Pick the Perfect Plant: Battery-Powered Sun Tracking

Created by Tyler Cooper

https://learn.adafruit.com/battery-powered-sun-tracking

Last updated on 2026-07-10 06:19:06 PM UTC

Table of Contents

Overview 3

- Using AI to help you pick the right plant (optional)
- How it works
- Deep Sleep?
- Parts

Get Started with Adafruit IO 7

- I have an Adafruit.com Account already
- Create an Adafruit Account (for Adafruit IO)

Install CircuitPython 9

- CircuitPython Quickstart
- Safe Mode
- Flash Resetting UF2

Create Your settings.toml File 13

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

Coding the Sun Tracker 16

- Add Your settings.toml File
- Understanding the Code

Circuit Diagram 23

- Wiring Connections
- Assembling Your Sun Tracker

Viewing the Data 25

- Interpreting Your Light Data
- Understanding the Graph
- Matching Data to Plant Needs
- How to Pick Your Plant

AI Plant Recommendations (optional) 28

- Step 1: Exporting Your Feed Data from Adafruit IO
- Step 2: Quick Cleanup
- Step 3: Let Gemini Interpret the Data
- Step 4: The Response

Overview

Finding the perfect spot in your yard for a new plant can be tricky. Does a nursery tag's "partial shade" mean harsh morning sun, or dappled light all day? **Instead of guessing and risking your new greenery, you can build a simple, battery-powered data logger to measure the exact light levels (lux) in any spot in your garden over time.**

By combining an Adafruit ESP32-S2 Feather with a highly accurate VEML7700 Lux sensor and a LiPo battery, you can track the sun's path and intensity throughout the day. To keep it weather-resistant outside without firing up a 3D printer or buying a specialized enclosure, the whole setup slips perfectly into a standard clear Rubbermaid food storage container straight from your kitchen.

Using Adafruit IO and a bit of code to manage power, this guide will show you how to log environmental data straight to the cloud. Best of all, there's zero soldering required thanks to the plug-and-play STEMMA QT system.

Using AI to help you pick the right plant (optional)

Once you've logged a few days or weeks of sunlight data, Adafruit IO allows you to easily export your history as a simple CSV file. **You can then take this raw data and**

feed it into your favorite AI to act as a virtual master gardener. Just provide your downloaded light readings alongside your specific USDA hardiness zone (like my chilly Zone 4 or 5). **The AI can analyze the exact peak sun hours and lux intensity of your garden's microclimate to generate a list of vegetables or perennials that would grow well in that exact spot.**

How it works

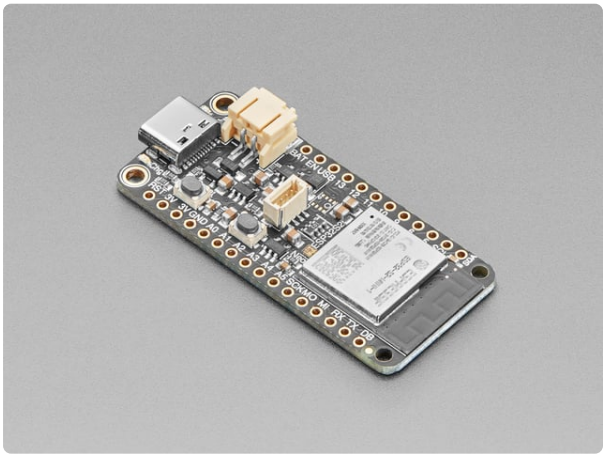
This project relies on the power of Adafruit IO to log your garden's ambient light data, and custom code to keep your hardware running as efficiently as possible. The ESP32-S2 Feather wakes up at a set interval, connects to your home WiFi, reads the light levels from the VEML7700 sensor over a simple STEMMA QT cable, and sends that data directly to an Adafruit IO Dashboard before going back to sleep.

Placed in its clear, water-resistant Rubbermaid housing, it can safely sit in the dirt among your garden beds. You can then log into Adafruit IO to view a beautiful, continuous graph of your yard's sunlight over days or weeks, making it perfectly clear whether that corner is meant for sun-loving tomatoes or shade-tolerant hostas.

Deep Sleep?

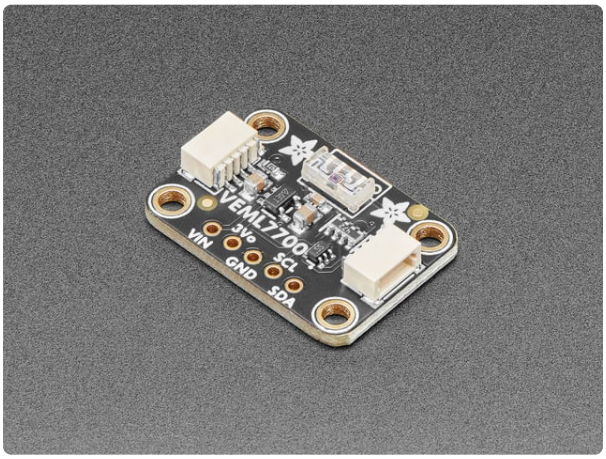
When running a project on battery power outdoors, energy efficiency is critical. If the microcontroller and WiFi radio run continuously, a standard LiPo battery would drain in a matter of hours. By writing code that utilizes CircuitPython's deep sleep mode, the board essentially powers down between readings. It only wakes up for a few seconds to take a lux measurement and transmit it via WiFi. This software trick extends your battery life from hours to weeks, giving you long-term environmental data without constant recharging.

Parts



Adafruit ESP32-S2 Feather - 4 MB Flash + 2 MB PSRAM

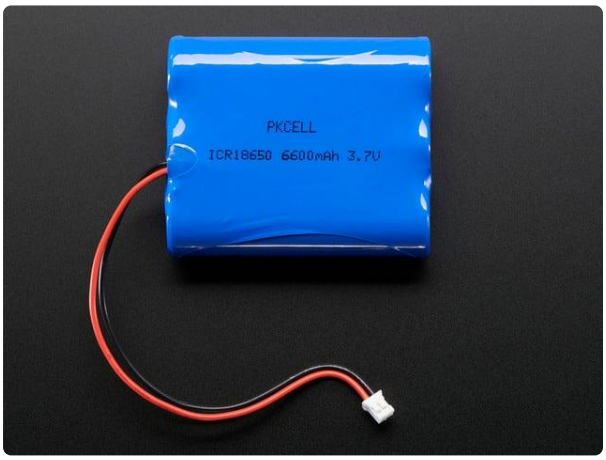
What's Feather-shaped and has an ESP32-S2 WiFi module? What has a STEMMA QT connector for I2C devices? What has your favorite Espressif WiFi microcontroller and lots of Flash and...
https://www.adafruit.com/product/5000



