections to the board. For example the wiring for a LED matrix to Feather HUZZAH ESP8266, and 14-segment display to Feather M0 might look like:



https://adafru.it/zdA

https://adafru.it/zdA

- **Board SCL / I2C clock** to **LED Backpack SCL / C**.
- **Board SDA / I2C data** to **LED Backpack SDA / D**.
- **Board 3.3V power** to **LED Backpack VCC** / + (note some larger backpacks prefer 5V power but might still at 3.3V with less light output, check your product's guide for details).
- **Board 3.3V power** to **LED Backpack VI2C / logic level voltage** (not all backpacks have a VI2C or logic level pin, check your product's guide for details).
- **Board GND / ground** to **LED Backpack GND / -**.

Remember to read about your backpack's power requirements in the LED backpack guide (https://adafru.it/dEM). Some of the large backpack displays (like the 1.2" 7-segment display) need 5V of power to run, and others require an explicit VI2C or logic level voltage input.

CircuitPython

Adafruit CircuitPython Module Install

To use the LED backpack with your Adafruit CircuitPython (https://adafru.it/BIM) board you'll need to install the Adafruit_CircuitPython_HT16K33 (https://adafru.it/u1E) module on your board. **Remember this module is for Adafruit CircuitPython firmware and not MicroPython.org firmware!**

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board. Next you'll need to install the necessary libraries to use the hardware--read below and carefully follow the referenced steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx).

Bundle Install

For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with Adafruit's CircuitPython bundle (https://adafru.it/zdx). This is an all-in-one package that includes the necessary libraries to use the LED backpack display with CircuitPython. For details on installing the bundle, read about CircuitPython Libraries (https://adafru.it/ABU).

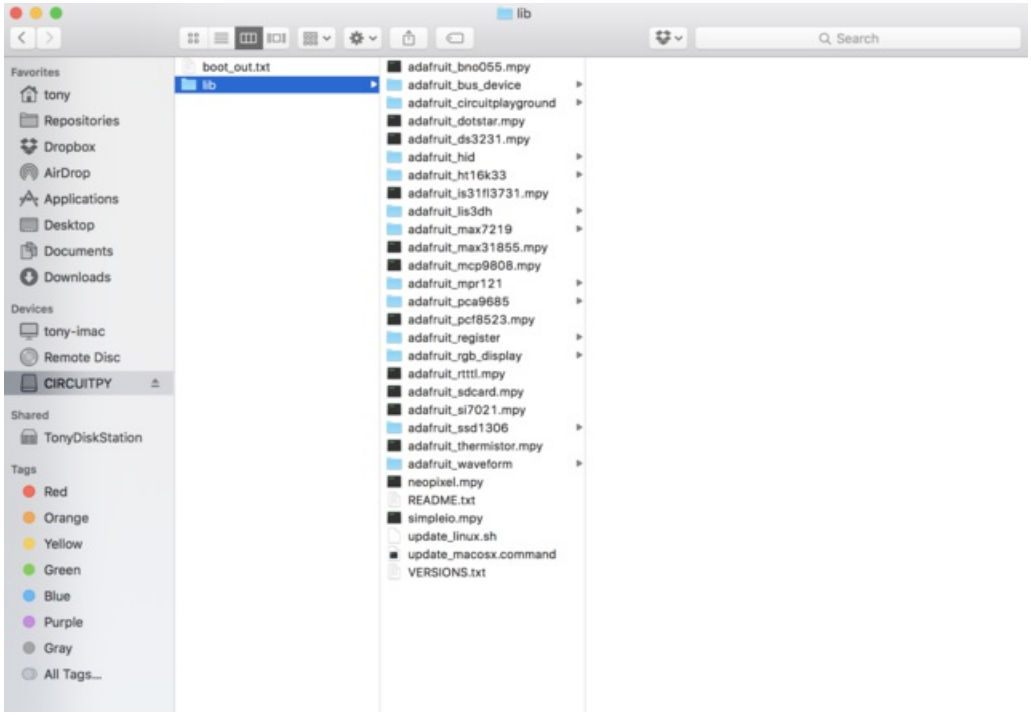
Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries (https://adafru.it/ABU) from the bundle:

- **adafruit_ht16k33**
- **adafruit_bus_device**
- **adafruit_register**

If your board supports USB mass storage, like the M0-based boards, then simply drag the files to the board's file system. **Note on boards without external SPI flash, like a Feather M0 or Trinket/Gemma M0, you might run into issues on Mac OSX with hidden files taking up too much space when drag and drop copying, see this page for a workaround (https://adafru.it/u1d).**

If your board doesn't support USB mass storage, like the ESP8266, then use a tool like ampy to copy the file to the board (https://adafru.it/s1f). You can use the latest version of ampy and its new directory copy command (https://adafru.it/q2A) to easily move module directories to the board.

Before continuing make sure your board's **lib** folder or root filesystem has at least the **adafruit_ht16k33**, **adafruit_bus_device**, and **adafruit_register** folders/modules copied over.



Usage

The following section will show how to control the LED backpack from the board's Python prompt / REPL. You'll walk through how to control the LED display and learn how to use the CircuitPython module built for the display.

First connect to the board's serial REPL (https://adafru.it/Awz)so you are at the CircuitPython >>> prompt.

I2C Initialization

First you'll need to initialize the I2C bus for your board. First import the necessary modules:

```
import board
import busio as io
```

Note if you're using the ESP8266 or other boards which do not support hardware I2C you need to import from the bitbangio module instead of busio:

```
import board
import bitbangio as io
```

Now for either board run this command to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board):

```
i2c = io.I2C(board.SCL, board.SDA)
```

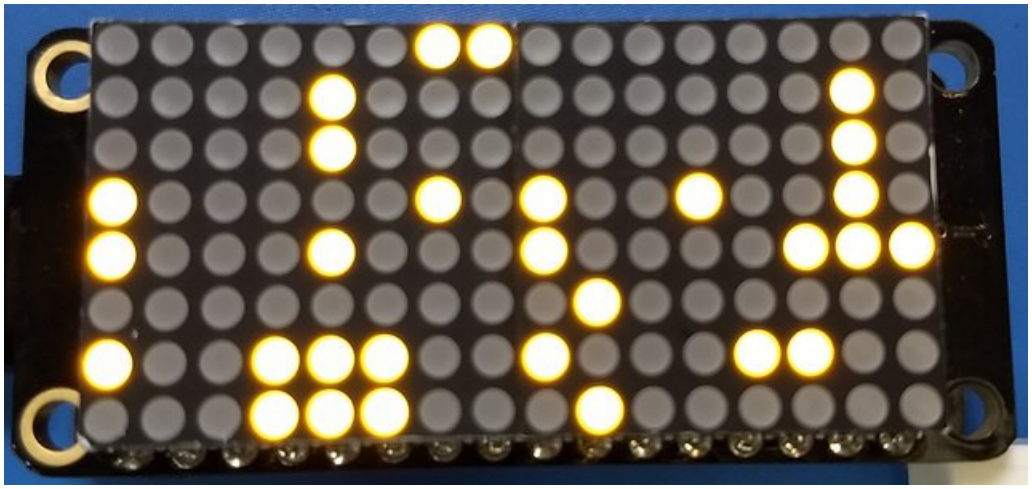
LED Matrix

To use a LED matrix you'll first need to import the **adafruit_ht16k33.matrix** module and create an instance of the appropriate Matrix class. There are three classes currently available to use:

- **Matrix8x8** - This is for a simple 8x8 matrix (square or round pixels, they're both the same driver and code).
- **Matrix16x8** - This is for a 16x8 matrix (i.e. double the width of the 8x8 matrices). For the LED Matrix FeatherWing you want to use this Matrix16x8 class.
- **Matrix8x8x2** - This is for a 8x8 bi-color matrix.

