



# PyPortal Thingiverse Viewer

Created by John Park



<https://learn.adafruit.com/pyportal-thingiverse-viewer>

Last updated on 2025-02-26 02:59:53 PM EST

# Table of Contents

<b>Overview</b>	<b>3</b>
<ul style="list-style-type: none"><li>• <a href="#">Parts</a></li></ul>	
<b>Install CircuitPython</b>	<b>5</b>
<ul style="list-style-type: none"><li>• <a href="#">Set up CircuitPython Quick Start!</a></li><li>• <a href="#">PyPortal Default Files</a></li></ul>	
<b>PyPortal CircuitPython Setup</b>	<b>7</b>
<ul style="list-style-type: none"><li>• <a href="#">Adafruit CircuitPython Bundle</a></li></ul>	
<b>Create Your settings.toml File</b>	<b>9</b>
<ul style="list-style-type: none"><li>• <a href="#">CircuitPython settings.toml File</a></li><li>• <a href="#">settings.toml File Tips</a></li><li>• <a href="#">Accessing Your settings.toml Information in code.py</a></li></ul>	
<b>Internet Connect!</b>	<b>11</b>
<ul style="list-style-type: none"><li>• <a href="#">Connect to WiFi</a></li><li>• <a href="#">Advanced Requests Usage</a></li><li>• <a href="#">WiFi Manager</a></li><li>• <a href="#">Further Information</a></li></ul>	
<b>Code the Thingiverse Thing Viewer</b>	<b>20</b>
<ul style="list-style-type: none"><li>• <a href="#">Thingiverse Apps API</a></li><li>• <a href="#">Adafruit IO Image Converter Server</a></li><li>• <a href="#">Add CircuitPython Code and Assets</a></li><li>• <a href="#">Editing the Code</a></li><li>• <a href="#">boot.py</a></li><li>• <a href="#">How It Works</a></li><li>• <a href="#">Display Stand</a></li></ul>	

---

# Overview



Thingiverse is a terrific repository of Things that you can download and build using various digital fabrication techniques, with an emphasis on 3D printable models.

You can display a collection of your favorite Thingiverse maker Things with the PyPortal and admire these builds in convenient 2D image form!

## Parts

You can build this project with parts in AdaBox 011 or order parts separately.



### [AdaBox011 - PyPortal](#)

Reach out beyond your desk - to the stars and beyond - with PyPortal! This ADABOX features a new, easy-to-use IoT device that allows you to customize and create your...

<https://www.adafruit.com/product/4061>



### Adafruit PyPortal - CircuitPython Powered Internet Display

PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...

<https://www.adafruit.com/product/4116>



### Pink and Purple Braided USB A to Micro B Cable - 2 meter long

This cable is super-fashionable with a woven pink and purple Blinka-like pattern! First let's talk about the cover and over-molding. We got these in custom colors,...

<https://www.adafruit.com/product/4148>



### Black Nylon Machine Screw and Stand-off Set – M2.5 Thread

Totaling 380 pieces, this M2.5 Screw Set is a must-have for your workstation. You'll have enough screws, nuts, and hex standoffs to fuel your maker...

<https://www.adafruit.com/product/3299>



### PLA Filament for 3D Printers - 2.85mm Diameter - Magenta - 1 Kg

Having a 3D printer without filament is sort of like having a regular printer without paper or ink. And while a lot of printers come with some filament there's a good chance...

<https://www.adafruit.com/product/3732>

---

# Install CircuitPython

[CircuitPython](https://adafru.it/tB7) (<https://adafru.it/tB7>) is a derivative of [MicroPython](https://adafru.it/BeZ) (<https://adafru.it/BeZ>) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** "flash" drive to iterate.

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

## Set up CircuitPython Quick Start!

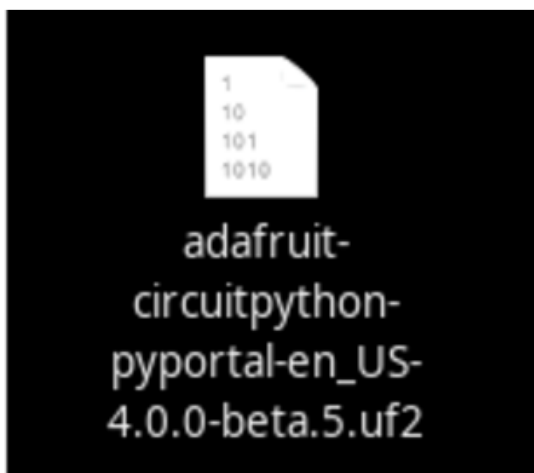
Follow this quick step-by-step for super-fast Python power :)

Download the latest version of  
CircuitPython for the PyPortal via  
CircuitPython.org

<https://adafru.it/Egk>

Download the latest version of  
CircuitPython for the PyPortal Pynt  
via CircuitPython.org

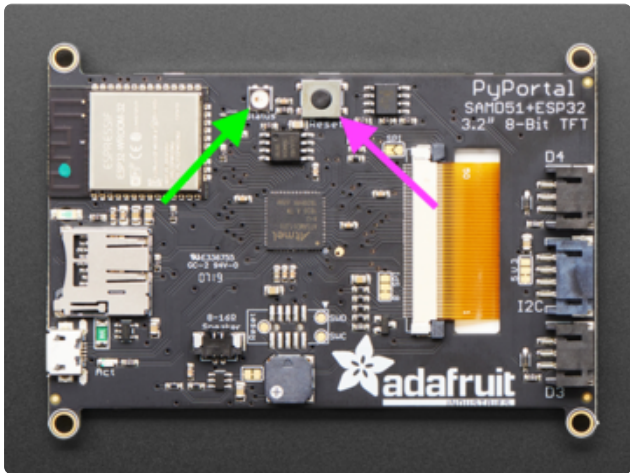
<https://adafru.it/HFd>



Click the link above to download the latest version of CircuitPython for the PyPortal.

Download and save it to your desktop (or wherever is handy).