Adafruit VEML7700 Lux Sensor - I2C Light Sensor

Vishay has a lot of light sensors out there, and this is a nice simple lux sensor that's easy to add to any microcontroller. Most light sensors just give you a number for...
https://www.adafruit.com/product/4162

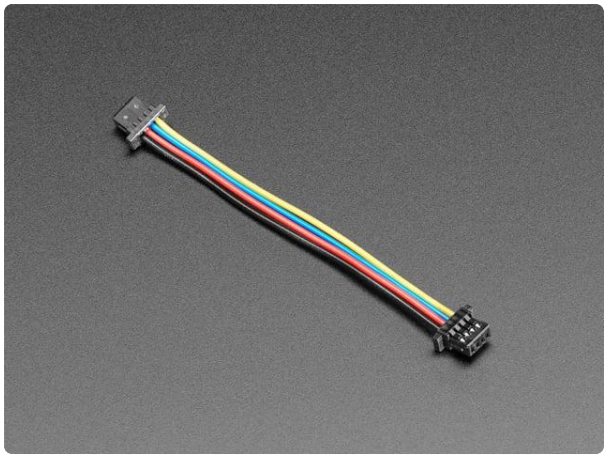
Either of the below batteries will work. Because we put this device into deep sleep, it really doesn't consume that much power. If you are doing a shorter test you can get away with an even smaller capacity battery.



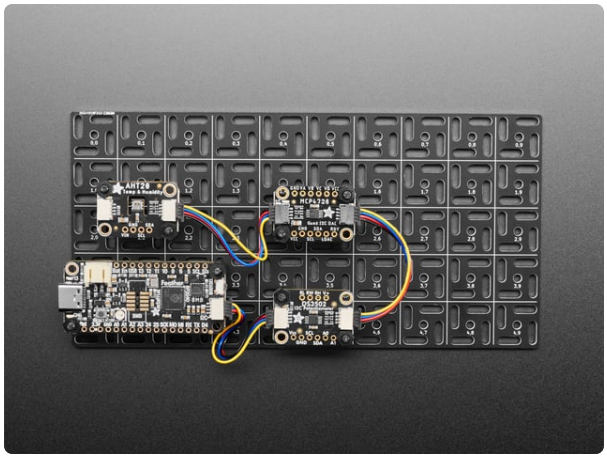
Lithium Ion Battery Pack - 3.7V 6600mAh

Need a massive battery for your project? This lithium-ion pack is made of 3 balanced 2200mAh cells for a total of 6600mA capacity! The cells are connected in parallel and spot-welded...
https://www.adafruit.com/product/353

[Lithium Ion Battery Pack - 3.7V 4400mAh]
Need a big battery for your project? This lithium-ion pack is made of 2 balanced 2200mAh cells for a total of 4400mA capacity! The cells are connected in parallel and spot-welded to a...
https://www.adafruit.com/product/354



[STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long]
This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...
https://www.adafruit.com/product/4399



[Adafruit Swirly Aluminum Mounting Grid for 0.1" Spaced PCBs]
With most of our dev boards, sensors and feathers now sporting plug-and-play stemma QT ports it can be very fast to...
https://www.adafruit.com/product/5774

[Black Nylon Machine Screw and Stand-off Set – M2.5 Thread]
Totaling 380 pieces, this M2.5 Screw Set is a must-have for your workstation. You'll have enough screws, nuts, and hex standoffs to fuel your maker...
https://www.adafruit.com/product/3299

Get Started with Adafruit IO

Adafruit IO is integrated with your adafruit.com account (https://adafru.it/dyy) so you don't need to create yet another online account! You need an Adafruit account to use Adafruit IO because we want to make sure the data you upload is available to only you (unless you decide to publish your data).

I have an Adafruit.com Account already

If you already have an Adafruit account, then you already have access to Adafruit IO. It doesn't matter how you signed up, your account will make all three available.

To access Adafruit IO, simply visit https://io.adafruit.com (https://adafru.it/eZ8) to start streaming, logging, and interacting with your data.

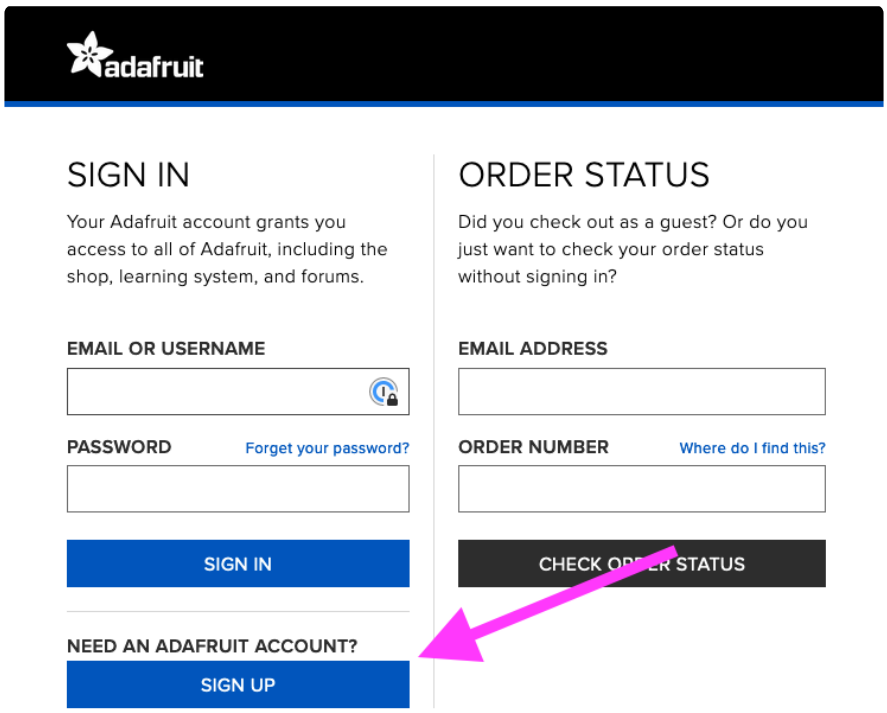
Create an Adafruit Account (for Adafruit IO)

An Adafruit account makes Adafruit content and services available to you in one place. Your account provides access to the Adafruit shop (https://adafru.it/dAR), the Adafruit Learning System (https://adafru.it/dIu), and Adafruit IO (https://adafru.it/fsU). This means only one account, one username, and one password are necessary to engage with the content and services that Adafruit offers.

If you do not have an Adafruit account, signing up for a new Adafruit account only takes a couple of steps.