For example to use the Matrix16x8 class import the module and create an instance with:

```
import adafruit_ht16k33.matrix
matrix = adafruit_ht16k33.matrix.Matrix16x8(i2c)
```



The above command will create the matrix class using the default I2C address of the display (0x70). If you've changed the I2C address (like when using multiple backpacks or displays) you can override it in the initializer using an optional **address** keyword argument.

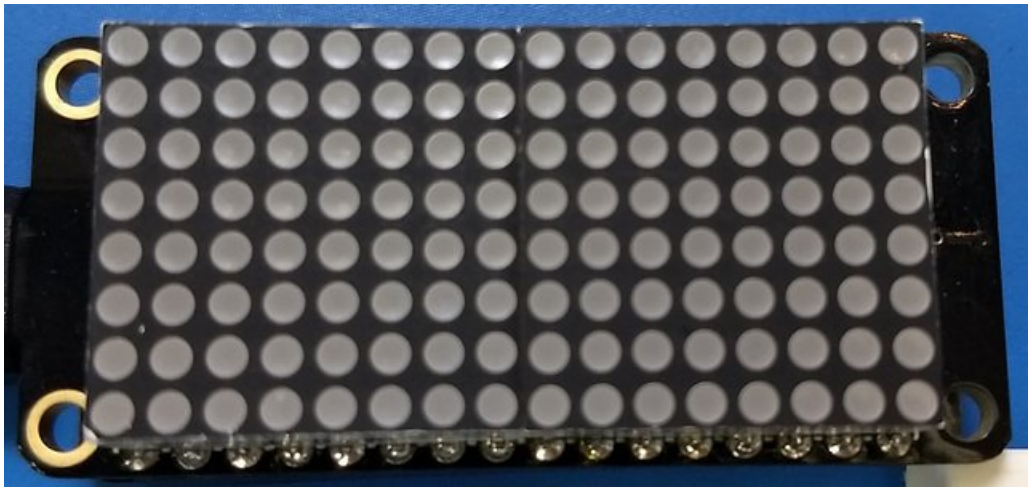
For example to create an instance of the Matrix8x8 class on address 0x74:

```
matrix = adafruit_ht16k33.matrix.Matrix8x8(i2c, address=0x74)
```

You might notice the matrix turns on to a 'jumbled' collection of random LEDs. Don't worry! The display isn't broken, right now the module that controls it doesn't clear the display state on startup so you might see noise from random memory values.

To clear the display and turn all the pixels off you can use the **fill** command with a color of 0 (off):

```
matrix.fill(0)
```

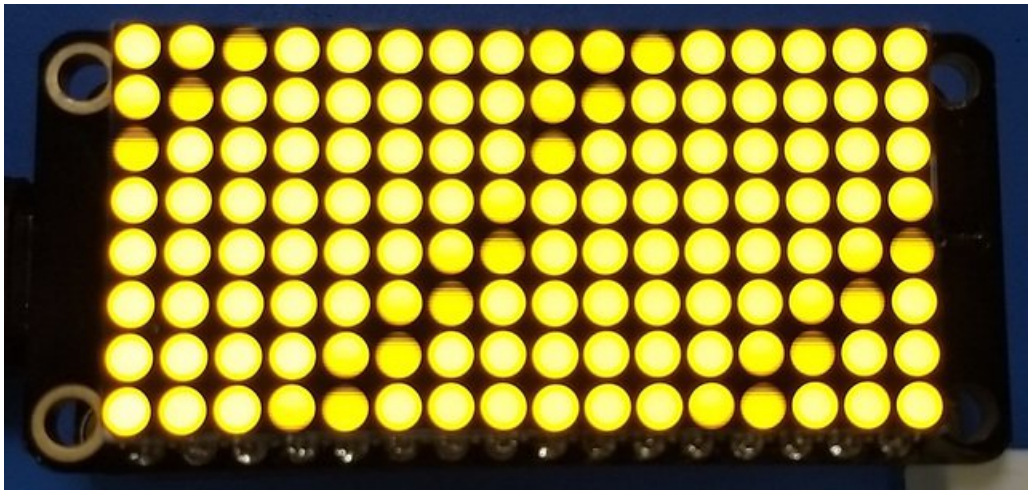


By default the display will update automatically. This way you do not need to call the **show** function every time you update the display buffer. However, this means it is being sent the contents of the display buffer with every change. This can slow things down if you're trying to do something fancy. If you think you're running into that issue, you can simply turn the auto write feature off. Then you will need to call **show** to update the display.

```
# auto write can be turned off
matrix.auto_write = False
# and fill is same as before
matrix.fill(0)
# but now you have to call show()
matrix.show()
```