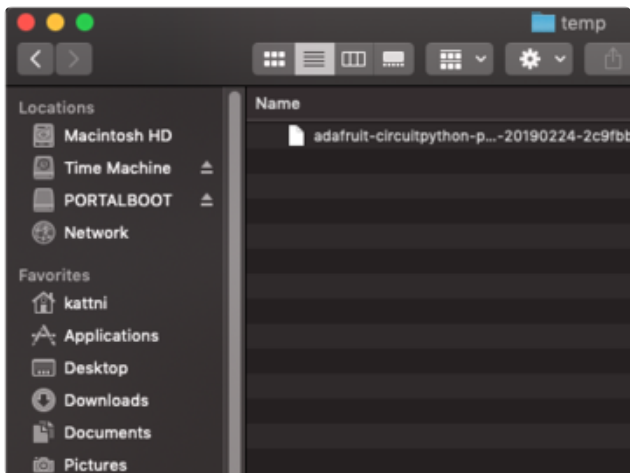
Plug your PyPortal into your computer using a known-good USB cable.



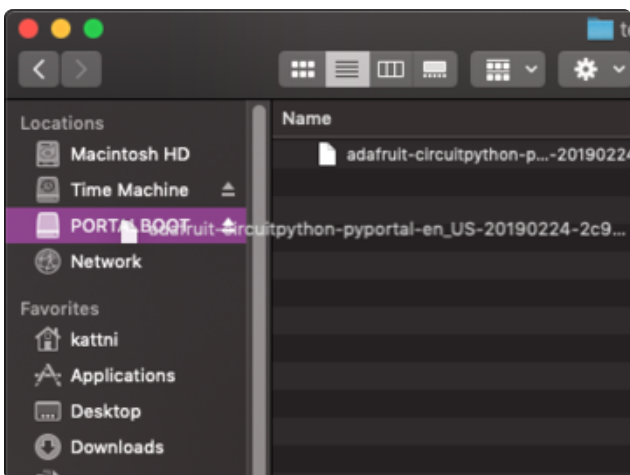
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

Double-click the **Reset** button on the top in the middle (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

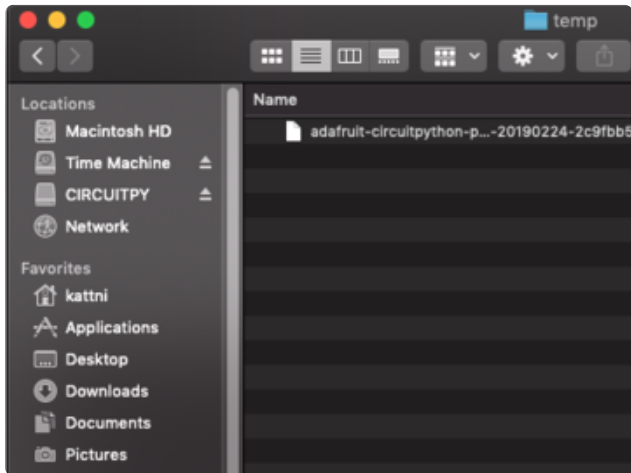
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



You will see a new disk drive appear called **PORTALBOOT**.



Drag the **adafruit-circuitpython-pyportal-  
<whatever>.uf2** file to **PORTALBOOT**.



The LED will flash. Then, the **PORTALBOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

If you haven't added any code to your board, the only file that will be present is **boot\_out.txt**. This is absolutely normal! It's time for you to add your **code.py** and get started!

That's it, you're done! :)

## PyPortal Default Files

Click below to download a zip of the files that shipped on the PyPortal or PyPortal Pynt.

PyPortal Default Files

<https://adafru.it/UF->

PyPortal Pynt Default Files

<https://adafru.it/UGa>

---

## PyPortal CircuitPython Setup

To use all the amazing features of your PyPortal with CircuitPython, you must first install a number of libraries. This page covers that process.

## Adafruit CircuitPython Bundle

Download the Adafruit CircuitPython Library Bundle. You can find the latest release here:

Latest Adafruit CircuitPython  
Library Bundle

<https://adafru.it/ENC>

Download the **adafruit-circuitpython-bundle-\*.x-mpy-.zip** bundle zip file where **\*.x MATCHES THE VERSION OF CIRCUITPYTHON YOU INSTALLED**, and unzip a folder of the same name. Inside you'll find a **lib** folder. You have two options:

- You can add the **lib** folder to your **CIRCUITPY** drive. This will ensure you have all the drivers. But it will take a bunch of space on the 8 MB disk
- Add each library as you need it, this will reduce the space usage but you'll need to put in a little more effort.

At a minimum we recommend the following libraries, in fact we more than recommend. They're basically required. So grab them and install them into **CIRCUITPY/lib** now!

- **adafruit\_esp32spi** - This is the library that gives you internet access via the ESP32 using (you guessed it!) SPI transport. You need this for anything Internet
- **adafruit\_requests** - This library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- **adafruit\_connection\_manager** - used by **adafruit\_requests**.
- **adafruit\_pyportal** - This is our friendly wrapper library that does a lot of our projects, displays graphics and text, fetches data from the internet. Nearly all of our projects depend on it!
- **adafruit\_portalbase** - This library is the base library that **adafruit\_pyportal** library is built on top of.
- **adafruit\_touchscreen** - a library for reading touches from the resistive touchscreen. Handles all the analog noodling, rotation and calibration for you.
- **adafruit\_io** - this library helps connect the PyPortal to our free datalogging and viewing service
- **adafruit\_imageload** - an image display helper, required for any graphics!
- **adafruit\_display\_text** - not surprisingly, it displays text on the screen
- **adafruit\_bitmap\_font** - we have fancy font support, and its easy to make new fonts. This library reads and parses font files.
- **adafruit\_slideshow** - for making image slideshows - handy for quick display of graphics and sound
- **neopixel** - for controlling the onboard neopixel
- **adafruit\_adt7410** - library to read the temperature from the on-board Analog Devices ADT7410 precision temperature sensor (not necessary for Titano or Pynt)
- **adafruit\_bus\_device** - low level support for I2C/SPI
- **adafruit\_fakerequests** - This library allows you to create fake HTTP requests by using local files.



---

# Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of [CircuitPython 8 \(https://adafru.it/Em8\)](https://adafru.it/Em8), there is support for a **settings.toml** file. This is a file that is stored on your **CIRCUITPY** drive, that contains all of your secret network information, such as your SSID, SSID password and any API keys for IoT services. It is designed to separate your sensitive information from your **code.py** file so you are able to share your code without sharing your credentials.

