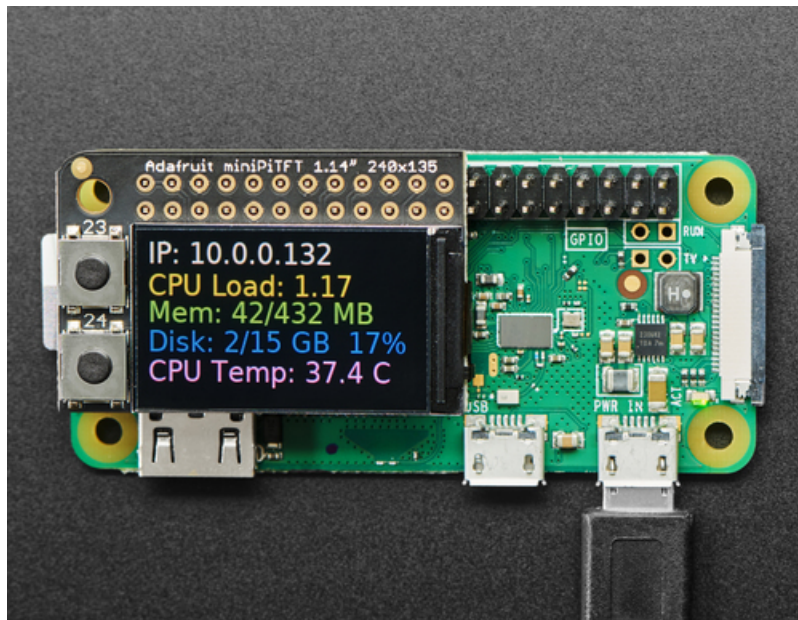


Adafruit Mini PiTFT - Color TFT Add-ons for Raspberry Pi

Created by lady ada



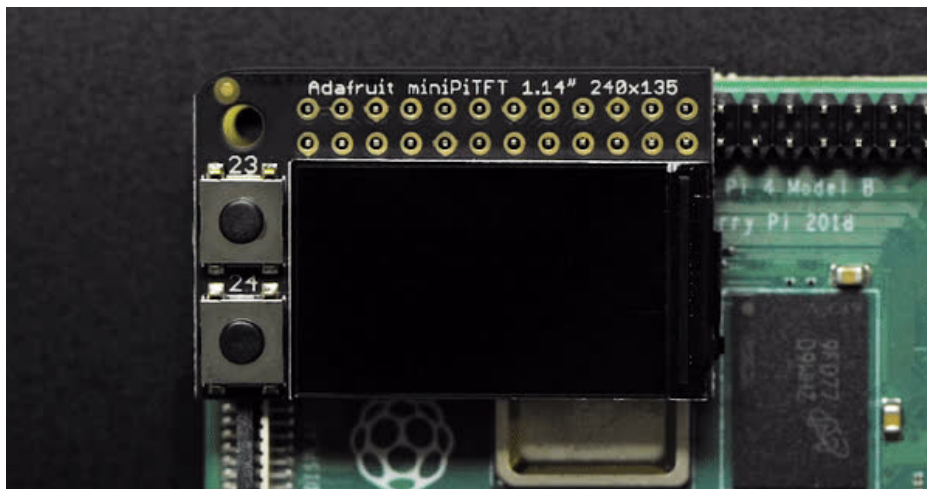
Last updated on 2021-03-03 01:32:55 PM EST

Guide Contents

Guide Contents	2
Overview	3
Pinouts	6
1.14" 240x135 Kernel Module Install	8
Prepare the Pi!	8
1.3" 240x240 Kernel Module Install	12
Prepare the Pi!	12
Kernel Module Troubleshooting	15
BrainCraft Audio Driver Reinstall	16
Python Setup	17
Attaching	17
Setup	18
Python Installation of RGB Display Library	19
DejaVu TTF Font	19
Pillow Library	19
NumPy Library	19
Quickstart Button Test	19
Python Stats Example	21
Modifications for the 1.3" Display	23
Running Stats on Boot	24
Troubleshooting Stats on Boot	24
Python Usage	25
Turning on the Backlight	25
Displaying an Image	25
Drawing Shapes and Text	29
Displaying System Information	33
Downloads	38
Files	38
Datasheets	38
Schematic	38
Schematic and Fab print for 1.3" MiniTFT	38

Overview

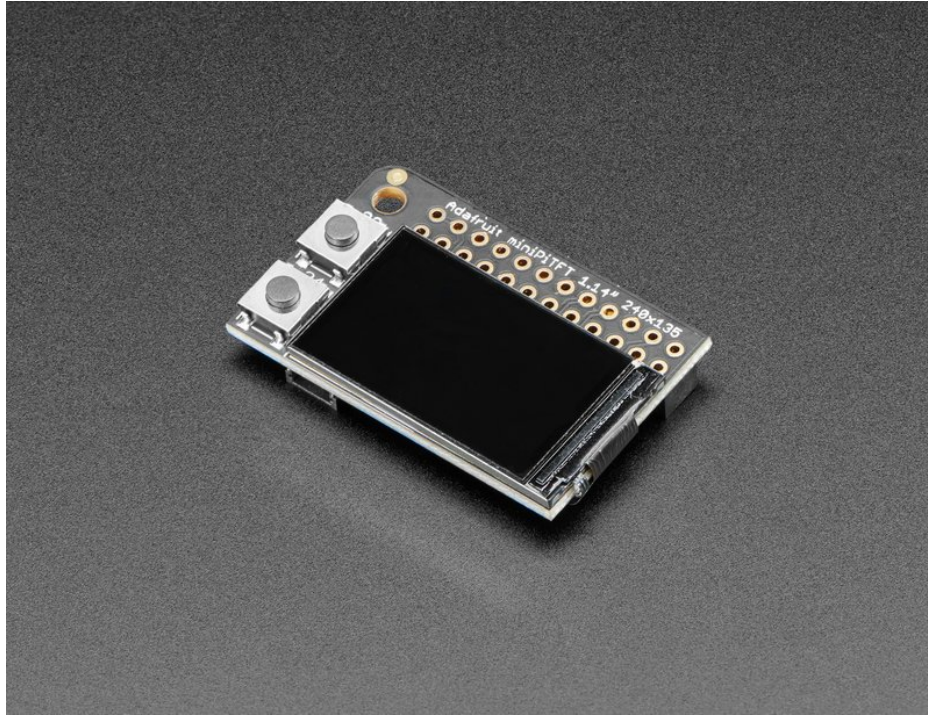
If you're looking for the most compact li'l color display for a [Raspberry Pi \(https://adafru.it/wF8\)](https://adafru.it/wF8) (most likely a [Pi Zero \(https://adafru.it/vla\)](https://adafru.it/vla)) project, this might be just the thing you need!



