



MagTag Literary Quote Clock

Created by Eva Herrada



<https://learn.adafruit.com/magtag-literary-quote-clock>

Last updated on 2024-11-29 10:06:42 AM EST

Table of Contents

Overview	3
<hr/>	
• Parts	
Install CircuitPython	5
<hr/>	
• Set Up CircuitPython	
• Option 1 - Load with UF2 Bootloader	
• Try Launching UF2 Bootloader	
• Option 2 - Use esptool to load BIN file	
• Option 3 - Use Chrome Browser To Upload BIN file	
CircuitPython Internet Test	9
<hr/>	
• The settings.toml File	
• IPv6 Networking	
Getting The Date & Time	16
<hr/>	
• Step 1) Make an Adafruit account	
• Step 2) Sign into Adafruit IO	
• Step 3) Get your Adafruit IO Key	
• Step 4) Upload Test Python Code	
Code the MagTag Quote Clock	19
<hr/>	
• Installing Project Code	
• Code run-through	
Using The MagTag Quote Clock	28
<hr/>	

Overview



This project is perfect for the book lover in all of us. In this project you will create a clock that tells the time using quotes from books. It'll update every five or so minutes (there are enough quotes to update more often but doing so has the potential to wear out your elnk display). Each time, it will display a quote that has the time highlighted in bold.

The code for this project also has some useful functions for mixing multiple fonts in a relatively seamless text block, so if you're trying to do that check out the code attached.

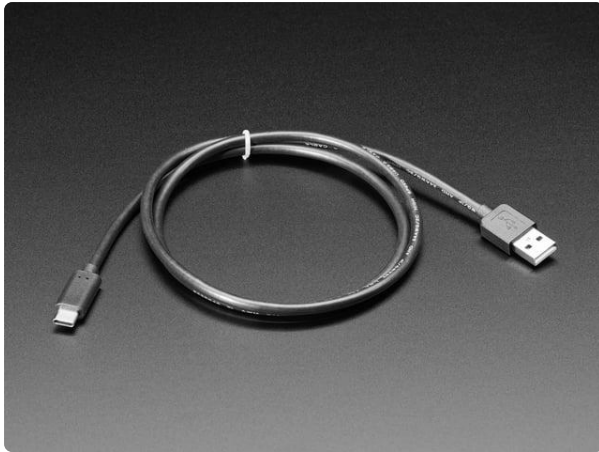
Parts



[Adafruit MagTag - 2.9" Grayscale E-Ink WiFi Display](https://www.adafruit.com/product/4800)

The Adafruit MagTag combines the new 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...

<https://www.adafruit.com/product/4800>



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>



5V 2A Switching Power Supply w/ USB-A Connector

Our 5V 2A USB power adapter is the perfect choice for powering single-board computers like Raspberry Pi, BeagleBone, or anything else that's power-hungry! This adapter was...

<https://www.adafruit.com/product/1994>

If you'd like to mount yours like I did in the pictures:

This project is not designed with low power consumption in mind. I would not recommend running it off of a LiPo

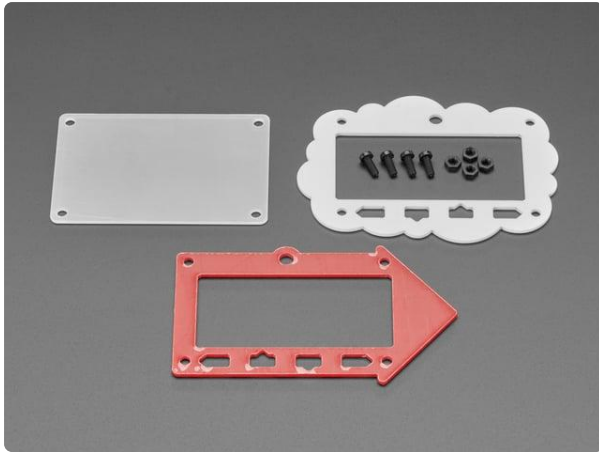


Mini Magnet Feet for RGB LED Matrices (Pack of 4)

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>

If you would like an acrylic frame:



[Acrylic + Hardware Kit for Adafruit MagTag](https://www.adafruit.com/product/4807)

Here is the perfect kit with two faceplate options for your MagTag, including a vivid Red Arrow plate and a dreamy white Cloud plate. And of course, the mounting hardware is included,...

<https://www.adafruit.com/product/4807>

Install CircuitPython

[CircuitPython](https://adafru.it/tB7) (<https://adafru.it/tB7>) is a derivative of [MicroPython](https://adafru.it/BeZ) (<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.

Download the latest CircuitPython
for your board from
circuitpython.org

<https://adafru.it/OBd>

CircuitPython 6.1.0-beta.2

This is the latest unstable release of CircuitPython that will work with the MagTag - 2.9" Grayscale E-Ink WiFi Display.

