Getting Started with Micro Dot pHAT

This tutorial will show you how to install the Micro Dot pHAT Python library, and then walk through its functionality, finishing with an example of how to use Micro Dot pHAT as a clock.

If you haven't already, then follow our guide on how to assemble and solder your Micro Dot pHAT.

Installing the software

We always recommend using the most up-to-date version of Raspbian, as this is what we test our boards and software against, and it often helps to start with a completely fresh install of Raspbian, although this isn't necessary.

As with most of our boards, we've created a really quick and easy one-line-installer to get your Micro Dot pHAT set up. We'd suggest that you use this method to install the Micro Dot pHAT software.

Open a new terminal, and type the following, making sure to type 'y' or 'n' when prompted:

```
curl -sS get.pimoroni.com/microdotphat | bash
```

Once that's done, it probably a good idea to reboot your Pi to let the changes propagate.

Using the software

Open a new terminal and type `sudo python` to open a new Python prompt.

Lighting individual pixels

We'll begin by looking at the `set_pixel` function from the Python library. This function lets you set the state of individual pixels. Note that you can't individually control the brightness of each pixel, only the brightness of all of the pixels at once.

Type the following in your Python prompt to import the functions and variables we'll need to light some pixels:

```
from microdotphat import set_pixel, clear, show, WIDTH, HEIGHT
```

Before we start, we'll clear any pixels that may already have been set.

```
clear()
```

The pixels on Micro Dot pHAT are numbered from the top left hand corner of display 1 and, as is the convention in Python, are numbered from 0. So, the top left pixel of display 1 is `0, 0`.

We'll light the top left pixel first.

```
set_pixel(0, 0, 1)
```

The first two arguments passed to the `set_pixel` function are the x and y (row and column) coordinates of the pixel we want to light, and the third, `1` tells the function that we want to switch the pixel on rather than off.

You'll probably notice that... nothing has happened. That's because we need to call the `show` function to push the changes we've set onto the displays.

```
show()
```

You should now see the top left hand pixel of the first display lit.

We'll now `clear` the displays and call `show` again to push the changes to the displays.

```
clear()

show()
```

We're going to extend this now to the whole of the first column on the first display. The library includes `WIDTH` and `HEIGHT` variables that define the width and height of all six displays.

If you `print(WIDTH, HEIGHT)`, you'll see that it will tell you that the displays are 45 pixels wide by 7 pixels high. Why not 30 pixels wide (5x6)?

Because of the spacing of the displays, to make text and animations display correctly across the displays, we've added some imaginary pixels between the displays (3 between each) to make things look sensible.

This means that the top left pixel of display 2 is `0, 8` rather than `0, 5`.

We're going to use the the `HEIGHT` variable to light all of the pixels in the first column of display 1.

```
for y in range(HEIGHT):

    set_pixel(0, y, 1)


show()
```

All of the pixels in the first column should now be lit.

You've now seen how to light individual pixels and we could, in theory, use this method to light all of the pixels across all of the displays, but there is a function that makes lighting all of the pixels a whole lot easier.

