PyPortal Trivia Time with the Open Trivia Database

Created by Isaac Wellish

https://learn.adafruit.com/pyportal-trivia-time-open-trivia-database

Last updated on 2021-11-15 07:44:01 PM EST
# Table of Contents

## Overview
- Parts 3
- Other materials 4
- Tools and construction 5
- Optional 6

## Connect Electronics

## Install CircuitPython
- Set up CircuitPython Quick Start! 8
- PyPortal Default Files 10

## PyPortal CircuitPython Setup
- Adafruit CircuitPython Bundle 10

## Internet Connect!
- What's a secrets file? 12
- Connect to WiFi 13
- Requests 17
- HTTP GET with Requests 19
- HTTP POST with Requests 20
- Advanced Requests Usage 20
- WiFi Manager 22

## Code PyPortal with CircuitPython
- CircuitPython Code 24

## Using the App
- How it works 30
- The Open Trivia Database API 32
- PyPortal Constructor 35
- Fetch 35
- “Cleaning” JSON with HTML code 35
- Main loop 36

## Build the Box
- Install arcade buttons 37
- Feed wires through back and connect to buttons 40
- Finishing touches 42
- Going Further 44
Overview

Grab a friend and quiz-off with this internet-connected trivia app, powered by the Adafruit PyPortal!

Using the Open Trivia Database API (https://adafruit.com/FiK), the PyPortal can connect to thousands of verified questions online. Connect two arcade buttons to the PyPortal and face-off with a friend!

Parts

You can pick up an Adafruit PyPortal and a USB cable (if needed). If you like, you can mount the PyPortal in the Adafruit laser-cut acrylic stand. All these parts are bundled in AdaBox 011 if you'd like to buy them together.

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

Instead of AdaBox 011, you can buy parts separately:
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

Other materials

Arcade Button - 30mm Translucent Red
A button is a button, and a switch is a switch, but these translucent arcade buttons are in a class of their own. They’re the same size as common arcade controls (often referred to...
https://www.adafruit.com/product/473
2 x JST PH 3-Pin to Male Header Cable - 200mm  https://www.adafruit.com/product/3893

Connect buttons to PyPortal

2 x Arcade Button - 30mm Translucent Blue
A button is a button, and a switch is a switch, but these translucent arcade buttons are in a class of their own. They're the same size as common arcade controls (often referred to... https://www.adafruit.com/product/476

Arcade Button Quick-Connect Wire Pairs - 0.11" (10 pack)
Quick connector wire sets will make wiring up our arcade-style or metal buttons quicky-quick. Each wire comes as a 'pair' with two 0.11" quick-connects pre-crimped onto... https://www.adafruit.com/product/1152

Tools and construction

- small cardboard box
- hobby knife
- pencil
- ruler
- wire cutters
- tape
Optional

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

Connect Electronics

First we'll connect the electronics before we add the code and build the box.
• On the back of the PyPortal, connect one JST cable to D4 and the other to D3.
• Next connect the ground and data breadboard cables from the JST connector to the arcade button quick connect wire pairs. Orientation will not matter.
• Lastly connect the quick connect pairs to buttons. Make sure to keep track which color button corresponds to which input on the PyPortal (D4 vs D3).
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](https://adafru.it/Egk)

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

[https://adafru.it/HFd](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_espm32spi - 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_sdcard - support for reading/writing data from the onboard SD card slot.
- 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': 'fawfj23rakjnfwiefa',
    '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](https://adafru.it/EcP) 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. Let’s use the ESP32SPI and the Requests libraries - you’ll need to visit the CircuitPython bundle and install (https://adafruit.io/ENC):

- adafruit_bus_device
- adafruit_esp32spi
- adafruit_requests
- neopixel

Into your lib folder. Once that’s done, load up the following example using Mu or your favorite editor:

```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("\t%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 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")))
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.text)
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 `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](https://adafruit.com).
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` ([https://adafru.it/E9o](https://adafru.it/E9o)) - 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)

```python
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 (https://adafru.it/Kpa) library for web interfacing named Adafruit_CircuitPython_Requests (https://adafru.it/FpW). This library allows you to send HTTP/1.1 requests without "crafting" them and provides helpful methods for parsing the response from the server.