To turn all the pixels on you can use **fill** with a color of 1 (on):

```
matrix.fill(1)
```



If you're using the bi-color matrix you can even use a fill color of 2 or 3 to change to different colors of red, green, and yellow (red + green).

Next you can set pixels on the display by accessing them using x,y coordinates and setting a color:

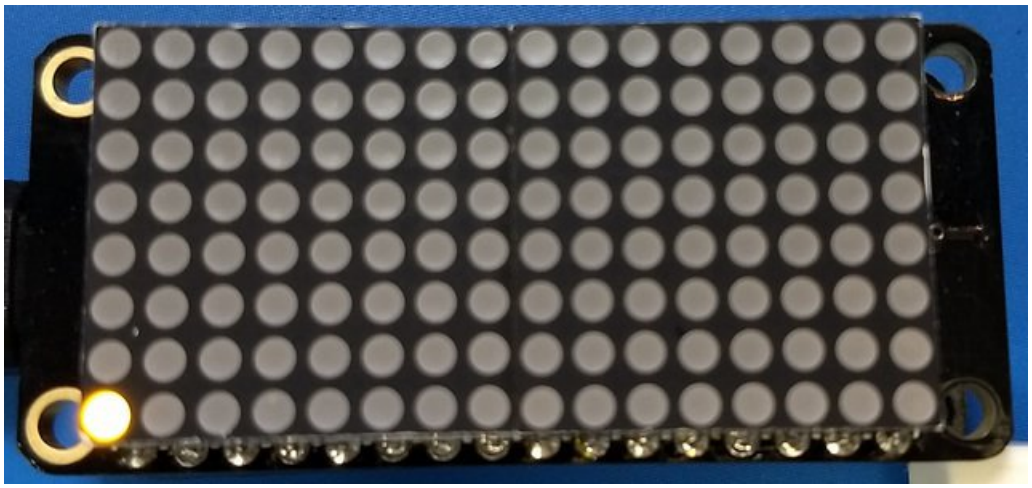
- **X position** - X position of the pixel on the matrix.
- **Y position** - Y position of the pixel on the matrix.
- **Color** - 0 for off, 1 for on (or one of the LEDs for bi-color display), 2 or 3 for other bi-color display colors.

The general way to set a pixel is:

```
matrix[x,y] = color
```

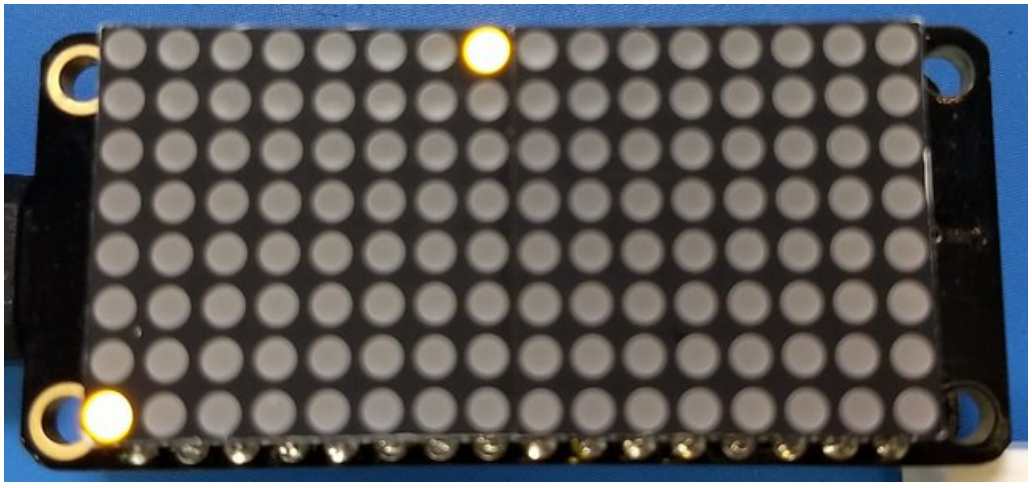
For example to set the first pixel at position 0, 0 to on:

```
matrix[0, 0] = 1
```



