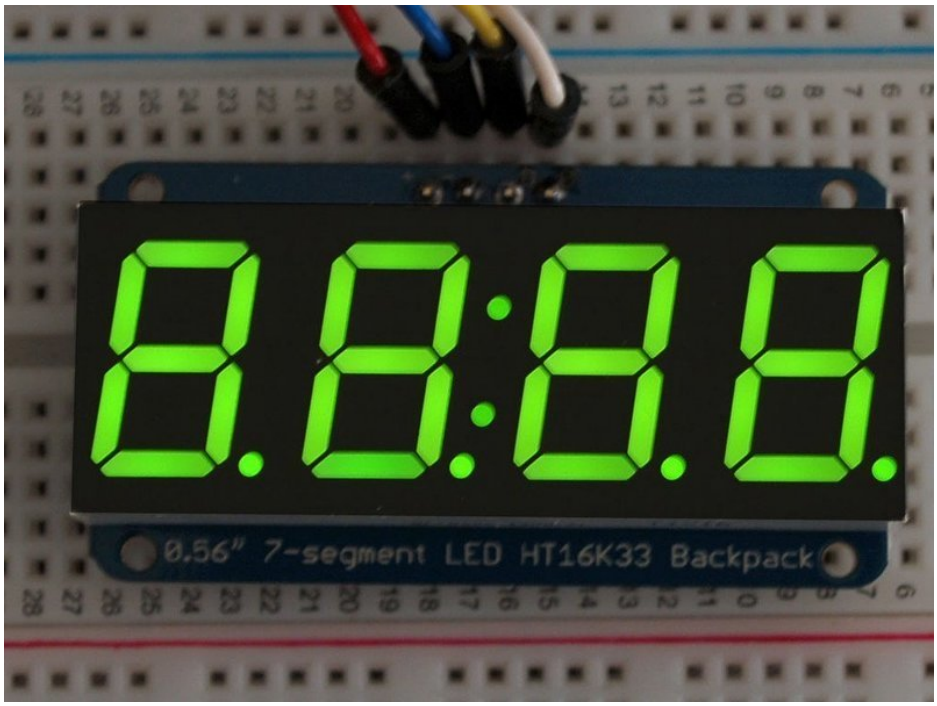
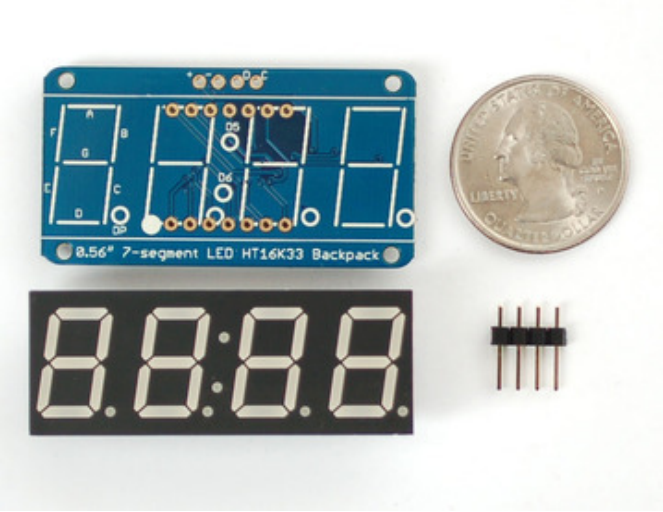
0.56" 7-Segment Backpack

This version of the LED backpack is designed for these big bright 7-segment displays. These 7-segment displays normally require 13 pins (5 'characters' and 8 total segments each) This backpack solves the annoyance of using 13 pins or a bunch of chips by having an I2C constant-current matrix controller sit neatly on the back of the PCB. The controller chip takes care of everything, drawing all the LEDs in the background. All you have to do is write data to it using the 2-pin I2C interface. There are three address select pins so you can select one of 8 addresses to control up to 8 of these on a single 2-pin I2C bus (as well as whatever other I2C chips or sensors you like). The driver chip can 'dim' the entire display from 1/16 brightness up to full brightness in 1/16th steps. It cannot dim individual LEDs, only the entire display at once.



