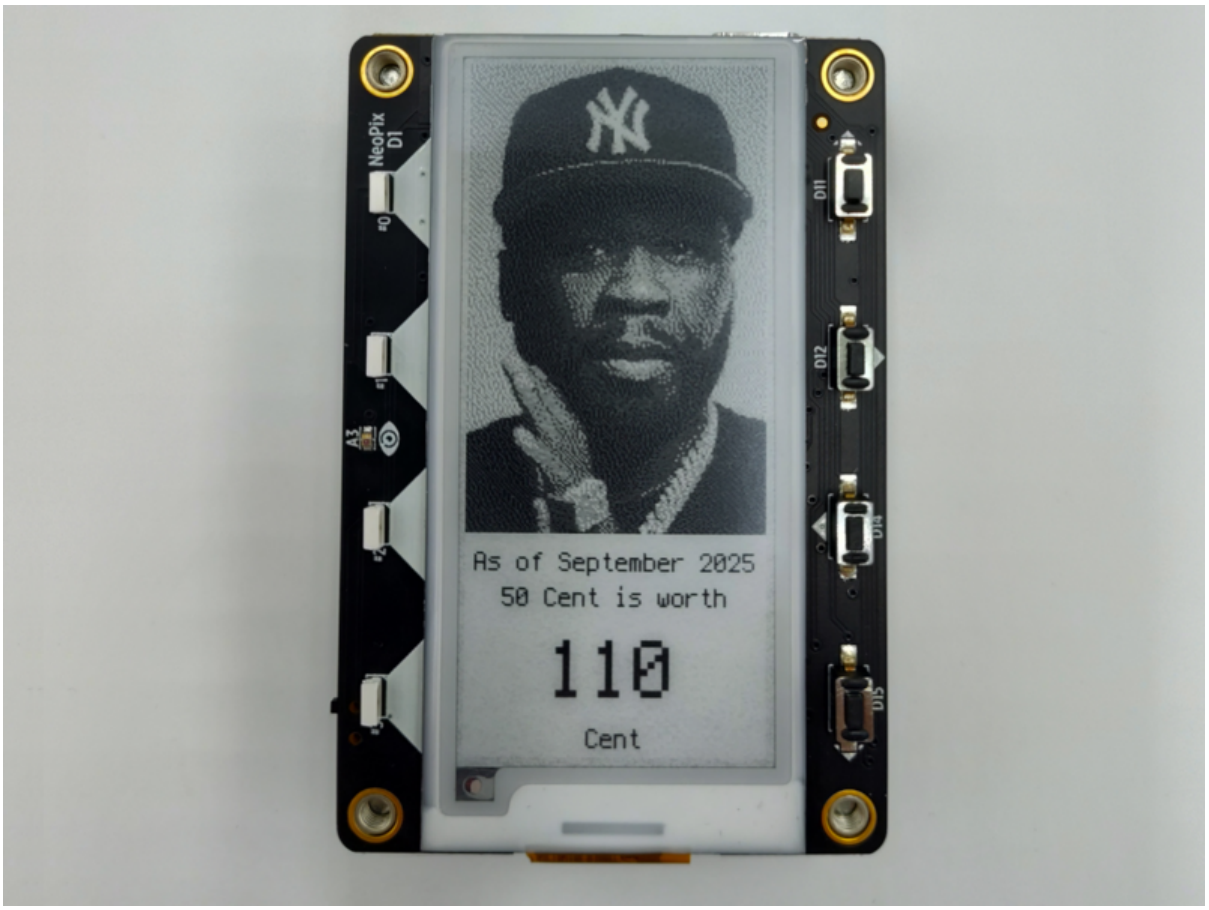
50 Cent CPI Tracker for MagTag

Created by Tim C



https://learn.adafruit.com/50-cent-cpi-tracker-for-magtag

Last updated on 2025-12-18 02:41:48 PM UTC

Table of Contents

Overview

The US Bureau of Labor Statistics public API (https://adafru.it/1aws) contains all sorts of interesting data, including current and historical Consumer Price Index or CPI. CPI is a measure of the average change over time in the prices paid by US consumers for common goods and services. There is data broken out regionally and across different market categories, but this project uses the universal all items data as a baseline for comparison.

