2.2" TFT Display
Created by lady ada
### Guide Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>Pinouts</td>
<td>6</td>
</tr>
<tr>
<td>Assembly</td>
<td>7</td>
</tr>
<tr>
<td>Arduino Wiring</td>
<td>9</td>
</tr>
<tr>
<td>Arduino UNO or Compatible Wiring</td>
<td>9</td>
</tr>
<tr>
<td>Wiring for Other Boards</td>
<td>10</td>
</tr>
<tr>
<td>Arduino Code</td>
<td>12</td>
</tr>
<tr>
<td>Install Library</td>
<td>12</td>
</tr>
<tr>
<td>Run Graphics Test</td>
<td>13</td>
</tr>
<tr>
<td>Adafruit GFX Library</td>
<td>14</td>
</tr>
<tr>
<td>Bitmaps</td>
<td>14</td>
</tr>
<tr>
<td>CircuitPython Displayio Quickstart</td>
<td>22</td>
</tr>
<tr>
<td>Preparing the Breakout</td>
<td>22</td>
</tr>
<tr>
<td>Wiring the Breakout to the Feather</td>
<td>22</td>
</tr>
<tr>
<td>Required CircuitPython Libraries</td>
<td>23</td>
</tr>
<tr>
<td>Code Example Additional Libraries</td>
<td>23</td>
</tr>
<tr>
<td>CircuitPython Code Example</td>
<td>24</td>
</tr>
<tr>
<td>Code Details</td>
<td>26</td>
</tr>
<tr>
<td>Where to go from here</td>
<td>30</td>
</tr>
<tr>
<td>Python Wiring and Setup</td>
<td>31</td>
</tr>
<tr>
<td>Wiring</td>
<td>31</td>
</tr>
<tr>
<td>ILS9341 and HX-8357-based Displays</td>
<td>31</td>
</tr>
<tr>
<td>2.2&quot; Display</td>
<td>31</td>
</tr>
<tr>
<td>2.4&quot;, 2.8&quot;, 3.2&quot;, and 3.5&quot; Displays</td>
<td>32</td>
</tr>
<tr>
<td>ST7789 and ST7735-based Displays</td>
<td>33</td>
</tr>
<tr>
<td>1.3&quot;, 1.54&quot;, and 2.0&quot; IPS TFT Display</td>
<td>33</td>
</tr>
<tr>
<td>0.96&quot;, 1.14&quot;, and 1.44&quot; Displays</td>
<td>34</td>
</tr>
<tr>
<td>1.8&quot; Display</td>
<td>35</td>
</tr>
<tr>
<td>SSD1351-based Displays</td>
<td>36</td>
</tr>
<tr>
<td>1.27&quot; and 1.5&quot; OLED Displays</td>
<td>36</td>
</tr>
<tr>
<td>SSD1331-based Display</td>
<td>37</td>
</tr>
<tr>
<td>0.96&quot; OLED Display</td>
<td>37</td>
</tr>
<tr>
<td>Setup</td>
<td>38</td>
</tr>
<tr>
<td>Python Installation of RGB Display Library</td>
<td>38</td>
</tr>
<tr>
<td>DejaVu TTF Font</td>
<td>39</td>
</tr>
<tr>
<td>Pillow Library</td>
<td>39</td>
</tr>
<tr>
<td>Python Usage</td>
<td>40</td>
</tr>
<tr>
<td>Turning on the Backlight</td>
<td>40</td>
</tr>
<tr>
<td>Displaying an Image</td>
<td>40</td>
</tr>
<tr>
<td>Drawing Shapes and Text</td>
<td>45</td>
</tr>
<tr>
<td>Displaying System Information</td>
<td>48</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>54</td>
</tr>
<tr>
<td>Display does not work on initial power but does work after a reset.</td>
<td>54</td>
</tr>
<tr>
<td>Downloads</td>
<td>56</td>
</tr>
<tr>
<td>Files:</td>
<td>56</td>
</tr>
</tbody>
</table>

© Adafruit Industries  
https://learn.adafruit.com/2-2-tft-display  
Page 2 of 57
This lovely little display breakout is the best way to add a small, colorful and bright display to any project. Since the display uses 4-wire SPI to communicate and has its own pixel-addressable frame buffer, it can be used with every kind of microcontroller. Even a very small one with low memory and few pins available!

NOTE: This tutorial no longer covers the 176x220 pixel version of the display - only the newer 320x240 pixel version. Chances are you DO NOT have the older version!
The 2.2" display has 320x240 color pixels. Unlike the low cost "Nokia 6110" and similar LCD displays, which are CSTN type and thus have poor color and slow refresh, this display is a true TFT! The TFT driver (ILI9340) can display full 16-bit color. And the LCD will always come with the same driver chip so there's no worries that your code will not work from one to the other.

The breakout has the TFT display soldered on (it uses a delicate flex-circuit connector) as well as a ultra-low-dropout 3.3V regulator and a 3/5V level shifter so you can use it with 3.3V or 5V power and logic. We also had a little space so we placed a microSD card holder so you can easily load full color bitmaps from a
FAT16/FAT32 formatted microSD card. The microSD card is not included but you can pick one up here (http://adafruit.it/102).
This color display uses SPI to receive image data. That means you need at least 4 pins - clock, data in, TFT CS and D/C. If you'd like to have SD card usage too, add another 2 pins - data out and card cs. However, there's a couple other pins you may want to use, lets go thru them all!