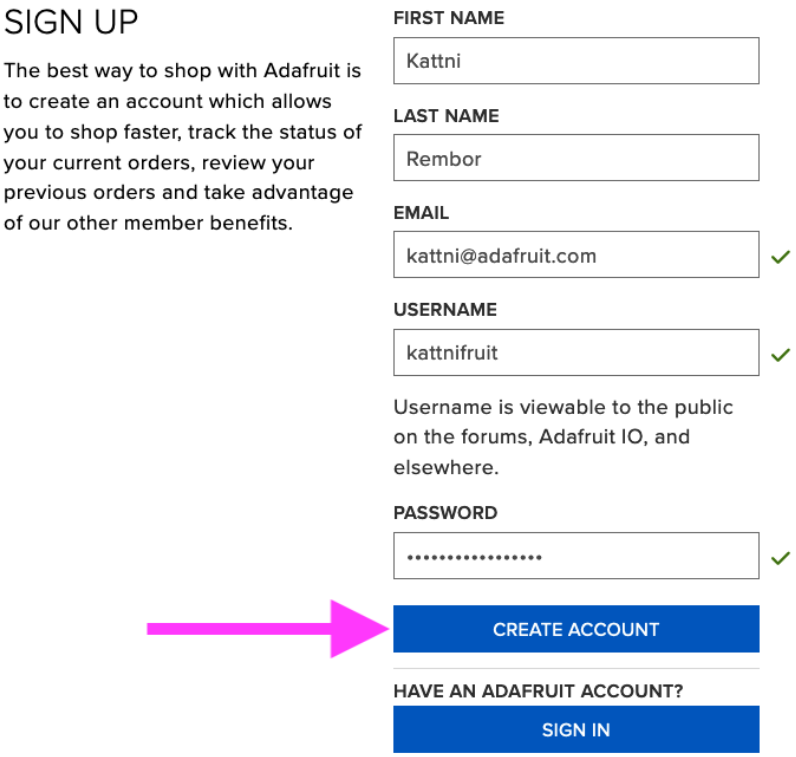
Begin by visiting https://accounts.adafruit.com (https://adafru.it/18ta).

Click the Sign Up button under the "Need An Adafruit Account?" title, below the Sign In section.



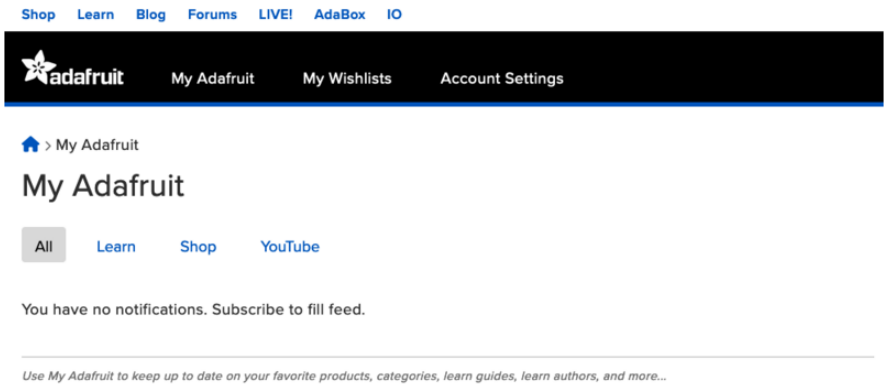
This will take you to the **Sign Up** page.

Fill in the requested information, and **click the Create Account button.**



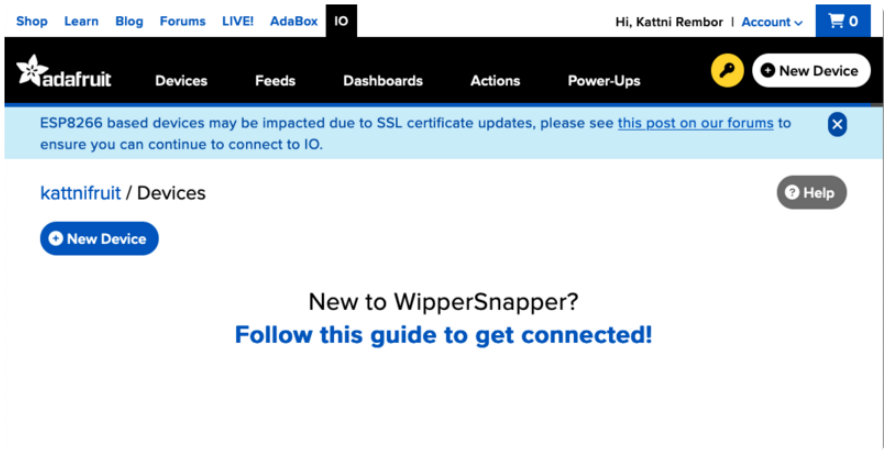
This takes you to your Adafruit Account home page. From here, you can access all the features of your account.

You can also access the Adafruit content and services right from this page. Along the top of the page, you'll see a series of links beginning with "Shop". To access any of these, simply click the link.



For example, **to begin working with Adafruit IO, click the IO link** to the right of the rest of the links. This is the same for the other links as well.

That's all there is to creating a new Adafruit account, and navigating to Adafruit IO.



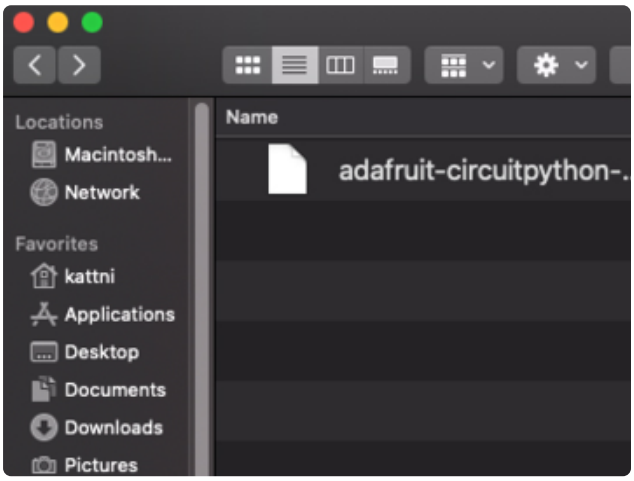
Install CircuitPython

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** drive to iterate.

