League of Legends Level Trophy for PyPortal

Created by John Park

https://learn.adafruit.com/league-of-legends-level-trophy-for-pyportal

Last updated on 2025-11-05 02:26:39 PM EST

Table of Contents

Overview 3

- Parts

Install CircuitPython 5

- Set up CircuitPython Quick Start!
- PyPortal Default Files

PyPortal CircuitPython Setup 8

- Adafruit CircuitPython Bundle

Create Your settings.toml File 9

- CircuitPython settings.toml File
- settings.toml File Tips
- Accessing Your settings.toml Information in code.py

Internet Connect! 12

- Connect to WiFi
- Advanced Requests Usage
- WiFi Manager
- Further Information

Code League Level Viewer 21

- Riot API Key
- Add CircuitPython Code and Assets
- Editing the Code
- Add Summoner Name
- How It Works
- Personal App

Build the Trophy 31

- Prep
- Mounting
- Alternate Mounting

Overview

Proudly display your League of Legends Summoner's Level on the PyPortal display, with automatic updates!

This project uses the PyPortal's built-in WiFi co-processor to import up-to-the-moment stats on League of Legends! Using the Riot Games API, you can automatically keep your Summoner's level front and center. Or, enter the name of a different Summoner in the easy-to-edit CircuitPython code!

Mount the PyPortal to a trophy or other display stand for a great look.

Parts

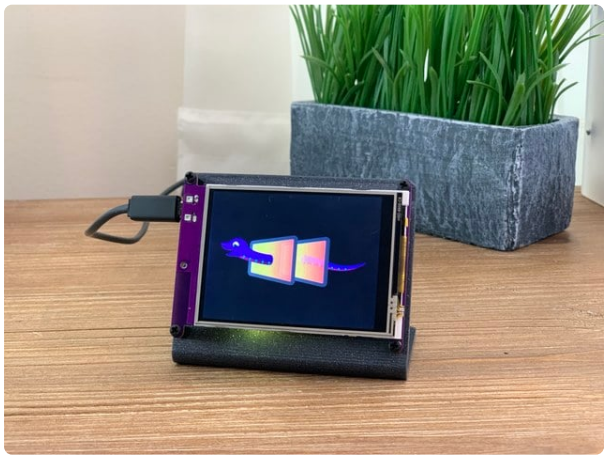
All the parts are in AdaBox 011 but this can be made from separate parts also.

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

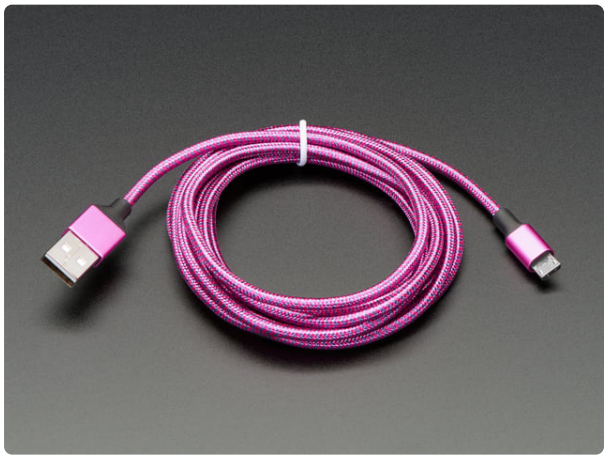
Separate Parts



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



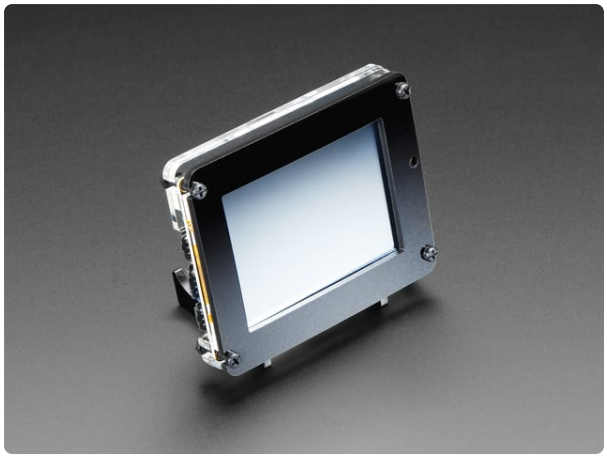
Pink and Purple Braided USB A to Micro B Cable - 2 meter long

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

Optional

You can build your own stand with a trophy and zip ties, or use the stand listed below:



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

Install CircuitPython

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (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

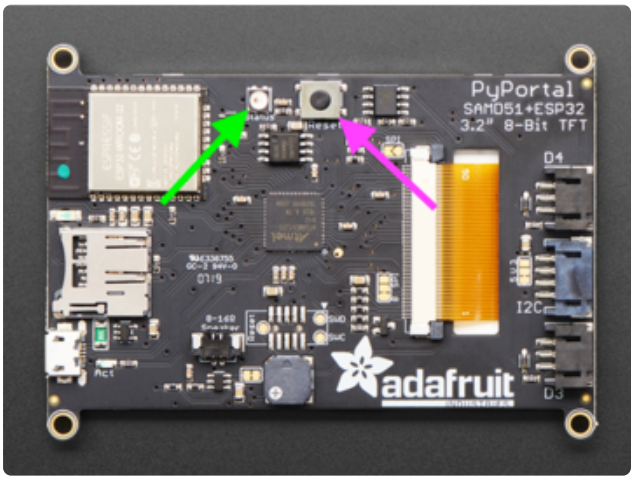
Download the latest version of CircuitPython for the PyPortal Pynt via CircuitPython.org

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).