CircuitPython previously used a **secrets.py** file for this purpose. The **settings.toml** file is quite similar.

Your settings.toml file should be stored in the main directory of your CIRCUITPY drive. It should not be in a folder.

## CircuitPython settings.toml File

This section will provide a couple of examples of what your **settings.toml** file should look like, specifically for CircuitPython WiFi projects in general.

The most minimal **settings.toml** file must contain your WiFi SSID and password, as that is the minimum required to connect to WiFi. Copy this example, paste it into your **settings.toml**, and update:

- `your_wifi_ssid`
- `your_wifi_password`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"  
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
```

Many CircuitPython network-connected projects on the Adafruit Learn System involve using Adafruit IO. For these projects, you must also include your Adafruit IO username and key. Copy the following example, paste it into your settings.toml file, and update:

- `your_wifi_ssid`
- `your_wifi_password`
- `your_aio_username`
- `your_aio_key`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_AIO_USERNAME = "your_aio_username"
ADAFRUIT_AIO_KEY = "your_aio_key"
```

Some projects use different variable names for the entries in the `settings.toml` file. For example, a project might use `ADAFRUIT_AIO_ID` in the place of `ADAFRUIT_AIO_USERNAME`. If you run into connectivity issues, one of the first things to check is that the names in the `settings.toml` file match the names in the code.

Not every project uses the same variable name for each entry in the `settings.toml` file! Always verify it matches the code.

## settings.toml File Tips

Here is an example `settings.toml` file.

```
# Comments are supported
CIRCUITPY_WIFI_SSID = "guest wifi"
CIRCUITPY_WIFI_PASSWORD = "guessable"
CIRCUITPY_WEB_API_PORT = 80
CIRCUITPY_WEB_API_PASSWORD = "passw0rd"
test_variable = "this is a test"
thumbs_up = "\U0001f44d"
```

In a `settings.toml` file, it's important to keep these factors in mind:

- Strings are wrapped in double quotes; ex: `"your-string-here"`
- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
  - Floats, octal (`0o567`) and binary (`0b11011`) are not supported.
- Use `\u` escapes for weird characters, `\x` and `\ooo` escapes are not available in `.toml` files
  - Example: `\U0001f44d` for 👍 (thumbs up emoji) and `\u20ac` for € (EUR sign)
- Unicode emoji, and non-ASCII characters, stand for themselves as long as you're careful to save in "UTF-8 without BOM" format



When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

## Accessing Your **settings.toml** Information in **code.py**

In your **code.py** file, you'll need to **import** the **os** library to access the **settings.toml** file. Your settings are accessed with the **os.getenv()** function. You'll pass your settings entry to the function to import it into the **code.py** file.

```
import os
print(os.getenv("test_variable"))
```

```
CircuitPython REPL
code.py output:
this is a test

Code done running.

Press any key to enter the REPL. Use CTRL-D to reload.
```

In the upcoming CircuitPython WiFi examples, you'll see how the **settings.toml** file is used for connecting to your SSID and accessing your API keys.

---

# Internet Connect!

## Connect to WiFi

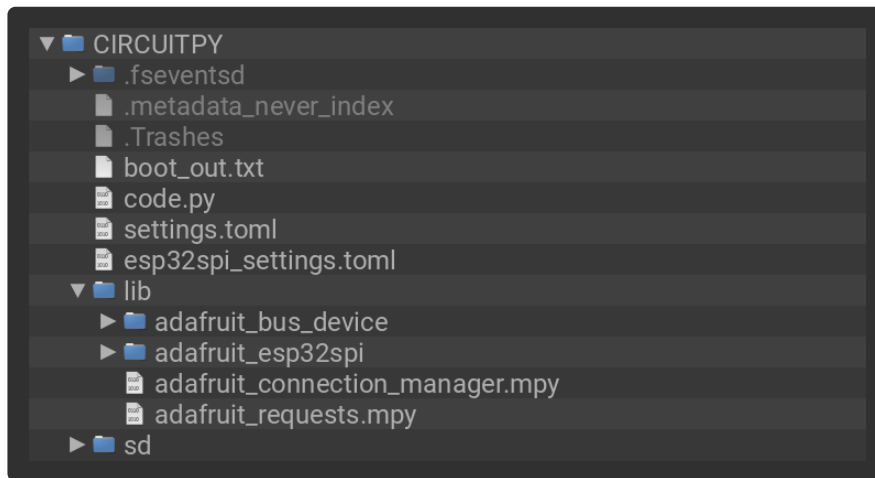
OK, now that you have your **settings.toml** file set up - you can connect to the Internet.

To do this, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **examples/** and then

click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:



Update to CircuitPython 9.2.x or later to use this example.

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

from os import getenv
import board
import busio
from digitalio import DigitalInOut
import adafruit_connection_manager
import adafruit_requests
from adafruit_esp32spi import adafruit_esp32spi

# Get wifi details and more from a settings.toml file
# tokens used by this Demo: CIRCUITPY_WIFI_SSID, CIRCUITPY_WIFI_PASSWORD
ssid = getenv("CIRCUITPY_WIFI_SSID")
password = getenv("CIRCUITPY_WIFI_PASSWORD")

print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://wifitest.adafruit.com/testwifi/sample.json"

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an AirLift Shield:
# esp32_cs = DigitalInOut(board.D10)
# esp32_ready = DigitalInOut(board.D7)
# esp32_reset = DigitalInOut(board.D5)

# If you have an AirLift Featherwing or ItsyBitsy AirLift:
```

```

# esp32_cs = DigitalInOut(board.D13)
# esp32_ready = DigitalInOut(board.D11)
# esp32_reset = DigitalInOut(board.D12)

# If you have an externally connected ESP32:
# NOTE: You may need to change the pins to reflect your wiring
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

# Secondary (SCK1) SPI used to connect to WiFi board on Arduino Nano Connect RP2040
if "SCK1" in dir(board):
    spi = busio.SPI(board.SCK1, board.MOSI1, board.MISO1)
else:
    spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

pool = adafruit_connection_manager.get_radio_socketpool(esp)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(esp)
requests = adafruit_requests.Session(pool, ssl_context)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", ":".join("%02X" % byte for byte in esp.MAC_address))

for ap in esp.scan_networks():
    print("\t%-23s RSSI: %d" % (ap.ssid, ap.rssi))