CircuitPython Quickstart

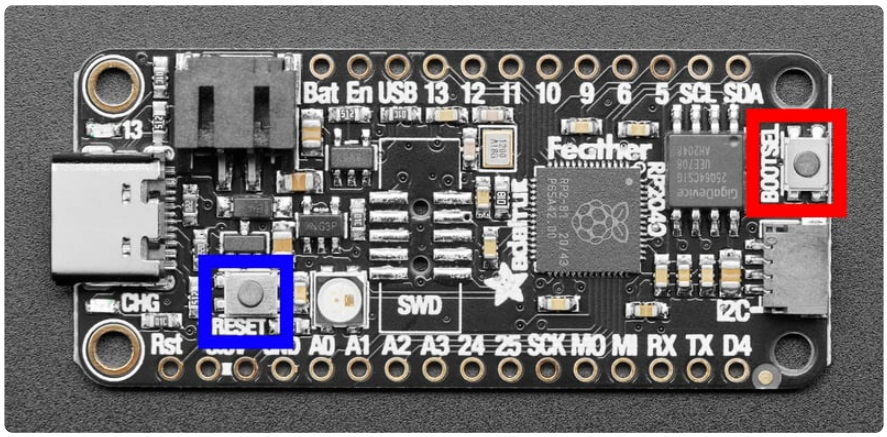
Follow this step-by-step to quickly get CircuitPython running on your board.

https://adafru.it/R1D



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

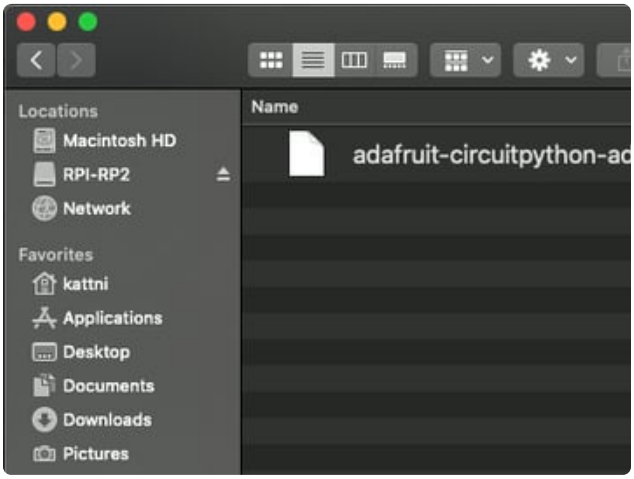


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in red or blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

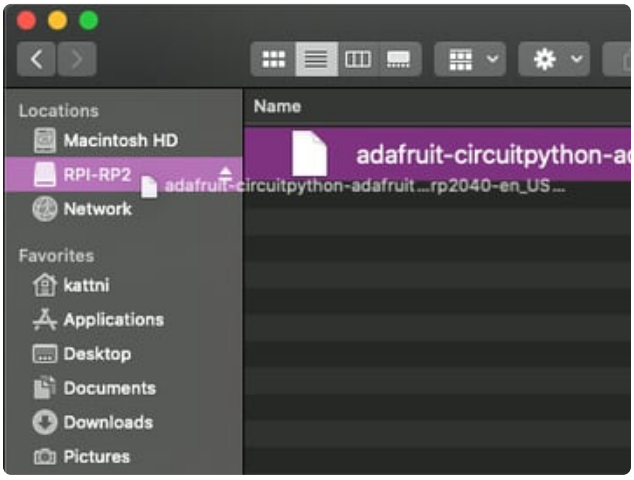
If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.

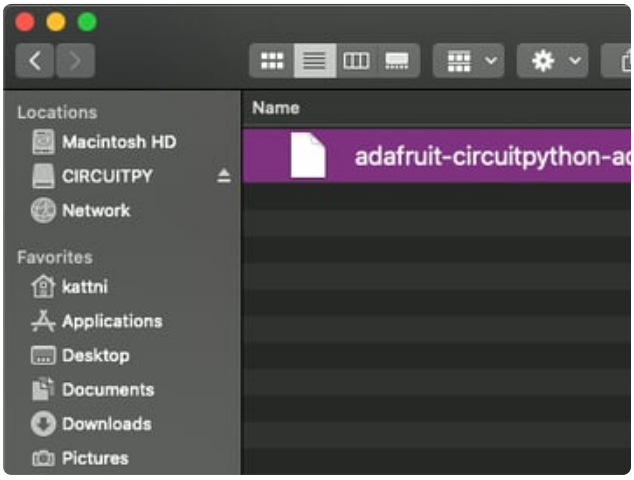
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **RPI-RP2**.



Drag the **adafruit_circuitpython_etc.uf2** file to **RPI-RP2.**



The **RPI-RP2** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

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

Safe Mode

You want to edit your **code.py** or modify the files on your **CIRCUITPY** drive, but find that you can't. Perhaps your board has gotten into a state where **CIRCUITPY** is read-only. You may have turned off the **CIRCUITPY** drive altogether. Whatever the reason, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in **boot.py** (where you can set **CIRCUITPY** read-only or turn it off completely). Second, it does not run the code in **code.py**. And finally, it does not automatically soft-reload when data is written to the **CIRCUITPY** drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the **CIRCUITPY** drive.

Entering Safe Mode

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

If you successfully enter safe mode on CircuitPython, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot. Press again to

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

You can now edit the contents of the **CIRCUITPY** drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.

Flash Resetting UF2

If your board ever gets into a really weird state and CIRCUITPY doesn't show up as a disk drive after installing CircuitPython, try loading this 'nuke' UF2 to RPI-RP2. which will do a 'deep clean' on your Flash Memory. **You will lose all the files on the board**, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

https://adafru.it/RLE

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.

Text emphasized with a yellow exclamation:	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.**

Text emphasized with a yellow exclamation:	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"))
```

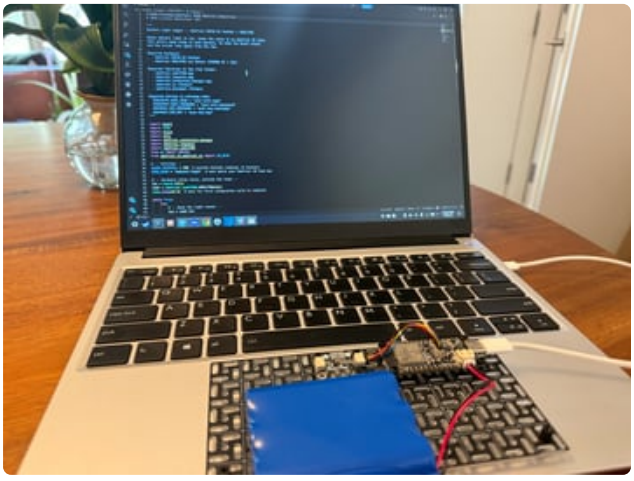
```
CircuitPython REPL
code.py output:
this is a test

Code done running.

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

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

Coding the Sun Tracker



Once you've finished setting up your ESP32 Feather with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

To do this, click on the **Download Project Bundle** button in the window below. It will download to your computer as a zipped folder.

```
# SPDX-FileCopyrightText: 2026 Adafruit Industries
# SPDX-License-Identifier: MIT

"""
Outdoor Light Logger -- Adafruit ESP32-S2 Feather + VEML7700

Reads ambient light in lux, sends the value to an Adafruit IO feed,
then enters deep sleep to save battery. On wake the board resets
and the script runs again from the top.