Or to set the opposite corner pixel at position 7, 7 to on:

```
matrix[7, 7] = 1
```



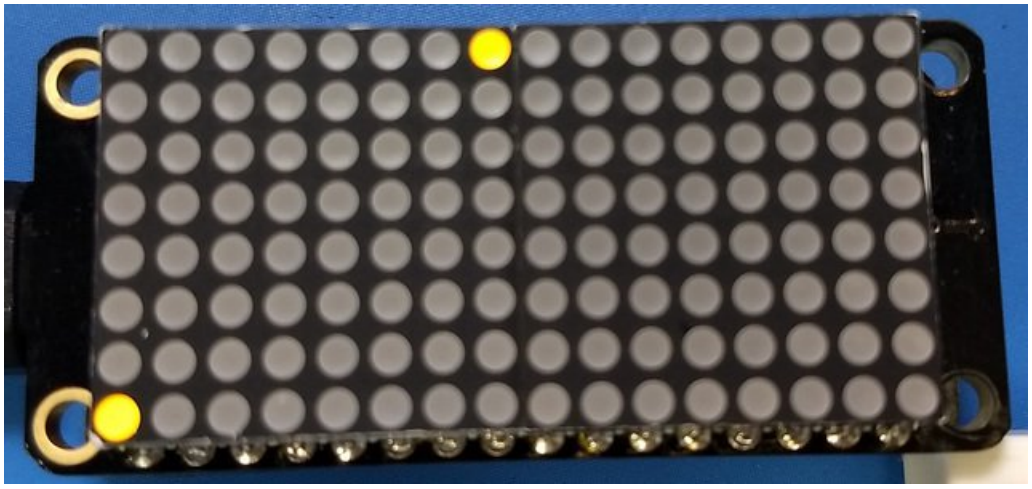
That's all there is to controlling the pixels on a LED matrix! Right now the matrix library is simple and only exposes basic pixel control. In the future more advanced drawing commands might be avaialable.

Brightness and Blinking

You can change the brightness of the entire display with the **brightness** property. This property has a value from 0 to 15 where 0 is the lowest brightness and 15 is the highest brightness. Note that you don't need to call **show** after calling **brightness**, the change is instant.

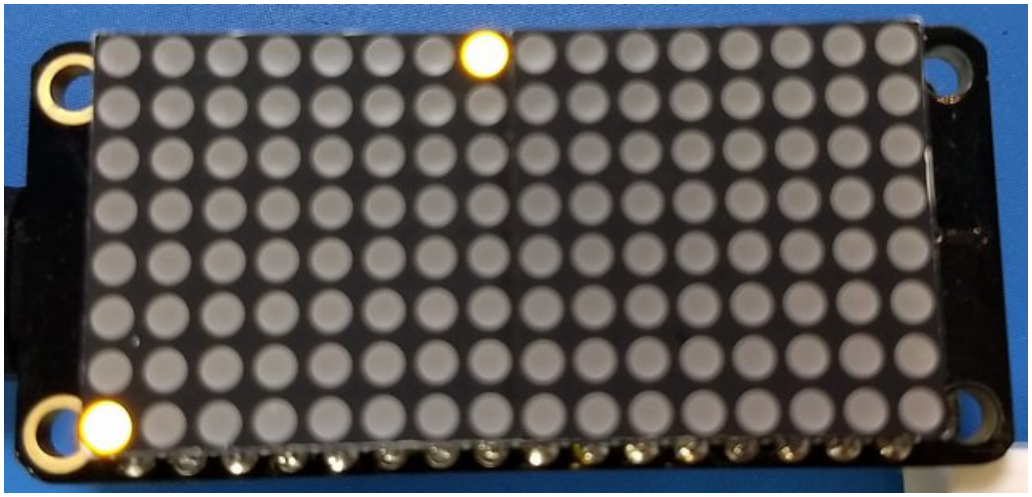
For example to set the brightness to the minimum:

```
matrix.brightness = 0
```