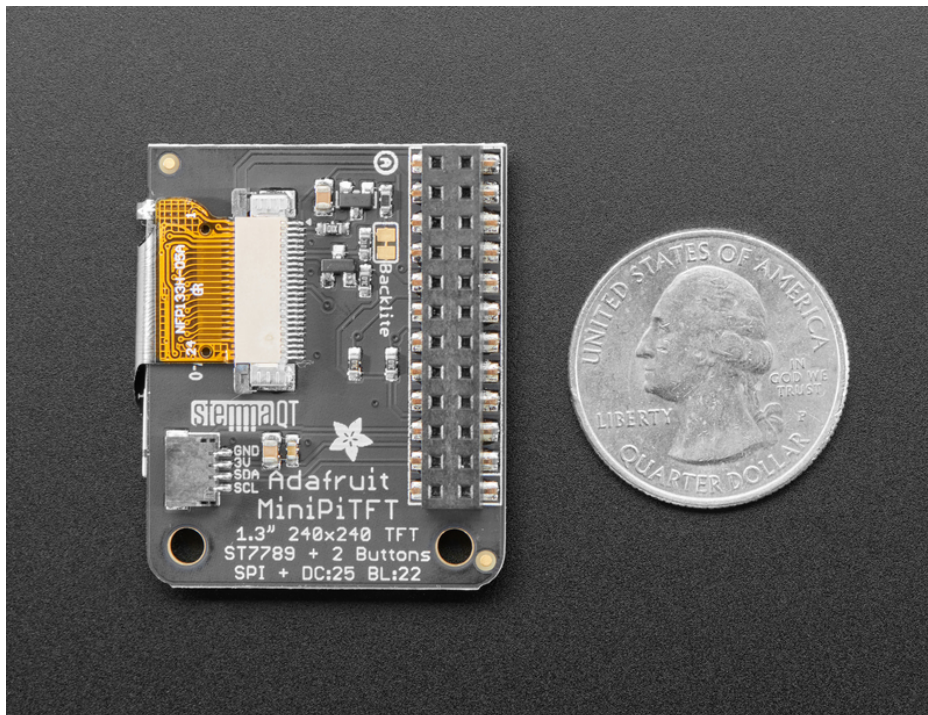
The **Adafruit Mini PiTFT - 135x240 Color TFT Add-on for Raspberry Pi** is your little TFT pal, ready to snap onto any and all Raspberry Pi computers, to give you a little display. The Mini PiTFT comes with a full color 240x135 pixel IPS display with great visibility at all angles. The TFT uses only the SPI port so its very fast, and we leave plenty of pins remaining available for buttons, LEDs, sensors, etc. It's also nice and compact so it will fit into any case.



The **Adafruit Mini PiTFT - 240x240 Color TFT Add-on for Raspberry Pi** is a bit larger, 1.3" diagonal and has 240x240 pixels instead of 240x135

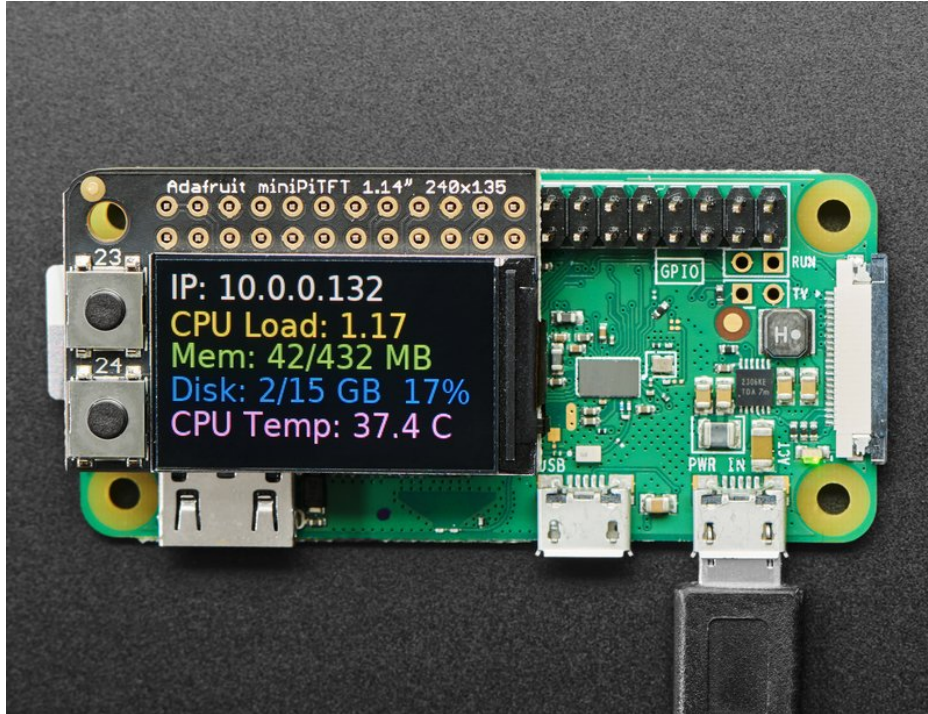


These display are super small, only about 1.14" or 1.3" diagonal, but since they are IPS displays, both are very readable with high contrast and visibility. We had a little space on the PCB so we give you two tactile buttons on GPIO pins so you can create a simple user interface. [On the bottom we have a Qwiic/STEMMA QT connector for I2C sensors and device so you can plug and play any of our STEMMA QT devices \(<https://adafru.it/GfR>\).](#)

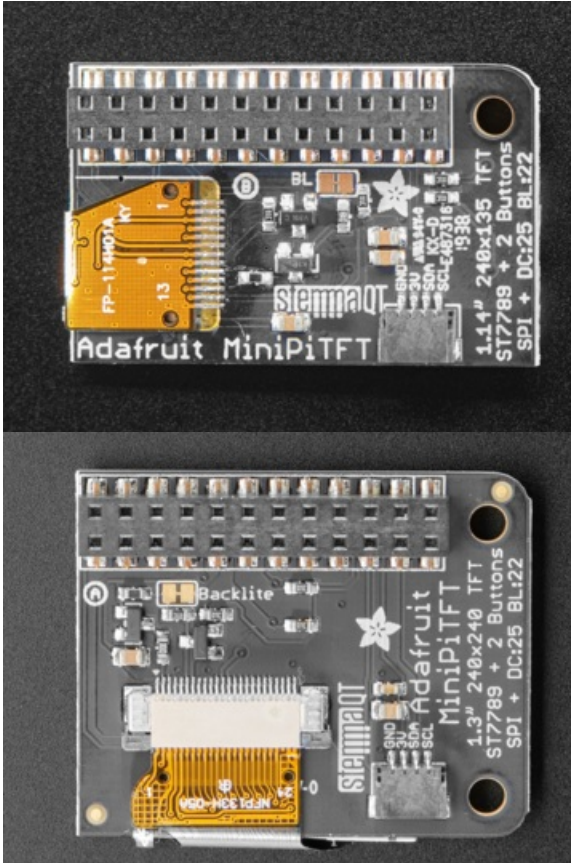


Using the display is very easy, we have a kernel driver and Python library for the ST7789 chipset. You can set it up as a console output so you can have text and user interface through the Raspberry Pi OS *or* you draw images, text, whatever you like, using the Python imaging library. Our tests showed ~15 FPS update rates so you can do animations or simple video.