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

# Adafruit_Requests usage with an esp32spi_socket

# 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)
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)

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
The code first sets up the ESP32SPI interface. Then, it initializes a `requests` object using an ESP32 `socket` and the `esp` object.

```python
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

# 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:

    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)
    print("-" * 40)

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

    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)

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

©Adafruit Industries
try:
    esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_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
requests.set_socket(socket, esp)

HTTP GET with Requests

The code makes a HTTP GET request to Adafruit's WiFi testing website - [http://wifitest.adafruit.com/testwifi/index.html](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`.

While we requested data from the server, we'd want to know 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 collects the response's data.

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

HTTP POST with Requests

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

data = '31F'
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print('-'*40)

json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:"), json_resp['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)

json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:"), json_resp['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.
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(espssid, "utf-8"), "\RSSI:", 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": "blinka/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 example works but it's a little fussy - 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:

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

import time
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi_wifimanager 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)

# 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)

status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
    # Uncomment for Most Boards

status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
    # Uncomment for Most Boards

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",
headers={"X-AIO-KEY": secrets["aio_key"]},
)
        print(response.json())
counter = counter + 1
print("OK")
    except (ValueError, RuntimeError) 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_'
}
```

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](https://adafruit.it/f5k)
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!

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_Trivia_Time.

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

```
<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>boot_out.txt</td>
</tr>
<tr>
<td>code.py</td>
</tr>
<tr>
<td>fonts</td>
</tr>
<tr>
<td>fonts</td>
</tr>
<tr>
<td>adafruit_bitmap_font</td>
</tr>
<tr>
<td>adafruit_button.mpy</td>
</tr>
<tr>
<td>adafruit_display_shapes</td>
</tr>
<tr>
<td>adafruit_display_text</td>
</tr>
<tr>
<td>adafruit_esp32spi</td>
</tr>
<tr>
<td>adafruit_imageload</td>
</tr>
<tr>
<td>adafruit_pyportal.py</td>
</tr>
<tr>
<td>adafruit_sdcard.mpy</td>
</tr>
<tr>
<td>adafruit_touchscreen.mpy</td>
</tr>
<tr>
<td>neopixel.mpy</td>
</tr>
<tr>
<td>secrets.py</td>
</tr>
<tr>
<td>trivia_title.bmp</td>
</tr>
<tr>
<td>trivia.bmp</td>
</tr>
</tbody>
</table>
```

Make sure you are using CircuitPython 4.1.0 or later or text may be slow to display!

```python
import time
import random
import board
from adafruit_pyportal import PyPortal
from digitalio import DigitalInOut, Direction, Pull
from adafruit_display_text import label
from adafruit_bitmap_font import bitmap_font

# initialize hardware
led = DigitalInOut(board.L) # For debugging
led.direction = Direction.OUTPUT
button1 = DigitalInOut(board.D4)
button1.direction = Direction.INPUT
button2 = DigitalInOut(board.D3)
button2.direction = Direction.INPUT
button1.pull = Pull.UP
button2.pull = Pull.UP
display = board.DISPLAY

# determine the current working directory
# needed so we know where to find files
```
# Set up where we'll be fetching data from
DATA_SOURCE = "https://opentdb.com/api.php?amount=1&type=multiple"
Q_LOCATION = ['results', 0, 'question']
WA_LOCATION1 = ['results', 0, 'incorrect_answers', 0]
WA_LOCATION2 = ['results', 0, 'incorrect_answers', 1]
WA_LOCATION3 = ['results', 0, 'incorrect_answers', 2]
CA_LOCATION = ['results', 0, 'correct_answer']

# Text info
trivia_font = bitmap_font.load_font("/fonts/Arial-ItalicMT-17.bdf")
loading_color = 0x8080FF
loading_position = (100, 120)
loading_text_area = label.Label(trivia_font, color=loading_color,
                               x=loading_position[0], y=loading_position[1])

def shuffle(aList):
    for i in range(len(aList)):
        j = random.randint(0, len(aList)-1)
        # Swap arr[i] with the element at random index
        aList[i], aList[j] = aList[j], aList[i]
    return aList

# convert html codes to normal text
def unescape(s):
    s = s.replace("&quot;", ""
    s = s.replace("&#039;", ""
    s = s.replace("&amp;", "&")
    return s

# A function to handle the timer and determine which player answers first
def faceOff(timerLength):
    timer_text = str(timerLength) + " seconds!"
    timer_text_area = ''
    timer_text_area.text = str(timer_text)
    timerStart = time.monotonic()
    while time.monotonic() - timerStart < timerLength:
        if button1.value:
            led.value = False # For debugging
        else: # If button 1 pressed, print player 1 on screen and exit function
            led.value = True # For debugging
            q_text_area.text = ''
reveal_text_area.text = "Player 1!"
break
if button2.value:
    led.value = False # For debugging
else: # If button 2 pressed, print player 2 on screen and exit function
    led.value = True # For debugging
    q_text_area.text = '
    reveal_text_area.text = "Player 2!"
break
time.sleep(0.05) # debounce delay
else: # Timer runs out
    q_text_area.text = ''
    reveal_text_area.text = "Times up!"