Assembly

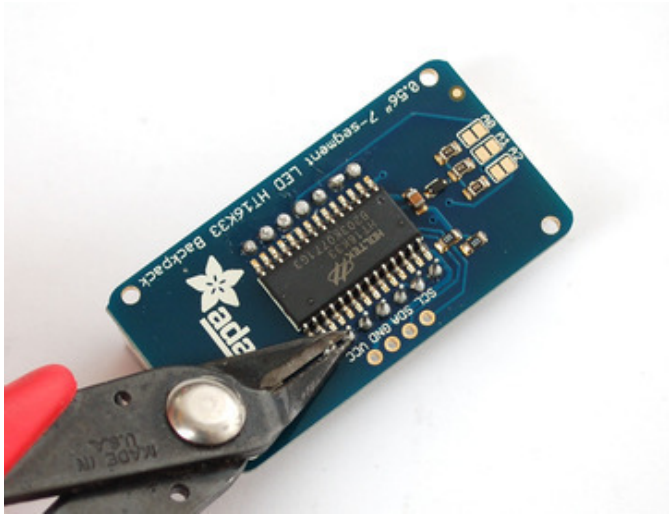


When you buy a pack from Adafruit, it comes with the fully tested and assembled backpack as well as a 7-segment display in one of the colors we provide (say, red, yellow, blue or green). You'll need to solder the matrix onto the backpack but it's an easy task.

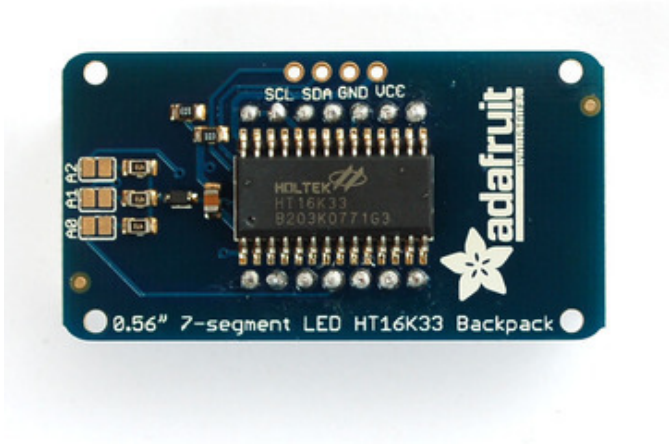
Remove the parts from packaging and place the LED matrix OVER the silkscreen side. **DO NOT PUT THE DISPLAY ON UPSIDE DOWN OR IT WONT WORK!! Check the image below to make sure the 'decimal point' dots are on the bottom, matching the silkscreen.**

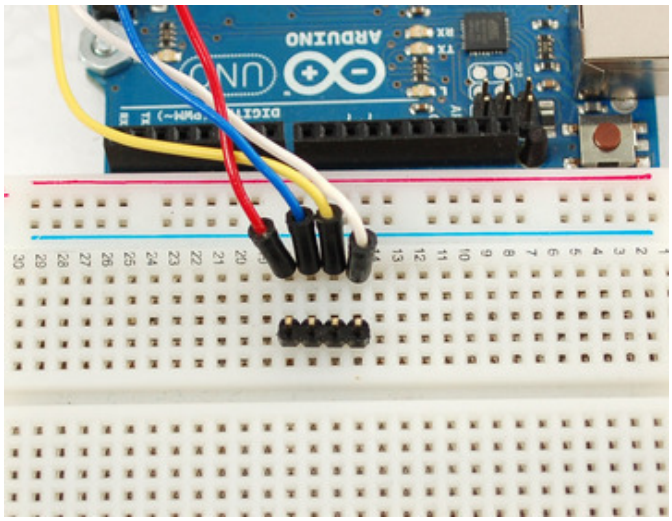
Turn the backpack over so it is sitting flat on the matrix.

Solder all 14 pins.

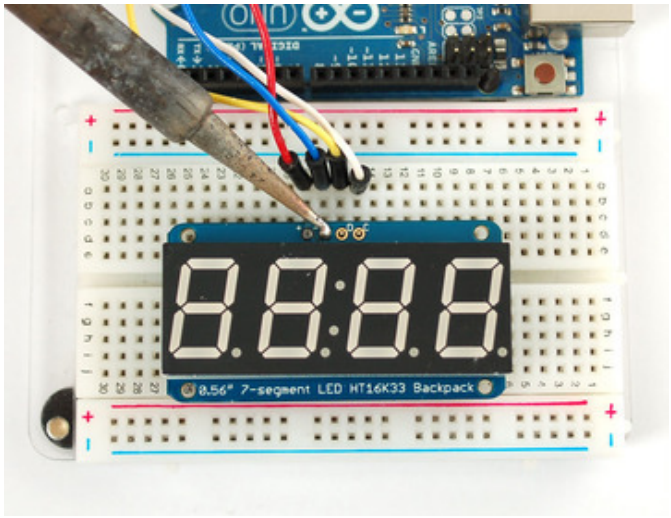


Clip the long pins.