Or to set it back to maximum:

```
matrix.brightness = 15
```



You can also make the entire display blink at 3 different rates using the **blink_rate** property, which has a value 0 to 3:

- **0** = no blinking
- **1** = fast blinking (~once a 1/2 second)
- **2** = moderate blinking (~once a second)
- **3** = slow blinking (~once every 2 seconds)

Again you don't need to call **show** after setting the blink rate, the change will immediately take effect.

For example to blink quickly:

```
matrix.blink_rate = 1
```

And to stop blinking:

```
matrix.blink_rate = 0
```

LED 7-segment Display

To use a 7-segment display you'll first need to import the **adafruit_ht16k33.segments** module and create an instance of the **Seg7x4** class.

```
import adafruit_ht16k33.segments
display = adafruit_ht16k33.segments.Seg7x4(i2c)
```

The above command will create the 7-segment class using the default I2C address of the display (0x70). If you've changed the I2C address (like when using multiple backpacks or displays) you can override it in the initializer using an optional **address** keyword argument.

For example to create an instance of the **Seg7x4** class on address 0x74:

```
display = adafruit_ht16k33.Seg7x4(i2c, address=0x74)
```

You might notice the display turns on to a 'jumbled' collection of random LEDs. Don't worry! The display isn't broken, right now the module that controls it doesn't clear the display state on startup so you might see noise from random memory values.

To clear the display and turn all the LEDs off you can use the **fill** command with a color of 0 (off):

```
display.fill(0)
```

Setting Individual Digits

You can put a numeric value in any of the display's 4 positions by accessing it using the index of the position. For example to set position 0 to the number 1 and position 1 to the number 2 call:

```
display[0] = '1'
display[1] = '2'
display.show()
```

Display Numbers and Hex Values

You can also use the **print** function to write to the entire display. Remember the display only has 4 digits so a best effort will be made to display the number--you might need to round the number or adjust it to fit!

```
display.print(1234)
display.show()
display.print(3.141)
display.show()
```

To display hex values, pass in a string to **print**. The hex characters A-F can be displayed.

```
display.print('FEED')
display.show()
```

If you want to work with actual integer values, then use the built in string formatting.

```
display.print('{:x}'.format(65261))
display.show()
```

You can pass some special characters to the display to control extra capabilities:

- **Colon** - Use ':' to turn the colon on, you don't need to specify the position parameter. Use ';' to turn the colon off.
- **Hex character** - Use a character 'a' through 'f' to display a high hex character value at a specified position.