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(ssid, password)
    except OSError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", esp.ap_info.ssid, "\tRSSI:", esp.ap_info.rssi)
print("My IP address is", esp.ipv4_address)
print(
    "IP lookup adafruit.com: %s" %
    esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
)
print("Ping google.com: %d ms" % esp.ping("google.com"))

# esp._debug = True
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print("-" * 40)
print(r.text)
print("-" * 40)
r.close()

print()
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print("-" * 40)
print(r.json())
print("-" * 40)
r.close()

print("Done!")

```

And save it to your board, with the name **code.py**.

Don't forget you'll also need to create the **settings.toml** file as seen above, with your WiFi ssid and password.

In a serial console, you should see something like the following. For more information about connecting with a serial console, view the guide [Connecting to the Serial Console \(https://adafru.it/Bec\)](https://adafru.it/Bec).

```
&gt;&gt;&gt; import wifitest
ESP32 SPI webclient test
ESP32 found and in idle mode
Firmware vers. 1.7.5
MAC addr: 24:C9:DC:BD:0F:3F
  HomeNetwork      RSSI: -46
  HomeNetwork      RSSI: -76
  Fios-12345        RSSI: -92
  FiOS-AB123        RSSI: -92
  NETGEAR53         RSSI: -93
Connecting to AP...
Connected to HomeNetwork  RSSI: -45
My IP address is 192.168.1.245
IP lookup adafruit.com: 104.20.39.240
Ping google.com: 30 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
-----
This is a test of Adafruit WiFi!
If you can read this, its working :)
-----

Fetching json from http://wifitest.adafruit.com/testwifi/sample.json
-----
{'fun': True, 'company': 'Adafruit', 'founded': 2005, 'primes': [2, 3, 5], 'pi':
3.14, 'mixed': [False, None, 3, True, 2.7, 'cheese']}
-----
Done!
```

Going over the example above, here's a breakdown of what the program is doing:

- Initialize the ESP32 over SPI using the SPI port and 3 control pins:

```
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

#...

else:
    spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
    esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
```

- Get the socket pool and the SSL context, and then tell the `adafruit_requests` library about them.

```
pool = adafruit_connection_manager.get_radio_socketpool(esp)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(esp)
requests = adafruit_requests.Session(pool, ssl_context)
```

- Verify an ESP32 is found, checks the firmware and MAC address

```

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", ":".join("%02X" % byte for byte in esp.MAC_address))

```

- Perform a scan of all access points it can see and print out the name and signal strength.

```

for ap in esp.scan_networks():
    print("\t%-23s RSSI: %d" % (ap.ssid, ap.rssi))

```

- Connect to the AP we've defined here, then print out the local IP address. Then attempt to do a domain name lookup and ping google.com to check network connectivity. (Note sometimes the ping fails or takes a while; this isn't a big deal.)

```

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(ssid, password)
    except OSError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", esp.ap_info.ssid, "\tRSSI:", esp.ap_info.rssi)
print("My IP address is", esp.ipv4_address)
print(
    "IP lookup adafruit.com: %s" %
    esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
)

```

Now we're getting to the really interesting part of the example program. We've written a library for web fetching web data, named [adafruit\\_requests](https://adafru.it/FpW) (<https://adafru.it/FpW>). It is a lot like the regular Python library named [requests](https://adafru.it/1af4) (<https://adafru.it/1af4>). This library allows you to send HTTP and HTTPS requests easily and provides helpful methods for parsing the response from the server.

- Here is the part of the example program is fetching text data from a URL.

```

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html" # Further up in the
program

# ...

print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print('-' * 40)
print(r.text)

```

```
print('-' * 40)
r.close()
```

- Finally, here the program is fetching some JSON data. The `adafruit_requests` library will parse the JSON into a Python dictionary whose structure is the same as the structure of the JSON.

```
JSON_URL = "http://wifitest.adafruit.com/testwifi/sample.json" # Further up in
the program

# ...

print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print('-' * 40)
print(r.json())
print('-' * 40)
r.close()
```

## Advanced Requests Usage

Want to send custom HTTP headers, parse the response as raw bytes, or handle a response's http status code in your CircuitPython code?

We've written an example to show advanced usage of the requests module below.

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update `code.py` with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the `code.py` file in a zip file. Extract the contents of the zip file, open the directory `examples/` and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

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

import os

import adafruit_connection_manager
import board
import busio
from adafruit_esp32spi import adafruit_esp32spi
from digitalio import DigitalInOut

import adafruit_requests

# Get WiFi details, ensure these are setup in settings.toml
ssid = os.getenv("CIRCUITPY_WIFI_SSID")
password = os.getenv("CIRCUITPY_WIFI_PASSWORD")

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
```



```

esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

# If you have an AirLift Featherwing or ItsyBitsy Airlift:
# esp32_cs = DigitalInOut(board.D13)
# esp32_ready = DigitalInOut(board.D11)
# esp32_reset = DigitalInOut(board.D12)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
radio = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

print("Connecting to AP...")
while not radio.is_connected:
    try:
        radio.connect_AP(ssid, password)
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(radio.ssid, "utf-8"), "\tRSSI:", radio.rssi)

# Initialize a requests session
pool = adafruit_connection_manager.get_radio_socketpool(radio)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(radio)
requests = adafruit_requests.Session(pool, ssl_context)

JSON_GET_URL = "https://httpbin.org/get"

# Define a custom header as a dict.
headers = {"user-agent": "blinka/1.0.0"}

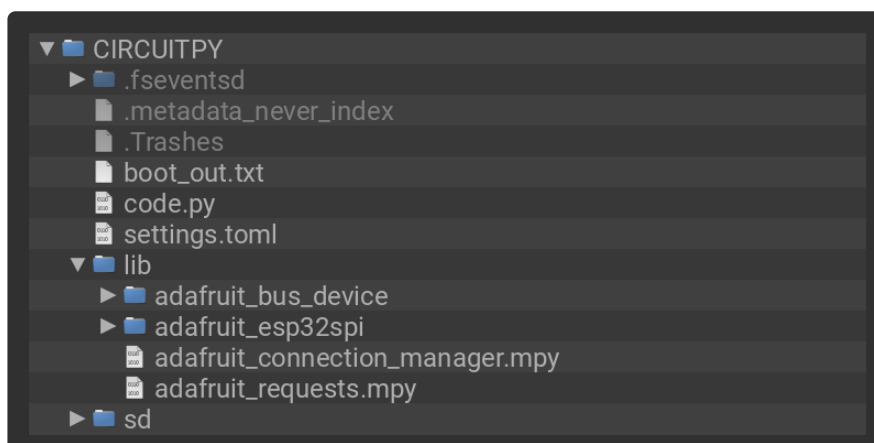
print("Fetching JSON data from %s..." % JSON_GET_URL)
with requests.get(JSON_GET_URL, headers=headers) as response:
    print("-" * 60)

    json_data = response.json()
    headers = json_data["headers"]
    print("Response's Custom User-Agent Header: {0}".format(headers["User-Agent"]))
    print("-" * 60)

    # Read Response's HTTP status code
    print("Response HTTP Status Code: ", response.status_code)
    print("-" * 60)