- **GND** - this is the power and signal ground pin
- **Vin** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **D/C** - this is the TFT SPI data or command selector pin. Use 3-5V logic level
- **RESET** - this is the TFT reset pin. Connect to ground to reset the TFT! Its best to have this pin controlled by the library so the display is reset cleanly, but you can also connect it to the Arduino Reset pin, which works for most cases. There is an automatic-reset chip connected so it will reset on power-up. Use 3-5V logic level
- **SD Card CS / SDCS** - this is the SD card chip select, used if you want to read from the SD card. Use 3-5V logic level
- **LCD_CS** - this is the TFT SPI chip select pin. Use 3-5V logic level
- **MOSI** - this is the SPI Microcontroller Out Serial In pin, it is used to send data from the microcontroller to the SD card and/or TFT. Use 3-5V logic level
- **MISO** - this is the SPI Microcontroller In Serial Out pin, its used for the SD card. It isn't used for the TFT display which is write-only. It is 3.3V logic out (but can be read by 5V logic)
- **CLK** - this is the SPI clock input pin. Use 3-5V logic level
- **Backlite** - this is the PWM input for the backlight control. It is by default pulled high (backlight on) you can PWM at any frequency or pull down to turn the backlight off. Use 3-5V logic level
Assembly

Start by connecting a piece of header to the display. This will make breadboarding much easier. Break off a piece of 0.1” header 9 pins long and place it into a breadboard, long pins facing down into the breadboard.

Place the display on top

Solder all the pins
Arduino Wiring

There are two ways to wire up these displays:

**Software SPI is a more flexible method** (you can use any pins on the Arduino) and **hardware SPI is much faster** (4-8x faster) but you are required to use the hardware SPI pins.

Since the display is quite large, we found that drawing would seem really slow if using 'software' SPI. For that reason, we'll show primarily how to wire up using hardware SPI and then how you can change the pins if desired.

Hardware SPI means that we have to connect the **CLK** and **MOSI** pins to fixed digital pins.

On '328 and '168 Arduinos, **CLK** must connect to digital **13** and **MOSI** must connect to digital **11**. If using an Arduino Mega, connect **CLK** to **52** and **MOSI** to **51**. If you're using another kind of Arduino you'll need to use the SPI hardware port ([https://adafru.it/iCE](https://adafru.it/iCE))

Digital **10** (53 on Arduino Mega) must also be an output (but doesn't need to be connected to any particular pin).

**Arduino UNO or Compatible Wiring**

We'll use the following pin connections:

- **GND** connects to ground - black wire
- **VIN** connects to +5V - red wire
- **DC** (data/clock) connects to digital **9** on Atmega328
- Skip **SDCS** (SD card chip select - used for SD card interfacing)
- **CS** (chip select) connects to digital **10** on Atmega328
- **MOSI** (data out) connects to digital **11** on Atmega328
- **SCK** (clock) connects to digital **13** on Atmega328
- Skip **MISO** (data in - used for SD card interfacing)
Wiring for Other Boards

We'll use the following pin connections:

- **GND** connects to ground - black wire
- **VIN** connects to +5V - red wire
- **DC (data/clock)** connects to digital 9
- Skip **SDCS** (SD card chip select - used for SD card interfacing)
- **CS (chip select)** connects to digital 10
- **MOSI (data out)** connects to **MOSI**
- **SCK (clock)** connects to **SCK**
- Skip **MISO** (data in - used for SD card interfacing)
You can later change the CS and RST pins but to match the tutorial, use this connection diagram.
Arduino Code

Once you have the display wired up, its time to test your wiring by uploading the example code we have written. Again, we suggest using an Arduino to test.

Install Library

Go to the Arduino Library manager under Sketch → Include Library → Manage Libraries...

From within the Library manager, start by installing Adafruit GFX:

Then look for and install the Adafruit ILI9341 library

Note that this display has an ILI9340 but we still use the ILI9341 library, it's OK! The chips are nearly identical
If using an older Arduino IDE (pre-1.8.10), also locate and install Adafruit_BusIO (newer versions do this one automatically).

You can read more about installing libraries in our tutorial (https://adafruit.it/aYG).

Run Graphics Test

Restart the Arduino IDE. You should now be able to select File → Examples → Adafruit_ILI9341 → graphicstest sketch. Upload the sketch to your Arduino wired as before.

Once uploaded, the Arduino should perform all the test display procedures! If you're not seeing anything - first check if you have the backlight on, if the backlight is not lit something is wrong with the power/backlight wiring. If the backlight is lit but you see nothing on the display make sure you're using our suggested wiring.
Adafruit GFX Library

We've written a full graphics library specifically for this display which will get you up and running quickly. The code is written in C/C++ for Arduino but is easy to port to any microcontroller by rewriting the low level pin access functions. Here are some of the functions we've included in the library.

The TFT LCD library is based off of the Adafruit GFX graphics core library. GFX has many ready to go functions that should help you start out with your project. Its not exhaustive and we'll try to update it if we find a really useful function. Right now it supports pixels, lines, rectangles, circles, round-rects, triangles and printing text as well as rotation.

Read more about it here! (https://adafru.it/aPx)

Bitmaps

In this example, we'll show how to display a 220x176 pixel full color bitmap from a microSD card.
We have an example sketch in the library showing how to display full color bitmap images stored on an SD card. You'll need a [microSD card such as this one](http://adafruit.it/102). You'll also need to be running Arduino 1.0 or later, as the SD library was updated.

You'll also need an image. [We suggest starting with this bitmap of a rose](https://adafruit.it/cmm). If you want to later use your own image, use an image editing tool and crop your image to no larger than 160 pixels high and 128 pixels wide. Save it as a 24-bit color BMP file - it must be 24-bit color format to work, even if it was originally a 16-bit color image - because of the way BMPs are stored and displayed!

Names for bitmap files must not exceed 8 characters with a 3 character extension. "mybitmap.bmp" is fine. "myotherbitmap.bmp" is too long and will not be readable by the SD file system.

Copy the rose.bmp to the microSD card and insert it into the back of the breakout board.
Wire up the TFT according to the high-speed SPI diagram above. Test that your wiring is correct by uploading the graphics test sketch with the high speed SPI line uncommented and the flexible-low-speed wiring commented.