Now you're ready to wire it up to a microcontroller. We'll assume you want to use a 4pin header. You can also of course solder wires directly. Place a 4-pin piece of header with the LONG pins down into the breadboard.



Place the soldered backpack on top of the header and Solder 'em!

That's it! now you're ready to run the firmware!

Arduino Setup

Seven-Segment Backpack Firmware

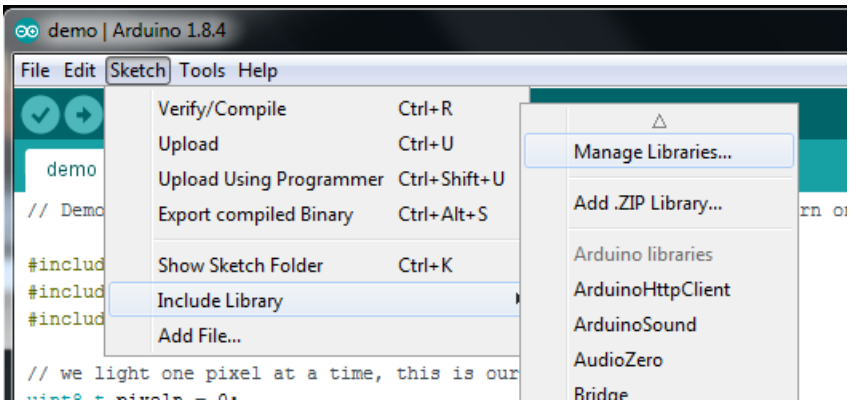
We wrote a basic library to help you work with the 7-segment backpack. The library is written for the Arduino and will work with any Arduino as it just uses the I2C pins. The code is very portable and can be easily adapted to any I2C-capable micro.

Wiring to the matrix is really easy

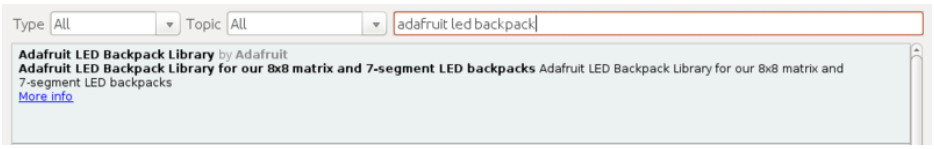
- Connect **CLK** to the I2C clock - on Arduino UNO thats Analog #5, on the Leonardo it's Digital #3, on the Mega it's digital #21
- Connect **DAT** to the I2C data - on Arduino UNO thats Analog #4, on the Leonardo it's Digital #2, on the Mega it's digital #20
- Connect **GND** to common ground
- Connect **VCC+** to power - 5V is best but 3V also seems to work for 3V microcontrollers.

Next, download the **Adafruit LED Backpack** library and the **Adafruit GFX** library from the Arduino library manager.

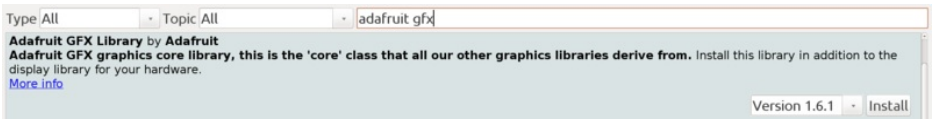
Open up the Arduino library manager:



Search for the **Adafruit LED Backpack** library and install it



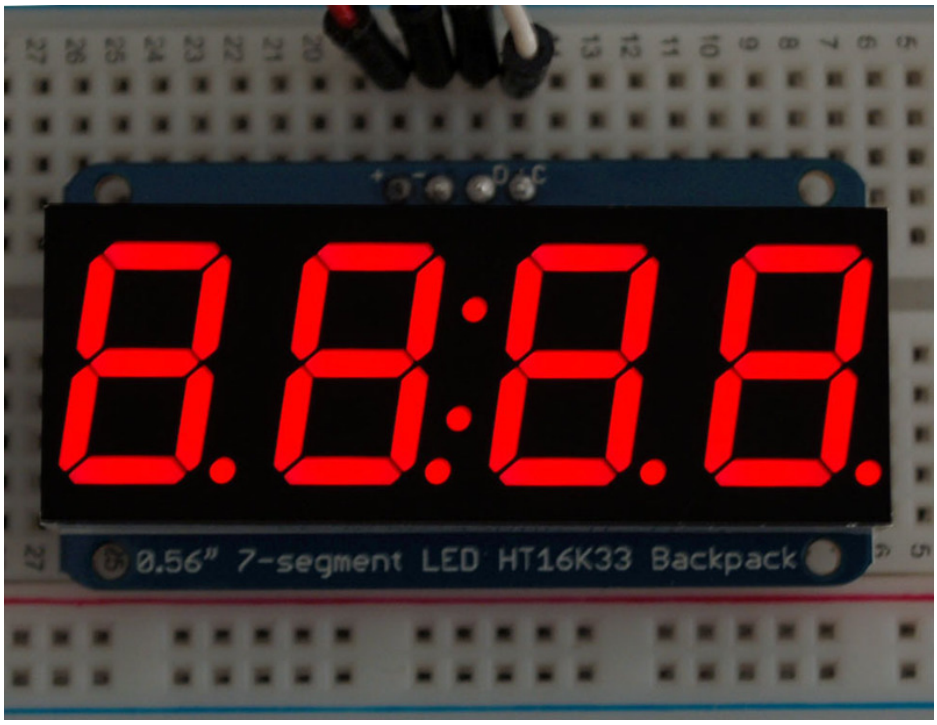
Search for the **Adafruit GFX** library and install it



Once you've restarted you should be able to select the **File>Examples>Adafruit_LEDBackpack>sevenseg** example sketch. Upload it to your Arduino as usual. You should see a basic test program that goes through a bunch of different routines.

We also have a great tutorial on Arduino library installation at:

http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)



Once you're happy that the matrix works, you can write your own sketches.

There's a few ways you can draw to the display. The easiest is to just call **print -** just like you do with **Serial**

- **print(*variable*,HEX) -** this will print a hexidecimal number, from 0000 up to FFFF
- **print(*variable*,DEC)** or **print(*variable*) -** this will print a decimal integer, from 0000 up to 9999

If you need more control, you can call **writeDigitNum(*location*, *number*)** - this will write the *number* (0-9) to a single location. Location #0 is all the way to the left, location #2 is the colon dots so you probably want to skip it, location #4 is all the way to the right. If you want a decimal point, call **writeDigitNum(*location*, *number*, true)** which will paint the decimal point. To draw the colon, use**drawColon(*true* *or* *false*)**

If you want even more control, you can call **writeDigitRaw(*location*,*bitmask*)** to draw a raw 8-bit mask (as stored in a uint8_t) to that location.

All the drawing routines only change the display memory kept by the Arduino. Don't forget to call **writeDisplay()** after drawing to 'save' the memory out to the matrix via I2C.

There are also a few small routines that are special to the backpack:

- **setBrightness(*brightness*)**- will let you change the overall brightness of the entire display. 0 is least bright, 15 is brightest and is what is initialized by the display when you start
- **blinkRate(*rate*)** - You can blink the entire display. 0 is no blinking. 1, 2 or 3 is for display blinking.

CircuitPython Wiring and Setup

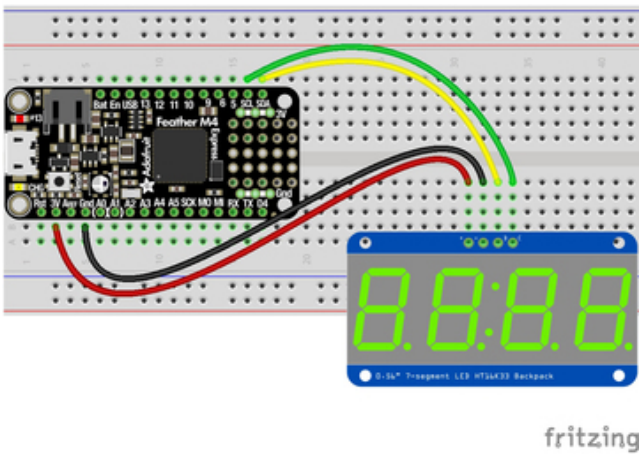
Wiring

It's easy to use LED 7-Segment Displays with CircuitPython and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This module allows you to easily write CircuitPython code to control the display.