```

Your **CIRCUITPY** drive should now look similar to the following image:



# WiFi Manager

The way the examples above connect to WiFi works but it's a little finicky. Since WiFi is not necessarily so reliable, you may have disconnects and need to reconnect. For more advanced uses, we recommend using the `WiFiManager` class. It will wrap the connection/status/requests loop for you - reconnecting if WiFi drops, resetting the ESP32 if it gets into a bad state, etc.

Here's a more advanced example that shows using the `WiFiManager` and also how to fetch the current time from a web source.

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
from os import getenv
import board
import busio
from digitalio import DigitalInOut
import neopixel
import rtc
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi.adafruit_esp32spi_wifimanager import WiFiManager

# Get wifi details and more from a settings.toml file
# tokens used by this Demo: CIRCUITPY_WIFI_SSID, CIRCUITPY_WIFI_PASSWORD
ssid = getenv("CIRCUITPY_WIFI_SSID")
password = getenv("CIRCUITPY_WIFI_PASSWORD")

print("ESP32 local time")

TIME_API = "http://worldtimeapi.org/api/ip"

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

# Secondary (SCK1) SPI used to connect to WiFi board on Arduino Nano Connect RP2040
if "SCK1" in dir(board):
    spi = busio.SPI(board.SCK1, board.MOSI1, board.MISO1)
else:
    spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

"""Use below for Most Boards"""
status_pixel = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
"""Uncomment below for ItsyBitsy M4"""
# status_pixel = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1,
# brightness=0.2)
"""Uncomment below for an externally defined RGB LED (including Arduino Nano
Connect)"""
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
```

```

# status_pixel = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)

wifi = WiFiManager(esp, ssid, password, status_pixel=status_pixel)

the_rtc = rtc.RTC()

response = None
while True:
    try:
        print("Fetching json from", TIME_API)
        response = wifi.get(TIME_API)
        break
    except OSError as e:
        print("Failed to get data, retrying\n", e)
        continue

json = response.json()
current_time = json["datetime"]
the_date, the_time = current_time.split("T")
year, month, mday = [int(x) for x in the_date.split("-")]
the_time = the_time.split(".")[0]
hours, minutes, seconds = [int(x) for x in the_time.split(":")]

# We can also fill in these extra nice things
year_day = json["day_of_year"]
week_day = json["day_of_week"]
is_dst = json["dst"]

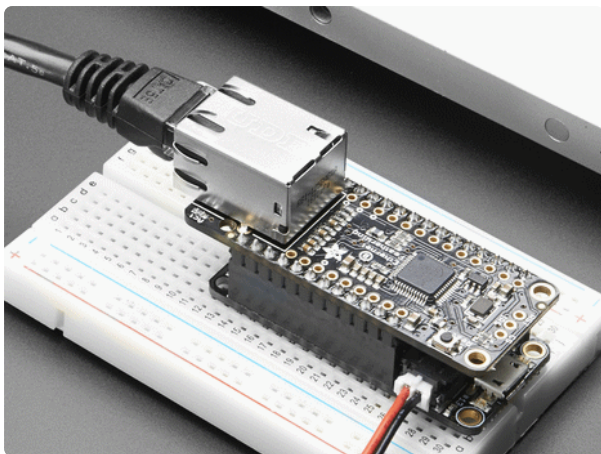
now = time.struct_time(
    (year, month, mday, hours, minutes, seconds, week_day, year_day, is_dst)
)
print(now)
the_rtc.datetime = now