Unstable builds have the latest features but are more likely to have critical bugs.

[Release Notes for 6.1.0-beta.2](#)

ENGLISH

DOWNLOAD .BIN NOW

DOWNLOAD .UF2 NOW

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

(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

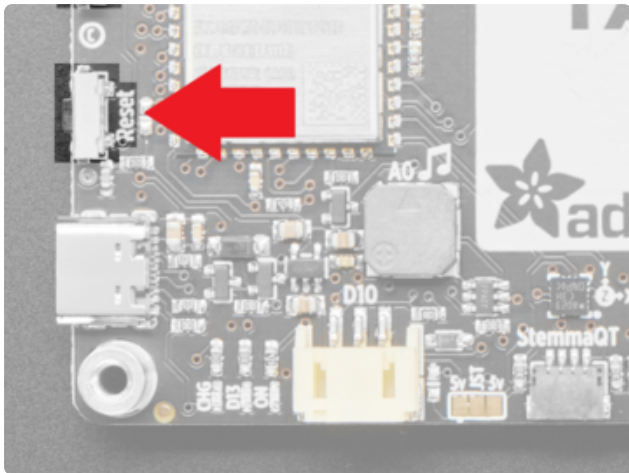
This is by far the easiest way to load CircuitPython. However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.

Still, try this first!

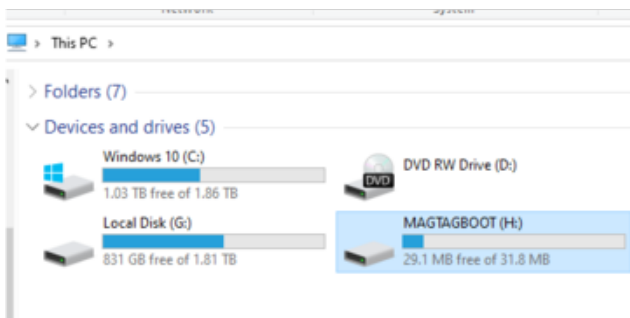


Try Launching UF2 Bootloader

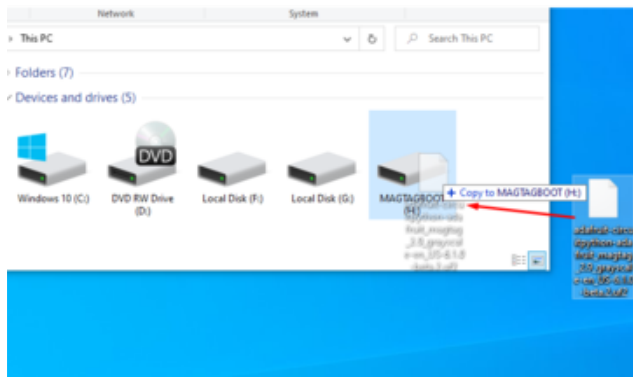
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



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.

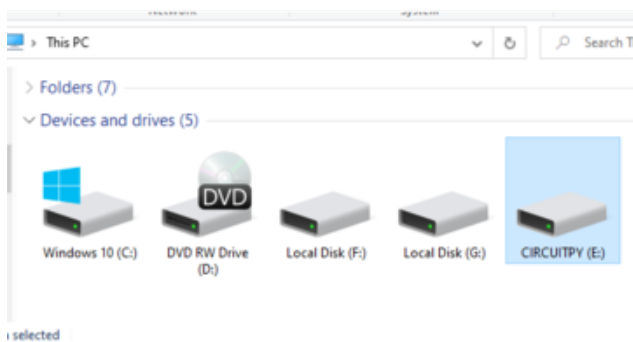


If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



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 \(the latest version of the UF2 bootloader fixes this warning\) \(https://adafru.it/Pfk\)](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 white 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!


```

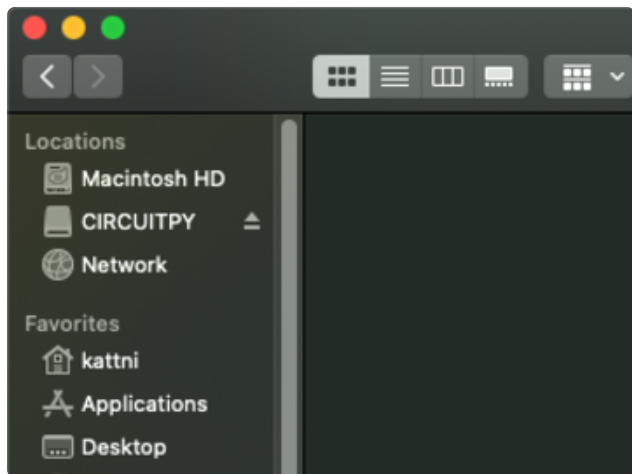
kattni@roborepe:esptool $ python ./esptool.py --port /dev/cu.usbmodem01 --after-no-reset
write_flash 0x0 ~/adafruit-circuitpython-adafruit_metro_esp32-en_US-20201103-5a07925.bin
esptool.py v3.0-dev
Serial port /dev/cu.usbmodem01
Connecting...
Detecting chip type... ESP32-S2
Chip is ESP32-S2
Features: WiFi, ADC and temperature sensor calibration in BLK2 of efuse
Crystal is 40MHz
MAC: 7c:df:a1:00:4a:a2
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Compressed 1305184 bytes to 844014...
Wrote 1305184 bytes (844014 compressed) at 0x00000000 in 11.9 seconds (effective 878.2 kbit/s)...
Flash of data verified.
Leaving...
Staying in bootloader.