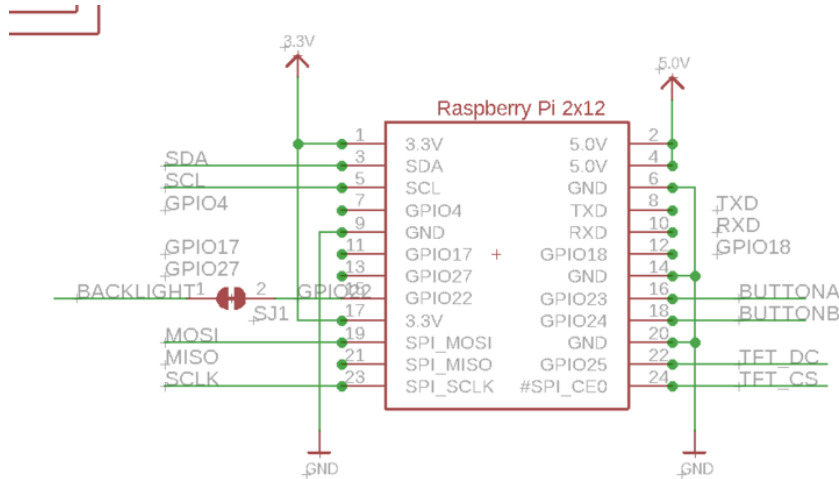
Comes completely pre-assembled and tested so you don't need to do anything but plug it in and install our Python code! Works with any Raspberry Pi computer.



Pinouts



Both the 1.3" and 1.14" versions of the Mini PiTFT have the same 2x12 connector and pinouts.



The mini PiTFT connects to the 'top' 2x12 headers on the Pi's 2x20 header connection. It uses the following pins:

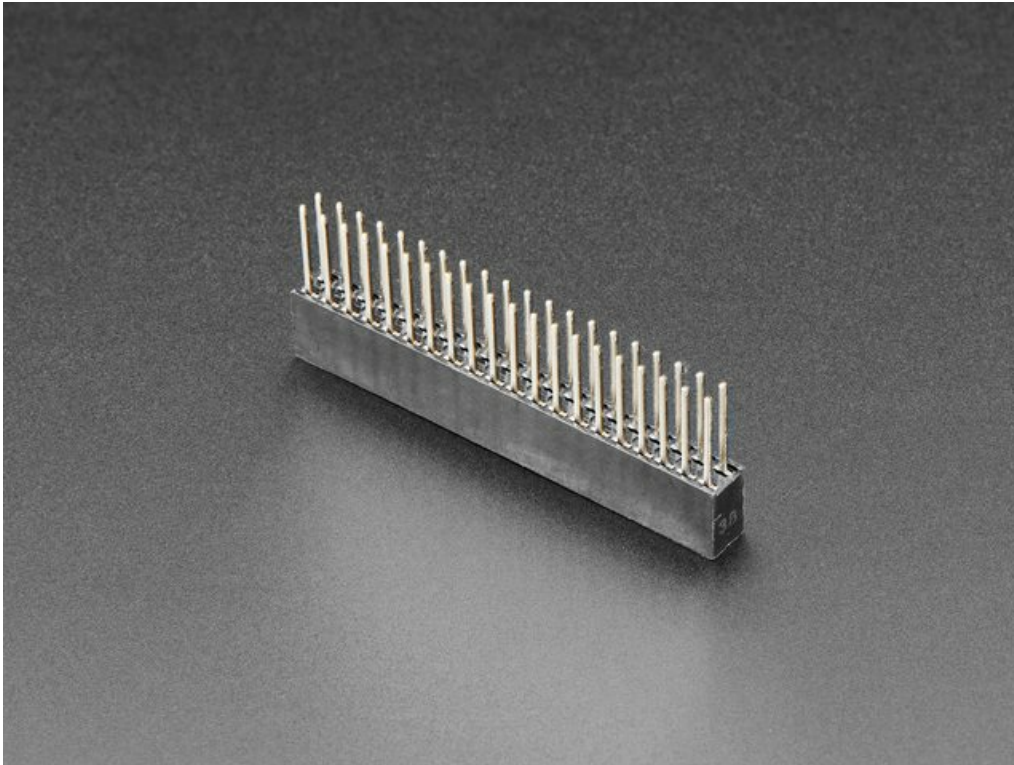
- **5.0V** - Connected to the display backlight
- **3.3V** - Connected to the display power and also the STEMMA QT / Qwiic connector
- **GND** - Ground for everything
- **SDA & SCL** - I2C data for the STEMMA QT / Qwiic connector. Not used by buttons or display
- **GPIO22** - Used to turn the backlight on and off. If you never want to turn the backlight off, cut the

small jumper on the bottom of the PiTFT to free up GPIO22

- **GPIO23 & GPIO24** - Connected to the two front buttons. These pins have 10K pullups to 3.3V so when the button is pressed, you will read a LOW voltage on these pins
- **SCK, MOSI, CE0 & GPIO25** - These are the display control pins. Note that MISO is not connected even though it is a SPI pin because you cannot read back from the display.

Not used: **GPIO4, GPIO17, GPIO18, GPIO27**

If you are using the 240x135 1.14" (small rectangular) Mini PiTFT , you can attach other hardware to the Pi on those pins if you use a stacking header - it will go through the 2x12 connector on the Mini PiTFT. This wont work on the 1.3" because the holes don't go through the PCB



[GPIO Stacking Header for Pi A+/B+/Pi 2/Pi 3](#)

Connect your own PCB to a Raspberry Pi B+ and stack on top with this normal-height female header with extra long pins. The female header part is about 8.5mm tall, good for small...