Once you are sure that the TFT is wired correctly, add the two wires for talking to the SD card. Connect **CDCS** (the unconnected pin in the middle) to digital pin 4 (you can change this later to any pin you want) that's the orange wire below. Connect **MISO** (last unconnected pin) to the Arduino's hardware SPI **MISO** pin, that's the white wire below. For Classic arduinos, this is pin 12. For Mega's this is pin 50. You can't change the **MISO** pin, its fixed in the chip hardware.
Now load the **bitmap** example sketch into the Arduino. It should display the parrot image. If you have any problems, check the serial console for any messages such as not being able to initialize the microSD card or not finding the image.
Bitmaps

There is a built in microSD card slot into the breakout, and we can use that to load bitmap images! You will need a microSD card formatted **FAT16 or FAT32** (they almost always are by default).

It's really easy to draw bitmaps. We have a library for it, Adafruit ImageReader, which can be installed through the Arduino Library Manager (Sketch→Include Library→Manage Libraries...). Enter "imageread" in the search field and the library is easy to spot:

Lets start by downloading this image of pretty flowers (pix by johngineer)
Copy `purple.bmp` into the base directory of a microSD card and insert it into the microSD socket in the breakout.

You'll need to connect up the **SDCS** pin to **Digital 4** on your Arduino, and the **MISO** to **MISO** (or **Digital #12** on an Uno) as well. In the below image, those are the extra purple & light blue wires.

You may want to try the **SD library** examples before continuing, especially one that lists all the files on the SD card.

Now upload the **File→examples→Adafruit ImageReader Library→ShieldILI9341** example to your Arduino + breakout. You will see the flowers appear!
To make new bitmaps, make sure they are less than 240 by 320 pixels and save them in **24-bit BMP format**! They must be in 24-bit format, even if they are not 24-bit color as that is the easiest format for the Arduino. You can rotate images using the `setRotation()` procedure.

You can draw as many images as you want - don't forget the names must be less than 8 characters long. Just copy the BMP drawing routines below `loop()` and call

```cpp
bmpDraw(bmpfilename, x, y);
```

For each bitmap. They can be smaller than 320x240 and placed in any location on the screen.
CircuitPython Displayio Quickstart

You will need a board capable of running CircuitPython such as the Metro M0 Express or the Metro M4 Express. You can also use boards such as the Feather M0 Express or the Feather M4 Express. We recommend either the Metro M4 or the Feather M4 Express because it’s much faster and works better for driving a display. For this guide, we will be using a Feather M4 Express. The steps should be about the same for the Feather M0 Express or either of the Metros. If you haven't already, be sure to check out our Feather M4 Express guide (https://adafru.it/EEm).

Adafruit Feather M4 Express - Featuring ATSAMD51

It's what you've been waiting for, the Feather M4 Express featuring ATSAMD51. This Feather is fast like a swift, smart like an owl, strong like a ox-bird (it's half ox,...

For this guide, we'll assume you have a Feather M4 Express. The steps should be about the same for the Feather M0 Express. To start, if you haven't already done so, follow the assembly instructions for the Feather M4 Express in our Feather M4 Express guide (https://adafru.it/EEm).

Preparing the Breakout

Before using the TFT Breakout, you will need to solder the headers or some wires to it. Be sure to check out the Adafruit Guide To Excellent Soldering (https://adafru.it/drI). After that, the breakout should be ready to go.

Wiring the Breakout to the Feather

- 3-5V Vin connects to the Feather 3V pin
- GND connects to Feather ground
- CLK connects to SPI clock. On the Feather that's SCK.
- MISO connects to SPI MISO. On the Feather that's MI
- MOSI connects to SPI MOSI. On the Feather that's MO
- CS connects to our SPI Chip Select pin. We'll be using Digital 9 but you can later change this to any pin
- D/C connects to our SPI data/command select pin. We'll be using Digital 10 but you can later change this pin too.
- RST connects to our reset pin. We'll be using Digital 6 but you can later change this pin too.
Required CircuitPython Libraries

To use this display with `displayio`, there is only one required library.

First, make sure you are running the latest version of Adafruit CircuitPython (https://adafruit.it/Amd) for your board.

Next, you'll need to install the necessary libraries to use the hardware—carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafruit.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafruit.it/ABU) for both express and non-express boards.

Remember for non-express boards, you'll need to manually install the necessary libraries from the bundle:

- adafruit_ili9341

Before continuing make sure your board's lib folder or root filesystem has the adafruit_ili9341 file copied over.

Code Example Additional Libraries
For the Code Example, you will need an additional library. We decided to make use of a library so the code didn't get overly complicated.

Go ahead and install this in the same manner as the driver library by copying the adafruit_display_text folder over to the lib folder on your CircuitPython device.

CircuitPython Code Example
This test will initialize the display using displayio and draw a solid green background, a smaller purple rectangle, and some yellow text. All drawing is done using native displayio modules.

Pinouts are for the 2.4" TFT FeatherWing or Breakout with a Feather M4 or M0.

```python
# Release any resources currently in use for the displays
displayio.release_displays()

spi = board.SPI()
tft_cs = board.D9
tft_dc = board.D10
display_bus = displayio.FourWire(
    spi, command=tft_dc, chip_select=tft_cs, reset=board.D6
)
display = adafruit_ili9341.ILI9341(display_bus, width=320, height=240)

# Draw a green background
color_bitmap = displayio.Bitmap(320, 240, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00  # Bright Green
text_group = displayio.Group(max_size=10, scale=3, x=57, y=120)
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00)
text_group.append(text_area)  # Subgroup for text scaling
while True:
    pass
```
Code Details

Let's take a look at the sections of code one by one. We start by importing the board so that we can initialize `SPI`, `displayio`, `terminalio` for the font, a `label`, and the `adafruit_ili9341` driver.

```python
import board
import displayio
import terminalio
from adafruit_display_text import label
import adafruit_ili9341
```

Next we release any previously used displays. This is important because if the Feather is reset, the display pins are not automatically released and this makes them available for use again.

```python
displayio.release_displays()
```

Next, we set the SPI object to the board's SPI with the easy shortcut function `board.SPI()`. By using this function, it finds the SPI module and initializes using the default SPI parameters. Next we set the Chip Select and Data/Command pins that will be used.

```python
spi = board.SPI()
tft_cs = board.D9
tft_dc = board.D10
```