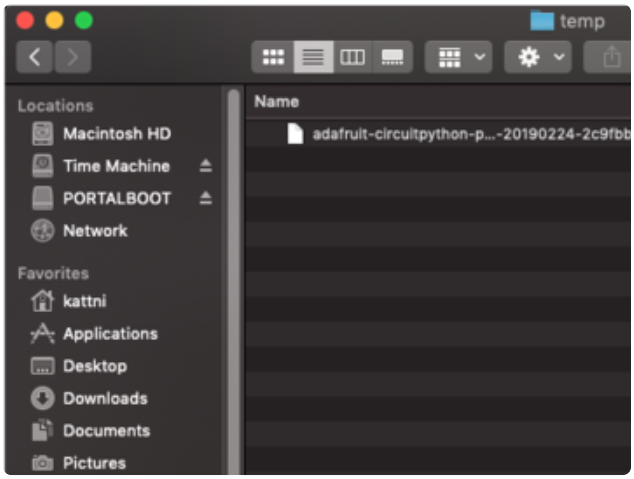
Plug your PyPortal into your computer using a known-good USB cable.

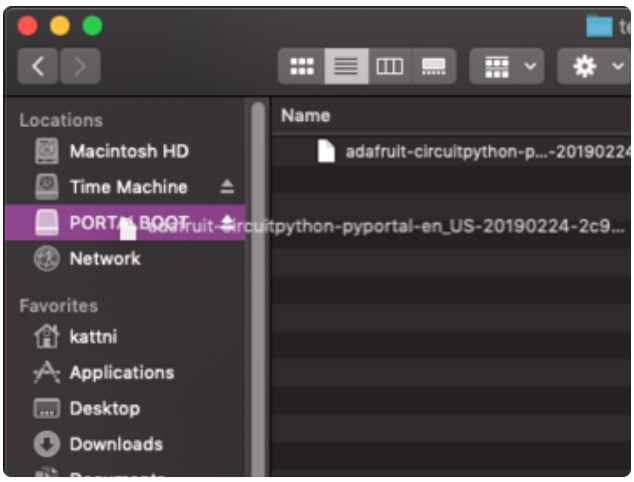


A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

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!

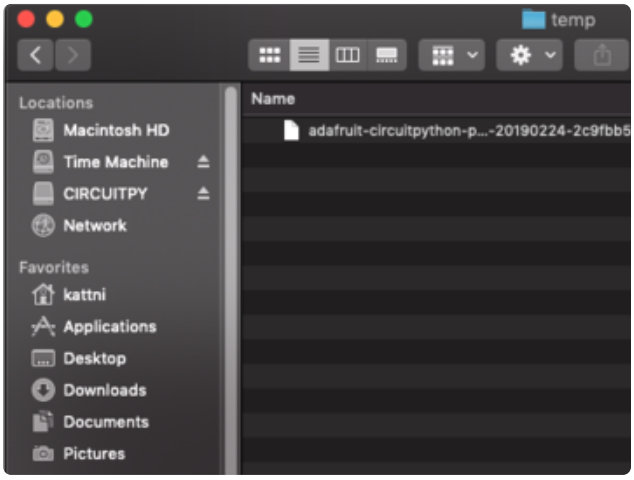
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!





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), 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:

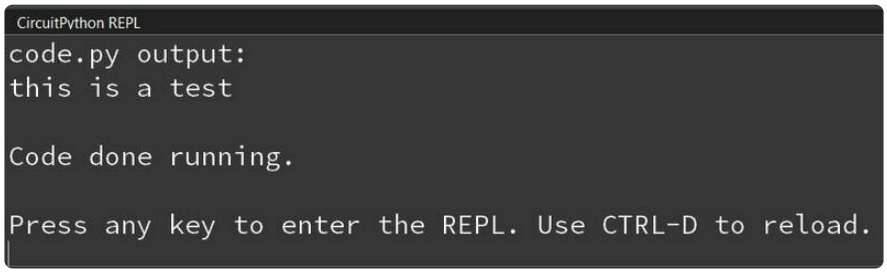
- Strings are wrapped in double quotes; ex: `"your-string-here"`
- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
    - Floats (decimal numbers), octal (`0o567`) and binary (`0b11011`) are not supported.
- Use `\u` escapes 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"))
```



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!

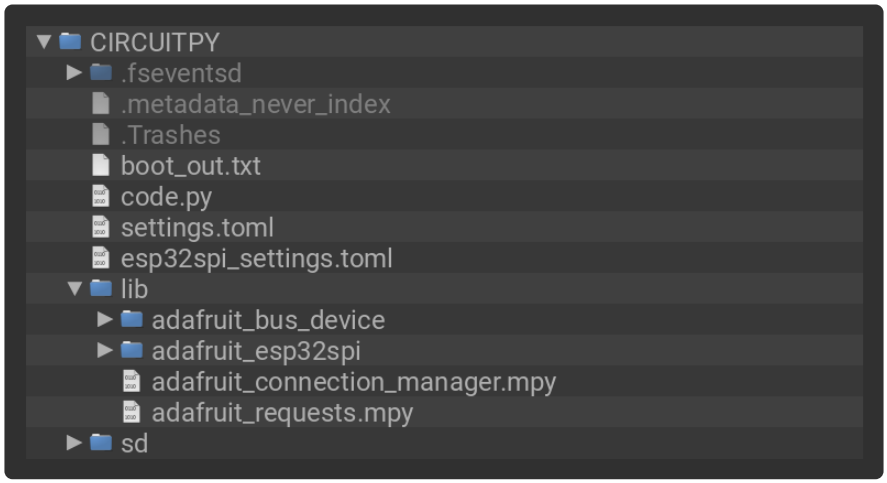
Connect to WiFi

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 adafruit_connection_manager
import adafruit_requests
import board
import busio
from digitalio import DigitalInOut

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).