You can use this sensor with any CircuitPython microcontroller board.

We'll cover how to wire the 7-Segment Display to your CircuitPython microcontroller board. First assemble your 7-Segment Display.

Connect the 7-Segment Display to your microcontroller board as shown below.



- **Microcontroller 3V** to **7-Segment Display VIN**
- **Microcontroller GND** to **7-Segment Display GND**
- **Microcontroller SCL** to **7-Segment Display SCL**
- **Microcontroller SDA** to **7-Segment Display SDA**

https://adafru.it/ICk

https://adafru.it/ICk

Library Setup

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) library on your board.

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

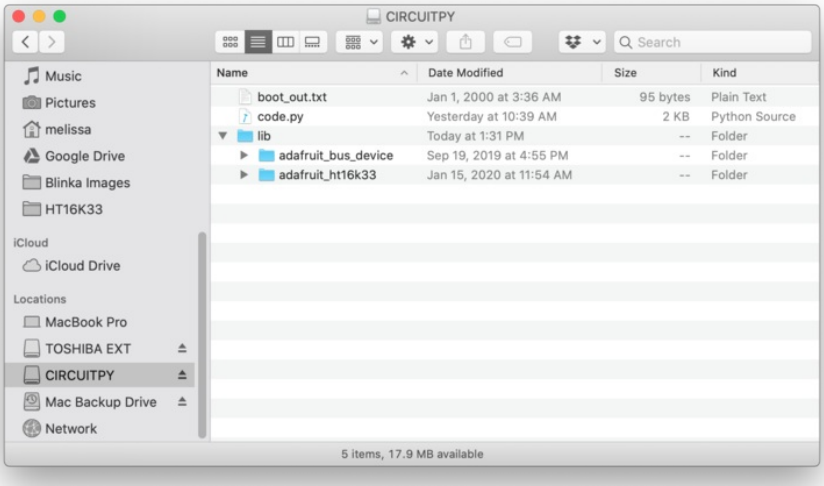
For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with Adafruit's CircuitPython bundle (https://adafru.it/zdx). This is an all-in-one package that includes the necessary libraries to use the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

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).**

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



Python Wiring and Setup

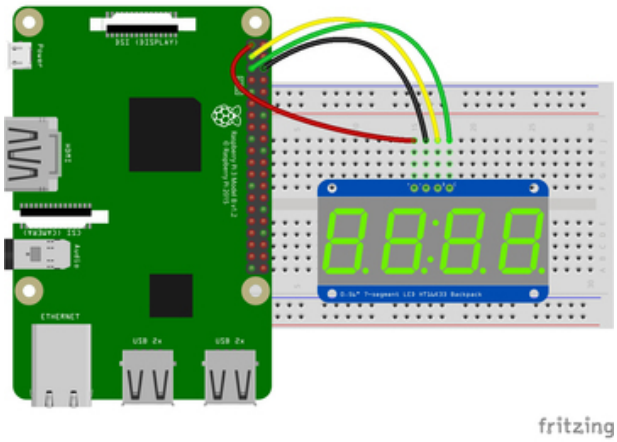
Wiring

It's easy to use 7-Segment Displays with Python and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This library allows you to easily write Python code to control the display.

We'll cover how to wire the 7-Segment Display to your Raspberry Pi. First assemble your 7-Segment Display.

Since there's *dozens* of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Connect the 7-Segment Display as shown below to your Raspberry Pi.



- **Raspberry Pi 3.3V** to **7-Segment Display VIN**
- **Raspberry Pi GND** to **7-Segment Display GND**
- **Raspberry Pi SCL** to **7-Segment Display SCL**
- **Raspberry Pi SDA** to **7-Segment Display SDA**

https://adafru.it/ICI

https://adafru.it/ICI

Setup

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN)!

Python Installation of HT16K33 Library

Once that's done, from your command line run the following command:

- `pip3 install adafruit-circuitpython-ht16k33`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

Pillow Library

We also need PIL, the Python Imaging Library, to allow using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go.

CircuitPython and Python 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.