Out of Stock

Out of
Stock

1.14" 240x135 Kernel Module Install

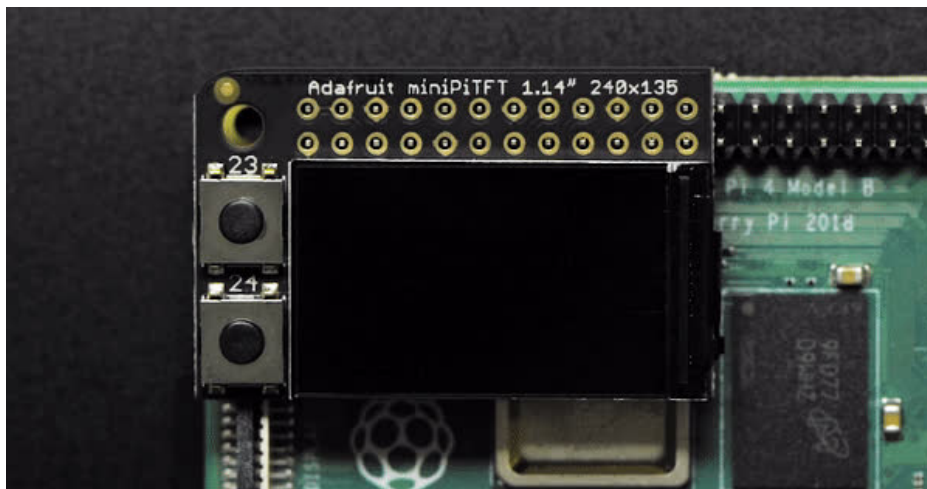
The 240x135 Mini PiTFT is so small, its not a default-supported resolution for small TFTs. This technique will update your kernel to the latest, and if you upgrade your Raspberry Pi which replaces the kernel you'll need to re-run the instructions! You'll also need to re-run if you change from a Pi Zero / Pi 2 / Pi 3 / Pi 4 as these all use different kernel types.

There's two ways you can use the 240x135 display.

Be aware that you can only choose to do one way at a time. If you choose the hard way, it will install the kernel driver, which will prevent you from doing it the easy way.

The easy way is to use 'pure Python 3' and Pillow library to draw to the display from within Python. This is great for showing text, stats, images etc that you design yourself. If you want to do that, skip this page and go to the Python install/usage page

The hard way is to install a kernel module to add support for the TFT display that will make the console appear on the display. This is cute because you can have any program print text or draw to the framebuffer (or, say, with pygame) and Linux will take care of displaying it for you. If you don't need the console or direct framebuffer access, please consider using the 'pure Python' technique instead as it is not as delicate.



You will not get a GUI/LXDE display, this is only for text console usage. The display is waaay too small for LXDE

Prepare the Pi!

Before you begin, its a good idea to get your Pi completely updated and upgraded. We assume you have burned an SD card and can log into the console to install stuff.

Run


```
sudo apt update -y
sudo apt-get update -y
sudo apt-get upgrade -y
```

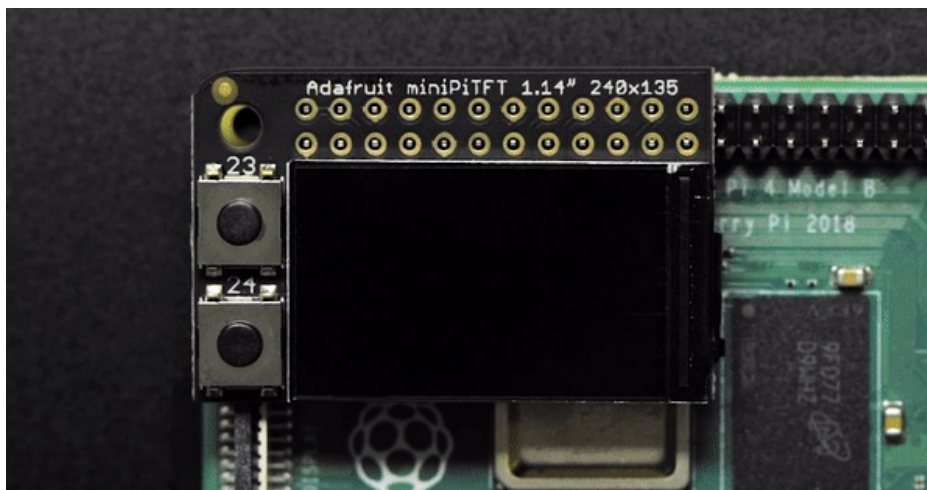
To fully update and upgrade your Pi!

```
pi@raspberrypi:~ $ sudo apt-get upgrade -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages have been kept back:
  raspi-config
The following packages will be upgraded:
  e2fsprogs firmware-atheros firmware-brcm80211 firmware-libertas
  firmware-misc-nonfree firmware-realtek libcom-err2 libext2fs2
  libraspberrypi-bin libraspberrypi-dev libraspberrypi-doc libraspberrypi0
  libssl2 libssl1.1 libxml2 libxmu1 openssh-client openssh-server
  openssh-sftp-server openssl raspberrypi-bootloader raspberrypi-kernel
  rpi-eeprom rpi-eeprom-images ssh sudo wpasupplicant
```

After that is complete run

```
sudo shutdown -h now
```

to shutdown the Pi safely. Remove power and attach the miniPiTFT. Watch that the pins plug into the **first 2x12 headers!** The rounded corner and mounting hole should line up.



Attach power to the Pi and re-log in. The PiTFT should be lit but nothing on the screen.

Run the following at the terminal

```
cd ~
sudo pip3 install --upgrade adafruit-python-shell click
sudo apt-get install -y git
git clone https://github.com/adafruit/Raspberry-Pi-Installer-Scripts.git
cd Raspberry-Pi-Installer-Scripts
sudo python3 adafruit-pitft.py --display=st7789_240x135 --rotation=270 --install-type=console
```

```
pi@raspberrypi:~ $ cd ~
pi@raspberrypi:~ $ sudo pip3 install --upgrade adafruit-python-shell click==7.0
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting adafruit-python-shell
  Using cached https://www.piwheels.org/simple/adafruit-python-shell/adafruit_python_shell-1.2.0-py3-none-any.whl
Collecting click==7.0
  Using cached https://files.pythonhosted.org/packages/fa/37/45185cb5abb30d7257104c434fe0b07e5a195a6847506c074527aa599ec/Click-7.0-py2.py3-none-any.whl
Requirement already satisfied, skipping upgrade: clint in /usr/local/lib/python3.7/dist-packages (from adafruit-python-shell) (0.5.1)
Requirement already satisfied, skipping upgrade: Adafruit-PlatformDetect in /usr/local/lib/python3.7/dist-packages (from adafruit-python-shell) (2.17.0)
Requirement already satisfied, skipping upgrade: args in /usr/local/lib/python3.7/dist-packages (from clint->adafruit-python-shell) (0.1.0)
Installing collected packages: adafruit-python-shell, click
Successfully installed adafruit-python-shell-1.2.0 click-7.0
pi@raspberrypi:~ $ sudo apt-get install -y git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.20.1-2+deb10u3).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@raspberrypi:~ $ git clone https://github.com/adafruit/Raspberry-Pi-Installer-Scripts.git
Cloning into 'Raspberry-Pi-Installer-Scripts'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 544 (delta 13), reused 18 (delta 7), pack-reused 514
Receiving objects: 100% (544/544), 181.33 KiB | 1.63 MiB/s, done.
Resolving deltas: 100% (305/305), done.
```