while True:
    print(time.localtime())
    time.sleep(1)

```

## Further Information

For more information on the basics of doing networking in CircuitPython, see this guide:



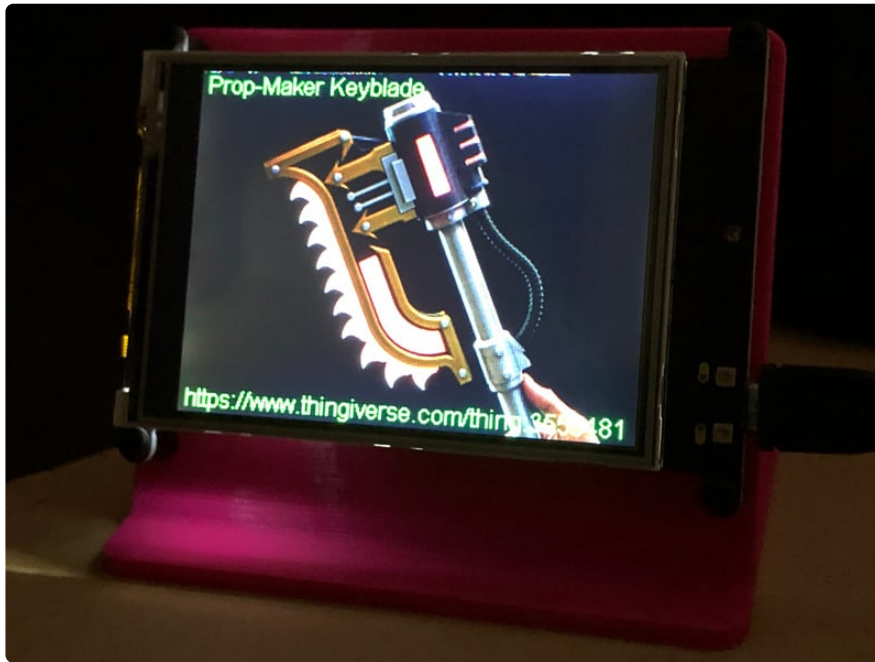
[Networking in CircuitPython](https://learn.adafruit.com/networking-in-circuitpython)

By Anne Barela

<https://learn.adafruit.com/networking-in-circuitpython>

---

# Code the Thingiverse Thing Viewer



## Thingiverse Apps API

As with most Internet of Things (IoT) projects on the PyPortal, we are able to perform our dynamic data and image gathering through the wonderful magic of a REST Application Programming Interface (API) request, that returns a JSON text file.

The Thingiverse Apps API makes it easy for us to request essential info about a Thing, including its name, URL, and image.

In order to use the Thingiverse API, you'll need to sign up for a free developer apps account. This will grant you your own Thingiverse app token needed for running queries.

Here's how to sign up:

1. First, log in or sign up for a regular Thingiverse account here <https://accounts.thingiverse.com/register> (<https://adafru.it/Ey9>)
2. Once you're logged in, go to the Apps create page here: <https://www.thingiverse.com/apps/create> (<https://adafru.it/Eya>)
3. On this page, click the **Web App** radio button
4. In the Basic Information area, enter an **App Name**, such as **PyPortal Thingiverse Viewer** (if you use a name that's already in use, you'll need to adjust it slightly to make it unique). Also enter an **App Description**
5. Click the checkbox to agree to the MakerBot API terms
6. Up at the top, right corner of the screen, click on the **Create & Get App Key button**

**Thingiverse** DASHBOARD EXPLORE EDUCATION CREATE Search Thingiverse You

THINGIVERSE DEVELOPERS / Create an App CANCEL CREATE & GET APP KEY

**SELECT A PLATFORM**  
Please select one platform that users can find and use your app. Unsure? [Learn more about platforms.](#)

Thing App Thingiverse.com Web App Desktop App Win / Mac / Linux IOS App Android App

**THING APP TYPE**  
What does your app do? This will help categorize your App on a Thing page. Unsure? [Learn more about Thing App types.](#)

Customization Your App allows for the customization or remaking of a Thing already found on Thingiverse.  
Printing Your app can print to 3rd party services or directly to a connected printer.  
Tools & Utilities

**BASIC INFORMATION**  
Let us know your app name and a brief description

App Name  
PyPortal Thingiverse Adafruit Viewer

App Description  
For viewing Things on PyPortal.

I agree to the MakerBot API terms and Privacy Policy

This will generate your Client ID, Client Secret, and App Token keys. Do not share these, keep them secret! You'll need to copy and paste them into a secure note somewhere on your computer. Later, we'll enter the App Token into the PyPortal's `secrets.py` file in order to query the Thingiverse API in read-only mode.

**API INFORMATION**  
Basic description copy for API information stuff goes here....

Client ID  
[Blurred]

Client Secret  
This key allows your application to interact with Thingiverse. Keep it secret!  
[Blurred]

App Token  
This token allows read only access to the Thingiverse API without requiring a user to authenticate. However, it also allows you to make changes to orders created by this app such as refunding and cancelling orders. So keep it secret and don't use it in client side javascript apps if your app involves payments!  
[Blurred]

GENERATE NEW APP TOKEN

## Adafruit IO Image Converter Server

In order to use the Adafruit image converter, this project will require you to have an Adafruit IO username and key. Adafruit IO is absolutely free to use, but you'll need to log in with your Adafruit account to use it. If you don't already have an Adafruit login, create [one here \(https://adafru.it/dAQ\)](https://adafru.it/dAQ).

If you haven't used Adafruit IO before, [check out this guide for more info \(https://adafru.it/Ef8\)](https://adafru.it/Ef8).

Once you have logged into your account, there are two pieces of information you'll need to place in your `settings.toml` file: **Adafruit IO username**, and **Adafruit IO key**. Head to [io.adafruit.com \(https://adafru.it/fsU\)](https://adafru.it/fsU) and simply click the **View AIO Key** link on the left hand side of the Adafruit IO page to get this information.

Then, add them to the `settings.toml` file, along with your **Thingiverse App Token** like this:

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
AIO_USERNAME = "your_aio_username"
AIO_KEY = "your_aio_key"
THINGIVERSE_TOKEN = "very_long_thingiverse_app_token"
```

At this point make sure your WiFi SSID and password are also in `settings.toml`. You only have to do this once when you set it up. If you move the PyPortal to a different WiFi served location, use a text editor to enter the new values in this file.

## Add CircuitPython Code and Assets

In the embedded code element below, click on the **Download Project Bundle** button, and save the `.zip` archive file to your computer.

Then, uncompress the `.zip` file, it will unpack to a folder named **PyPortal\_Thingiverse**.

Copy the contents of the **PyPortal\_Thingiverse** directory to your PyPortal **CIRCUITPY** drive.



## Editing the Code

You can edit the `code.py` file with any text editor you like. Adafruit suggests installing the free Mu Python editor as it's super handy, recognizes Adafruit boards, and has a built in serial monitor/REPL to interact with the board. [Find out more about Mu here \(https://adafru.it/ANO\)](https://adafru.it/ANO).

