PyPortal YouTube Views and Subscribers Display

Created by John Park

https://learn.adafruit.com/pyportal-youtube-views-and-subscribers-display

Last updated on 2023-08-29 04:02:17 PM EDT
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>• Parts</td>
<td></td>
</tr>
<tr>
<td>• Materials</td>
<td></td>
</tr>
<tr>
<td>Install CircuitPython</td>
<td>5</td>
</tr>
<tr>
<td>• Set up CircuitPython Quick Start!</td>
<td></td>
</tr>
<tr>
<td>• PyPortal Default Files</td>
<td></td>
</tr>
<tr>
<td>PyPortal CircuitPython Setup</td>
<td>8</td>
</tr>
<tr>
<td>• Adafruit CircuitPython Bundle</td>
<td></td>
</tr>
<tr>
<td>Internet Connect!</td>
<td>10</td>
</tr>
<tr>
<td>• What's a secrets file?</td>
<td></td>
</tr>
<tr>
<td>• Connect to WiFi</td>
<td></td>
</tr>
<tr>
<td>• Requests</td>
<td></td>
</tr>
<tr>
<td>• HTTP GET with Requests</td>
<td></td>
</tr>
<tr>
<td>• HTTP POST with Requests</td>
<td></td>
</tr>
<tr>
<td>• Advanced Requests Usage</td>
<td></td>
</tr>
<tr>
<td>• WiFi Manager</td>
<td></td>
</tr>
<tr>
<td>Code PyPortal with CircuitPython</td>
<td>24</td>
</tr>
<tr>
<td>• CircuitPython Code</td>
<td></td>
</tr>
<tr>
<td>• YouTube Token</td>
<td></td>
</tr>
<tr>
<td>• How It Works</td>
<td></td>
</tr>
<tr>
<td>• Keys</td>
<td></td>
</tr>
<tr>
<td>• PyPortal Constructor</td>
<td></td>
</tr>
<tr>
<td>• Fetch</td>
<td></td>
</tr>
<tr>
<td>• Ba-Ding!</td>
<td></td>
</tr>
<tr>
<td>• Customization</td>
<td></td>
</tr>
<tr>
<td>• Text Position</td>
<td></td>
</tr>
<tr>
<td>• Text Color</td>
<td></td>
</tr>
<tr>
<td>• Background Image</td>
<td></td>
</tr>
<tr>
<td>PyPortal YouTube Stand</td>
<td>36</td>
</tr>
<tr>
<td>• Mounting</td>
<td></td>
</tr>
<tr>
<td>• Attach Bezel</td>
<td></td>
</tr>
</tbody>
</table>
Overview

It's not all about the stats, right? Well, in YouTube land, maybe it is sometimes! If you're a little bit obsessed with tracking your stats, read on!

This PyPortal YouTube Views and Subscribers Display runs on CircuitPython and uses the Google APIs to grab JSON data and display your view count and subscriber count. And it does so through the thin air with its on-board WiFi ESP32 co-processor.

The text is displayed on top of a custom background graphic and mounted to a 3D printed stand with a matching retro-style red bezel!

Plus, you'll hear a satisfying "ba-ding!" each time you gain a new subscriber.
Parts

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

USB cable - USB A to Micro-B
This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...
https://www.adafruit.com/product/592

Materials

You may want to create the optional, stylish mounting stand for your PyPortal. For this you'll need:

• 3D printer ()
• Filament ()
• M2.5 screws ()

If you don't have access to a 3D printer you can optionally use an online service such as 3D Hubs to have it printed for you on demand.
Install CircuitPython

CircuitPython () is a derivative of MicroPython () 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

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

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

PyPortal Pynt Default Files

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
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_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.
Internet Connect!

Once you have CircuitPython setup and libraries installed we can get your board connected to the Internet. Note that access to enterprise level secured WiFi networks is not currently supported, only WiFi networks that require SSID and password.

To get connected, you will need to start by creating a secrets file.

What's a secrets file?