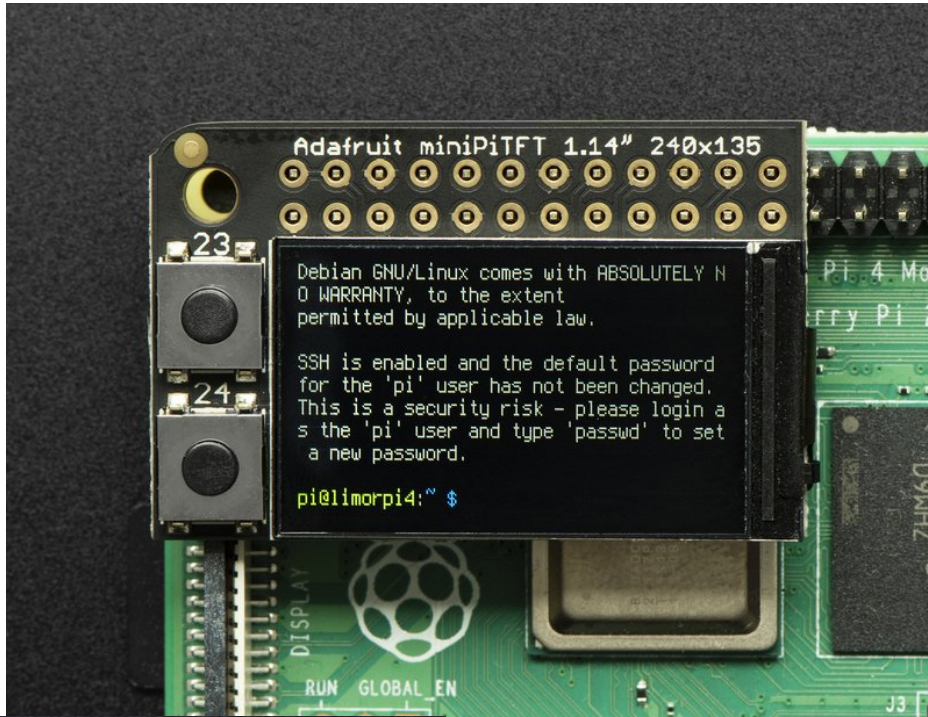
When you get asked to reboot, reboot!

```
Removing old section...
##### UPGRADING KERNEL #####
Updating packages...
Upgrading packages...
Installing Kernel Headers...
PITFT make: Entering directory '/usr/src/linux-headers-5.4.51-v7l+'
PITFT Building modules, stage 2.
PITFT MODPOST 2 modules
PITFT LD [M] /home/pi/Raspberry-Pi-Installer-Scripts/st7789_module/fb_st7789v.ko
PITFT LD [M] /home/pi/Raspberry-Pi-Installer-Scripts/st7789_module/st7789v_ada.ko
PITFT make: Leaving directory '/usr/src/linux-headers-5.4.51-v7l+'
PITFT Success!

Settings take effect on next boot.

REBOOT NOW? [Y/n] y
```

Zat's it! You will now have the miniPiTFT with a console display on it



If you ever get a display like this, it means your kernel changed - either due to an upgrade/update or because you switched Pi boards. The solution is to simply re-run the scripts above!

1.3" 240x240 Kernel Module Install

There's two ways you can use the 1.3" 240x240 display.

Be aware that you can only choose to do one way at a time. If you choose the hard way, it will install the kernel driver, which will prevent you from doing it the easy way.

The easy way is to use 'pure Python 3' and Pillow library to draw to the display from within Python. This is great for showing text, stats, images etc that you design yourself. If you want to do that, skip this page and go to the Python install/usage page

The hard way is to install a kernel module to add support for the TFT display that will make the console appear on the display. This is cute because you can have any program print text or draw to the framebuffer (or, say, with pygame) and Linux will take care of displaying it for you. If you don't need the console or direct framebuffer access, please consider using the 'pure Python' technique instead as it is not as delicate.



We don't recommend using the 240x240 display for GUI/PIXEL desktop, this is only for text console usage. The display is waaay too small for a desktop

Prepare the Pi!

Before you begin, its a good idea to get your Pi completely updated and upgraded. We assume you have burned an SD card and can log into the console to install stuff.

Run

```
sudo apt update -y
sudo apt-get update -y
sudo apt-get upgrade -y
```

To fully update and upgrade your Pi!

```
pi@raspberrypi:~ $ sudo apt-get upgrade -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages have been kept back:
  raspi-config
The following packages will be upgraded:
  e2fsprogs firmware-atheros firmware-brcm80211 firmware-libertas
  firmware-misc-nonfree firmware-realtek libcom-err2 libext2fs2
  libraspberrypi-bin libraspberrypi-dev libraspberrypi-doc libraspberrypi0
  libssl2 libssl1.1 libxml2 libxmu1 openssl-client openssl-server
  openssl-sftp-server openssh raspberrypi-bootloader raspberrypi-kernel
  rpi-eeeprom rpi-eeeprom-images ssh sudo wpasupplicant
```

After that is complete run

sudo shutdown -h now

to shutdown the Pi safely. Remove power and attach the miniPiTFT. Watch that the pins plug into the **first 2x12 headers!** The rounded corner and mounting hole should line up.

Attach power to the Pi and re-log in. The PiTFT should be lit but nothing on the screen.