```
&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). It is a lot like the regular Python library named requests (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.

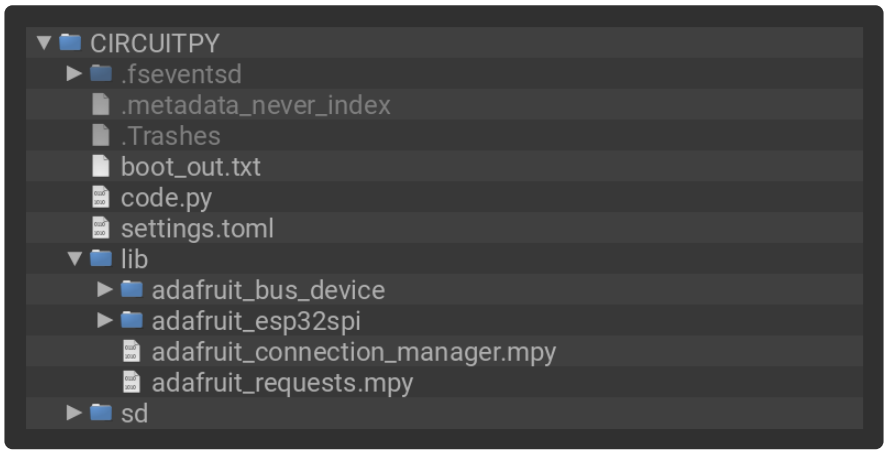
To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

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.ap_info.ssid, "utf-8"), "\tRSSI:",
radio.ap_info.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.