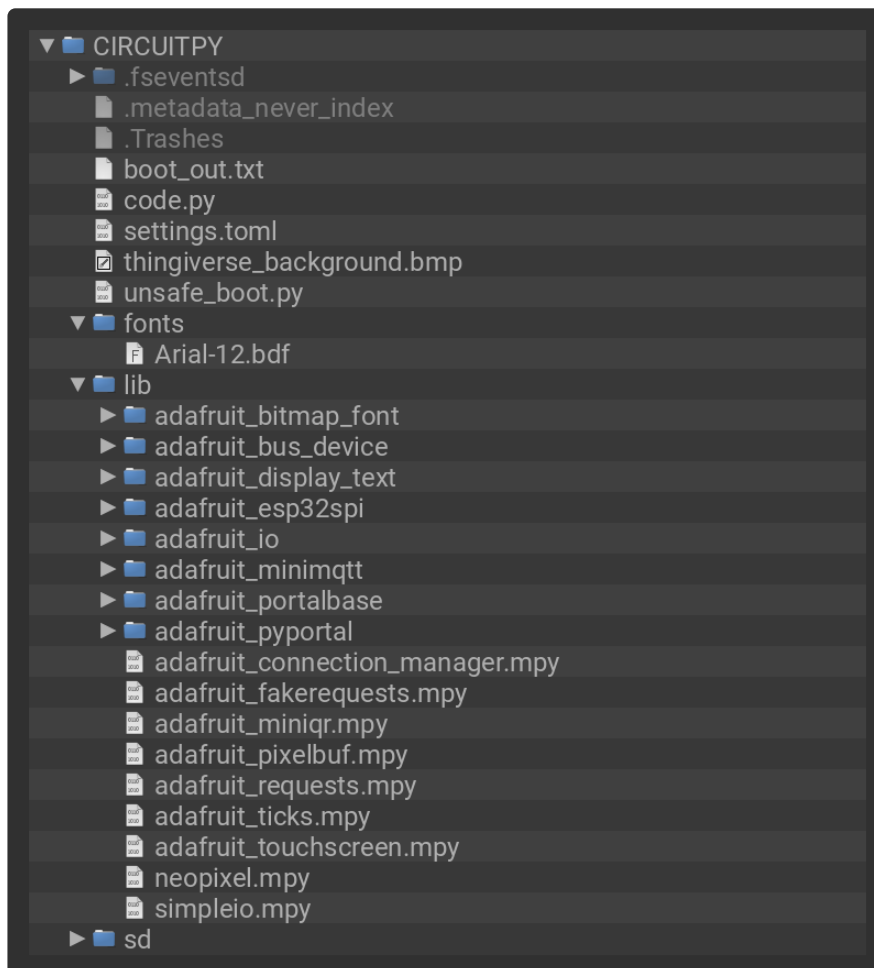
## boot.py

We're using a special file to ensure the .bmp cache writes to the flash properly. This is the `unsafe_boot.py` file you copied to the drive. Rename it to `boot.py` now.

Note that you'll see this scary looking text appear during restart, don't worry, it's supposed to say that!

```
***** WARNING *****  
Using the filesystem as a write-able cache!  
This is risky behavior, backup your files!  
***** WARNING *****
```

This is what the final contents of the `CIRCUITPY` drive will look like:



```
# SPDX-FileCopyrightText: 2019 Limor Fried for Adafruit Industries  
#  
# SPDX-License-Identifier: MIT  
  
import os  
import time  
import random  
import board  
import adafruit_pyportal
```

```

# Set up where we'll be fetching data from
NUM_THINGS=25 # how many things to select from (we randomize between em)
DATA_SOURCE = "https://api.thingiverse.com/users/adafruit/things?
per_page="+str(NUM_THINGS)
DATA_SOURCE += "&access_token=" + os.getenv("THINGIVERSE_TOKEN")
IMAGE_LOCATION = [0, "thumbnail"]
TITLE_LOCATION = [0, "name"]
URL_LOCATION = [0, "public_url"]

# determine the current working directory needed so we know where to find files
cwd = ("/" + __file__).rsplit('/', 1)[0]
pyportal = adafruit_pyportal.PyPortal(url=DATA_SOURCE,
                                     json_path=(TITLE_LOCATION, URL_LOCATION,
IMAGE_LOCATION),
                                     status_neopixel=board.NEOPIXEL,
                                     default_bg=cwd+"/thingiverse_background.bmp",
                                     text_font=cwd+"/fonts/Arial-12.bdf",
                                     text_position=((5, 10), (5, 230)),
                                     text_color=(0x00FF00, 0x00FF00),
                                     text_transform=(None, None))

pyportal.preload_font()

while True:
    response = None
    try:
        response = pyportal.fetch()
        print("Response is", response)
        pyportal.set_background(None)
        image_url = response[2].replace('_thumb_medium.', '_display_large.')
        pyportal.wget(pyportal.image_converter_url(image_url, 320,
240, color_depth=16),
                    "/cache.bmp",
                    chunk_size=512)
        pyportal.set_background("/cache.bmp")

    except (IndexError, RuntimeError, ValueError) as e:
        print("Some error occured, retrying! -", e)

    # next thingy should be random!
    thingy = random.randint(0, NUM_THINGS - 1)
    URL_LOCATION[0] = TITLE_LOCATION[0] = IMAGE_LOCATION[0] = thingy

    time.sleep(60 * 3) # cycle every 3 minutes

```

## How It Works

The Thingiverse Viewer works like this: first, when it starts up it connects to your Wifi access point as specified (and authenticated) in the **secrets.py** file.

### Background Splash Screen

Next, it displays the **thingiverse\_background.bmp** image file splash screen. This is a 320x240 pixel RGB 16-bit raster graphic in **.bmp** format.

### JSON

In order to retrieve images, we'll be making a query to the Thingiverse API.

When you make a request of the server, you'll get a JSON file returned as the response.



In fact, you can run the same query as the PyPortal does to see the result. Copy and paste this link `https://api.thingiverse.com/users/adafruit/things?per_page=1&access_token=` into your browser, where you paste your Thingiverse token in at the end of that URL.

When you enter this in your web browser, you'll see a result returned like this (in the actual code we request 25 things and then randomly pick which one to show, so you may see a different one than this):

```
[
  {
    "id": 3553481,
    "name": "Prop-Maker Keyblade",
    "url": "https://api.thingiverse.com/things/3553481",
    "public_url": "https://www.thingiverse.com/thing:3553481",
    "thumbnail": "https://cdn.thingiverse.com/renders/e2/a8/15/5b/c1/41109f706f73d0cbaa74e0f38e597173_thumb_medium.jpg",
    "creator": {
      "id": 491,
      "name": "adafruit",
      "first_name": "Adafruit",
      "last_name": "Industries",
      "url": "https://api.thingiverse.com/users/adafruit",
      "public_url": "https://www.thingiverse.com/adafruit",
      "thumbnail": "https://cdn.thingiverse.com/renders/30/1b/8b/a1/70/adafruit-tv-avatar_thumb_medium.jpg"
    },
    "is_private": false,
    "is_purchased": false,
    "is_published": true
  }
]
```

That result is a JSON (JavaScript Object Notation) array. It is comprised of a single element with a number **key:value** pairs, or sub-hierarchies of additional **key:value** pairs. For example, there is one **key** called `name` which has a **value** of `"Prop-Maker Keyblade"`

which is expressed this way:

```
"name": "Prop-Maker Keyblade"
```

Since this JSON object array has a consistent way to return the results to us, the code we're running on the PyPortal can easily parse the data and display it!