1994 is reportedly the year (https://adafru.it/1awt) that Curtis Jackson adopted the nickname 50 Cent. This project uses the latest CPI value compared with the value from June 1994 to calculate how much 50 cents from 1994 would be worth in today's money adjusted for inflation. A dithered grayscale photo of the half-dollar Yankee's fan himself is displayed as a reminder to hustle harder and smarter. It's too early to tell what effect the removal of pennies from circulation might have on inflation - luckily the half dollar coin (https://adafru.it/1awu) is safe for now.

This project was inspired by the 50centadjustedforinflation.com (https://adafru.it/1awv) website made by Brian Moore.

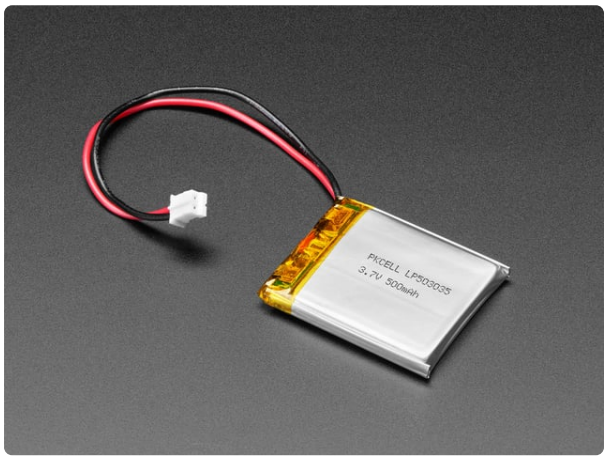
Parts

Adafruit MagTag - 2.9" Grayscale E-Ink WiFi Display

The Adafruit MagTag combines the ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen even when power is...
https://www.adafruit.com/product/4800

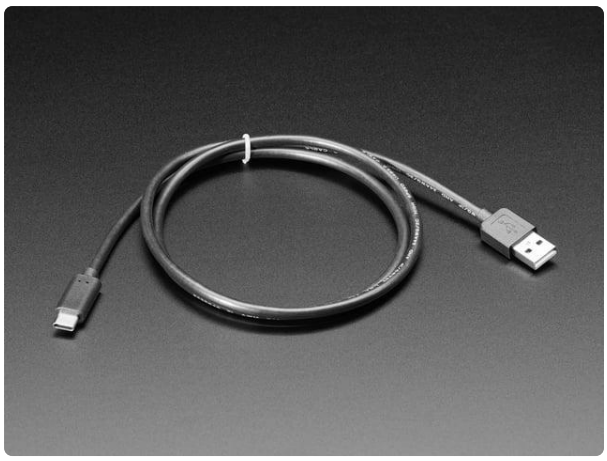
Mini Magnet Feet for RGB LED Matrices (Pack of 4)

Got a glorious RGB Matrix project you want to mount and display in your workspace or home? If you have one of the matrix panels listed below, you'll need a pack of these...
https://www.adafruit.com/product/4631



Lithium Ion Polymer Battery - 3.7v 500mAh

Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/1578



USB Type A to Type C Cable - approx 1 meter / 3 ft long
As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...
https://www.adafruit.com/product/4474

Install CircuitPython

e sure that you update the TinyUF2 Bootloader before installing uitPython!

Adafruit MagTag
By Kattni Rembor
 Update TinyUF2 Bootloader for CircuitPython 10 and Later

https://learn.adafruit.com/adafruit-magtag/update-tinyuf2-bootloader-for-circuitpython-10-4mb-boards-only

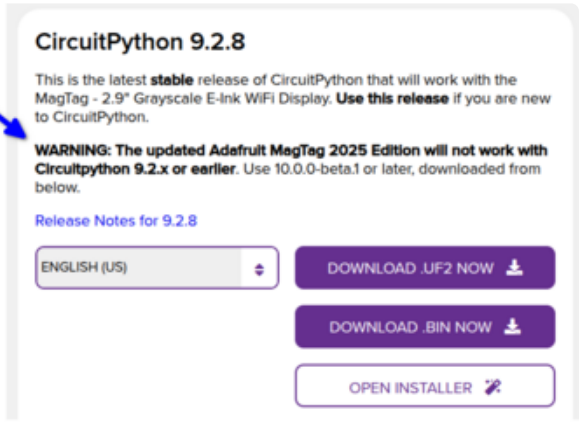
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