Lighting all of the pixels

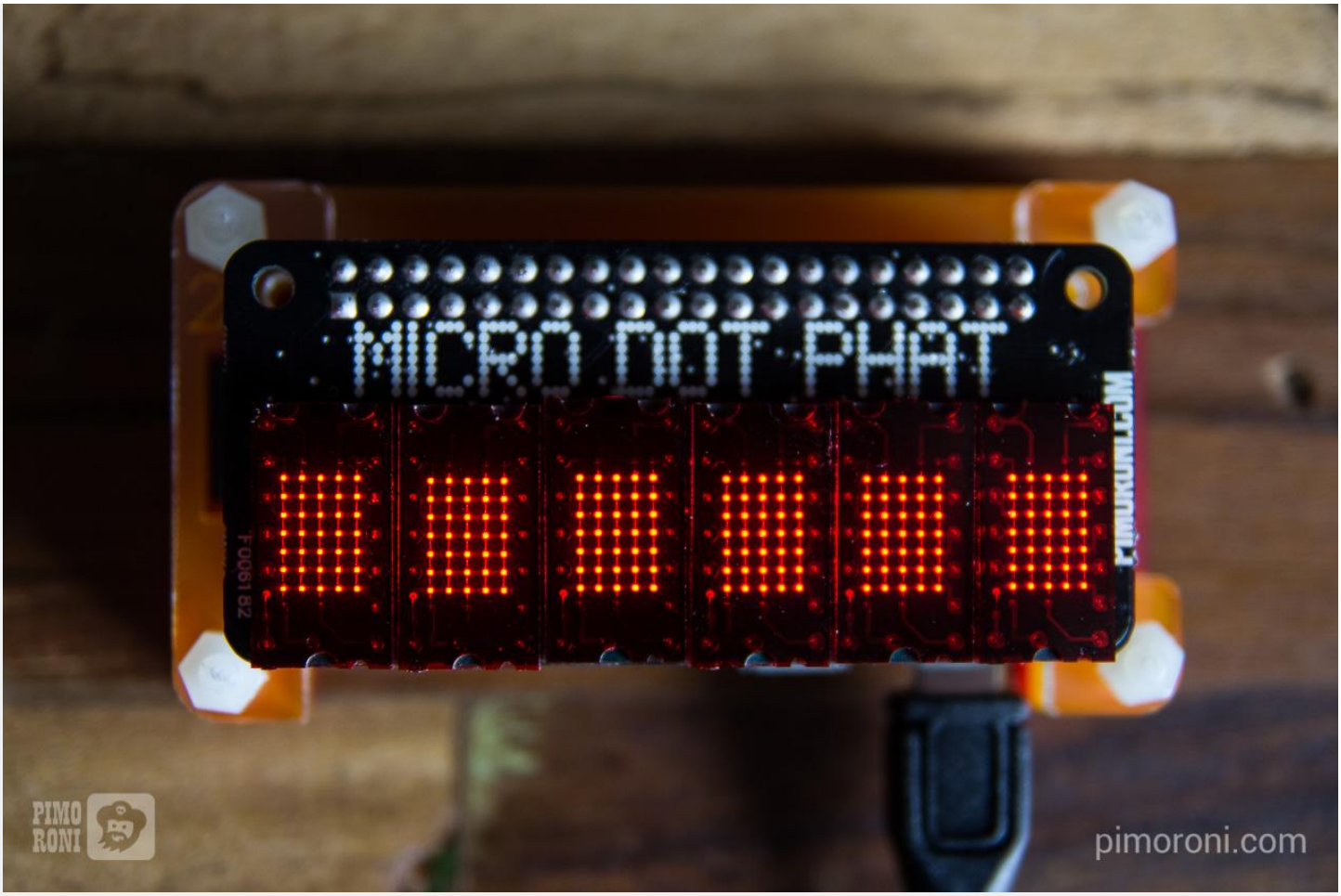
Before we start this section, we'll clear the displays and import what we need.

```
from microdotphat import fill, clear, show, WIDTH, HEIGHT


clear()
show()
```

Lighting all of the pixels is as simple as calling the `fill` function, making sure to pass `1` in, to tell it to switch the pixels on rather than off, and then calling `show` again to propagate the change.

```
fill(1)
show()
```



Taking this slightly further, if you wanted to flash all of the pixels on and off, you could do the following:

```
import time
from microdotphat import fill, clear, show, WIDTH, HEIGHT

clear()

while True:
    fill(1)
    show()
    time.sleep(0.2)
    clear()
    show()
    time.sleep(0.2)
```

We use the `time.sleep` function to introduce a small delay of 0.2 seconds each time the displays are either filled or cleared, and wrap the whole thing in a `while True` loop to keep it running continuously.

Now that we've seen how to control the pixels, we'll move on to displaying text and scrolling it.

Displaying and scrolling text

The Micro Dot pHAT library has a convenient `write_string` function, that can be passed a string which is passed into the buffer to be displayed on the matrices. Depending on how long your string is, i.e. if it's longer than six characters, you may want to scroll it across the displays so that all of it can be read.