```

```
Required hardware:
  - Adafruit ESP32-S2 Feather
  - Adafruit VEML7700 Lux Sensor (STEMMA QT / I2C)

Required libraries in the /lib folder:
  - adafruit_veml7700.mpy
  - adafruit_requests.mpy
  - adafruit_connection_manager.mpy
  - adafruit_io (folder)
  - adafruit_minimqtt (folder)

Required entries in settings.toml:
  CIRCUITPY_WIFI_SSID = "your-wifi-name"
  CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"
  ADAFRUIT_AIO_USERNAME = "your-aio-username"
  ADAFRUIT_AIO_KEY = "your-aio-key"
"""

import time
from os import getenv

import alarm
import board
import wifi
import adafruit_connection_manager
import adafruit_requests
import adafruit_veml7700
from adafruit_io.adafruit_io import IO_HTTP

# -- Settings --
SLEEP_INTERVAL = 300  # seconds between readings (5 minutes)
FEED_NAME = "ambient-light"  # must match your Adafruit IO feed key

# -- Hardware setup (once, outside the loop) --
i2c = board.I2C()
veml = adafruit_veml7700.VEML7700(i2c)
time.sleep(0.5)  # wait for first integration cycle to complete

while True:
    try:
        # -- Read the light sensor --
        lux = veml.lux
        print(f"Light: {lux:.1f} lux")

        # -- Connect to WiFi and send to Adafruit IO --
        if not wifi.radio.ipv4_address:
            wifi.radio.connect(
                getenv("CIRCUITPY_WIFI_SSID"),
                getenv("CIRCUITPY_WIFI_PASSWORD"),
            )
        print(f"WiFi connected - IP: {wifi.radio.ipv4_address}")

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

```

```
        io = IO_HTTP(
            getenv("ADAFRUIT_AIO_USERNAME"),
            getenv("ADAFRUIT_AIO_KEY"),
            requests,
        )

        io.send_data(FEED_NAME, lux)
        print("Sent to Adafruit IO!")

    except Exception as e:  # pylint: disable=broad-except
        print(f"ERROR: {e}")

    # -- Deep sleep (battery) or wait (USB) --
    print(f"Sleeping {SLEEP_INTERVAL} seconds...")
    time_alarm = alarm.time.TimeAlarm(
        monotonic_time=time.monotonic() + SLEEP_INTERVAL
    )
    alarm.exit_and_deep_sleep_until_alarms(time_alarm)
    # On battery: board resets, script runs from the top.
    # On USB: pretend sleep returns here, loop continues.

```

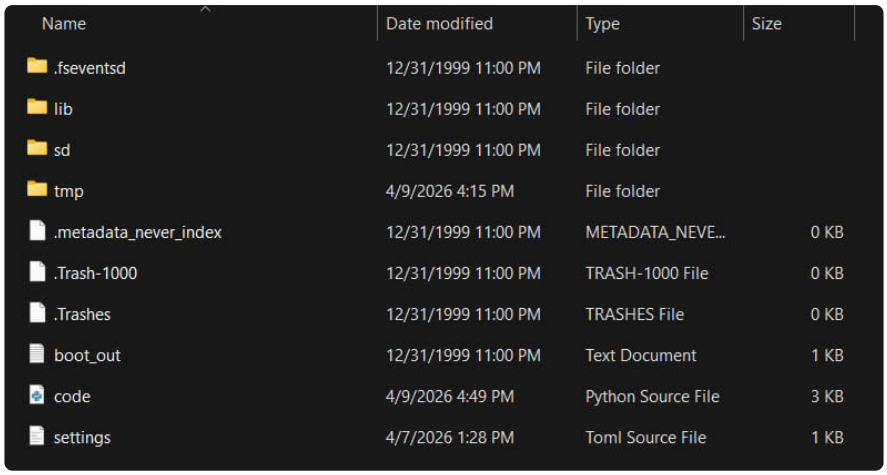
https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/Battery_Powered_Sun_Tracking/code.py

Upload the code and libraries to your ESP32-S2 Feather

After downloading the Project Bundle, plug your ESP32-S2 Feather into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called **CIRCUITPY**. Unzip the folder and copy the following items to the ESP32-S2 Feather's **CIRCUITPY** drive.

- **lib** folder
- **code.py**

Your ESP32-S2 Feather **CIRCUITPY** drive should look like this after copying the **lib** folder and the **code.py** file.



Add Your **settings.toml** File

As of CircuitPython 8, there is support for Environment Variables (https://adafru.it/11wE). These Environmental Variables are stored in a **settings.toml** file. Similar to **secrets.py**, the **settings.toml** file separates your sensitive information from your main **code.py** file. Add your **settings.toml** file as described in the Create Your settings.toml File page (https://adafru.it/18IC) earlier in this guide. You'll need to include your `CIRCUITPY_WIFI_SSID`, `CIRCUITPY_WIFI_PASSWORD`, `aio_username` and `aio_key` in the file.

```
CIRCUITPY_WIFI_SSID = "your-wifi-ssid-here"
CIRCUITPY_WIFI_PASSWORD = "your-wifi-password-here"

aio_username = "your-Adafruit-IO-username-here"
aio_key = "your-Adafruit-IO-key-here"
```

Understanding the Code

At first glance, connecting to WiFi and beaming sensor data to the cloud might sound a bit intimidating, but CircuitPython makes it incredibly straightforward!

The code for the light logger follows a simple, repeatable routine:

1. The microcontroller wakes up
2. Grabs the current ambient light reading from the VEML7700 sensor
3. Connects to your local network to push that data to your Adafruit IO dashboard
4. Immediately goes into a deep sleep to conserve battery life

The sections below break down exactly what is happening in each section so you can see how all the pieces fit together.

1. Importing Libraries

```
import board
import time
import alarm
import wifi
import adafruit_connection_manager
import adafruit_requests
import adafruit_veml7700
from os import getenv
from adafruit_io.adafruit_io import IO_HTTP
```

What this does: Microcontrollers don't know how to do everything right out of the box. The `import` statements tell the board to load specific "instruction manuals" (libraries) so it knows how to talk to your hardware and the internet.

- `board`, `time`, and `alarm` handle the physical pins on the board, time delays, and the deep-sleep functionality.
- `wifi`, `adafruit_connection_manager`, `adafruit_requests`, and `IO_HTTP` give your board the ability to connect to your local Wi-Fi and send data securely to the Adafruit IO servers.
- `adafruit_veml7700` contains the specific instructions for reading the light sensor.
- `getenv` allows your code to securely read the WiFi passwords and API keys you stored in your **settings.toml** file.

2. User Settings

```
# -- Settings --
SLEEP_INTERVAL = 300  # seconds between readings (5 minutes)
FEED_NAME = "ambient-light"  # must match your Adafruit IO feed key
```

What this does: a couple of variables are defined up top so they are easy to find and change without having to dig through the rest of the code.

- `SLEEP_INTERVAL` tells the board how long to snooze between readings to save battery.

- `FEED_NAME` is the exact name of the digital "bucket" on Adafruit IO where you want to send your light data.