# PyPortal constructor
pyportal = PyPortal(url=DATA_SOURCE,
    json_path=(Q_LOCATION, CA_LOCATION, WA_LOCATION1, WA_LOCATION2,
    WA_LOCATION3),
    status_neopixel=board.NEOPIXEL,
    default_bg=cwd + "trivia_title.bmp")
pyportal.preload_font() # speed things up by preloading font
pyportal.splash.append(loading_text_area) #loading...
pyportal.splash.append(q_text_area)
pyportal.splash.append(reveal_text_area)
pyportal.splash.append(timer_text_area)
for textarea in ans_text_areas:
    pyportal.splash.append(textarea)

while True:
    # Load new question when screen is touched
    while not pyportal.touchscreen.touch_point:
        pass
    reveal_text_area.text = ''
    q_text_area.text = ''
    for textarea in ans_text_areas:
        textarea.text = ''
    timer_text_area.text = ''
    pyportal.set_background(cwd + "trivia.bmp")
    loading_text_area.text = "Loading question..."

    while True:
        try:
            value = pyportal.fetch()
            break
        except RuntimeError as e:
            print("Some error occurred, retrying! -", e)
            continue
        print("Response is", value)
        question = value[0]
        correct_answer = value[1]
        answers = shuffle(value[1:5])
        loading_text_area.text = ''

        # Format text and wrap with display text library
        try: # sometimes gives a runtime error: Group full
            q_text_area.text = '\n'.join(pyportal.wrap_nicely(unescape(question), 35))
        except RuntimeException as e:
            print("Group full", e)
            continue
        for k, answer in enumerate(answers):
            ans_text_areas[k].text = answer_choices[k] + unescape(answer)

        faceOff(10) # 10 seconds with question
        time.sleep(2) # pause for 2 seconds to show which player tapped first
        faceOff(5) # 5 seconds to answer
Using the App

When the PyPortal is turned on or reset, the following events occur:

- Display the title page and wait until user taps screen.
- If user taps screen, display "loading..." while question data is being pulled from API.
- Display question, answer choices then timer length.
- Which ever player hits their button first, will get their name displayed on the screen.
- They then get 5 seconds to answer and hit their button again.
- Display correct answer and prompt user to tap screen for next question.
If no player hits their button within the 10 seconds after the question is displayed, "Time's Up!" appears on the screen.

- Tap any button to reveal the answer.

How it works

Trivia Time is doing a couple of nifty things to deliver your quiz-off experience!
Background

First, the program displays a bitmap graphic as the screen's background. After the screen is tapped, the second bitmap graphic displays as the background for displaying the trivia questions. These are 320 x 240 pixel RGB 16-bit raster graphics in .bmp format.

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:

```python
default_bg=cwd+ '/trivia_title.bmp'
```

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

```python
default_bg=cwd+'/my_new_background.bmp'
```
Font

First we see "loading" appear on the screen along with the source at the bottom (part of the bitmap).

Next the question, answers and timer appear on the screen.

The fonts used here are bitmap fonts made from the Arial Italic typeface. You can learn more about converting type in this guide (https://adafru.it/E7E).

JSON

The neat part is that the text is not coming from a file on the device, but rather it is taken from a website!

The Open Trivia Database API

The Open Trivia Database (https://adafru.it/FiL) API is where we'll be grabbing our trivia questions from. The API (https://adafru.it/FiK) has over 3,000 verified, user-contributed questions to use!
You can choose a number of presets to filter the type of questions you get such as number, category, difficulty, type and encoding.
Change the number of questions to 1 then click the "Generate API URL" button.
A link will appear on the top of the screen. Copy and paste that link into a new web browser window (or just click here (https://adafruit.it/FiM)).

```
{"response_code":0,"results":[{"category":"Entertainment: Japanese Anime & Manga","type":"multiple","difficulty":"medium","question":"In "Highschool DxD", what is the name of the item some humans are born with?","correct_answer":"Sacred Gear","incorrect_answers":["Imperial Arm","Hallowed Relic","Blessed Artifact"]}
```

You may see a large mess of numbers, letters and symbols. This is JSON data!