We'll look now at an example of how to write a string into the buffer and then scroll it across the displays.

```
import time
from microdotphat import write_string, scroll, clear, show


clear()
write_string('YARR!!      ')


while True:
    scroll()
    show()
    time.sleep(0.05)
```

We've added a bit of empty padding to the end of our string, to make it more legible, and you'll notice that we've put the `scroll()` inside a `while True` loop, to keep it scrolling constantly.

Each time you call the `scroll` function, it will shift one place through the buffer, and therefore give the impression of scrolling the text from right to left.

Because we only have six characters in our string, we probably don't need to scroll it constantly. Try the following to just write the string and then display it static-ly:

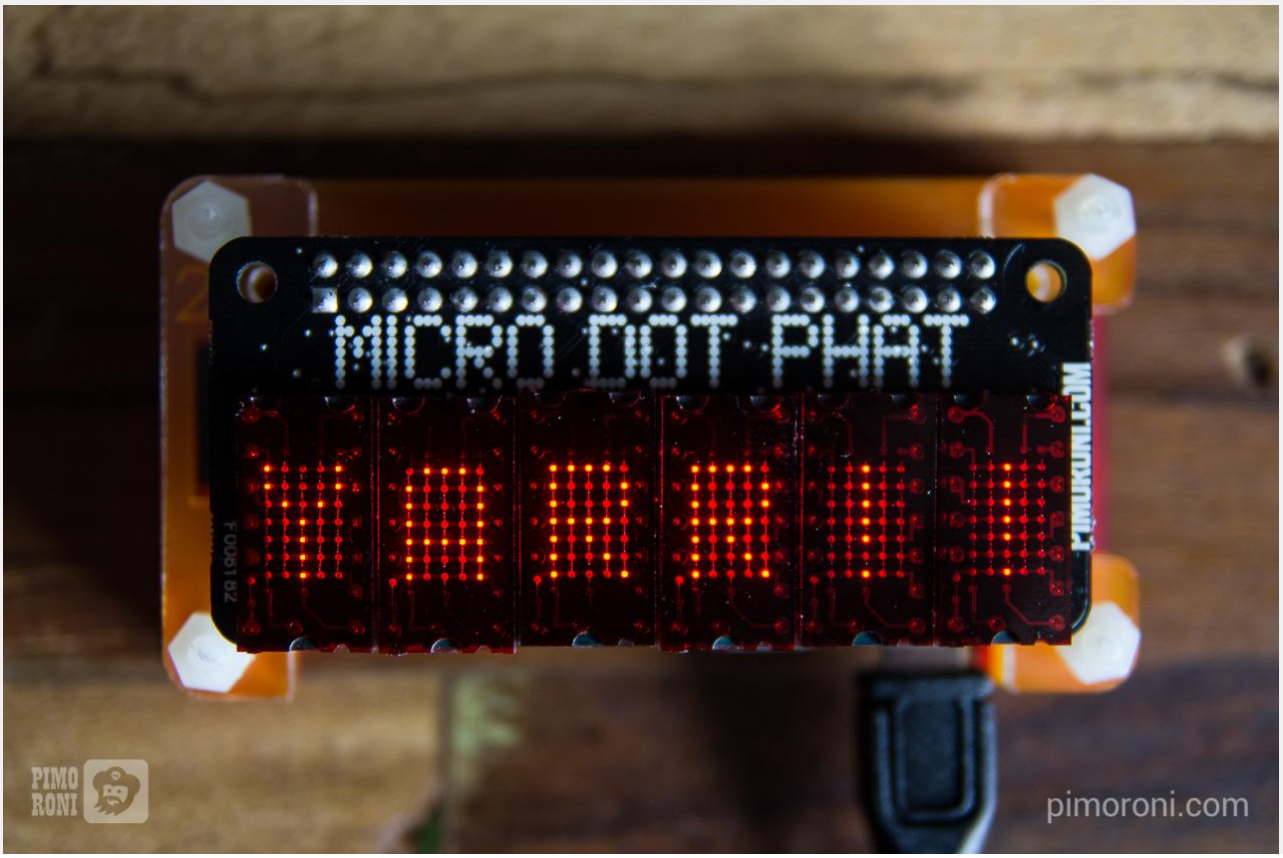
```
clear()
write_string('YARR!!')
show()
```