Run the following at the terminal

```
cd ~
sudo pip3 install --upgrade adafruit-python-shell click
sudo apt-get install -y git
git clone https://github.com/adafruit/Raspberry-Pi-Installer-Scripts.git
cd Raspberry-Pi-Installer-Scripts
sudo python3 adafruit-pitft.py --display=st7789_240x240 --rotation=0 --install-type=console
```

```
pi@raspberrypi:~ $ cd ~
pi@raspberrypi:~ $ sudo pip3 install --upgrade adafruit-python-shell click==7.0
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting adafruit-python-shell
  Using cached https://www.piwheels.org/simple/adafruit-python-shell/adafruit_python_shell-1.2.0-py3-none-any.whl
Collecting click==7.0
  Using cached https://files.pythonhosted.org/packages/fa/37/45185cb5abbc30d7257104c434fe0b07e5a195a6847506c074527aa599ec/Click-7.0-py2.py3-none-any.whl
Requirement already satisfied, skipping upgrade: clint in /usr/local/lib/python3.7/dist-packages (from adafruit-python-shell) (0.5.1)
Requirement already satisfied, skipping upgrade: Adafruit-PlatformDetect in /usr/local/lib/python3.7/dist-packages (from adafruit-python-shell) (2.17.0)
Requirement already satisfied, skipping upgrade: args in /usr/local/lib/python3.7/dist-packages (from clint->adafruit-python-shell) (0.1.0)
Installing collected packages: adafruit-python-shell, click
Successfully installed adafruit-python-shell-1.2.0 click-7.0
pi@raspberrypi:~ $ sudo apt-get install -y git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.20.1-2+deb10u3).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@raspberrypi:~ $ git clone https://github.com/adafruit/Raspberry-Pi-Installer-Scripts.git
Cloning into 'Raspberry-Pi-Installer-Scripts'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 544 (delta 13), reused 18 (delta 7), pack-reused 514
Receiving objects: 100% (544/544), 181.33 KiB | 1.63 MiB/s, done.
Resolving deltas: 100% (205/205), done.
```

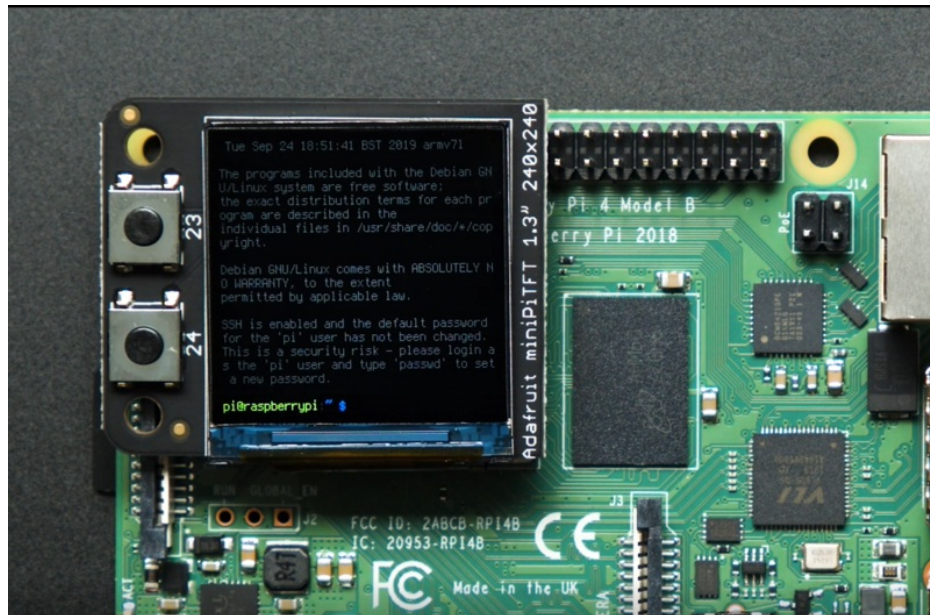
When you get asked to reboot, reboot!

```
Removing old section...
##### UPGRADING KERNEL #####
Updating packages...
Updating packages...
Installing Kernel Headers...
PITFT make: Entering directory '/usr/src/linux-headers-5.4.51-v7l+'
PITFT Building modules, stage 2.
PITFT MODPOST 2 modules
PITFT LD [M] /home/pi/Raspberry-Pi-Installer-Scripts/st7789_module/fb_st7789v.ko
PITFT LD [M] /home/pi/Raspberry-Pi-Installer-Scripts/st7789_module/st7789v_ada.ko
PITFT make: Leaving directory '/usr/src/linux-headers-5.4.51-v7l+'
PITFT Success!

Settings take effect on next boot.

REBOOT NOW? [Y/n] y
```

Zat's it! You will now have the miniPiTFT with a console display on it



Kernel Module Troubleshooting

The Raspberry Pi Kernel sometimes updates firmware, which can which can break the Frame Buffer Copy mechanism. In this particular case, it only seems to affect the Raspberry Pi 4. The issue appears as a garbled screen that looks like static.



To check your kernel version, run the following command:

```
dpkg -l raspberrypi-kernel
```

You should see output similar to the following. If the kernel version is 20210104 or later, then the following fix should work.

```
pi@raspberrypi:~$ dpkg -l raspberrypi-kernel
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name          Version          Architecture Description
++-----+-----+-----+-----+
ii raspberrypi-kernel 1.20210108-1 armhf      Raspberry Pi bootloader
```

We have a script that is able to set the kernel version to the kernel version prior to it breaking. To "pin" the kernel version to an older version prior to it breaking, you'll need to run a few commands. You can either SSH into the Pi or hook up an HDMI cable, though the display may appear small.

Once you'd at a command prompt, run the following commands:

- `cd ~`
- `wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/rpi-pin-kernel-firmware.sh`
- `sudo sh rpi-pin-kernel-firmware.sh 1.20201126-1`

After it finishes, reboot the Pi.

Once the Pi is back up, the display may appear inverted. To fix this, just run the Adafruit PiTFT script again and reboot a second time.

You can check the new kernel version by running the `dpkg` command again:

```
dpkg -l raspberrypi-kernel
```

This time, your version should be 20201126-1.

```
pi@raspberrypi:~$ dpkg -l raspberrypi-kernel
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name          Version          Architecture Description
+++-----+-----+-----+-----+
ii raspberrypi-kernel 1.20201126-1 armhf      Raspberry Pi bootloader
```

BrainCraft Audio Driver Reinstall

If your display is a BrainCraft HAT and you have pinned your kernel, you should be running a kernel version of around 5.4. You can check this by typing `uname -r`.

```
pi@raspberrypi:~$ uname -r
5.4.79-v7l+
```

You may need to reinstall the audio drivers at this point to get sound working. Be sure to follow the [BrainCraft HAT Audio Setup instructions \(https://adafru.it/Pf8\)](https://adafru.it/Pf8) for a kernel version around 5.4 when reinstalling.

Python Setup

You can use this technique with any PiTFT, from the 240x135 mini PiTFT up to the 320x480. It isn't as fast as the kernel module support version but it'll work no matter what kernel/OS/version/etc and so is a lot less painful

Attaching

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

Since the PiTFT comes preassembled, all you need to do is place it onto the GPIO pins.