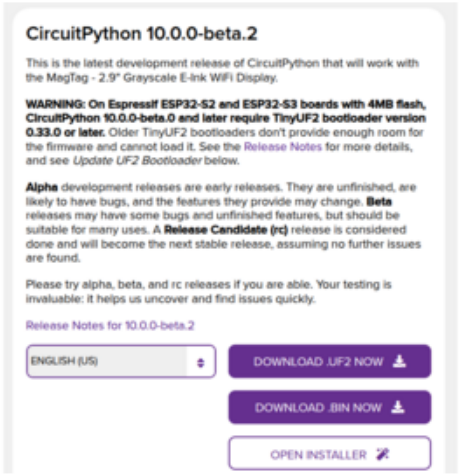
Set Up CircuitPython

Follow the steps to get CircuitPython installed on your MagTag.

https://adafru.it/OBd

ʔNING: The updated Adafruit MagTag 2025 Edition will not work with uitPython 9.2.x or earlier. Make sure you install 10.x.x or later!





Click the link above and download the latest .BIN and .UF2 file

You can use a 9.x.x release for a pre-2025 MagTag. You **must** use a 10.x.x release for the updated MagTag 2025 Edition.

(depending on how you program the ESP32S2 board you may need one or the other, might as well get both)

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

Plug your MagTag 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.

Option 1 - Load with UF2 Bootloader

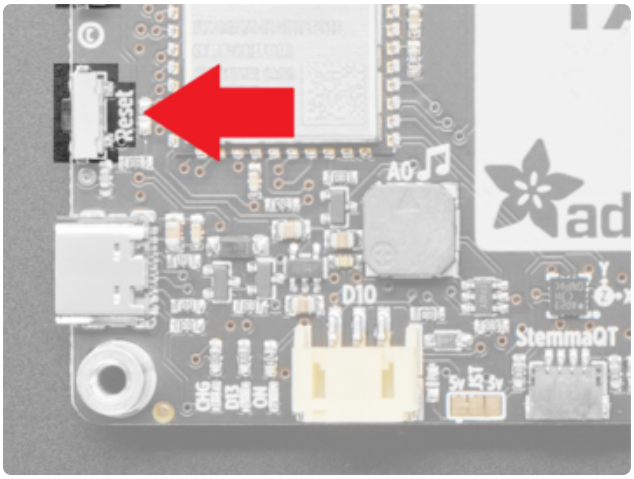
This is by far the easiest way to load CircuitPython. **However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.**

Still, try this first!

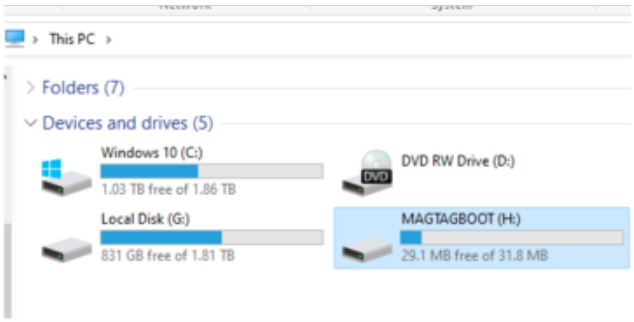
e sure that you update the TinyUF2 Bootloader before following these s for the UF2 bootloader!

Try Launching UF2 Bootloader

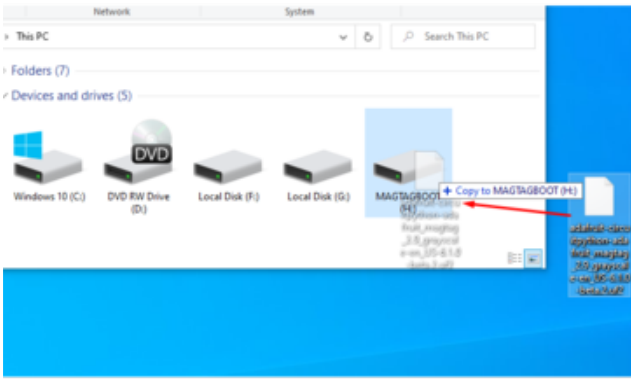
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



Launch UF2 by **double-clicking** the Reset button (the one next to the USB C port). You may have to try a few times to get the timing right.