In the next line, we set the display bus to FourWire which makes use of the SPI bus.

```python
display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=board.D6)
```

Finally, we initialize the driver with a width of 320 and a height of 240. If we stopped at this point and ran the code, we would have a terminal that we could type at and have the screen update.

```python
display = adafruit_ili9341.ILI9341(display_bus, width=320, height=240)
```
Next we create a background splash image. We do this by creating a group that we can add elements to and adding that group to the display. In this example, we are limiting the maximum number of elements to 10, but this can be increased if you would like. The display will automatically handle updating the group.

```python
splash = displayio.Group(max_size=10)
display.show(splash)
```

Next we create a Bitmap which is like a canvas that we can draw on. In this case we are creating the Bitmap to be the same size as the screen, but only have one color. The Bitmaps can currently handle up to 256 different colors. We create a Palette with one color and set that color to 0x00FF00 which happens to be green. Colors are Hexadecimal values in the format of RRGGBB. Even though the Bitmaps can only handle 256 colors at a time, you get to define what those 256 different colors are.

```python
color_bitmap = displayio.Bitmap(320, 240, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00 # Bright Green
```

With all those pieces in place, we create a TileGrid by passing the bitmap and palette and draw it at (0, 0) which represents the display's upper left.

```python
bg_sprite = displayio.TileGrid(color_bitmap,
                                pixel_shader=color_palette,
                                x=0, y=0)
splash.append(bg_sprite)
```

This creates a solid green background which we will draw on top of.
Next we will create a smaller purple rectangle. The easiest way to do this is to create a new bitmap that is a little smaller than the full screen with a single color and place it in a specific location. In this case we will create a bitmap that is 20 pixels smaller on each side. The screen is 320x240, so we'll want to subtract 40 from each of those numbers.

We'll also want to place it at the position \((20, 20)\) so that it ends up centered.

```python
inner_bitmap = displayio.Bitmap(280, 200, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088 # Purple
inner_sprite = displayio.TileGrid(inner_bitmap,
                                pixel_shader=inner_palette,
                                x=20, y=20)
splash.append(inner_sprite)
```

Since we are adding this after the first rectangle, it's automatically drawn on top. Here's what it looks like now.
Next let's add a label that says "Hello World!" on top of that. We're going to use the built-in Terminal Font and scale it up by a factor of three. To scale the label only, we will make use of a subgroup, which we will then add to the main group.

Labels are centered vertically, so we'll place it at 120 for the Y coordinate, and around 57 pixels make it appear to be centered horizontally, but if you want to change the text, change this to whatever looks good to you. Let's go with some yellow text, so we'll pass it a value of 0xFFFF00.

```python
text_group = displayio.Group(max_size=10, scale=3, x=57, y=120)
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00)
text_group.append(text_area) # Subgroup for text scaling
splash.append(text_group)
```

Finally, we place an infinite loop at the end so that the graphics screen remains in place and isn't replaced by a terminal.

```python
while True:
    pass
```
Where to go from here

Be sure to check out this excellent guide to CircuitPython Display Support Using displayio (https://adafruit.it/EGh)
Python Wiring and Setup

Wiring

It's easy to use display breakouts with Python and the Adafruit CircuitPython RGB Display (https://adafru.it/u1C) module. This module allows you to easily write Python code to control the display.

We'll cover how to wire the display to your Raspberry Pi. First assemble your 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 display as shown below to your Raspberry Pi.

ILI9341 and HX-8357-based Displays

2.2" Display

- CLK connects to SPI clock. On the Raspberry Pi, that's SLCK
- MOSI connects to SPI MOSI. On the Raspberry Pi, that's also MOSI
- CS connects to our SPI Chip Select pin. We'll be using CEO
- D/C connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later.
- RST connects to our Reset pin. We'll be using GPIO 24 but this can be changed later as well.
- Vin connects to the Raspberry Pi's 3V pin
- GND connects to the Raspberry Pi's ground
2.4", 2.8", 3.2", and 3.5" Displays

These displays are set up to use the 8-bit data lines by default. We want to use them for SPI. To do that, you'll need to either solder bridge some pads on the back or connect the appropriate IM lines to 3.3V with jumper wires. Check the back of your display for the correct solder pads or IM lines to put it in SPI mode.

- **Vin** connects to the Raspberry Pi's 3V pin
- **GND** connects to the Raspberry Pi's ground
- **CLK** connects to SPI clock. On the Raspberry Pi, that's SLCK
- **MOSI** connects to SPI MOSI. On the Raspberry Pi, that's also MOSI
- **CS** connects to our SPI Chip Select pin. We'll be using CE0
- **D/C** connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later.
- **RST** connects to our Reset pin. We'll be using GPIO 24 but this can be changed later as well.

These larger displays are set to use 8-bit data lines by default and may need to be modified to use SPI.
ST7789 and ST7735-based Displays

1.3", 1.54", and 2.0" IPS TFT Display

- **Vin** connects to the Raspberry Pi's 3V pin
- **GND** connects to the Raspberry Pi's ground
- **CLK** connects to SPI clock. On the Raspberry Pi, that's **SLCK**
- **MOSI** connects to SPI MOSI. On the Raspberry Pi, that's also **MOSI**
- **CS** connects to our SPI Chip Select pin. We'll be using **CE0**
- **RST** connects to our Reset pin. We'll be using **GPIO 24** but this can be changed later.
- **D/C** connects to our SPI Chip Select pin. We'll be using **GPIO 25**, but this can be changed later as well.
0.96", 1.14", and 1.44" Displays

- **Vin** connects to the Raspberry Pi's **3V** pin
- **GND** connects to the Raspberry Pi's **ground**
- **CLK** connects to SPI clock. On the Raspberry Pi, that's **SLCK**
- **MOSI** connects to SPI MOSI. On the Raspberry Pi, that's also **MOSI**
- **CS** connects to our SPI Chip Select pin. We'll be using **CE0**
- **RST** connects to our Reset pin. We'll be using **GPIO 24** but this can be changed later.
- **D/C** connects to our SPI Chip Select pin. We'll be using **GPIO 25**, but this can be changed later as well.
1.8" Display