3. Hardware Setup

```
# -- Hardware setup (once, outside the loop) --
i2c = board.I2C()
veml = adafruit_veml7700.VEML7700(i2c)
time.sleep(0.5)  # wait for first integration cycle to complete
```

What this does: Before starting to taking measurements, you have to introduce the sensor to the microcontroller.

- `board.I2C()` sets up the communication pipeline (the STEMMA QT cable) between the Feather and the sensor.
- `veml = adafruit_veml7700.VEML7700(i2c)` creates a code object representing your physical sensor.
- A tiny half-second delay (`time.sleep(0.5)`) is added to give the sensor a moment to wake up and take its very first light reading before we ask for the data.

4. The Main Loop & Reading Data

```
while True:
    try:
        # -- Read the light sensor --
        lux = veml.lux
        print(f"Light: {lux:.1f} lux")
```

What this does:

- `while True:` creates an infinite loop. Everything indented under this line will run over and over again.
- `try:` is the start of our error handling. We are telling the board, "Try to do the following steps, but if something goes wrong (like the WiFi dropping), don't completely crash."

- `lux = veml.lux` is where the magic happens! We ask the sensor for the current ambient light level and store it in a variable called `lux`. We then print it out to the Serial console so you can see it working on your computer.

5. Connecting to Wi-Fi & Adafruit IO

```
# -- Connect to WiFi and send to Adafruit IO --
    if not wifi.radio.ipv4_address:
        wifi.radio.connect(
            getenv("CIRCUITPY_WIFI_SSID"),
            getenv("CIRCUITPY_WIFI_PASSWORD"),
        )
    print(f"WiFi connected - IP: {wifi.radio.ipv4_address}")

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

    io = IO_HTTP(
        getenv("ADAFRUIT_AIO_USERNAME"),
        getenv("ADAFRUIT_AIO_KEY"),
        requests,
    )

    io.send_data(FEED_NAME, lux)
    print("Sent to Adafruit IO!")
```

What this does: This chunk gets your data onto the internet.

- First, it checks if you are already connected to WiFi. If not, it uses the credentials from your **settings.toml** file to log into your router.
- Next, it sets up a `pool` and `ssl_context`. Think of this as opening a secure browser window so your board can talk to websites safely.
- Then, it logs into your specific Adafruit IO account using your username and secret key.
- Finally, `io.send_data(FEED_NAME, lux)` packages up your light reading and beams it directly into your Adafruit IO dashboard!

6. Error Catching & Deep Sleep

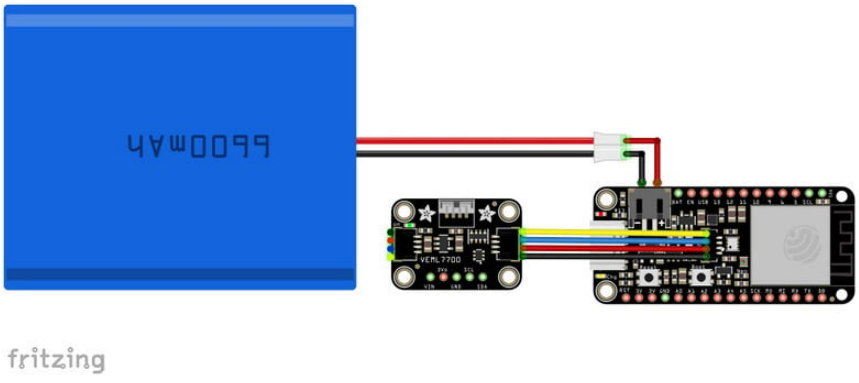
```
except Exception as e:
        print(f"ERROR: {e}")

    # -- Deep sleep (battery) or wait (USB) --
    print(f"Sleeping {SLEEP_INTERVAL} seconds...")
    time_alarm = alarm.time.TimeAlarm(
        monotonic_time=time.monotonic() + SLEEP_INTERVAL
    )
    alarm.exit_and_deep_sleep_until_alarms(time_alarm)
    # On battery: board resets, script runs from the top.
    # On USB: pretend sleep returns here, loop continues.
```

What this does:

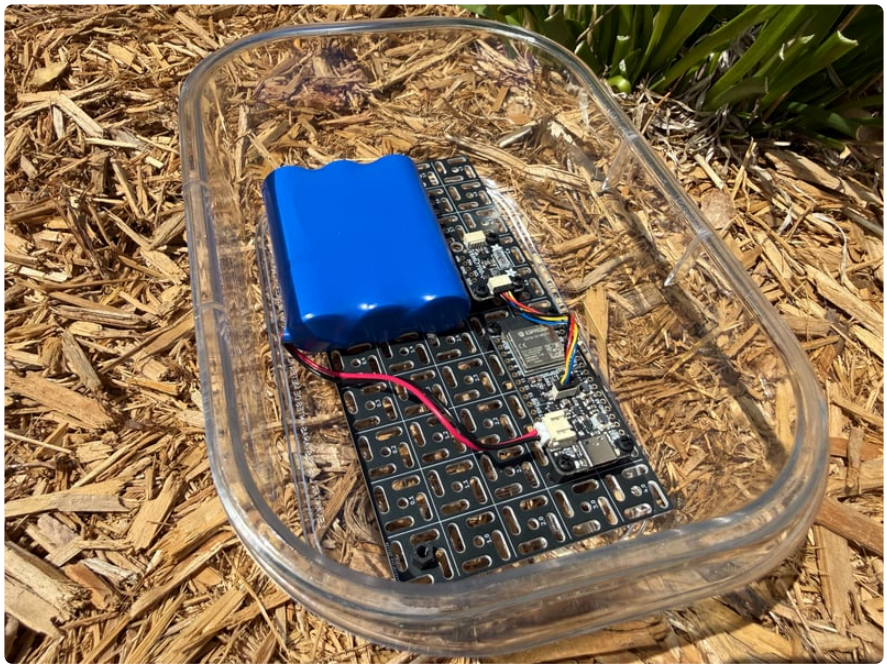
- `except Exception as e:` pairs with the `try:` from earlier. If the Wi-Fi disconnects or Adafruit IO is unreachable, it simply prints the error to the console instead of freezing the board.
- To make this project run for a long time on a battery, we use **Deep Sleep**. We set a `time_alarm` for 5 minutes (our 300-second interval).
- `alarm.exit_and_deep_sleep_until_alarms` shuts down almost the entire microcontroller to sip the absolute minimum amount of power.
- When the 5 minutes are up, the board wakes up. If it's running on battery, waking from deep sleep acts like pressing the reset button: the code starts entirely over from line 1. (Note: If your board is plugged into your computer via USB, it skips the deep sleep and just waits 5 minutes before looping, so you don't lose your serial connection!).