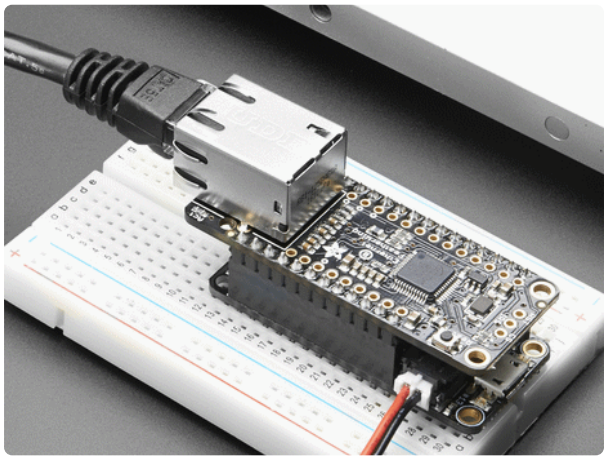
```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
from os import getenv

import board
import busio
import neopixel
import rtc
from digitalio import DigitalInOut

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
By Anne Barela

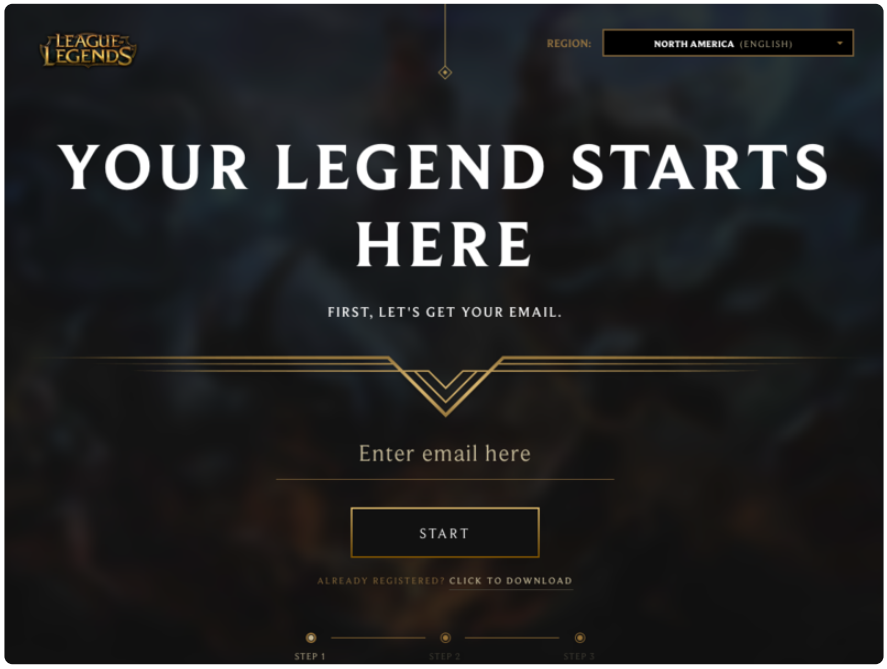
https://learn.adafruit.com/networking-in-circuitpython

Code League Level Viewer

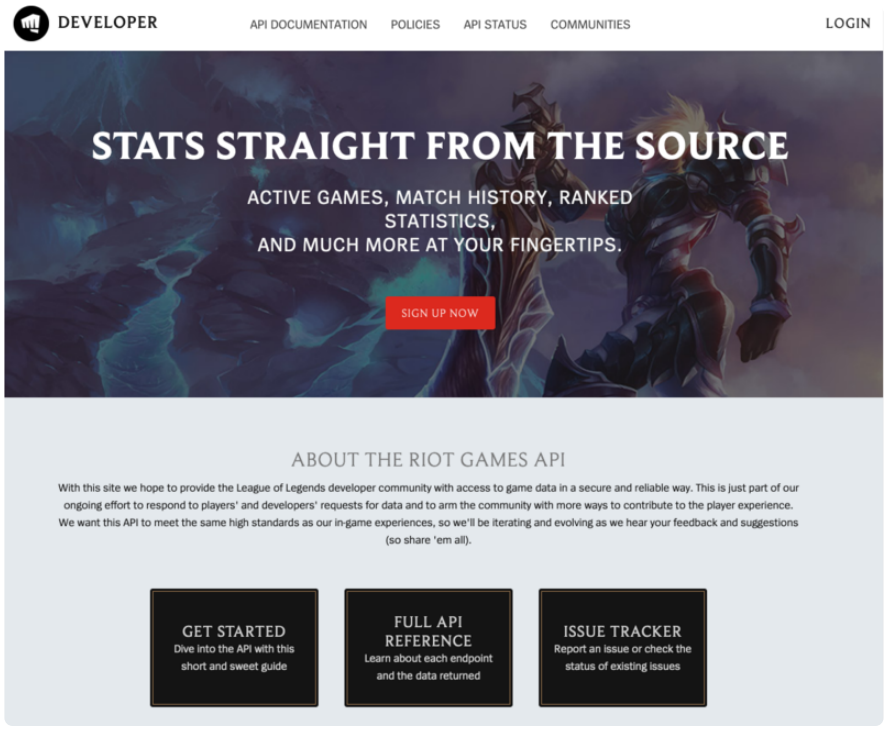
Riot API Key

We'll use the **Riot Games developer API** to make our queries and retrieve our Summoner's level statistic. In order to do so, you'll need to register for a free account with Riot Games and get your API key.

First, you'll need to have a League of Legends account login, and have played the game so that you have selected a Summoner name. [Head here to get started or to log in (https://adafru.it/EyV)](https://adafru.it/EyV).

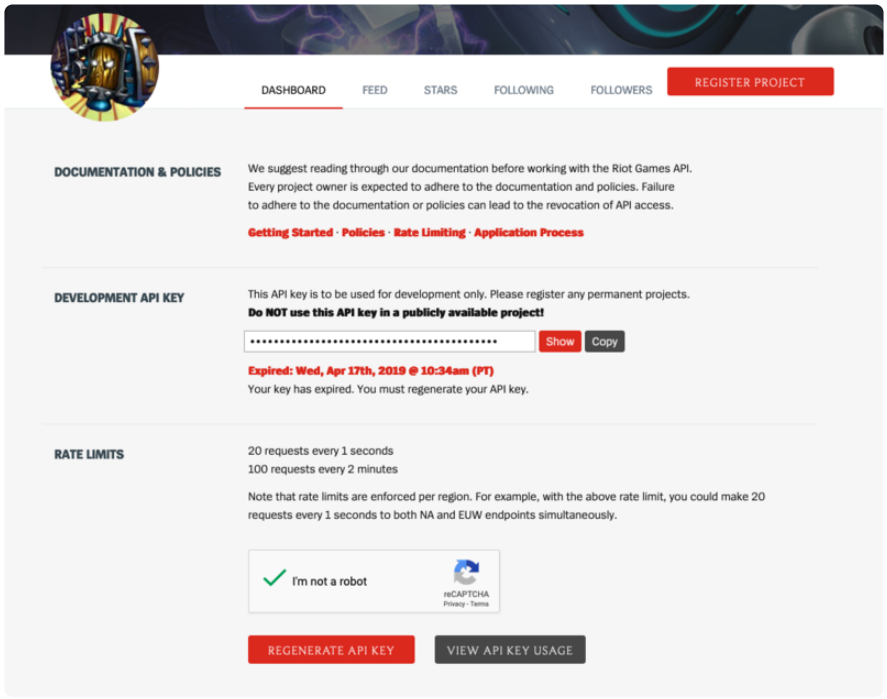


Once you've verified your account via email, and created your Summoner name, you can proceed to create a developer account. Head here to do so [https://developer.riotgames.com/ (https://adafru.it/EyW)](https://developer.riotgames.com/)



When you create your developer account, you'll be given a development API key. Keep that key (or the email containing it) handy, we'll need to copy and paste it into our code in a moment. The key will appear in your Riot Games dashboard for future reference.

Note, this key will expire after 24 hours and needs to be regenerated. It will get us going at first, but later we'll look at creating a **Personal App** with a permanent key.



You'll need to place the API key in your **settings.toml** file as shown below using a text editor or Mu:

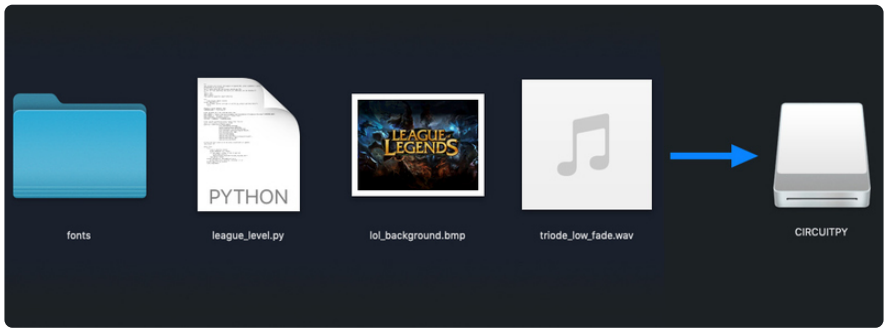
```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
LEAGUE_TOKEN = "your_very_long_riot_dev_api_key"
```

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_LeagueLevel**.

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