- GND connects to the Raspberry Pi's ground
- Vin connects to the Raspberry Pi's 3V pin
- RST connects to our Reset pin. We'll be using GPIO 24 but this can be changed later.
- D/C connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later as well.
- CS connects to our SPI Chip Select pin. We'll be using CE0
- MOSI connects to SPI MOSI. On the Raspberry Pi, that's also MOSI
- CLK connects to SPI clock. On the Raspberry Pi, that's SLCK
- LITE connects to the Raspberry Pi's 3V pin. This can be used to separately control the backlight.
SSD1351-based Displays

1.27" and 1.5" OLED Displays

- GND connects to the Raspberry Pi's ground
- Vin connects to the Raspberry Pi's 3V pin
- CLK connects to SPI clock. On the Raspberry Pi, that's SLCK
- MOSI connects to SPI MOSI. On the Raspberry Pi, that's also MOSI
- CS connects to our SPI Chip Select pin. We'll be using CE0
- RST connects to our Reset pin. We'll be using GPIO 24 but this can be changed later.
- D/C connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later as well.
SSD1331-based Display

0.96" OLED Display

- **MOSI** connects to SPI MOSI. On the Raspberry Pi, thats also **MOSI**
- **CLK** connects to SPI clock. On the Raspberry Pi, thats **SLCK**
- **D/C** connects to our SPI Chip Select pin. We'll be using **GPIO 25**, but this can be changed later.
- **RST** connects to our Reset pin. We'll be using **GPIO 24** but this can be changed later as well.
- **CS** connects to our SPI Chip Select pin. We'll be using **CE0**
- **Vin** connects to the Raspberry Pi's **3V** pin
- **GND** connects to the Raspberry Pi's **ground**
Setup

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling SPI 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)!

If you have previously installed the Kernel Driver with the PiTFT Easy Setup, you will need to remove it first in order to run this example.

Python Installation of RGB Display Library

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

- `sudo pip3 install adafruit-circuitpython-rgb-display`

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`

**DejaVu TTF Font**

Raspberry Pi usually comes with the DejaVu font already installed, but in case it didn't, you can run the following to install it:

- `sudo apt-get install ttf-dejavu`

**Pillow Library**

We also need PIL, the Python Imaging Library, to allow graphics and 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.
Python Usage

Now that you have everything setup, we're going to look over three different examples. For the first, we'll take a look at automatically scaling and cropping an image and then centering it on the display.

Turning on the Backlight

On some displays, the backlight is controlled by a separate pin such as the 1.3" TFT Bonnet with Joystick. On such displays, running the below code will likely result in the display remaining black. To turn on the backlight, you will need to add a small snippet of code. If your backlight pin number differs, be sure to change it in the code:

```python
# Turn on the Backlight
backlight = DigitalInOut(board.D26)
backlight.switch_to_output()
backlight.value = True
```

Displaying an Image

Here's the full code to the example. We will go through it section by section to help you better understand what is going on. Let's start by downloading an image of Blinka. This image has enough border to allow resizing and cropping with a variety of display sizes and rations to still look good.

Make sure you save it as `blinka.jpg` and place it in the same folder as your script. Here's the code we'll be
loading onto the Raspberry Pi. We'll go over the interesting parts.

```python
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

---

Be sure to check the learn guides for more usage information.

This example is for use on (Linux) computers that are using CPython with Adafruit Blinka to support CircuitPython libraries. CircuitPython does not support PIL/pillow (python imaging library)!

Author(s): Melissa LeBlanc-Williams for Adafruit Industries
---

import digitalio
import board
from PIL import Image, ImageDraw
import adafruit_rgb_display.ili9341 as ili9341
import adafruit_rgb_display.st7789 as st7789  # pylint: disable=unused-import
import adafruit_rgb_display.hx8357 as hx8357  # pylint: disable=unused-import
import adafruit_rgb_display.st7735 as st7735  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1351 as ssd1351  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1331 as ssd1331  # pylint: disable=unused-import

# Configuration for CS and DC pins (these are PiTFT defaults):
# disp = st7789.ST7789(spi, rotation=90,     # 2.0" ST7789
# disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,  # 1.3", 1.54" ST7789
# disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53, y_offset=40, # 1.14" ST7789
# disp = hx8357.HX8357(spi, rotation=180,     # 3.5" HX8357
# disp = st7735.ST7735R(spi, rotation=90,     # 1.8" ST7735R
# disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3, # 1.44" ST7735R
# disp = st7735.ST7735R(spi, rotation=90, bgr=True,     # 0.96" MiniTFT ST7735R
# disp = ssd1351.SSD1351(spi, rotation=180,    # 1.5" SSD1351
# disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
# disp = ssd1331.SSD1331(spi, rotation=180,    # 0.96" SSD1331
# disp = ili9341.ILI9341(  
#     spi, 
#     rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
#     cs=cs_pin,
#     dc=dc_pin,
#     rst=reset_pin,
#     baudrate=BAUDRATE,
# )
# -pylint: enable=line-too-long

# Create blank image for drawing.
```
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width # we swap height/width to rotate it to landscape!
    height = disp.height
image = Image.new("RGB", (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0))

disp.image(image)

image = Image.open("blinka.jpg")

# Scale the image to the smaller screen dimension
image_ratio = image.width / image.height
screen_ratio = width / height
if screen_ratio < image_ratio:
    scaled_width = image.width * height // image.height
    scaled_height = height
else:
    scaled_width = width
    scaled_height = image.height * width // image.width
image = image.resize((scaled_width, scaled_height), Image.BICUBIC)

# Crop and center the image
x = scaled_width // 2 - width // 2
y = scaled_height // 2 - height // 2
image = image.crop((x, y, x + width, y + height))

# Display image.
disp.image(image)

So we start with our usual imports including a couple of Pillow modules and the display drivers. That is followed by defining a few pins here. The reason we chose these is because they allow you to use the same code with the PiTFT if you chose to do so.

import digitalio
import board
from PIL import Image, ImageDraw
import adafruit_rgb_display.ili9341 as ili9341
import adafruit_rgb_display.st7789 as st7789
import adafruit_rgb_display.hx8357 as hx8357
import adafruit_rgb_display.st7735 as st7735
import adafruit_rgb_display.ssd1351 as ssd1351
import adafruit_rgb_display.ssd1331 as ssd1331

# Configuration for CS and DC pins
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)
Next we'll set the baud rate from the default 24 MHz so that it works on a variety of displays. The exception to this is the SSD1351 driver, which will automatically limit it to 16MHz even if you pass 24MHz. We'll set up out SPI bus and then initialize the display.