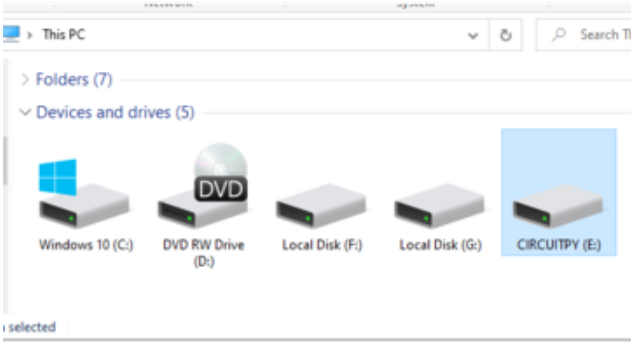


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



Copy the **UF2** file you downloaded at the first step of this tutorial onto the **MAGTAGBOOT** drive

If you're using Windows and you get an error at the end of the file copy that says **Error from the file copy, Error 0x800701B1: A device which does not exist was specified.** You can ignore this error, the bootloader sometimes disconnects without telling Windows, the install completed just fine and you can continue. If its really annoying, you can also upgrade the bootloader (the latest version of the UF2 bootloader fixes this warning) (https://adafru.it/Pfk)

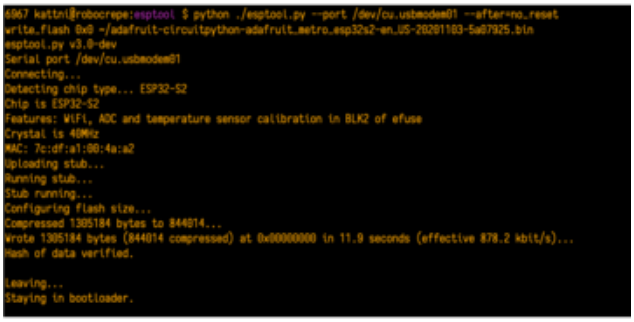


Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

Option 2 - Use esptool to load BIN file

If you have an original MagTag with while soldermask on the front, we didn't have UF2 written for the ESP32S2 yet so it will not come with the UF2 bootloader.

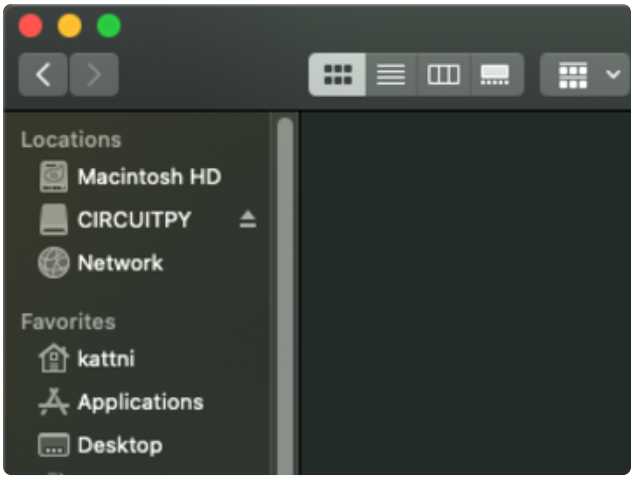
You can upload with **esptool** to the ROM (hardware) bootloader instead!



Follow the initial steps found in the Run esptool and check connection section of the ROM Bootloader page (https://adafru.it/OBc) to verify your environment is set up, your board is successfully connected, and which port it's using.

In the final command to write a binary file to the board, replace the port with your port, and replace "firmware.bin" with the the file you downloaded above.

The output should look something like the output in the image.



Press reset to exit the bootloader.

Your **CIRCUITPY** drive should appear!

You're all set! Go to the next pages.

Option 3 - Use Chrome Browser To Upload BIN file

If for some reason you cannot get esptool to run, you can always try using the Chrome-browser version of esptool we have written. This is handy if you don't have Python on your computer, or something is really weird with your setup that makes esptool not run (which happens sometimes and isn't worth debugging!) You can follow along on the Web Serial ESPTool (https://adafru.it/Pdq) page and either load the UF2 bootloader and then come back to Option 1 on this page, or you can download the CircuitPython BIN file directly using the tool in the same manner as the bootloader.