Initialization

First you'll need to initialize the I2C bus for your board. It's really easy, first import the necessary modules. In this case, we'll use `board` and `Seg7x4`.

Then just use `board.I2C()` 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).

Then to initialize the display, you just pass `i2c` in.

```
import board
from adafruit_ht16k33.segments import Seg7x4

i2c = board.I2C()
display = Seg7x4(i2c)
```

If you bridged the address pads on the back of the display, you could pass in the address. The addresses for the HT16K33 can range between 0x70 and 0x77 depending on which pads you have bridged, with 0x70 being used if you haven't bridged any of them. For instance, if you bridge only the **A0** pad, you would use `0x71` like this:

```
display = Seg7x4(i2c, address=0x71)
```

Setting the Brightness

You can set the brightness of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/16 increments **between 0 and 15** with 15 being the brightest. So to set the display to half brightness, you would use the following:

```
display.brightness = 7
```

Setting the Blink Rate

You can set the blink rate of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/4 increments **between 0 and 3** with 3 being the fastest blinking. So to set the display to blink at full speed, you would use the following:

```
display.blink_rate = 3
```

Printing Text

To print text to the display, you just use the print function. For the 7-segment display, valid characters are 0-9, letters A-F, a period, and a hyphen. So if we want to print ABCD, we would use the following:

```
display.print("ABCD")
```

Printing Numbers

Printing numbers is done similar to printing text, except without the quotes, though you can still print numbers in a string as well.

```
display.print(1234)
```

Printing Hexidecimal Values

To print hexidecimal values, you use the `print_hex` function:

```
display.print_hex(0x1A2B)
```

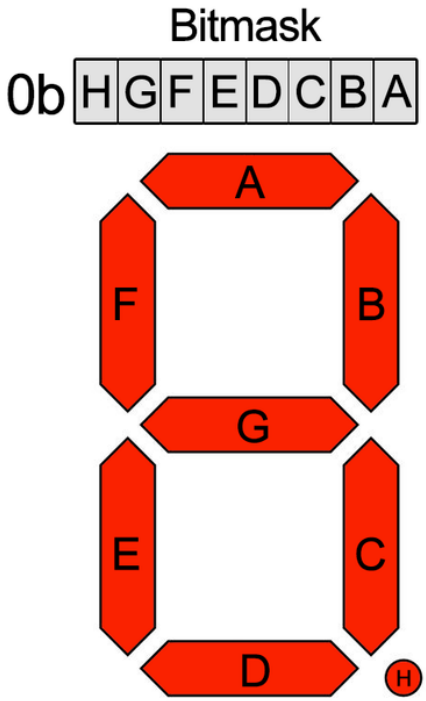
Setting Individual Characters

To set individual characters, you simply treat the `display` object as a list and set it to the value that you would like.

```
display[0] = '1'
display[1] = '2'
display[2] = 'A'
display[3] = 'B'
```