LED 14-segment Quad Alphanumeric Display

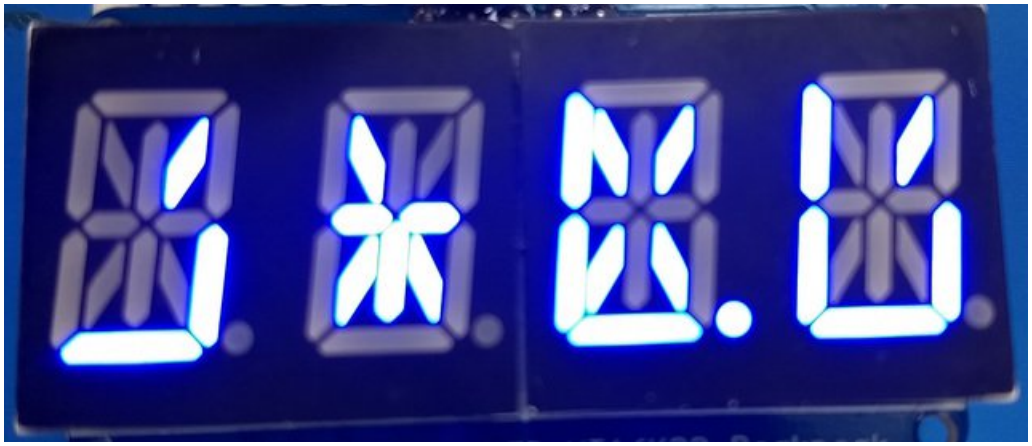
To use a 14-segment quad alphanumeric display it's almost exactly the same as the 7-segment display, but with a slightly different class name. Import the **adafruit_ht16k33.segments** module again but this time create an instance of the **Seg14x4** class.

```
import adafruit_ht16k33.segments
display = adafruit_ht16k33.segments.Seg14x4(i2c)
```

The above command will create the 14-segment class using the default I2C address of the display (0x70). If you've changed the I2C address (like when using multiple backpacks or displays) you can override it in the initializer using an optional **address** keyword argument.

For example to create an instance of the **Seg14x4** class on address 0x74:

```
display = adafruit_ht16k33.segments.Seg14x4(i2c, address=0x74)
```



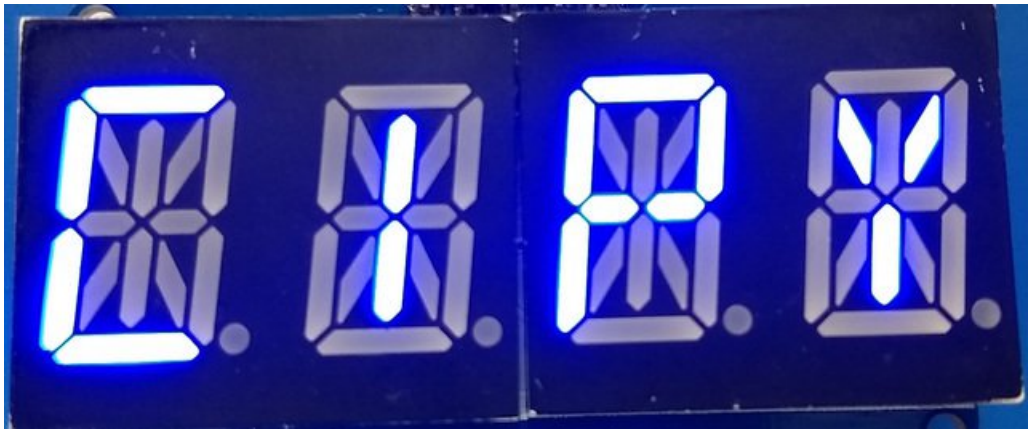
You might notice the display turns on to a 'jumbled' collection of random LEDs. Don't worry! The display isn't broken, right now the module that controls it doesn't clear the display state on startup so you might see noise from random memory values.

To clear the display and turn all the LEDs off you can use the **fill** command with a color of 0 (off):

```
display.fill(0)
display.show()
```