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.

· **settings.toml** file should be stored in the main directory of your **CUITPY** 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.**

every project uses the same variable name for each entry in the **ngs.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

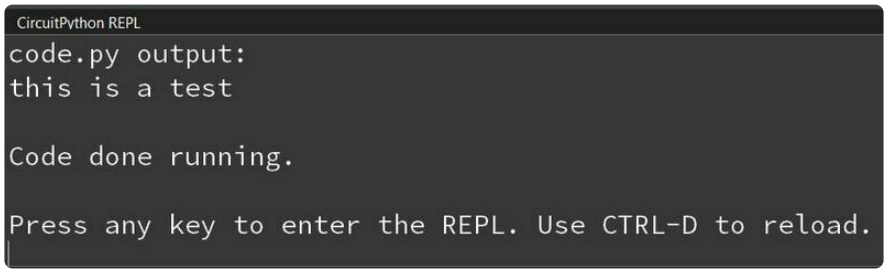
When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

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.

Code

Getting the Program's Files

To use the application, you need to obtain **code.py** with the program, and the **50_cent.bmp** file to place on the MagTag **CIRCUITPY** drive.

Thankfully, this can be done 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.

Connect your board to your computer via a known good data+power USB cable. The board should show up in your File Explorer/Finder (depending on your operating system) as a flash drive named **CIRCUITPY**.

Extract the contents of the zip file, copy the **lib** directory files to **CIRCUITPY/lib**. Copy the **code.py** and **50_cent.bmp** file to your **CIRCUITPY** drive. The program should self start.

Drive Structure

After copying the files, your drive should look like the listing below. It can contain other files as well, but must contain these at a minimum.

Code

The **code.py** for the 50 Cent CPI Tracker app is shown below.

```
# SPDX-FileCopyrightText: 2025 Tim C, written for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""
50 Cent Consumer Price Index Tracker

This project illustrates how to fetch CPI data from the
US Bureau of Labor Statistics API. CPI data from 1994,
when 50 Cent adopted his moniker, is compared with the latest
data to determine how much 50 Cent from 1994 is worth today.

This project was inspired by: https://50centadjustedforinflation.com/
"""
```

```
import json
import os
import time

import alarm
import displayio
from displayio import OnDiskBitmap, TileGrid, Group
import supervisor
from terminalio import FONT
import wifi

import adafruit_connection_manager
from adafruit_display_text.text_box import TextBox
import adafruit_requests


# CPI Data URL
latest_cpi_url = "https://api.bls.gov/publicAPI/v2/timeseries/data/CUUR0000SA0"
# CUUR0000SA0 the series ID for CPI-U (All items)
# Note: bls.gov API is limited to 20 requests per day for unregistered users

# CPI value for June 1994
june_94_cpi = 148.0

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

# Initalize Wifi, Socket Pool, Request Session
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)

try:
    # Connect to the WiFi network
    wifi.radio.connect(ssid, password)
except OSError as ose:
    print(f"OSError: {ose}")
print("Wifi connected!")

# Get display reference
display = supervisor.runtime.display

# Set to portrait orientation
display.rotation = 0

# Group to hold all visual elements
main_group = Group()

# Full display sized white background
white_bg_group = Group(scale=8)
white_bg_bmp = displayio.Bitmap(display.width // 8, display.height // 8, 1)
white_bg_palette = displayio.Palette(1)
white_bg_palette[0] = 0xFFFFFF
white_bg_tg = TileGrid(bitmap=white_bg_bmp, pixel_shader=white_bg_palette)
white_bg_group.append(white_bg_tg)
main_group.append(white_bg_group)

# 50 Cent photo
photo_bmp = OnDiskBitmap("50_cent.bmp")
photo_tg = TileGrid(bitmap=photo_bmp, pixel_shader=photo_bmp.pixel_shader)
main_group.append(photo_tg)


def get_latest_cpi_value():
    """
    Fetch the latest CPI data from BLS API.

    :return: tuple containing (latest CPI value, month, year)
```

```
    """
    try:
        response = requests.get(latest_cpi_url)
        try:
            json_data = response.json()
        except json.decoder.JSONDecodeError as json_error:
            raise Exception(f"JSON Parse error: {response.text}") from json_error

        latest_data = json_data["Results"]["series"][0]["data"][0]

        return (
            float(latest_data["value"]),
            latest_data["periodName"],
            latest_data["year"],
        )

    except Exception as e:
        raise Exception(f"Error fetching data from BLS API: {e}") from e


# fetch the latest CPI data
latest_cpi_value, month, year = get_latest_cpi_value()

# Label for the message below the photo
message_lbl = TextBox(
    FONT,
    128,
    TextBox.DYNAMIC_HEIGHT,
    align=TextBox.ALIGN_CENTER,
    text=f"As of {month} {year}\n50 Cent is worth",
    color=0x000000,
    scale=1,
)
message_lbl.anchor_point = (0, 0)
message_lbl.anchored_position = (0, photo_tg.tile_height + 6)
main_group.append(message_lbl)

# calculate current value of 50 cents from 1994
current_value = round(50.0 * (latest_cpi_value / june_94_cpi) + 0.5)

# Label for the calculated value
amount_lbl = TextBox(
    FONT,
    128 // 3,
    TextBox.DYNAMIC_HEIGHT,
    align=TextBox.ALIGN_CENTER,
    text=f"{current_value}",
    color=0x000000,
    scale=3,
)
amount_lbl.anchor_point = (0, 0)
amount_lbl.anchored_position = (0, photo_tg.tile_height + message_lbl.height + 12)
main_group.append(amount_lbl)

# Cent label at bottom of display
cent_lbl = TextBox(
    FONT,
    128,
    TextBox.DYNAMIC_HEIGHT,
    align=TextBox.ALIGN_CENTER,
    text="Cent",
    color=0x000000,
    scale=1,
)
cent_lbl.anchor_point = (0, 1)
cent_lbl.anchored_position = (0, display.height - 2)
main_group.append(cent_lbl)

# set main_group showing on the display
```

```
display.root_group = main_group
display.refresh()

# Sleep for 1 day
al = alarm.time.TimeAlarm(monotonic_time=time.monotonic() + 86400)
alarm.exit_and_deep_sleep_until_alarms(al)
# Does not return. Exits, and restarts after 1 day
```