Head here (https://adafruit.it/Eb5) to beautify the code so we can actually read it!

"Beautified" code:

```
{
    "response_code": 0,
    "results": [
    {
        "category": "Entertainment: Japanese Anime & Manga",
        "type": "multiple",
        "difficulty": "medium",
        "question": "In "Highschool DxD", what is the name of the item some humans are born with?",
        "correct_answer": "Sacred Gear",
        "incorrect_answers": [
            "Imperial Arm",
            "Hallowed Relic",
            "Blessed Artifact"
        ]
    }
]
```

If we look through the JSON file, we'll see a key called question, and 2 others called correct_answer and incorrect_answers. incorrect_answers has a sub-tree below it hierarchically with 3 elements.

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

```
"question": "In "Highschool DxD", what is the name of the item some humans are born with?"
"correct_answer": "Sacred Gear"
"incorrect_answers": [
            "Imperial Arm",
            "Hallowed Relic",
```
"Blessed Artifact"

Our CircuitPython code is able to grab and parse this data using these variables:

```python
Q_LOCATION = ['results', 0, 'question']
CA_LOCATION = ['results', 0, 'correct_answer']
WA_LOCATION1 = ['results', 0, 'incorrect_answers', 0]
WA_LOCATION2 = ['results', 0, 'incorrect_answers', 1]
WA_LOCATION3 = ['results', 0, 'incorrect_answers', 2]
```

**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

**Fetch**

With the pyportal set up, we can then use `pyportal.fetch()` to do the query and parsing the pieces of Open Trivia data. You may notice some responses output data like `"`. This is HTML code. " Next we'll "clean" the text before it gets displayed.

"Cleaning" JSON with HTML code

In order to "clean" the JSON data (i.e. take HTML codes like `"` and make them `"`) we have to take in the JSON data, clean it, and then display it using the Adafruit_Display_Text library. After taking in the data from the API, we use the `unescape()` function to replace HTML code with desire text.

```python
def unescape(s):
    s = s.replace("\"", """)
    s = s.replace("\&lt;", """")
    s = s.replace("\&gt;", """")
    return s
```
Main loop

```python
while True:

In the repeating main loop, the following actions are performed:

1. Check to see if the screen was touched
2. If screen was touched, clear all text elements on the screen.
3. Next change the background, then load and display the question.
4. Call the `faceOff()` function to start the timer for 10 seconds and wait for the first player to tap their button.
5. Call `faceOff()` again to give 5 seconds for player to answer. Then display the answer.
6. Prompt user to touch screen to display next question.
Build the Box

Install arcade buttons

- First disconnect the quick connect wires from both arcade buttons.
- Grab a small - medium sized box.
- Using a ruler, draw a line approximately through the middle of the box top length-wise.
- According to the schematic (https://adafruit.it/udT) of the arcade button, the diameter of the button that we want is 29.5 cm. This smaller diameter will allow the button to snap into place and be snug against the cardboard.
- Measure two points 29.5 cm away from each other on both sides of the box.
- From the mid-point of line you just drew, draw another identical line that's perpendicular.
- Now we have 4 points on the circles, sketch in the arc to complete it. The circle does not have to be perfect!
- Now use a hobby knife to cut out the circle.
- Take an arcade button and press it in place.
- Repeat on the other side!
Feed wires through back and connect to buttons

- Poke a hole in the back of the box with a pen or pencil.
- Feed the quick connect cables through the hole.
- Now connect the corresponding cables to the arcade buttons.
Finishing touches

- Cut the red wires off of the JST connectors as we are not using them.
- Run the remaining wires around the left corner of the box and secure with some tape.
- Secure PyPortal the top of the box with some double sided tape.
Power up the PyPortal in and it's time-to-trivia!

Going Further

How can you make this trivia game even cooler? Here are some ideas:

- Create a point system! Use the buttons to tell the PyPortal if you got the question right or not and add up points accordingly.
- Use the question difficulty element from the API to determine the amount of points for each question.
- Use the category element from the API to have different "rounds" based on category.