You'll notice that it's awkward to read this text when it's not scrolling. For situations like this, where you have six or fewer characters (assuming you have all six displays installed), we've added an option that you can pass to `write_string` that specifies how to space the letters, also known as `kerning`.

Try the following to write one character of our `YARR!!` string to each of the six displays:

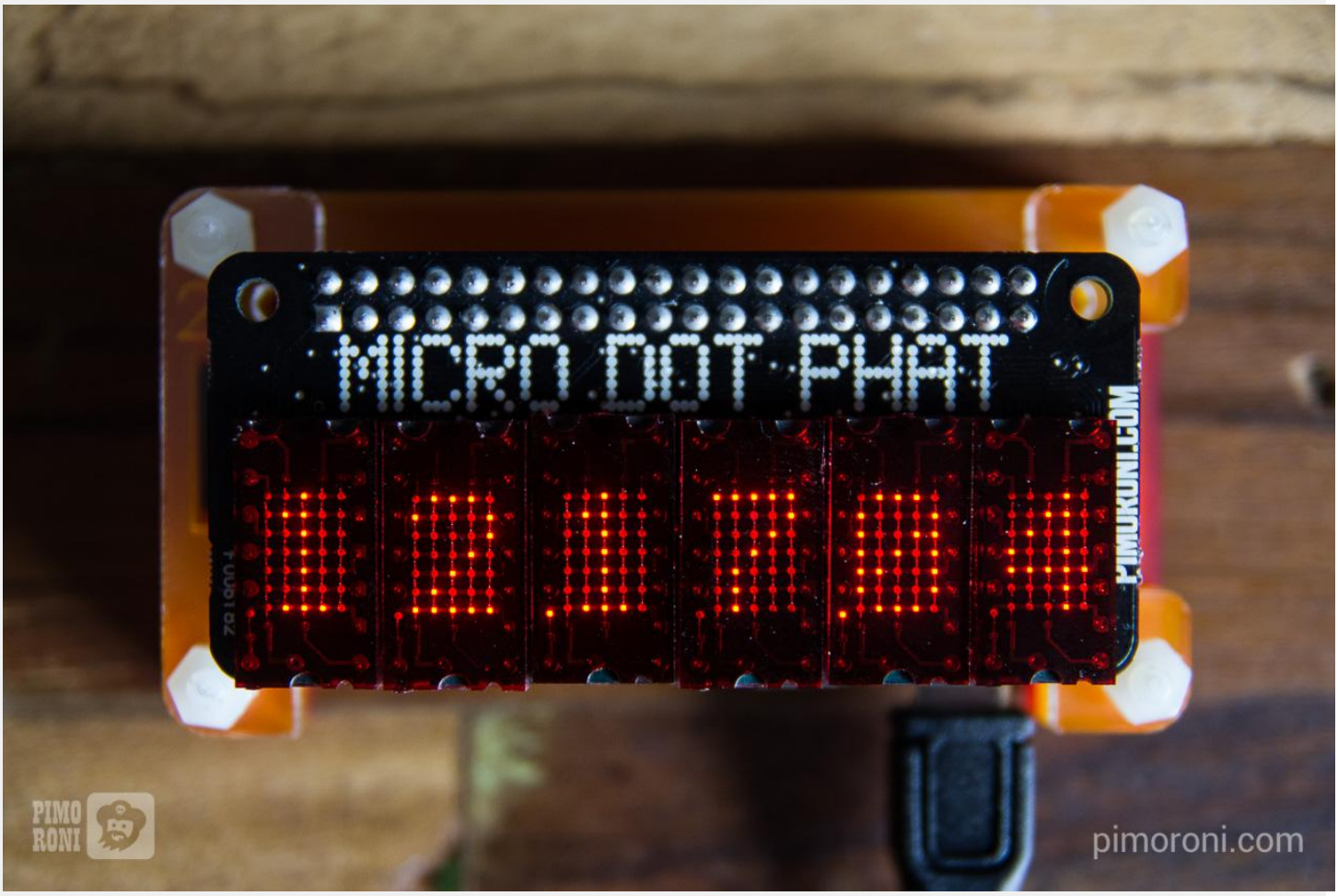
```
clear()
write_string('YARR!!', kerning=False)
show()
```