```

Follow the initial steps found in the [Run esptool and check connection section of the ROM Bootloader page \(https://adafruit.it/OBc\)](#) 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 \(https://adafruit.it/Pdq\)](https://adafruit.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.

CircuitPython Internet Test

One of the great things about the ESP32 is the built-in WiFi capabilities. This page covers the basics of getting connected using CircuitPython.

The first thing you need to do is update your **code.py** to the following. Click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py** file to your **CIRCUITPY** drive.

```
# SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import ipaddress
import ssl
import wifi
import socketpool
import adafruit_requests

# URLs to fetch from
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_QUOTES_URL = "https://www.adafruit.com/api/quotes.php"
JSON_STARS_URL = "https://api.github.com/repos/adafruit/circuitpython"

print("ESP32-S2 WebClient Test")

print(f"My MAC address: {[hex(i) for i in wifi.radio.mac_address]}")

print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
                                           network.rssi, network.channel))
wifi.radio.stop_scanning_networks()

print(f"Connecting to {os.getenv('CIRCUITPY_WIFI_SSID')}")
wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print(f"Connected to {os.getenv('CIRCUITPY_WIFI_SSID')}")
print(f"My IP address: {wifi.radio.ipv4_address}")

ping_ip = ipaddress.IPv4Address("8.8.8.8")
ping = wifi.radio.ping(ip=ping_ip)

# retry once if timed out
if ping is None:
    ping = wifi.radio.ping(ip=ping_ip)

if ping is None:
    print("Couldn't ping 'google.com' successfully")
else:
    # convert s to ms
    print(f"Pinging 'google.com' took: {ping * 1000} ms")

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())

print(f"Fetching text from {TEXT_URL}")
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)

print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)

print()
```

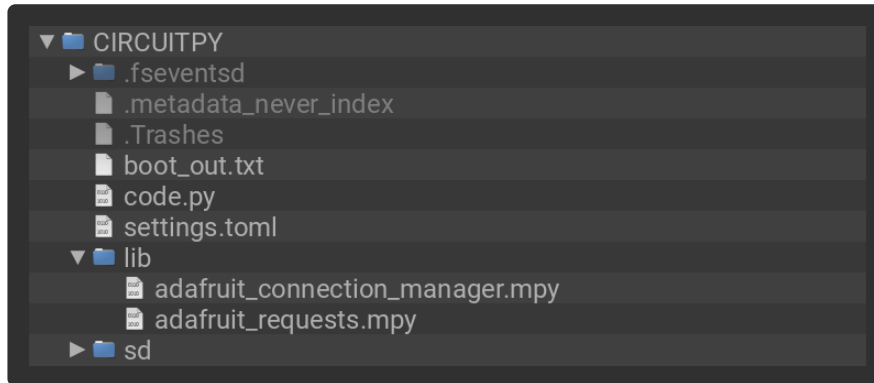
```

print(f"Fetching and parsing json from {JSON_STARS_URL}")
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print(f"CircuitPython GitHub Stars: {response.json()['stargazers_count']}")
print("-" * 40)

print("Done")

```

Your **CIRCUITPY** drive should resemble the following.



To get connected, the next thing you need to do is update the **settings.toml** file.

The settings.toml 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 **settings.toml** file, that is on your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

If you have a fresh install of CircuitPython on your board, the initial **settings.toml** file on your **CIRCUITPY** drive is empty.

To get started, you can update the **settings.toml** on your **CIRCUITPY** drive to contain the following code.

```

# SPDX-FileCopyrightText: 2023 Adafruit Industries
#
# SPDX-License-Identifier: MIT

# This is where you store the credentials necessary for your code.
# The associated demo only requires WiFi, but you can include any
# credentials here, such as Adafruit IO username and key, etc.
CIRCUITPY_WIFI_SSID = "your-wifi-ssid"
CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"

```

This file should contain a series of Python variables, each assigned to a string. Each variable should describe what it represents (say `wifi_ssid`), followed by an

= (equals sign), followed by the data in the form of a Python string (such as `"my-wifi-password"` including the quote marks).