We wanted to make these examples work on as many displays as possible with very few changes. The ILI9341 display is selected by default. For other displays, go ahead and comment out the line that starts with:

```python
disp = ili9341.ILI9341(spi,
```

and uncomment the line appropriate for your display. The displays have a rotation property so that it can be set in just one place.

```python
# Config for display baudrate (default max is 24mhz):
BAUDRATE = 24000000

# Setup SPI bus using hardware SPI:
spi = board.SPI()

#disp = st7789.ST7789(spi, rotation=90,                            # 2.0" ST7789
#disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,  # 1.3", 1.54" ST7789
#disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53, y_offset=40, # 1.14" ST7789
#disp = hx8357.HX8357(spi, rotation=180,                            # 3.5" HX8357
#disp = st7735.ST7735R(spi, rotation=90,                             # 1.8" ST7735R
#disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   # 1.44" ST7735R
#disp = st7735.ST7735R(spi, rotation=90, bgr=True,                             # 0.96" MiniTFT ST7735R
#disp = ssd1351.SSD1351(spi, rotation=90,                             # 1.5" SSD1351
#disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
#disp = ssd1331.SSD1331(spi, rotation=180,                             # 0.96" SSD1331

disp = ili9341.ILI9341(spi, rotation=90,                            # 2.2", 2.4", 2.8", 3.2" ILI9341
    cs=cs_pin, dc=dc_pin, rst=reset_pin, baudrate=BAUDRATE)
```

Next we read the current rotation setting of the display and if it is 90 or 270 degrees, we need to swap the width and height for our calculations, otherwise we just grab the width and height. We will create an `image` with our dimensions and use that to create a `draw` object. The `draw` object will have all of our drawing functions.

```python
# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width  # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    height = disp.width  # we swap height/width to rotate it to landscape!
    width = disp.height

image = Image.new('RGB', (width, height))

dl = ImageDraw.Draw(image)
```

© Adafruit Industries  https://learn.adafruit.com/2-2-tft-display  Page 43 of 57
Next we clear whatever is on the screen by drawing a black rectangle. This isn't strictly necessary since it will be overwritten by the image, but it kind of sets the stage.

```python
# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0))
disp.image(image)
```

Next we open the Blinka image, which we've named `blinka.jpg`, which assumes it is in the same directory that you are running the script from. Feel free to change it if it doesn't match your configuration.

```python
image = Image.open("blinka.jpg")
```

Here's where it starts to get interesting. We want to scale the image so that it matches either the width or height of the display, depending on which is smaller, so that we have some of the image to chop off when we crop it. So we start by calculating the width to height ration of both the display and the image. If the height is the closer of the dimensions, we want to match the image height to the display height and let it be a bit wider than the display. Otherwise, we want to do the opposite.

Once we've figured out how we're going to scale it, we pass in the new dimensions and using a Bicubic rescaling method, we reassign the newly rescaled image back to `image`. Pillow has quite a few different methods to choose from, but Bicubic does a great job and is reasonably fast.

```python
# Scale the image to the smaller screen dimension
image_ratio = image.width / image.height
screen_ratio = width / height
if screen_ratio < image_ratio:
    scaled_width = image.width * height // image.height
    scaled_height = height
else:
    scaled_width = width
    scaled_height = image.height * width // image.width
image = image.resize((scaled_width, scaled_height), Image.BICUBIC)
```

Next we want to figure the starting x and y points of the image where we want to begin cropping it so that it ends up centered. We do that by using a standard centering function, which is basically requesting the difference of the center of the display and the center of the image. Just like with scaling, we replace the `image` variable with the newly cropped image.

```python
# Crop and center the image
x = scaled_width // 2 - width // 2
y = scaled_height // 2 - height // 2
image = image.crop((x, y, x + width, y + height))
```

Finally, we take our image and display it. At this point, the image should have the exact same dimensions at the display and fill it completely.
Drawing Shapes and Text

In the next example, we'll take a look at drawing shapes and text. This is very similar to the displayio example, but it uses Pillow instead. Here's the code for that.

```python
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
import adafruit_rgb_display.ili9341 as ili9341
import adafruit_rgb_display.st7789 as st7789  # pylint: disable=unused-import
import adafruit_rgb_display.hx8357 as hx8357  # pylint: disable=unused-import
import adafruit_rgb_display.st7735 as st7735  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1351 as ssd1351  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1331 as ssd1331  # pylint: disable=unused-import

# First define some constants to allow easy resizing of shapes.
```

This demo will draw a few rectangles onto the screen along with some text on top of that.

This example is for use on (Linux) computers that are using CPython with Adafruit Blinka to support CircuitPython libraries. CircuitPython does not support PIL/pillow (python imaging library)!