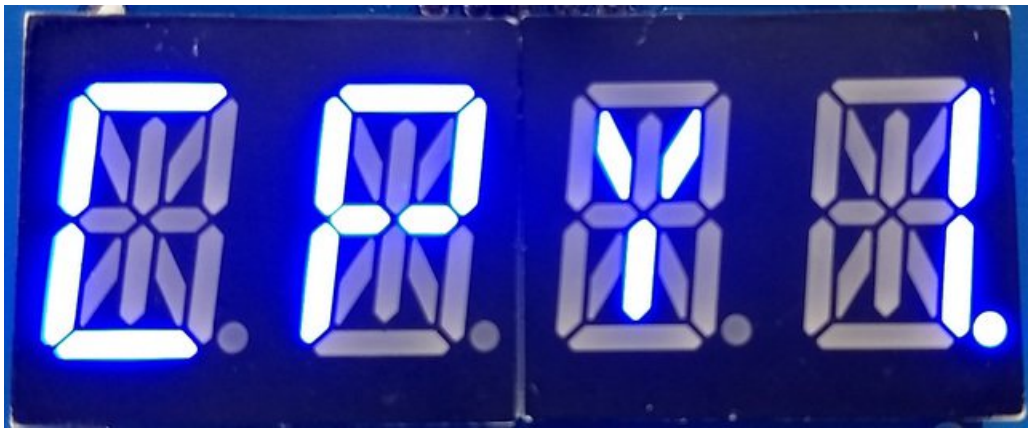
To access the individual digits, it's the same as with the 7-segment display. However, the 14-segment display can take any alphanumeric character. For example:

```
display[0] = 'C'
display[1] = 'I'
display[2] = 'P'
display[3] = 'Y'
display.show()
```



You can use the **print** function to write to the entire display.

```
display.print('CPY!')
display.show()
```



Python Docs

Python Docs (https://adafru.it/C4N)

MicroPython

> Note this page describes how to use a MicroPython.org version of this library with MicroPython boards. Skip back to the previous page if you're using a CircuitPython board like the Feather M0 express!

In addition to CircuitPython there's an older MicroPython version of the HT16K33 library that you can use with some MicroPython boards. Before you get started it will help to be familiar with these guides for working with MicroPython:

- MicroPython Basics: What is MicroPython? (https://adafru.it/pXa)
- MicroPython Basics: How to Load MicroPython on a Board (https://adafru.it/pNB)
- MicroPython Basics: Load Files & Run Code (https://adafru.it/s1f)

See all the MicroPython guides in the learning system (https://adafru.it/qzD) for more information.

MicroPython Module Install

To use the LED backpack with your MicroPython board you'll need to install the micropython-adafruit-ht16k33 MicroPython module (https://adafru.it/scu) on your board. **Remember this module is for MicroPython.org firmware and not Adafruit CircuitPython!**

First make sure you are running the latest version of MicroPython for your board. If you're using the **ESP8266 MicroPython** port you **must** be running version **1.8.5 or higher (https://adafru.it/sas)** as earlier versions do not support using .mpy modules as shown in this guide.

Next download the latest **ht16k33_matrix.mpy** and **ht16k33_seg.mpy** files from the releases page (https://adafru.it/scv) of the micropython-adafruit-ht16k33 GitHub repository (https://adafru.it/scu). You'll need to copy **all** of the files to your MicroPython board's file system using a tool like ampy to copy the files to the board (https://adafru.it/r2B).

Usage

The following section will show how to control the LED backpack from the board's Python prompt / REPL. You'll walk through how to control the LED display and learn how to use the MicroPython module built for the display. As a reference be sure to see the micropython-adafruit-ht16k33 module documentation (https://adafru.it/scw) too.

First connect to the board's serial REPL (https://adafru.it/pMf)so you are at the MicroPython >>> prompt.

MicroPython I2C Initialization

On MicroPython.org firmware which uses the machine API you can initialize I2C like the MicroPython I2C guide mentions (https://adafru.it/sau). For example on a board like the ESP8266 you can run (assuming you're using the default SDA gpio #4 and SCL gpio #5 pins like on a Feather & LED backpack FeatherWing):

```
import machine
i2c = machine.I2C(scl=machine.Pin(5), sda=machine.Pin(4))
```

Matrix & Segment Display Usage

Once you've initialized the I2C bus you're ready to create instances of the matrix and display classes. These classes are exactly the same as the CircuitPython version of the library, but instead live in the ht16k33_matrix module. For

example to create a 8x8 matrix you could run:

```
import ht16k33_matrix
matrix = ht16k33_matrix.Matrix16x8(i2c)
```

See the CircuitPython library usage (https://adafru.it/zdB) for information on the functions you can call to control the matrix and segment displays. Once you've created the display class like above its usage is the same as with CircuitPython!