Code Explanation

The code for the project is fairly basic, with only three main tasks to complete: fetch the current CPI data, calculate the adjusted value to show on the E-Ink display, and go into deep sleep waiting for a day to do it all over again.

WiFi & Fetch CPI Data

WiFi SSID and password are looked up from the **settings.toml** file using `os.getenv()` and a connection is made to the specified network. The code prepares to fetch API data by initializing the connection manager and requests `Session`.

The `get_latest_cpi_value()` function is responsible for making the request to the bls.gov API and extracting the relevant bits of data that are used, the current CPI value with the month and year that it came from. The function returns a tuple containing `(cpi_value, month, year)`.

Data inside the bls.gov API is broken into units called time series. Each series has a unique series ID that is used to fetch it from the API. In the most basic usage, a series ID is used at the end of the the API URL like this: `https://api.bls.gov/publicAPI/v2/timeseries/data/CUUR0000SA0`. More advanced usage allows fetching multiple series at once, or specifying other parameters to filter and find specific data of interest. This project sticks to the basics, see the API docs (https://adafru.it/1aws) for more details. `CUUR0000SA0` is the series ID for the "universal all items" CPI data that this project uses. If you're interested to go digging deeper into the API, the series ID format documentation page (https://adafru.it/1aww) lists other types of available data and how to format the ID for them.

Displayio Graphics

The code sets `display.rotation = 0` to put the display into the portrait orientation with the buttons on the right side of the E-Ink display.