Author(s): Melissa LeBlanc-Williams for Adafruit Industries

---
# First define some constants to allow easy resizing of shapes.
BORDER = 20
FONTSIZE = 24

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)

# Config for display baudrate (default max is 24mhz):
BAUDRATE = 24000000

# Setup SPI bus using hardware SPI:
spi = board.SPI()

# pylint: disable=line-too-long
# Create the display:
# disp = st7789.ST7789(spi, rotation=90, # 2.0" ST7789
# disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180, # 1.3", 1.54" ST7789
# disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53, y_offset=40, # 1.14" ST7789
# disp = hx8357.HX8357(spi, rotation=180, # 3.5" HX8357
# disp = st7735.ST7735R(spi, rotation=90, # 1.8" ST7735R
# disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3, # 1.44" ST7735R
# disp = st7735.ST7735R(spi, rotation=90, bgr=True, # 0.96" MiniTFT ST7735R
# disp = ssd1351.SSD1351(spi, rotation=180, # 1.5" SSD1351
# disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
# disp = ssd1331.SSD1331(spi, rotation=180, # 0.96" SSD1331
# disp = ili9341.ILI9341(spi, rotation=90, # 2.2", 2.4", 2.8", 3.2" ILI9341

cs=cs_pin,
dc=dc_pin,
rst=reset_pin,
baudrate=BAUDRATE,
)
# pylint: enable=line-too-long

# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width # we swap height/width to rotate it to landscape!
    height = disp.height

image = Image.new("RGB", (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a green filled box as the background
draw.rectangle((0, 0, width, height), fill=(0, 255, 0))
disp.image(image)

# Draw a smaller inner purple rectangle
draw.rectangle((BORDER, BORDER, width - BORDER - 1, height - BORDER - 1), fill=(170, 0, 136))

# Load a TTF Font
font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", FONTSIZE)

# Draw Some Text
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
draw.text(
    (width // 2 - font_width // 2, height // 2 - font_height // 2),
    text,
    font=font,
    fill=(255, 255, 0),
)

# Display image.
disp.image(image)

Just like in the last example, we'll do our imports, but this time we're including the Pillow module because we'll be drawing some text this time.

import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
import adafruit_rgb_display.ili9341 as ili9341

Next we'll define some parameters that we can tweak for various displays. The BORDER will be the size in pixels of the green border between the edge of the display and the inner purple rectangle. The FONTSIZE will be the size of the font in points so that we can adjust it easily for different displays.

BORDER = 20
FONTSIZE = 24

Next, just like in the previous example, we will set up the display, setup the rotation, and create a draw object. If you have are using a different display than the ILI9341, go ahead and adjust your initializer as explained in the previous example. After that, we will setup the background with a green rectangle that takes up the full screen. To get green, we pass in a tuple that has our Red, Green, and Blue color values in it in that order which can be any integer from 0 to 255.

draw.rectangle((0, 0, width, height), fill=(0, 255, 0))
disp.image(image)

Next we will draw an inner purple rectangle. This is the same color value as our example in displayio quickstart, except the hexadecimal values have been converted to decimal. We use the BORDER parameter to calculate the size and position that we want to draw the rectangle.

draw.rectangle((BORDER, BORDER, width - BORDER - 1, height - BORDER - 1),
    fill=(170, 0, 136))
Next we'll load a TTF font. The `DejaVuSans.ttf` font should come preloaded on your Pi in the location in the code. We also make use of the `FONTSIZE` parameter that we discussed earlier.

```python
# Load a TTF Font
font = ImageFont.truetype('/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf', FONTSIZE)
```

Now we draw the text Hello World onto the center of the display. You may recognize the centering calculation was the same one we used to center crop the image in the previous example. In this example though, we get the font size values using the `getsize()` function of the font object.

```python
# Draw Some Text
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
draw.text((width//2 - font_width//2, height//2 - font_height//2),
          text, font=font, fill=(255, 255, 0))
```

Finally, just like before, we display the image.

```python
disp.image(image)
```

Displaying System Information

In this last example we'll take a look at getting the system information and displaying it. This can be very handy for system monitoring. Here's the code for that example:
This will show some Linux Statistics on the attached display. Be sure to adjust to the display you have connected. Be sure to check the learn guides for more usage information.

This example is for use on (Linux) computers that are using CPython with Adafruit Blinka to support CircuitPython libraries. CircuitPython does not support PIL/pillow (python imaging library)!

```python
import time
import subprocess
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
import adafruit_rgb_display.ili9341 as ili9341
import adafruit_rgb_display.st7789 as st7789  # pylint: disable=unused-import
import adafruit_rgb_display.hx8357 as hx8357  # pylint: disable=unused-import
import adafruit_rgb_display.st7735 as st7735  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1351 as ssd1351  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1331 as ssd1331  # pylint: disable=unused-import

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)
# Config for display baudrate (default max is 24mhz):
BAUDRATE = 24000000
# Setup SPI bus using hardware SPI:
spi = board.SPI()
# pylint: disable=line-too-long
# Create the display:
# disp = st7789.ST7789(spi, rotation=90,                            # 2.0" ST7789
# disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180, # 1.3", 1.54" ST7789
# disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53, y_offset=40, # 1.14" ST7789
# disp = hx8357.HX8357(spi, rotation=180,                           # 3.5" HX8357
# disp = st7735.ST7735R(spi, rotation=90,                           # 1.8" ST7735R
# disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   # 1.44" ST7735R
# disp = st7735.ST7735R(spi, rotation=90, bgr=True,                 # 0.96" MiniTFT ST7735R
# disp = ssd1351.SSD1351(spi, rotation=180,                         # 1.5" SSD1351
# disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
# disp = ssd1331.SSD1331(spi, rotation=180,                         # 0.96" SSD1331
# disp = ili9341.ILI9341(
# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
```
if disp.rotation % 180 == 0:
    image = Image.new("RGB", (disp.width, disp.height), (0, 0, 0))
else:
    image = Image.new("RGB", (disp.width, disp.height), (0, 0, 0))
```
```python
draw = ImageDraw.Draw(image)
font = ImageFont.truetype("arial", 16)
```
```python
```
```python
```
```python
```
```python
```
```python
```
```python
```
if disp.rotation % 180 == 90:
    height = disp.width  # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width  # we swap height/width to rotate it to landscape!
    height = disp.height
image = Image.new("RGB", (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0))
disp.image(image)

# First define some constants to allow easy positioning of text.
padding = -2
x = 0

# Load a TTF font. Make sure the .ttf font file is in the
# same directory as the python script!
# Some other nice fonts to try: http://www.dafont.com/bitmap.php
font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 24)

while True:
    # Draw a black filled box to clear the image.
    draw.rectangle((0, 0, width, height), outline=0, fill=0)

    # Shell scripts for system monitoring from here:
    # https://unix.stackexchange.com/questions/119126/command-to-display-memory-usage-disk-usage-and-
cpu-load
    cmd = "hostname -I | cut -d' ' -f1"
    IP = "IP: " + subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "top -bn1 | grep load | awk '{printf \"CPU Load: %.2f\n\", $(NF-2)}'"
    CPU = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "free -m | awk \"NR==2{printf \"Mem: %s/%s MB %.2f\%
\", $3,$2,$3*100/$2 }\""
    MemUsage = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "df -h | awk \"\$NF==/\{printf \"Disk: %d/%d GB \%s\n\", $3,$2,$5\}\""
    Disk = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "cat /sys/class/thermal/thermal_zone0/temp \ awk \'printf \"CPU Temp: %.1f C\n\", $(NF-0) / 1000\"'"  # pylint: disable=line-too-long
    Temp = subprocess.check_output(cmd, shell=True).decode("utf-8")

    # Write four lines of text.
    y = padding
draw.text((x, y), IP, font=font, fill="#FFFFFF")
y += font.getsize(IP)[1]
draw.text((x, y), CPU, font=font, fill="#FF0000")
y += font.getsize(CPU)[1]
draw.text((x, y), MemUsage, font=font, fill="#00FF00")
y += font.getsize(MemUsage)[1]
draw.text((x, y), Disk, font=font, fill="#0000FF")
y += font.getsize(Disk)[1]
draw.text((x, y), Temp, font=font, fill="#FF00FF")

disp.image(image)
time.sleep(0.1)
Just like the last example, we'll start by importing everything we imported, but we're adding two more imports. The first one is `time` so that we can add a small delay and the other is `subprocess` so we can gather some system information.