```
# SPDX-FileCopyrightText: 2019 John Edgar Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
This project will access the League of Legends API, grab a Summoner's Level
and display it on a screen.
You'll need a Riot API key in your settings.toml file
If you can find something that spits out JSON data, we can display it!
"""

from os import getenv
import time
import board
from adafruit_pyportal import PyPortal

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

if None in [ssid, password]:
    raise RuntimeError(
        "WiFi settings are kept in settings.toml, "
        "please add them there. The settings file must contain "
        "'CIRCUITPY_WIFI_SSID', 'CIRCUITPY_WIFI_PASSWORD', "
        "at a minimum."
```

```
    )

#Choose a valid Summoner name
SUMMONER_NAME = "RiotSchmick"

# Set up where we'll be fetching data from
DATA_SOURCE = "https://na1.api.riotgames.com/lol/summoner/v4/summoners/by-
name/"+SUMMONER_NAME
DATA_SOURCE += "?api_key=" + getenv("LEAGUE_TOKEN")
DATA_LOCATION = ["summonerLevel"]
CAPTION = "SUMMONER  "+SUMMONER_NAME

# the current working directory (where this file is)
cwd = ("/"+__file__).rsplit('/', 1)[0]
pyportal = PyPortal(url=DATA_SOURCE,
                    json_path=(DATA_LOCATION),
                    status_neopixel=board.NEOPIXEL,
                    default_bg=cwd+"/lol_background.bmp",
                    text_font=cwd+"/fonts/Collegiate-50.bdf",
                    text_position=(135, 200),
                    text_color=0xffbe33,
                    caption_text=CAPTION,
                    caption_font=cwd+"/fonts/Collegiate-24.bdf",
                    caption_position=(5, 20),
                    caption_color=0xffbe33)

# track the last value so we can play a sound when it updates
last_value = 0

while True:
    try:
        value = pyportal.fetch()
        print("Response is", value)
        if last_value < value:  # ooh it went up!
            print("New level!")
            pyportal.play_file(cwd+"/triode_low_fade.wav")
        last_value = value
    except (ValueError, RuntimeError, ConnectionError, OSError) as e:
        print("Some error occurred, retrying! -", e)
    #check again in two minutes
    time.sleep(60*2)
```

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).

Add Summoner Name

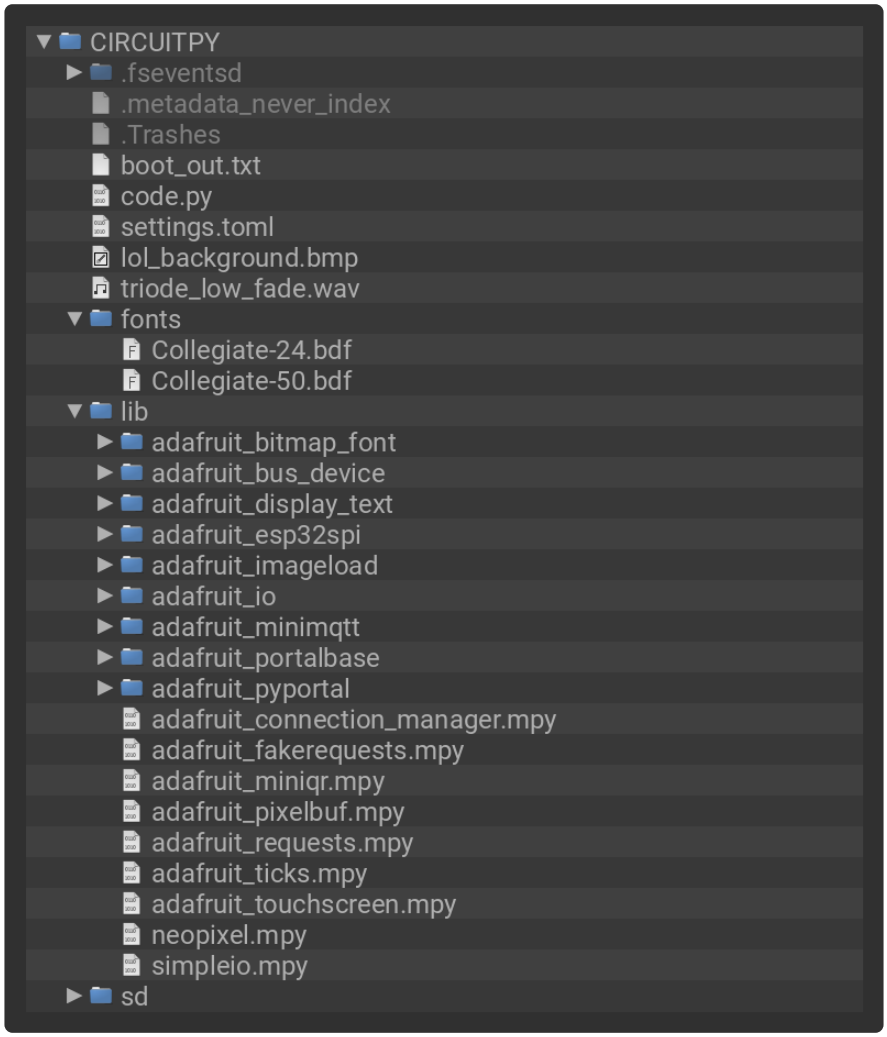
Open up **code.py** in Mu and then edit the `SUMMONER_NAME` variable to whichever one you want to display.

For the Summoner named **Ohbowz**, that line will look something like this:

```
SUMMONER_NAME = "Ohbowz"
```

When you're done, save the **code.py** file again to your PyPortal's **CIRCUITPY** drive.

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



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 League of Legends Level Viewer is doing a few cool things using CircuitPython and the PyPortal:

Background Splash Screen

First, we'll display a splash screen with the LoL logo. This is a 320x240 pixel RGB 16-bit raster graphic in **.bmp** format.

Font

We'll be displaying the image title and today's date as text created with a bitmapped font to overlay on top of the background image once that's loaded. The font used here is a bitmap font made from the Collegiate typeface. You can learn more about converting type in this guide (https://adafru.it/E7E).

JSON

In order to retrieve the Summoner's level statistic, we'll be making a query to the Riot Games API.

When you make a request of the with your API key added, 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://na1.api.riotgames.com/lol/summoner/v4/summoners/by-name/RiotSchmick?api_key=`

into your browser, and then copy and past your API key to the end.

When you enter this in your web browser, you'll see a result returned like this:

```
    {
  "id": "b6zzOOT1ATbtZtnX9nQ1HLoUNEfOZA8Mr9565oYyfhqO",
  "accountId": "9Fxgj1Zf5SZSNUWGBcy-aIlviyRwal9t8U8f5nMb-FVpMw",
  "puuid": "MNEzp8-ybiX49XIssYwrnuVgxDPvaeeYU8AFSKWcV7FnvRHaL8H5wb2HqrbwEU-
HOvNXfINHU0rzAw",
```

```
  "name": "RiotSchmick",
  "profileIconId": 3872,
  "revisionDate": 1555377428000,
  "summonerLevel": 138
}
```

That result is a JSON (JavaScript Object Notation) array. It is comprised of a single element with seven **key:value** pairs. For example, there is one **key** called `name` which has a **value** of `RiotSchmick`

which is expressed this way:

`"name": "RiotSchmick"`

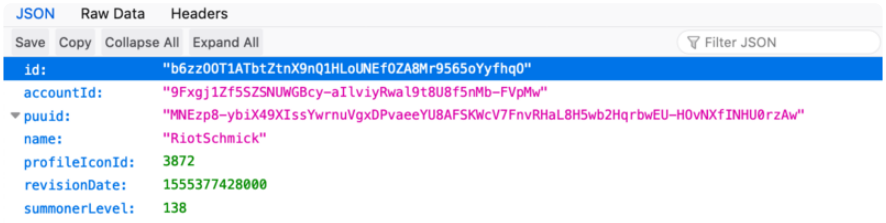
And another we care about is the `summonerLevel` **key**, which has a **value** of `138`

So that is expressed as:

`"summonerLevel": 138`

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!

Here's what the JSON file looks like in the "code beautifier" of Firefox.



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

```
SUMMONER_NAME = "RiotSchmick"
DATA_SOURCE = "https://na1.api.riotgames.com/lol/summoner/v4/summoners/by-
name/"+SUMMONER_NAME
DATA_SOURCE += "?api_key=" + os.getenv("LEAGUE_TOKEN")
DATA_LOCATION = ["summonerLevel"]
CAPTION = "SUMMONER  "+SUMMONER_NAME
```

Then, in the `pyportal` query we ask for the **summonerLevel** for a given Summoner **name** from that URL, and then use the `text_` arguments to set the `font`, `position`, and `color` of the text when it is displayed.