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a secrets.py file, that is in your CIRCUITPY drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

Your secrets.py file should look like this:

```python
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it
secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'github_token' : 'fawfj23rakjnfawiefa',
    'hackaday_token' : 'h4xx0rs3kret',
}
```

Inside is a python dictionary named secrets with a line for each entry. Each entry has an entry name (say 'ssid') and then a colon to separate it from the entry key ('home ssid') and finally a comma.

At a minimum you'll need the `ssid` and `password` for your local WiFi setup. As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing github or the hackaday API. Other non-secret data like your timezone can also go here, just cause it's called secrets doesn't mean you can't have general customization data in there!

For the correct time zone string, look at [http://worldtimeapi.org/timezones](http://worldtimeapi.org/timezones) and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.
Of course, don’t share your secrets.py - keep that out of GitHub, Discord or other project-sharing sites.

Connect to WiFi

OK now you have your secrets setup - 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:

![CIRCUITPY drive contents]

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

import board
import busio
from digitalio import DigitalInOut
import adafruit_requests as requests
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!"
raise
```
print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.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)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
requests.set_socket(socket, esp)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
    print("Firmware vers.", esp.firmware_version)
    print("MAC addr:", [hex(i) for i in esp.MAC_address])

for ap in esp.scan_networks():
    print("%s		RSSI: %d" % (str(ap["ssid"], "utf-8"), ap["rssi"]))

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except OSError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI: ", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_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("-")
    # print(r.text)
    # print("-")
    # r.close()

    print("Fetching json from", JSON_URL)
    r = requests.get(JSON_URL)
    print("-")
    print(r.json())
    print("-")
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 secrets.py 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.

In order, the example code...

Initializes the ESP32 over SPI using the SPI port and 3 control pins:

```python
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

Tells our requests library the type of socket we're using (socket type varies by connectivity type - we'll be using the adafruit_esp32spi_socket for this example).```
We'll also set the interface to an `esp` object. This is a little bit of a hack, but it lets us use `requests` like CPython does.

```python
requests.set_socket(socket, esp)
```

Verifies an ESP32 is found, checks the firmware and MAC address

```python
if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
    print("Firmware vers.", esp.firmware_version)
    print("MAC addr:", [hex(i) for i in esp.MAC_address])
```

Performs a scan of all access points it can see and prints out the name and signal strength:

```python
for ap in esp.scan_networks():
    print("%s		RSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))
```

Connects to the AP we've defined here, then prints out the local IP address, attempts 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)

```python
print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
print("IP lookup adafruit.com: %s" % esp.pretty_ip(esp.get_host_by_name("adafruit.com")))
```

OK now we're getting to the really interesting part. With a SAMD51 or other large-RAM (well, over 32 KB) device, we can do a lot of neat tricks. Like for example we can implement an interface a lot like `requests()` - which makes getting data really really easy

To read in all the text from a web URL call `requests.get()` - you can pass in `https` URLs for SSL connectivity

```python
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print('='*40)
print(r.text)
print('='*40)
r.close()
```
Or, if the data is in structured JSON, you can get the json pre-parsed into a Python dictionary that can be easily queried or traversed. (Again, only for nRF52840, M4 and other high-RAM boards)

```
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print(".*40")
print(r.json())
print(".*40")
r.close()
```

### Requests

We've written a requests-like() library for web interfacing named Adafruit_CircuitPython_requests(). This library allows you to send HTTP/1.1 requests without "crafting" them and provides helpful methods for parsing the response from the server.

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.