Circuit Diagram



Wiring Connections

- Plug the VEML7700 STEMMA QT Port to the Feather ESP32-S2's STEMMA QT port with a STEMMA QT cable
- LiPoly battery to the Feather ESP32-S3's LiPoly battery connector



Assembling Your Sun Tracker

You don't need to do anything fancy to get this project up and running. If you're in a hurry, you can simply toss the components into a clear, waterproof container and place it in your garden wherever you want to track light.

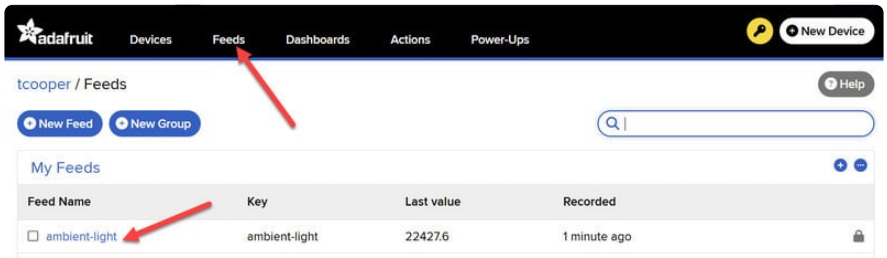
However, if you want to take a few extra minutes to secure your electronics, the Adafruit Swirly mounting system is a great option. It lets you cleanly screw down almost any Adafruit board or microcontroller. All you need is a Swirly board and a few plastic M2.5 mounting screws.

Learn more about the Adafruit Swirly grid system here. (https://adafru.it/1aFb)

Text emphasized with a red exclamation:

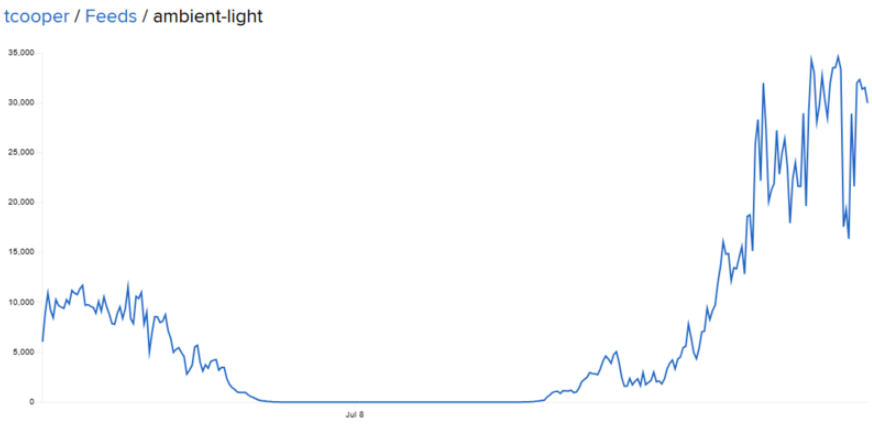
We ran this on a crisp early spring day, if its summer, make sure your battery does not get too hot (stay below 100 *F/40 *C)! Shade the battery if necessary or wait for a cool day.

Viewing the Data



If you have everything set up right, your new sun tracker should slowly start uploading data to Adafruit IO under the feed name 'ambient-light'.

Log into https://io.adafruit.com/ (https://adafru.it/fsU) and click on feeds, you should see this new feed in your My Feeds list. You can view the data by clicking the ambient-light feed name.



Interpreting Your Light Data

Once your sun tracker has been running for a few days, you'll have a clear picture of the "light profile" for that specific spot. While the raw numbers are interesting, the goal is to translate those peaks and valleys into a plant-friendly plan.

Understanding the Graph

The graph displays **Light Intensity** on the vertical axis and **Time** on the horizontal axis.

- **The Peaks:** Each peak represents the brightest part of the day. A tall, wide peak indicates a long stretch of direct, intense sunlight.
- **The Valleys:** These show when the area is in shadow, whether from a building, a tree, or simply because the sun has set.
- **Consistency:** Look for patterns over 3 to 5 days. A single sunny day is great, but seeing how clouds or shifting shadows affect the spot over a week gives you a more reliable average.

Matching Data to Plant Needs

Most plants come with a tag categorized by "Sun Requirements." Here is how to manually calculate which category your spot falls into based on your Adafruit IO data:

Light Category: Full Sun

- **Direct Sun Exposure (Daily):** 6+ Hours
- **What to look for on your graph:** A wide, high plateau of high-intensity values that lasts for the majority of the day.

Light Category: Partial Sun / Shade

- **Direct Sun Exposure (Daily):** 3 to 6 Hours
- **What to look for on your graph:** A narrower peak, or several smaller peaks if the spot gets "dappled" light through trees.

Light Category: Full Shade

- **Direct Sun Exposure (Daily):** Less than 3 Hours

- **What to look for on your graph:** Very low intensity values all day, or only one very brief spike of direct light.

How to Pick Your Plant

1. Now that you have your light data, it's time to cross-reference it with your "Hardiness Zone." Sunlight tells you if a plant will eat, but your climate zone tells you if it will survive.

    1. Count the "High" Hours: Look at your graph and count how many hours the light stays above a "direct sun" threshold.

        - 6+ Hours: Full Sun (Peppers, Tomatoes, Zinnias).
        - 3 to 6 Hours: Partial Sun/Shade (Root veggies, Herbs, Hydrangeas).
        - less than 3 Hours: Shade (Hostas, Ferns, Moss).

    2. Identify Your USDA Hardiness Zone: Before buying, look up your zip code on the USDA Plant Hardiness Zone Map (https://adafru.it/1aFc).

        - This number (e.g., Zone 4 here in Minneapolis) tells you the lowest temperature your area typically reaches in winter.
        - If you find a "Full Sun" plant that is only hardy to Zone 9, it will love your sunny deck in the summer, but it won't survive the winter outdoors. You'll need to treat it as an "annual" or bring it inside.

    3. Check for "Peak Heat" (10 AM to 4 PM): The intensity of the light matters as much as the duration.

        - Northern Climates (Zones 3-5): Morning sun is gentle. You often need those peak afternoon hours to get enough heat for "heat-loving" plants like peppers.
        - Southern Climates (Zones 8-11): Peak afternoon sun can actually be too intense. If your graph shows a massive spike at 2:00 PM in

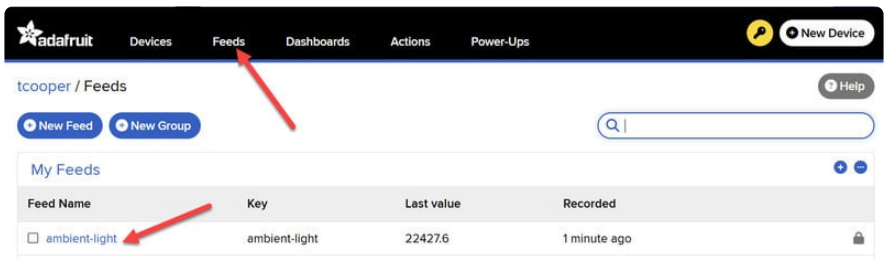
Arizona, you might actually need a "Partial Shade" plant to prevent scorching.