```
pyportal = PyPortal(url=DATA_SOURCE,
                    json_path=(DATA_LOCATION),
                    status_neopixel=board.NEOPIXEL,
                    default_bg=cwd+"/lol_background.bmp",
                    text_font=cwd+"/fonts/Collegiate-50.bdf",
                    text_position=(135, 200),
                    text_color=0xffbe33,
                    caption_text=CAPTION,
                    caption_font=cwd+"/fonts/Collegiate-24.bdf",
                    caption_position=(5, 20),
                    caption_color=0xffbe33)
```

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

Finally, the text will be displayed over the image.

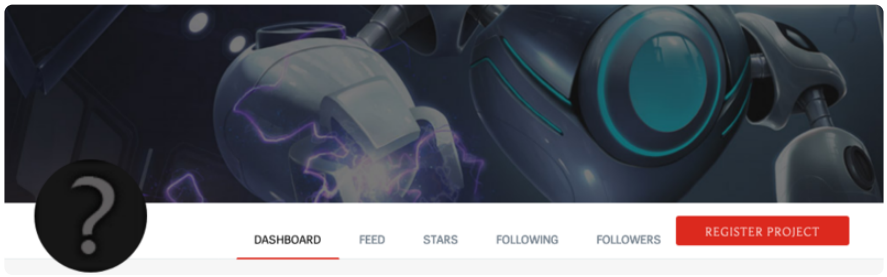
This updates every two minutes in order to stay current.

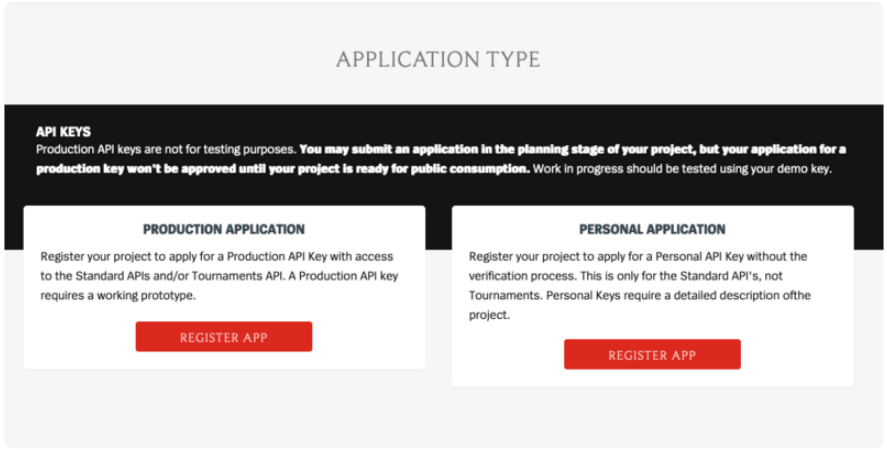
Clearly my Gulpfish Summoner has some leveling up to do.

Personal App

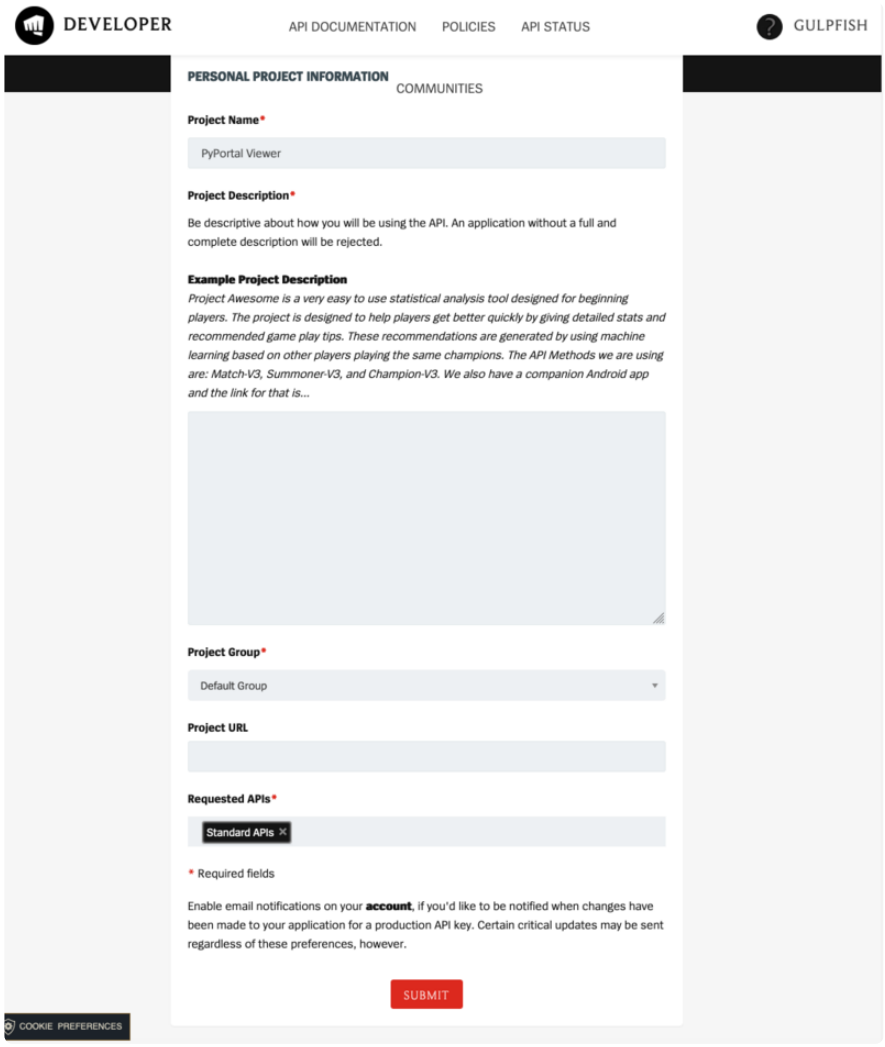
In order to use a permanent API key, you'll need to register a personal app. You can do so from your Riot Games developer portal.

Click on the **Register Project** button, then choose the **Register App** button under the Personal Application section.