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

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

# adafruit_requests usage with an esp32spi_socket
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
```
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# Add a secrets.py to your filesystem that has a dictionary called secrets with
"ssid" and
"password" keys with your WiFi credentials. DO NOT share that file or commit it
into Git or other
# source control.
# pylint: disable=no-name-in-module,wrong-import-order
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# 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)
est = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

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

# Initialize a requests object with a socket and esp32spi interface
socket.set_interface(esp)
requests.set_socket(socket, esp)

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_GET_URL = "https://httpbin.org/get"
JSON_POST_URL = "https://httpbin.org/post"

print("Fetching text from %s" % TEXT_URL)
response = requests.get(TEXT_URL)
print("\"\" * 40)
print("Text Response: ", response.text)
print("\"\" * 40)
response.close()

print("Fetching JSON data from %s" % JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print("\"\" * 40)
print("JSON Response: ", response.json())
print("\"\" * 40)
response.close()
data = "31F"
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
The code first sets up the ESP32SPI interface. Then, it initializes a request object using an ESP32 socket and the esp object.

HTTP GET with Requests

The code makes a HTTP GET request to Adafruit’s WiFi testing website - http://wifitest.adafruit.com/testwifi/index.html().
To do this, we'll pass the URL into `requests.get()`. We're also going to save the response from the server into a variable named `response`.

Having requested data from the server, we'd now like to see what the server responded with. Since we already saved the server's `response`, we can read it back. Luckily for us, requests automatically decodes the server's response into human-readable text, you can read it back by calling `response.text`.

Lastly, we'll perform a bit of cleanup by calling `response.close()`. This closes, deletes, and collect's the response's data.

```python
print("Fetching text from %s"%TEXT_URL)
response = requests.get(TEXT_URL)
print(''.join(['-']*40))

print("Text Response: ", response.text)
print(''.join(['-']*40))
response.close()
```

While some servers respond with text, some respond with json-formatted data consisting of attribute–value pairs.

CircuitPython_Requests can convert a JSON-formatted response from a server into a CPython `dict` object.

We can also fetch and parse json data. We'll send a HTTP get to a url we know returns a json-formatted response (instead of text data).

Then, the code calls `response.json()` to convert the response to a CPython `dict`.

```python
print("Fetching JSON data from %s"%JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print(''.join(['-']*40))

print("JSON Response: ", response.json())
print(''.join(['-']*40))
response.close()
```

**HTTP POST with Requests**

Requests can also POST data to a server by calling the `requests.post` method, passing it a `data` value.

```python
data = '31F'
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
```

©Adafruit Industries  Page 18 of 42
print('-'*40)
jsonResp = response.json()
# Parse out the 'data' key from jsonResp dict.
print("Data received from server:'", jsonResp['data'])
print('-'*40)
response.close()

You can also post json-formatted data to a server by passing json_data into the
requests.post method.

json_data = {"Date" : "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print('-'*40)

jsonResp = response.json()
# Parse out the 'json' key from jsonResp dict.
print("JSON Data received from server:'", jsonResp['json'])
print('-'*40)
response.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.

Your CIRCUITPY drive should now look similar to the following image:
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# Add a secrets.py to your filesystem that has a dictionary called secrets with
# "ssid" and
# "password" keys with your WiFi credentials. DO NOT share that file or commit it
# into Git or other
# source control.
# pylint: disable=no-name-in-module,wrong-import-order
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# 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)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
    print("Connected to ", str(esp.ssid, "utf-8"), " \tRSSI: ", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
socket.set_interface(esp)
requests.set_socket(socket, esp)

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

# Define a custom header as a dict.
headers = {"user-agent": "blinking/1.0.0"}
print("Fetching JSON data from %s..." % JSON_GET_URL)  
response = requests.get(JSON_GET_URL, headers=headers)  
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)  

# Close, delete and collect the response data  
response.close()

WiFi Manager

That simplest test example works but it's a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. 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 the WiFi manager and also how to POST data with some extra headers:

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.

Your CIRCUITPY drive should now look similar to the following image:
import time
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
print("ESP32 SPI webclient test")
# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise
# 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)
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

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

status_light = neopixel.NeoPixel(
    board.NEOPIXEL, 1, brightness=0.2
)  # Uncomment for Most Boards
"""Uncomment below for Most Boards"
"""status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
# Uncomment below for an externally defined RGB LED
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp32, 26)
# GREEN_LED = PWMOut.PWMOut(esp32, 27)
# BLUE_LED = PWMOut.PWMOut(esp32, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)
counter = 0
while True:
    try:
        print("Posting data...", end="")
        data = counter
        feed = "test"
        payload = {"value": data}
        response = wifi.post("https://io.adafruit.com/api/v2/"
            + secrets["aio_username"]
            + "/feeds/
            + feed
            + "/data",
        json=payload,
        headers=("X-AIO-KEY": secrets["aio_key"])),
        )
        print(response.json())
        response.close()
        counter = counter + 1
        print("OK")
    except OSError as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        continue
        response = None
        time.sleep(15)

You'll note here we use a secrets.py file to manage our SSID info. The wifimanager is
given the ESP32 object, secrets and a neopixel for status indication.

Note, you'll need to add a some additional information to your secrets file so that the
code can query the Adafruit IO API:

- **aio_username**
- **aio_key**

You can go to your adafruit.io View AIO Key link to get those two values and add
them to the secrets file, which will now look something like this:

```python
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid': '_your_ssid_",
    'password': '_your_wifi_password_",
    'timezone': "America/Los_Angeles", # http://worldtimeapi.org/timezones
    'aio_username': '_your_aio_username_",
    'aio_key': '_your_aio_key_",
}
```

©Adafruit Industries
Next, set up an Adafruit IO feed named **test**

- If you do not know how to set up a feed, [follow this page and come back when you've set up a feed named](#) **test**.

We can then have a simple loop for posting data to Adafruit IO without having to deal with connecting or initializing the hardware!

Take a look at your test feed on Adafruit.io and you'll see the value increase each time the CircuitPython board posts data to it!

![Graph of test feed](#)

**Code PyPortal with CircuitPython**

**CircuitPython Code**

In the embedded code element below, click on the Download: Project Zip link, and save the .zip archive file to your computer.

Then, uncompress the .zip file, it will unpack to a folder named PyPortal_Youtube.

Copy the contents of the PyPortal_Youtube directory to your PyPortal's CIRCUITPY drive.
This is what the final contents of the CIRCUITPY drive will look like:

<table>
<thead>
<tr>
<th>fonts</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Collegiate-24.bdf</td>
<td></td>
</tr>
<tr>
<td>Collegiate-50.bdf</td>
<td></td>
</tr>
<tr>
<td>lib</td>
<td></td>
</tr>
<tr>
<td>adafruit_bitmap_font</td>
<td></td>
</tr>
<tr>
<td>adafruit_bus_device</td>
<td></td>
</tr>
<tr>
<td>adafruit_display_text</td>
<td></td>
</tr>
<tr>
<td>adafruit_esp32spi</td>
<td></td>
</tr>
<tr>
<td>adafruit_image/load</td>
<td></td>
</tr>
<tr>
<td>adafruit_act7410.mpy</td>
<td></td>
</tr>
<tr>
<td>adafruit_pyportal.py</td>
<td></td>
</tr>
<tr>
<td>adafruit_sdcard.mpy</td>
<td></td>
</tr>
<tr>
<td>adafruit_slideshow.mpy</td>
<td></td>
</tr>
<tr>
<td>adafruit_touchscreen.mpy</td>
<td></td>
</tr>
<tr>
<td>neopixel.mpy</td>
<td></td>
</tr>
<tr>
<td>boot_out.txt</td>
<td></td>
</tr>
<tr>
<td>code.py</td>
<td></td>
</tr>
<tr>
<td>Coin.wav</td>
<td></td>
</tr>
<tr>
<td>secrets.py</td>
<td></td>
</tr>
<tr>
<td>youtube_background.bmp</td>
<td></td>
</tr>
</tbody>
</table>

YouTube Token

To prevent unwanted bot traffic, Google requires each query of the YouTube stats to be accompanied by a YouTube token.

First, you'll need to get set up with your YouTube Data API key. Follow these instructions to get this set up.
Click the dropdown menu for Select a project.

Then, click the New Project link at the upper right corner.

On the next page, give your project a name, such as **PyPortal-YouTube-Stats** and then click CREATE.
Next, click the + ENABLE APIS AND SERVICES link at the top of the page.

Now, you’ll be able to click the Library link on the left hand side so you can select the YouTube Data API v3 from the available libraries.

Click on the YouTube Data API v3 graphic button.

On the next page, click ENABLE.
Now we'll click on the CREATE CREDENTIALS button.

Choose the YouTube Data API v3 item from the first dropdown menu.

In the second menu, choose Other non-UI (e.g. cron job, daemon)

Then, click the Public data radio button.

Finally, click the What credentials do I need? button to get your API key.
One last setting to enable - click on the API restrictions tab for your API key and set the dropdown to YouTube Data API v3. Then click Save.

Your credentials will be generated and you should now copy the Key value and paste it into your `secrets.py` file.

Here's how to enter the token into your `secrets.py` file.