You should now see one character displayed on each matrix.



Clock example

Finally, we'll look at a simple little example of how you can use your Micro Dot pHAT as a retro clock. This example is included with the Python library, but we'll break down what the code does here to help you understand how it all works.



Here's all of the code:

```
import time
import datetime
from microdotphat import write_string, set_decimal, clear, show


while True:
    clear()
    t = datetime.datetime.now()
    if t.second % 2 == 0:
        set_decimal(2, 1)
        set_decimal(4, 1)
    else:
        set_decimal(2, 0)
        set_decimal(4, 0)
    write_string(t.strftime('%H%M%S'), kerning=False)
    show()
    time.sleep(0.05)
```

First, we import the libraries and functions that we'll be using elsewhere in our code.

```
import time
import datetime
```

```
from microdotphat import write_string, set_decimal, clear, show
```

We need the `time` library again to introduce a small delay between each iteration of the `while True` loop that will run all of our code.

The `datetime` library will allow us to grab the wall clock time and then reformat it into the specific format we need: the two digits each from the hour, minutes and seconds.

And we need several functions from our `microdotphat` library: `write_string`, `set_decimal`, `clear`, and `show`. `set_decimal` allows us to control the decimal points on the bottom left of each display, which will act as separators and also blink on and off each second.

We're going to wrap everything in a `while True:` loop to keep it running until we exit the program with control-c.

```
while True:

    clear()

    t = datetime.datetime.now()

    if t.second % 2 == 0:

        set_decimal(2, 1)

        set_decimal(4, 1)

    else:

        set_decimal(2, 0)

        set_decimal(4, 0)

    write_string(t.strftime('%H%M%S'), kerning=False)

    show()

    time.sleep(0.05)
```

Let's break down what all that does.

First, we `clear` the existing data in the buffer ready for the new data going in. Then, we grab the time and assign it to a variable `t` with `t = datetime.datetime.now()`.

Our `t` datetime object has lots of methods associated with it, two of which we'll use here. The first is the `second` method, and it returns exactly what you might expect, the current seconds value, as an integer. The second is the `strftime` method, which is an incredibly useful way of reformatting the datetime object into whichever format we'd like.

We have an `if` and `else` that check whether the current number of seconds is divisible by 2, `if t.second % 2 == 0:`, and if it is then it sets the decimal points between the hours and minutes, and the minutes and seconds on: `set_decimal(2, 1)` and `set_decimal(4, 1)`. The `else` catches oddly numbered seconds and sets the decimal points off accordingly.

Then, we write our formatted time string to the buffer with `write_string(t.strftime('%H%M%S'), kerning=False)`. `'%H%M%S'` tells `strftime` to return just the digits of the hour, minutes and seconds, in 24-hour format, e.g. 141500 for 2.15pm. We use `kerning=False` to display one digit on each of the six displays.

Finally, we call `show()` to push the buffer out onto the displays, and add a small delay of 50 milliseconds with `time.sleep(0.05)` to give sufficient resolution to our clock without running it an unnecessary number of times each second (at 50 milliseconds delay, it will run around 20 times each second).

Taking it further

That's all for this getting started guide, but why not use what you've learned to think up a fun new use for your Micro Dot pHAT?

Perhaps a Twitter ticker? Or a tiny game of snake or pong? Or a weather display, combined with our Enviro pHAT board? Or a really low resolution screen to play movies on? (probably not that last one...)