4. The Perfect Match: Your sun data matches the plant tag (e.g., 6+ hours) AND the plant is rated for your Hardiness Zone.

    - The "Summer Guest": The sun data matches, but the plant isn't hardy to your zone. Great for a seasonal pot, but don't plant it in the ground!
    - The "Reach" Plant: If you have 5 hours of sun but the plant wants 6, try a "Partial Sun" variety instead of forcing a "Full Sun" variety to struggle.

AI Plant Recommendations (optional)

While looking at a graph is great for a quick check, your sun tracker likely collected hundreds of data points. Instead of manually counting hours, you can use Gemini to analyze the "shape" of your data and cross-reference it with your specific climate.

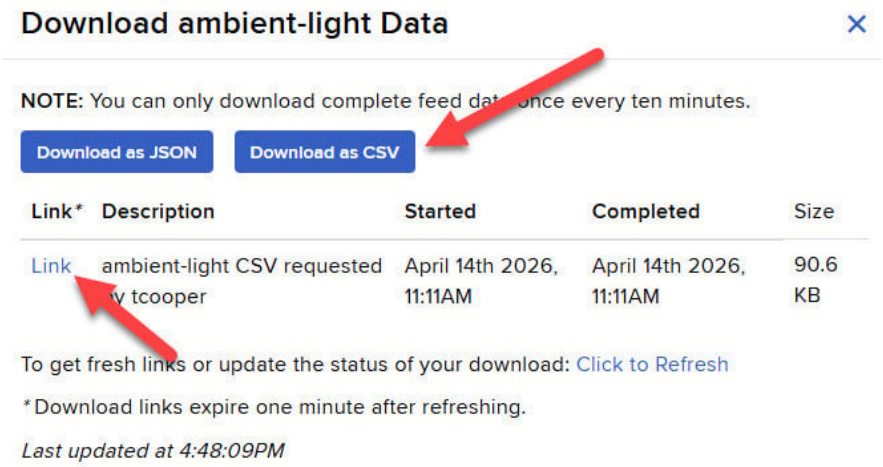


Step 1: Exporting Your Feed Data from Adafruit IO

First, to get the raw data out of the cloud and onto your computer.

1. Navigate to your Feeds page on Adafruit IO (https://adafru.it/fsU).
2. Select the feed you've been using for your sun tracker (e.g., `ambient-light`).
3. Right below the graph, you will see a big blue Download All Data button.

4. Click Download All Data. Then in the popup, choose the Download as CSV button. Wait a minute, then click the 'Click to Refresh' link and you should see a 'Link' show up on the left of the request.



Step 2: Quick Cleanup

AI works best when the data is "clean." Before we hand this over to Gemini, let's tidy it up.

1. Open your favorite spreadsheet app (like Google Sheets) and go to File > Import > Upload to open your CSV.
2. You'll see columns for `value`, `created_at`, and potentially some metadata.
3. Delete any columns that aren't the light value or the timestamp.
4. Filter for a "Good" couple of days. If you had a week of rain, that data isn't useful for finding "Full Sun" spots. Find a 24–48 hour window that was clear and sunny, and copy those rows into a new sheet.
5. Download as PDF or CSV that we can have an AI read.

Step 3: Let Gemini Interpret the Data

Now for the fun part. Give AI your data and your location (Hardiness Zone) so it can act as your personal botanist.

The Prompt

Upload your file to your AI of choice, then use a prompt similar to this:

"I am attaching a CSV of light intensity data collected by my sun tracker over the last 48 hours. I live in USDA Hardiness Zone 4.

Please analyze this data and:

1. Calculate the total number of 'High Intensity' sun hours per day.
2. Determine if this spot is Full Sun, Partial Sun, or Shade.
3. Recommend 5 plants (a mix of vegetables, herbs, and flowers) that would thrive in this specific light profile AND survive in Zone 4."

Step 4: The Response

Text emphasized with a yellow info:

As always with AI data, treat this as a good starting point for your own research. In my little test, I have found the data to be very accurate and it gave me some great ideas for what plants to consider for this spot in my garden.

I used Gemini to interpret my few days of data and here is what it came up with for me:

Based on the light measurements provided and the climate of Minneapolis, MN (USDA Hardiness Zone 4b), here is an analysis of your garden spot and recommendations for what will grow best there.

Light Analysis

The data indicates that this spot receives Partial Sun to Partial Shade.

- Intensity: The light levels peak between 10:00 AM and 3:00 PM (Central Time), with average values around 14,000 lux and maximum peaks reaching over 31,000 lux on sunny days.
- Duration: You are getting approximately 4 to 6 hours of significant light (above 10,000 lux) per day. This is ideal for plants that prefer "Part Sun" or "Part Shade"—they need some direct sunlight to thrive but may struggle or wilt in the intense, all-day heat of a full-sun exposure.

Recommended Plants for Minneapolis (Zone 4b)

Since Minneapolis has very cold winters, you need "hardy" plants that can survive temperatures down to -25°F.

1. Perennials (Flowers that come back every year)

- Hostas: These are the gold standard for partial shade in Minnesota. They come in hundreds of varieties and colors.
- Astilbe: These produce beautiful, plume-like flowers in summer and thrive in the 4–6 hour light window, especially if the soil stays moist.
- Bleeding Heart (Lamprocapnos spectabilis): A classic spring-blooming perennial that loves partial shade.
- Coral Bells (Heuchera): Grown primarily for their stunning foliage (purple, lime, silver, or red), they do very well in partial sun.
- Daylilies (Hemerocallis): Extremely hardy in Zone 4. While they love sun, many varieties will bloom well with 5–6 hours of light.
- Bee Balm (Monarda): Look for mildew-resistant varieties; they attract pollinators and handle partial sun well.

2. Shrubs

- Annabelle Hydrangea (Hydrangea arborescens): These are very hardy in Minnesota and produce massive white flower heads in partial shade.
- Serviceberry (Amelanchier): A native shrub/small tree that offers white spring flowers, edible berries, and great fall color.
- Dogwood (Cornus sericea): Specifically the Red-Osier Dogwood, which provides beautiful red stems in the winter.

3. Edibles

- Leafy Greens: Lettuce, spinach, kale, and Swiss chard actually prefer partial shade in the heat of a Minneapolis summer, as it prevents them from "bolting" (going to seed) too early.
- Herbs: Mint, chives, and cilantro will grow very well in this light profile.

- Root Vegetables: Carrots and beets can tolerate partial sun, though they may grow slightly slower than in full sun.