```python
secrets = {
    'ssid' : '_your_wifi_ssid_',
    'password' : '_your_wifi_password_',
    'youtube_token' : 'HUGE_LONG_YOUTUBE_API_TOKEN'
}
```

# SPDX-FileCopyrightText: 2019 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

This example will access the youtube API, grab a number like number of views or subscribers... and display it on a screen

If you can find something that spits out JSON data, we can display it!

Requires a youtube API key!

```python
import time
import board
from adafruit_pyportal import PyPortal

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# pylint: disable=line-too-long
DATA_SOURCE = "https://www.googleapis.com/youtube/v3/channels/?part=statistics&id=CHANNEL_ID&key=secrets['youtube_token']"
DATA_LOCATION1 = ["items", 0, "statistics", "viewCount"]
DATA_LOCATION2 = ["items", 0, "statistics", "subscriberCount"]

# pylint: enable=line-too-long

# the current working directory (where this file is)
cwd = ("/"+_file_).rsplit('/', 1)[0]
pyportal = PyPortal(url=DATA_SOURCE,
# track the last value so we can play a sound when it updates
last_subs = 0

while True:
    try:
        views, subs = pyportal.fetch()
        subs = int(subs)
        views = int(views)
        print("Subscribers:", subs)
        print("Views:", views)
        if last_subs < subs:  # ooh it went up!
            print("New subscriber!"")
            pyportal.play_file(cwd+"/coin.wav")
        last_subs = subs
    except RuntimeError as e:
        print("Some error occured, retrying! -", e)
        time.sleep(60)

If you run into any errors, such as "ImportError: no module named 'adafruit_display_text.label'" be sure to update your libraries to the latest release bundle!

How It Works

The PyPortal YouTube display is doing a few things to display the stats you want.

Background

First, it displays a bitmap graphic named youtube_background.bmp as the screen's background. This is a 320 x 240 pixel RGB 16-bit raster graphic in .bmp format.
Font

Then, it displays the YouTube account's name as a caption, created with bitmapped fonts to overlay on top of the background. The fonts used here is are bitmap fonts made from the Collegiate typeface. You can learn more about [converting type in this guide](#).

Next, the PyPortal will display the current number of Views and Subscribers for the account.

**JSON**

To keep things current, the stats are grabbed from the Google APIs website itself.

Google automatically generates a JSON file for each YouTube channel, in this case at the address: https://www.googleapis.com/youtube/v3/channels/?part=statistics&id=UCpOlOeQjj7EsVnDh3zuCgsA&key=YOUR_HUGE_LONG_YOUTUBE_API_TOKEN (Be sure to replace that last part with your actual token!)

This file contains all sorts of information, delivered in an easy-to-parse format. If you visit that URL in Firefox, you'll see the JSON data it returns. You can look at the "beautified" version and the raw data to compare by clicking the buttons at top.
Here it is in a raw-er form, but still using indentation and carriage returns to make it readable:

```json
{
  "kind": "youtube#channelListResponse",
  "etag": "\"XpPGQXPnxQJhLgs6enD_n8JR4Qk/XhYU4i1Tm18Eso0o_j9lmVMs5Va8\"",
  "pageInfo": {
    "totalResults": 1,
    "resultsPerPage": 1
  },
  "items": [
    {
      "kind": "youtube#channel",
      "etag": "\"XpPGQXPnxQJhLgs6enD_n8JR4Qk/Yv8z0GjQd8i90B_gmDeXP4c--W4\"",
      "id": "UCpOlOeQjj7EsVnD3zuCgsA",
      "statistics": {
        "viewCount": "55355282",
        "commentCount": "0",
        "subscriberCount": "310178",
        "hiddenSubscriberCount": false,
        "videoCount": "3784"
      }
    }
  ]
}
```

**Keys**

If we look through the JSON file, we'll see a key called `items` with a sub-tree below it hierarchically called `0` and a sub-tree below that called `statistics`. Within this key we see the two key : value pairs that we want, `viewCount` that has a value of 55355282 and `subscriberCount` with a value of 310178.