Then, simply fill out the application details, including a unique name for your app, and click **Submit**. It can take a few days for applications to be reviewed. In the meantime, you may need to regenerate your API key every 24 hours from the dashboard.



Build the Trophy

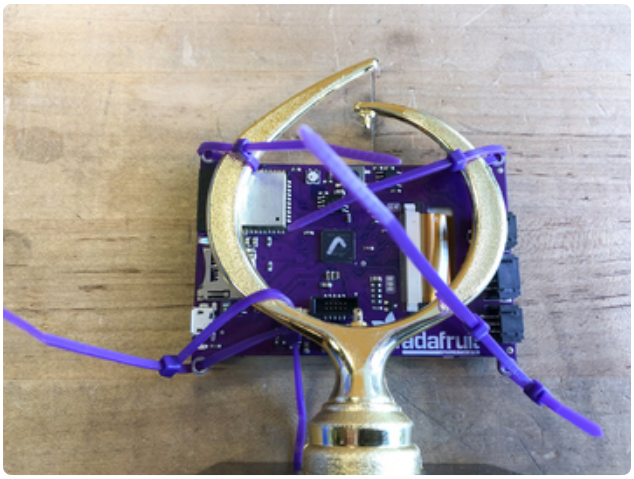
Now that you've got your PyPortal displaying your stats, dress things up a bit with a re-purposed trophy! C'mon, you know you've got one in a box somewhere. If not, get one at a yard sale or thrift shop.

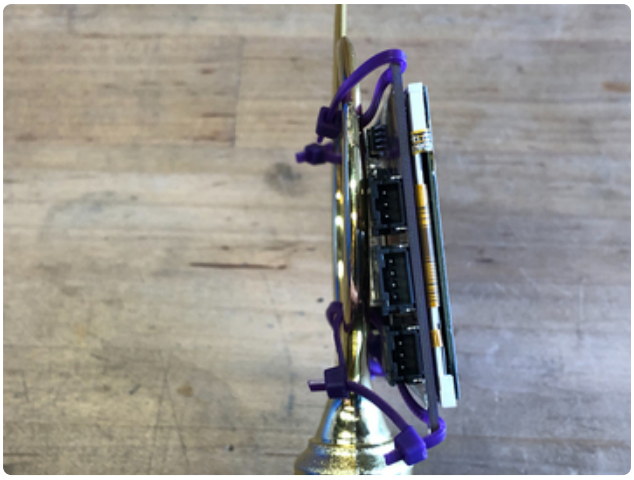
Prep

Remove off any unnecessary parts or labels from your trophy if you like.

In this case, I'm going to mount the PyPortal in the center, so I'll remove the (admittedly awesome) spinning "KICKBALL" element.

Mounting

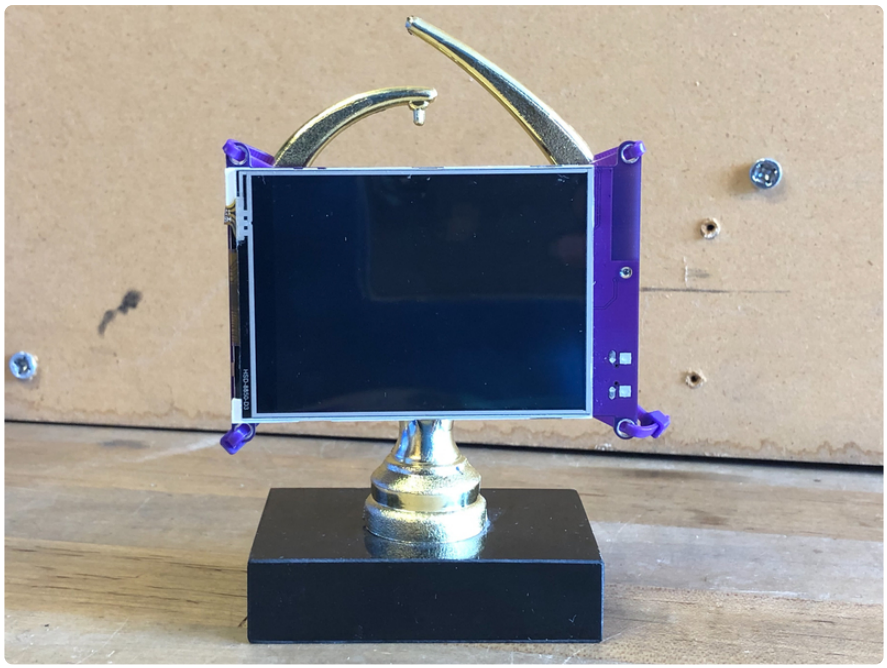




Using zip ties, connect each of the four mounting holes of the PyPortal to the trophy.

Carefully cut the excess plastic on the zip tie with diagonal cutters or a toenail clipper.

Alternately, you can use double-stick foam mounting tape or some other method to mount it.



Now, your display is complete, just plug it into USB power and admire your achievements!

Alternate Mounting

You may consider using the laser-cut acrylic frame in the Adafruit shop or one of the 3D printed stands available for PyPortal on Thingiverse. Be creative and have fun!