```python
import time
import subprocess
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
import adafruit_rgb_display.ili9341 as ili9341
```

Next, just like in the first two examples, we will set up the display, setup the rotation, and create a draw object. **If you have are using a different display than the ILI9341, go ahead and adjust your initializer as explained in the previous example.**

Just like in the first example, we're going to draw a black rectangle to fill up the screen. After that, we're going to set up a couple of constants to help with positioning text. The first is the `padding` and that will be the Y-position of the top-most text and the other is `x` which is the X-Position and represents the left side of the text.

```python
# First define some constants to allow easy positioning of text.
padding = 2
x = 0
```

Next, we load a font just like in the second example.

```python
font = ImageFont.truetype('/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf', 24)
```

Now we get to the main loop and by using `while True:`, it will loop until `Control+C` is pressed on the keyboard. The first item inside here, we clear the screen, but notice that instead of giving it a tuple like before, we can just pass `0` and it will draw black.

```python
draw.rectangle((0, 0, width, height), outline=0, fill=0)
```

Next, we run a few scripts using the `subprocess` function that get called to the Operating System to get information. The in each command is passed through awk in order to be formatted better for the display. By having the OS do the work, we don't have to. These little scripts came from [https://unix.stackexchange.com/questions/119126/command-to-display-memory-usage-disk-usage-and-cpu-load](https://unix.stackexchange.com/questions/119126/command-to-display-memory-usage-disk-usage-and-cpu-load)
cmd = "hostname -I | cut -d' ' -f1"
IP = "IP: "+subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "top -bn1 | grep load | awk '{printf "CPU Load: %.2f\", $(NF-2)}'"
CPU = subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "free -m | awk 'NR==2{printf "Mem: %s/%s MB %.2f\", $3,$2,$3*100/$2 }'"
MemUsage = subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "df -h | awk '$NF="/"{printf "Disk: %d/%d GB %.2\", $3,$2,$3}"
Disk = subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "cat /sys/class/thermal/thermal_zone0/temp | awk '{printf "CPU Temp: %.1f C", $(NF-0) / 1000}"
Temp = subprocess.check_output(cmd, shell=True).decode("utf-8")

Now we display the information for the user. Here we use yet another way to pass color information. We can pass it as a color string using the pound symbol, just like we would with HTML. With each line, we take the height of the line using `getsize()` and move the pointer down by that much.

```python
y = padding
draw.text((x, y), IP, font=font, fill="#FFFFFF")
y += font.getsize(IP)[1]
draw.text((x, y), CPU, font=font, fill="#FFFF00")
y += font.getsize(CPU)[1]
draw.text((x, y), MemUsage, font=font, fill="#00FF00")
y += font.getsize(MemUsage)[1]
draw.text((x, y), Disk, font=font, fill="#0000FF")
y += font.getsize(Disk)[1]
draw.text((x, y), Temp, font=font, fill="#FF00FF")
```

Finally, we write all the information out to the display using `disp.image()`. Since we are looping, we tell Python to sleep for 0.1 seconds so that the CPU never gets too busy.

```python
disp.image(image)
time.sleep(.1)
```
IP: 192.168.11.33
CPU Load: 0.08
Mem: 153/3854 MB 3.97%
Disk: 9/29 GB 33%
CPU Temp: 36.0 C
Troubleshooting

Display does not work on initial power but does work after a reset.

The display driver circuit needs a small amount of time to be ready after initial power. If your code tries to write to the display too soon, it may not be ready. It will work on reset since that typically does not cycle power. If you are having this issue, try adding a small amount of delay before trying to write to the display.

In Arduino, use `delay()` to add a few milliseconds before calling `tft.begin()`. Adjust the amount of delay as needed to see how little you can get away with for your specific setup.
Downloads

Files:

- Adafruit Fritzing Library (https://adafru.it/aP3)
- ILI9340 (datasheet) (https://adafru.it/CbV) controller with built in pixel-addressable video RAM buffer
- Display datasheet (https://adafru.it/CbW)
- EagleCAD files on GitHub (https://adafru.it/CbX)