The `thumbnail` key leads to this **value**: `"https://cdn.thingiverse.com/renders/e2/a8/15/5b/c1/41109f706f73d0cbaa74e0f38e597173_thumb_medium.jpg"`

Here's the image at that url:



In our code, we'll do a string substitution in order to get the "\_display\_large" version of that image in place of the "\_thumb\_medium" version. Here's what that looks like:



We also get the `public_url` key, which is the short URL that will be displayed at the bottom of the screen, in this case, `"https://www.thingiverse.com/thing:3553481"`

You can see how it's done in this part of `code.py`:

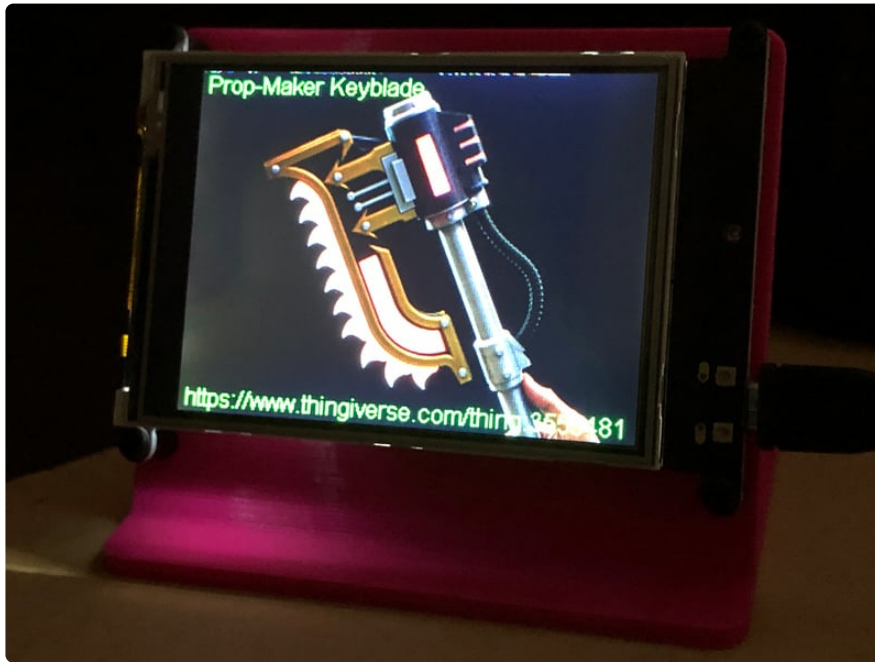
```
NUM_THINGS=25 # how many things to select from (we randomize between em)
DATA_SOURCE = "https://api.thingiverse.com/users/adafruit/things?
per_page="+str(NUM_THINGS)
DATA_SOURCE += "&access_token=" + secrets['thingiverse_token']
IMAGE_LOCATION = [0, "thumbnail"]
TITLE_LOCATION = [0, "name"]
URL_LOCATION = [0, "public_url"]
```

Then, in the `pyportal` query we ask for the `image` name from that URL to get the path to the .jpeg image file, as well as the `name` and `url` values.

```
pyportal = adafruit_pyportal.PyPortal(url=DATA_SOURCE,
                                       json_path=(TITLE_LOCATION, URL_LOCATION,
                                       IMAGE_LOCATION),
                                       status_neopixel=board.NEOPIXEL,
                                       default_bg=cwd+"/thingiverse_background.bmp",
                                       text_font=cwd+"/fonts/Arial-12.bdf",
                                       text_position=((5, 10), (5, 230)),
                                       text_color=(0x00FF00, 0x00FF00),
                                       text_transform=(None, None))
```

With all of this prepared, during the main loop of `while True:` the code will query the page for the JSON data.

The first of the 25 things will be selected. When it gets the path of the .jpeg file, the pyportal library passes it along to an Adafruit IO image converter server where the file is converted into the format the PyPortal can display, a 320x240 pixel RGB 16-bit .bmp.



This image is then cached onto the PyPortal's storage and displayed on the PyPortal TFT screen.

Then, the text for both the `name` and `public_url` values are drawn on top of the image background, using the specified font, color, and position in the PyPortal constructor.

Note, we're also specifying the `text_wrap` amount so that lines of text won't get cut off.

This updates every three minutes, so your Things stay fresh!

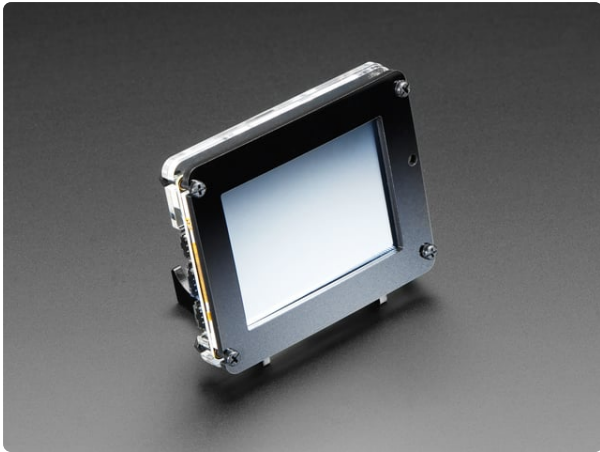
## Display Stand



Speaking of 3D printing -- if you'd like to make your own 3D printed PyPortal stand, check out the [Thingiverse model and instructions here \(https://adafru.it/EcN\)](https://adafru.it/EcN)!



If you have [AdaBox 011 \(http://adafru.it/4061\)](http://adafru.it/4061), the laser cut acrylic stand also works very well.



### Adafruit PyPortal Desktop Stand Enclosure Kit

PyPortal is our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Create little pocket...

<https://www.adafruit.com/product/4146>