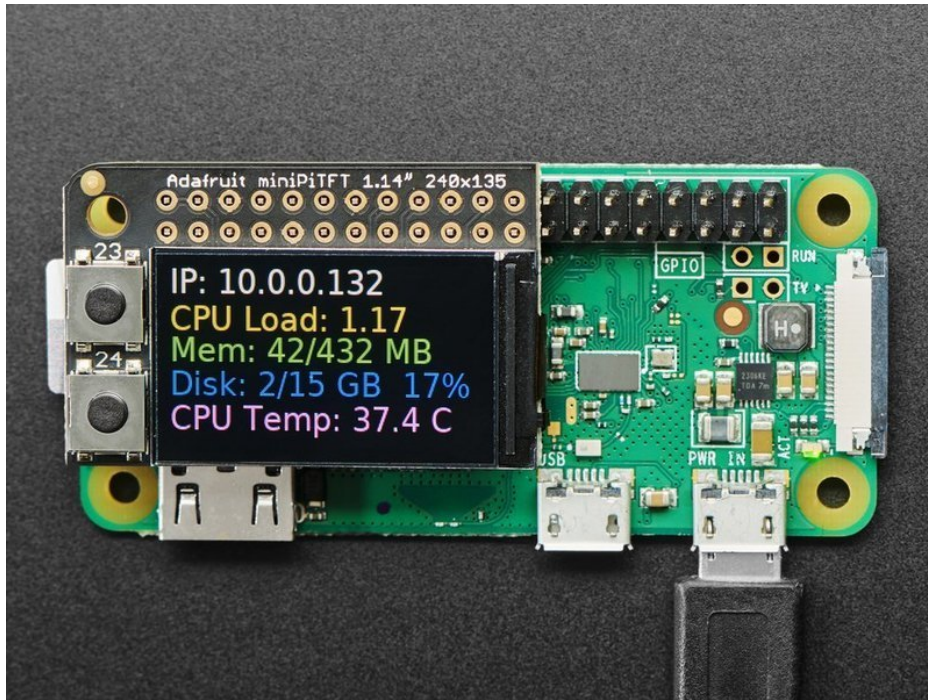
Since there's *dozens* of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, [please visit the guide for CircuitPython on Linux to see whether your platform is supported \(https://adafru.it/BSN\)](https://adafru.it/BSN).

Connect the display as shown below to your Raspberry Pi.

Note this is not a kernel driver that will let you have the console appear on the TFT. However, this is handy when you can't install an fbft driver, and want to use the TFT purely from 'user Python' code!

You can only use this technique with Linux/computer devices that have hardware SPI support, and not all single board computers have an SPI device so check before continuing

For the 1.14":



For the 1.3":



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 already installed the kernel module, you will need to remove it by editing your `/boot/config.txt` file before proceeding.

Python Installation of RGB Display Library

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

- `sudo pip3 install adafruit-circuitpython-rgb-display`
- `sudo pip3 install --upgrade --force-reinstall spidev`

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`

NumPy Library

A recent improvement of the RGB_Display library makes use of NumPy for some additional speed. This can be installed with the following command:

- `sudo apt-get install python3-numpy`

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

Quickstart Button Test

This button test demo will test to make sure you have everything setup correctly. Go ahead and save the file to your Raspberry Pi in your home directory as `rgb_display_minipitfttest.py`.

```

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

import digitalio
import board

from adafruit_rgb_display.rgb import color565
import adafruit_rgb_display.st7789 as st7789

# Configuration for CS and DC pins for Raspberry Pi
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = None
BAUDRATE = 64000000 # The pi can be very fast!
# Create the ST7789 display:
display = st7789.ST7789(
    board.SPI(),
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
    width=135,
    height=240,
    x_offset=53,
    y_offset=40,
)

backlight = digitalio.DigitalInOut(board.D22)
backlight.switch_to_output()
backlight.value = True
buttonA = digitalio.DigitalInOut(board.D23)
buttonB = digitalio.DigitalInOut(board.D24)
buttonA.switch_to_input()
buttonB.switch_to_input()

# Main loop:
while True:
    if buttonA.value and buttonB.value:
        backlight.value = False # turn off backlight
    else:
        backlight.value = True # turn on backlight
    if buttonB.value and not buttonA.value: # just button A pressed
        display.fill(color565(255, 0, 0)) # red
    if buttonA.value and not buttonB.value: # just button B pressed
        display.fill(color565(0, 0, 255)) # blue
    if not buttonA.value and not buttonB.value: # none pressed
        display.fill(color565(0, 255, 0)) # green

```

Go ahead and run it with this command:

```
sudo python3 rgb_display_minipitfttest.py
```

Once it is running, push the buttons. The top button should make the display light up **Red**, the bottom **Blue**, and pressing both at the same time should make it **Green**.

Python Stats Example

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.

We can also display some stats about your Pi such as the IP address, resource usage, and even the CPU Temperature. Start by saving the code below as **stats.py** in your home directory on your Raspberry Pi.

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

# -*- coding: utf-8 -*-

import time
import subprocess
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
import adafruit_rgb_display.st7789 as st7789

# Configuration for CS and DC pins (these are FeatherWing defaults on M0/M4):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = None

# Config for display baudrate (default max is 24mhz):
BAUDRATE = 64000000

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

# Create the ST7789 display:
disp = st7789.ST7789(
    spi,
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
    width=135,
    height=240,
    x_offset=53,
    y_offset=40,
)

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

# 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, rotation)
```

```

# Draw some shapes.
# First define some constants to allow easy resizing of shapes.
padding = -2
top = padding
bottom = height - padding
# Move left to right keeping track of the current x position for drawing shapes.
x = 0

# Alternatively 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)

# Turn on the backlight
backlight = digitalio.DigitalInOut(board.D22)
backlight.switch_to_output()
backlight.value = True

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\\\", $(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", $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\\\", $(NF-0) / 1000}'" # py
    Temp = subprocess.check_output(cmd, shell=True).decode("utf-8")