The graphics for the project are comprised of the following elements:

- White background - A white rectangle that matches the display size. Everything else is layered on top of it. This makes it so the app can display black text on a white background. A `displayio.Group` that is set to `scale=8` is used so the `Bitmap` object can be smaller than the actual display size allowing it to use less RAM.
- 50 Cent Photo - An `OnDiskBitmap` is used to show the photo of 50 Cent. The grayscale photo has been dithered and quantized down to 4 total colors: black, white, light grey, and dark grey. These colors match the ones available on the MagTag's E-Ink display. This process gives the illusion of more color depth. The image file is sized appropriately to fit the 128px width of the display in the portrait orientation.
- Message Label - This is the first of 3 TextBox (https://adafru.it/1awx) elements. It's placed underneath the 50 Cent photo, and it gets the month and year from the CPI data to show a message like "As of September 2025 50 Cent is worth". The alignment setting on the `TextBox` allows all of the text to be shown center-aligned.
- Amount Label - The next `TextBox` is placed below the message label. It shows the calculated numerical value of 50 cents from 1994 adjusted for current inflation. This `TextBox` is initialized with `scale=3` to make it's characters display 3x larger than the others.
- Cent Label - This is the last TextBox, placed at the bottom of the display. It displays the "Cent" label beneath the larger amount value.

Sleep

After fetching and displaying the latest CPI data the script uses deep sleep (https://adafru.it/Pyc) to put the microcontroller into a low power state for `86400` seconds, or

1 full day. The program will start over from the beginning when it wakes up repeating the whole process.

The **code.py** file is embedded below. It contains comments that explain the purpose of each line or section of code.

```
# SPDX-FileCopyrightText: 2025 Tim C, written for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""
50 Cent Consumer Price Index Tracker

This project illustrates how to fetch CPI data from the
US Bureau of Labor Statistics API. CPI data from 1994,
when 50 Cent adopted his moniker, is compared with the latest
data to determine how much 50 Cent from 1994 is worth today.

This project was inspired by: https://50centadjustedforinflation.com/
"""
import json
import os
import time

import alarm
import displayio
from displayio import OnDiskBitmap, TileGrid, Group
import supervisor
from terminalio import FONT
import wifi

import adafruit_connection_manager
from adafruit_display_text.text_box import TextBox
import adafruit_requests


# CPI Data URL
latest_cpi_url = "https://api.bls.gov/publicAPI/v2/timeseries/data/CUUR0000SA0"
# CUUR0000SA0 the series ID for CPI-U (All items)
# Note: bls.gov API is limited to 20 requests per day for unregistered users

# CPI value for June 1994
june_94_cpi = 148.0

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

# Initalize Wifi, Socket Pool, Request Session
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)

try:
    # Connect to the WiFi network
    wifi.radio.connect(ssid, password)
except OSError as ose:
    print(f"OSError: {ose}")
print("Wifi connected!")

# Get display reference
display = supervisor.runtime.display

# Set to portrait orientation
display.rotation = 0
```

```
# Group to hold all visual elements
main_group = Group()

# Full display sized white background
white_bg_group = Group(scale=8)
white_bg_bmp = displayio.Bitmap(display.width // 8, display.height // 8, 1)
white_bg_palette = displayio.Palette(1)
white_bg_palette[0] = 0xFFFFFF
white_bg_tg = TileGrid(bitmap=white_bg_bmp, pixel_shader=white_bg_palette)
white_bg_group.append(white_bg_tg)
main_group.append(white_bg_group)

# 50 Cent photo
photo_bmp = OnDiskBitmap("50_cent.bmp")
photo_tg = TileGrid(bitmap=photo_bmp, pixel_shader=photo_bmp.pixel_shader)
main_group.append(photo_tg)


def get_latest_cpi_value():
    """
    Fetch the latest CPI data from BLS API.

    :return: tuple containing (latest CPI value, month, year)
    """
    try:
        response = requests.get(latest_cpi_url)
        try:
            json_data = response.json()
        except json.decoder.JSONDecodeError as json_error:
            raise Exception(f"JSON Parse error: {response.text}") from json_error

        latest_data = json_data["Results"]["series"][0]["data"][0]

        return (
            float(latest_data["value"]),
            latest_data["periodName"],
            latest_data["year"],
        )

    except Exception as e:
        raise Exception(f"Error fetching data from BLS API: {e}") from e


# fetch the latest CPI data
latest_cpi_value, month, year = get_latest_cpi_value()

# Label for the message below the photo
message_lbl = TextBox(
    FONT,
    128,
    TextBox.DYNAMIC_HEIGHT,
    align=TextBox.ALIGN_CENTER,
    text=f"As of {month} {year}\n50 Cent is worth",
    color=0x000000,
    scale=1,
)
message_lbl.anchor_point = (0, 0)
message_lbl.anchored_position = (0, photo_tg.tile_height + 6)
main_group.append(message_lbl)

# calculate current value of 50 cents from 1994
current_value = round(50.0 * (latest_cpi_value / june_94_cpi) + 0.5)

# Label for the calculated value
amount_lbl = TextBox(
    FONT,
    128 // 3,
    TextBox.DYNAMIC_HEIGHT,
```

```
    align=TextBox.ALIGN_CENTER,
    text=f"{current_value}",
    color=0x000000,
    scale=3,
)
amount_lbl.anchor_point = (0, 0)
amount_lbl.anchored_position = (0, photo_tg.tile_height + message_lbl.height + 12)
main_group.append(amount_lbl)

# Cent label at bottom of display
cent_lbl = TextBox(
    FONT,
    128,
    TextBox.DYNAMIC_HEIGHT,
    align=TextBox.ALIGN_CENTER,
    text="Cent",
    color=0x000000,
    scale=1,
)
cent_lbl.anchor_point = (0, 1)
cent_lbl.anchored_position = (0, display.height - 2)
main_group.append(cent_lbl)

# set main_group showing on the display
display.root_group = main_group
display.refresh()

# Sleep for 1 day
al = alarm.time.TimeAlarm(monotonic_time=time.monotonic() + 86400)
alarm.exit_and_deep_sleep_until_alarms(al)
# Does not return. Exits, and restarts after 1 day
```