The raw JSON for these key : value pairs look like this: 

```
"viewCount": 157262
"subscriberCount": 310178
```

Our CircuitPython code is able to grab and parse this data using these variables:
You can get your own Channel ID by logging in to YouTube and then going to your advanced account settings page().

There, you'll see your YouTube Channel ID, which you can copy and paste into the code.py file's CHANNEL_ID variable to track that channel's stats.

PyPortal Constructor

When we set up the pyportal constructor, we are providing it with these things:

- url to query
- json_path to traverse and find the key:value pair we need
- status_neopixel pin
- default_bg path and name to display the background bitmap
- text_font path and name to the font used for displaying the follower count value
- text_position on the screen's x/y coordinate system
- text_color
- caption_text to display statically -- in this case the name of the repo
- caption_font
- caption_position
- caption_color

Fetch

With the pyportal set up, we can then use pyportal.fetch() to do the query and parsing of the two pieces of YouTube data and then display them on screen along with the caption text on top of the background image.

Ba-Ding!

Additionally, we use the last_value variable's state to compare against the latest value. If they differ, we play the coin.wav file for a satisfying ding over the PyPortal's built in speaker!

To make your own .wav files, check out this guide().
Customization

You can customize this project to make it your own and point to different website APIs as the source of your JSON data, as well as adjust the graphics and text.

Text Position

Depending on the design of your background bitmap and the length of the text you’re displaying, you may want to reposition the text and caption. You can do this with the `text_position` and `caption_position` options.

The PyPortal's display is 320 pixels wide and 240 pixels high. In order to refer to those positions on the screen, we use an x/y coordinate system, where x is horizontal and y is vertical.

The origin of this coordinate system is the upper left corner. This means that a pixel placed at the upper left corner would be (0,0) and the lower right corner would be (320, 240).
So, if you wanted to move the subscriber count text to the right and up closer to the top, your code may look like this for that part of the `pyportal` constructor:

```python
text_position=(250, 10)
```

**Text Color**

Another way to customize your stats trophy is to adjust the color of the text. The line `text_color=0xFFFFFF` in the constructor shows how. You will need to use the hexadecimal value for any color you want to display.

You can use something like [https://htmlcolorcodes.com/](https://htmlcolorcodes.com/) to pick your color and then copy the hex value, in this example it would be `0x0ED9EE`. 

![Color code example](image-url)
Background Image

If you would like to create your own background, awesome! You'll want to save the file with these specifications:

- 320 x 240 pixels
- 16-bit RGB color
- Save file as .bmp format

You can then copy the .bmp file to the root level of the CIRCUITPY drive. Make sure you refer to this new filename in the pyportal constructor line:

```
default_bg=cwd+"/youtube_background.bmp"
```

Change that line to use the new filename name, such as:

```
default_bg=cwd+"/my_new_background.bmp"
```

Now, we'll look at mounting the PyPortal onto a stylish stand!

PyPortal YouTube Stand

We can 3D print a nice display stand for the PyPortal and add a YouTube themed TV screen-styled bezel to the front.
First, 3D print the PyPortal Stand based on the Ruiz Bros. model and instructions here.
Mounting

Add nylon 2.5mm short hex standoffs and nuts to the stand as shown here.
Next, print the PyPortal YouTube Bezel model linked here.

Attach Bezel

Align the bezel with the front of the PyPortal so that the light sensor is visible through the bezel's side hole.

Use the four nylon M2.5 screws to fasten the bezel and PyPortal to the hex standoffs and stand.
Now, plug it into USB power and you're ready to go!