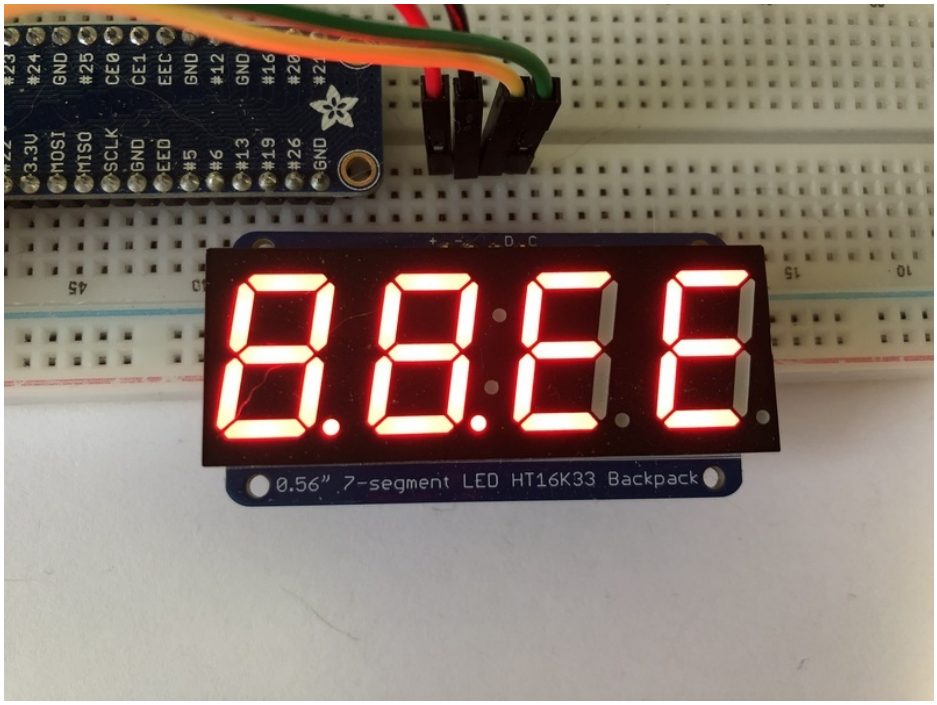
Setting Individual Segments

To set individual segments to turn on or off, you would use the set_digit_raw function to pass the digit that you want to change and the bitmask. This can be really useful for creating your own characters. The bitmask corresponds to the following diagram:



The bitmask is a single 8-bit number that can be passed in as a single Hexidecimal, Decimal, or binary number. This will use a couple different methods to display `8.8.EE`:

```
display.set_digit_raw(0, 0xFF)
display.set_digit_raw(1, 0b11111111)
display.set_digit_raw(2, 0x79)
display.set_digit_raw(3, 0b01111001)
```



Filling all Segments

To fill the entire display, just use the fill() function and pass in either 0 or 1 depending on whether you want all segments off or on. For instance, if you wanted to set everything to on, you would use:

```
display.fill(1)
```

Scrolling Display Manually

If you want to scroll the displayed data to the left, you can use the `scroll()` function. You can pass in the number of places that you want to scroll. The right-most digit will remain unchanged and you will need to set that manually. After scrolling, you will need to call the show function. For example if you wanted to print an A and then scroll it over to spaces, you would do the following.

```
display.print("A")
display.scroll(2)
display[3] = " "
display.show()
```

Displaying the Colon

There are a couple of different ways to display a colon on the 7-segment display. The first and easiest way is to use the print function:

```
display.print("12:30")
```

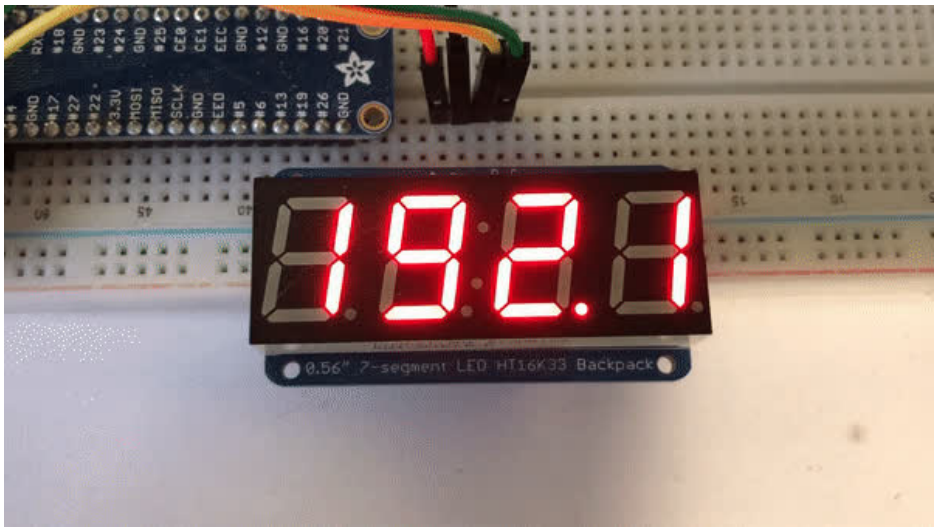
The other way to control it is to access the colon with the colon property and set it to `True` or `False`:

```
display.colon = False
```

Displaying an Automatic Scrolling Marquee

To make displaying long text easier, we've added a marquee function. You just pass it the full string. Optionally, you can pass it the amount of delay between each character. This may be useful for displaying an IP address, a phone number, or other numeric data:

```
display.marquee('192.168.100.102... ')
```



By default it is 0.25 seconds, but you can change this by providing a second parameter. You can optionally pass `False` for a third parameter if you would not like to have it loop. So if you wanted each character to display for half a second and didn't want it to loop, you would use the following:

```
display.marquee('192.168.100.102... ', 0.5, False)
```