Use

Magnet Feet & Battery

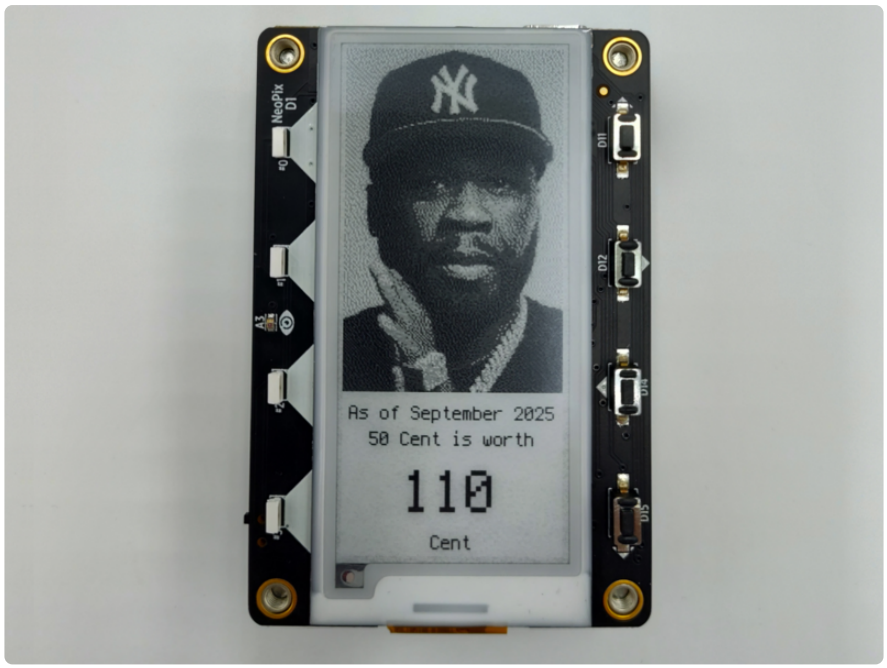
Attach the 4 M3 magnetic feet to the corner posts on the MagTag by screwing them in. Connect the 2-pin JST connector on the LiPo battery to the JST connector on the back of the MagTag.

Now the MagTag can be attached to a fridge, filing cabinet, or other magnetic surface. The battery allows the project to run even while a USB C cable is not connected.

Run the CPI Tracker

After copying the **code.py** and **50_cent.bmp** files to the MagTag's **CIRCUITPY** drive the project will automatically start. If you see an error message be sure that you have configured your WiFi credentials inside of **settings.toml** as shown on this guide page (https://adafru.it/1awy).

Charge the LiPo occasionally by plugging in the MagTag via the USB C plug to either a power adapter or computer. If the battery has died you can tell by the small green "on" LED on the back of the MagTag being off.


NeoPix
D1
D11
D12
D10
D5
A3
As of September 2025
50 Cent is worth
110
Cent