    # Write four lines of text.
    y = top
    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")

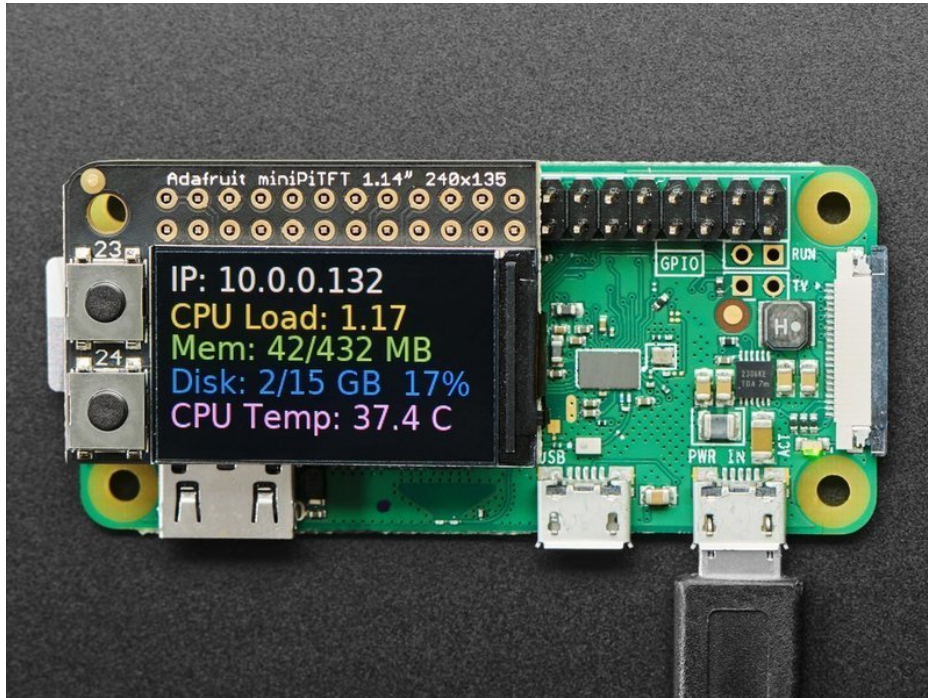
    # Display image.
    disp.image(image, rotation)
    time.sleep(0.1)

```

Go ahead and run the script by typing:

```
python3 stats.py
```

It should display some system information.



Modifications for the 1.3" Display

To get the `stats.py` example to display properly on the 1.3" TFT Display, you will need to make some changes due to the different geometry of the display. The parameters you will need to adjust are the `height`, `x_offset`, `y_offset`, and `rotation`.

The new values should be:

- `height = 240`
- `x_offset = 0`
- `y_offset = 80`
- `rotation = 180`

The easiest way to replace them may be to copy the following code block and replace it in the above code.

```
# Create the ST7789 display:
disp = st7789.ST7789(
    spi,
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
    width=240,
    height=240,
    x_offset=0,
    y_offset=80,
)

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

Running Stats on Boot

You can pretty easily make it so this handy program runs every time you boot your Pi.

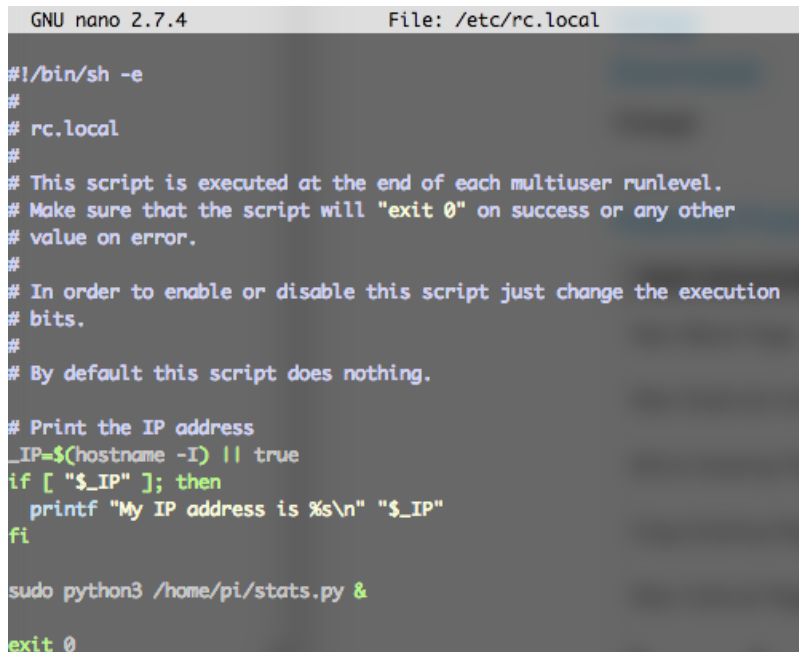
The fastest/easiest way is to put it in `/etc/rc.local`

Run `sudo nano /etc/rc.local` and add the line

```
sudo python3 /home/pi/stats.py &
```

on its own line right before `exit 0`

Then save and exit. Reboot to verify that the screen comes up on boot!



```
GNU nano 2.7.4 File: /etc/rc.local
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
#
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
sudo python3 /home/pi/stats.py &
exit 0
```

For more advanced usage, check out our linux system services guide (<https://adafru.it/wFR>)

Troubleshooting Stats on Boot

For the normal installation of Blinka on Raspberry Pi, we have you install stuff without the `sudo` keyword, which will install the libraries locally. However, to have the script run at boot, you will need to have the libraries available on a more system wide level. You can test this out by running the following command and see if the the stats come up:

```
sudo python3 /home/pi/stats.py
```

If you have any errors, most can be fixed by running the following command:

```
sudo pip3 install --upgrade adafruit-blinka adafruit-circuitpython-rgb-display spidev
```

Once you can get it to come up, go ahead and press `Control+C` and reboot the system. It should come up now.

Sometimes the Pi can boot too fast, so you may also need to add `sleep 10` on the line before the command you added in `/etc/rc.local`.

Python Usage

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.

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:

```
# 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 ratios 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.

```
# 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 # pylint: disable=unused-import
import adafruit_rgb_display.hx8357 # pylint: disable=unused-import
import adafruit_rgb_display.st7735 # pylint: disable=unused-import
import adafruit_rgb_display.ssd1351 # pylint: disable=unused-import
import adafruit_rgb_display.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(
    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 our 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:

```
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.

```

# 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=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)

```

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.

```

# 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)

```

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.

```

# 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 Blinky 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.

```

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.

```
# 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.

```
# 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.

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



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.

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

"""
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
"""

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.
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 `ImageFont` 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.

```
# 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.

```
# 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.

```
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:

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

"""
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)!
"""

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(
    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)

# 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\\\", $(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", $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\", $(NF-0) / 1000}'" # py
lint: 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="#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")

# Display image.
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.

```

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.

```

# 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.

```

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.

```
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>

```
cmd = "hostname -I | cut -d\ ' ' -f1"
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 %s\\\", $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\\\", $(NF-0) / 1000}'" # pylint: disable=line-too-long
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.

```
y = padding
draw.text((x, y), IP, font=font, fill="#FFFFFF")
y += font.getsize(IP)[1]
draw.text((x, y), CPU, font=font, fill="#FFF000")
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.

```
disp.image(image)
time.sleep(.1)
```