At a minimum you'll need to add/update your WiFi SSID and WiFi password, so do that now!

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.

For the correct time zone string, look at <http://worldtimeapi.org/timezones> (<https://adafruit.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 **settings.toml** - keep that out of GitHub, Discord or other project-sharing sites.

Don't share your settings.toml file! It has your passwords and API keys in it!

If you connect to the serial console, you should see something like the following:

```
1. screen /Users/brentrubell (screen)
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
ESP32-S2 WebClient Test
My MAC addr: ['0x7c', '0xdf', '0xa1', '0x0', '0x52', '0xa0']
Avaliable WiFi networks:
  Brunelleschi      RSSI: -84      Channel: 6
  Transit           RSSI: -54      Channel: 1
  Fios-5dLNb        RSSI: -66      Channel: 1
  disconnectededer   RSSI: -86      Channel: 1
  SKJFios-ZV007     RSSI: -83      Channel: 11
  Fios-QIVUQ        RSSI: -83      Channel: 11
  Fios-ZV007        RSSI: -85      Channel: 11
  [REDACTED]         RSSI: -58      Channel: 2
  [REDACTED]         RSSI: -76      Channel: 8
  NETGEAR52         RSSI: -81      Channel: 10
Connecting to Transit
Connected to Transit!
None
My IP address is 192.168.1.182
Ping google.com: 0.065000 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
-----
This is a test of Adafruit WiFi!
If you can read this, its working :)
-----
Fetching json from https://www.adafruit.com/api/quotes.php
-----
[{'text': 'Science, my lad, is made up of mistakes, but they are mistakes which it is u
seful to make, because they lead little by little to the truth', 'author': 'Jules Verne
'}]
-----
Fetching and parsing json from https://api.github.com/repos/adafruit/circuitpython
-----
CircuitPython GitHub Stars 1896
-----
done
```

In order, the example code...

Checks the ESP32's MAC address.

```
print(f"My MAC address: {[hex(i) for i in wifi.radio.mac_address]}")
```

Performs a scan of all access points and prints out the access point's name (SSID), signal strength (RSSI), and channel.

```
print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
                                             network.rssi, network.channel))
wifi.radio.stop_scanning_networks()
```

Connects to the access point you defined in the **settings.toml** file, and prints out its local IP address.

```
print(f"Connecting to {os.getenv('WIFI_SSID')}")
wifi.radio.connect(os.getenv("WIFI_SSID"), os.getenv("WIFI_PASSWORD"))
print(f"Connected to {os.getenv('WIFI_SSID')}")
print(f"My IP address: {wifi.radio.ipv4_address}")
```

Attempts to ping a Google DNS server to test connectivity. If a ping fails, it returns **None**. Initial pings can sometimes fail for various reasons. So, if the initial ping is successful (**is not None**), it will print the echo speed in ms. If the initial ping fails, it

will try one more time to ping, and then print the returned value. If the second ping fails, it will result in `"Ping google.com: None ms"` being printed to the serial console. Failure to ping does not always indicate a lack of connectivity, so the code will continue to run.

```
ping_ip = ipaddress.IPv4Address("8.8.8.8")
ping = wifi.radio.ping(ip=ping_ip) * 1000
if ping is not None:
    print(f"Ping google.com: {ping} ms")
else:
    ping = wifi.radio.ping(ip=ping_ip)
    print(f"Ping google.com: {ping} ms")
```

The code creates a socketpool using the wifi radio's available sockets. This is performed so we don't need to re-use sockets. Then, it initializes a new instance of the [requests](https://adafruit.it/E9o) (<https://adafruit.it/E9o>) interface - which makes getting data from the internet really really easy.

```
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
```

To read in plain-text from a web URL, call `requests.get` - you may pass in either a http, or a https url for SSL connectivity.

```
print(f"Fetching text from {TEXT_URL}")
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)
```

Requests can also display a JSON-formatted response from a web URL using a call to `requests.get`.

```
print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)
```

Finally, you can fetch and parse a JSON URL using `requests.get`. This code snippet obtains the `stargazers_count` field from a call to the GitHub API.

```
print(f"Fetching and parsing json from {JSON_STARS_URL}")
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print(f"CircuitPython GitHub Stars: {response.json()['stargazers_count']}")
print("-" * 40)
```


OK you now have your ESP32 board set up with a proper **settings.toml** file and can connect over the Internet. If not, check that your **settings.toml** file has the right SSID and password and retrace your steps until you get the Internet connectivity working!

IPv6 Networking

Starting in CircuitPython 9.2, IPv6 networking is available on most Espressif wifi boards. Socket-using libraries like **adafruit_requests** and **adafruit_ntp** will need to be updated to use the new APIs and for now can only connect to services on IPv4.

IPv6 connectivity & privacy

IPv6 addresses are divided into many special kinds, and many of those kinds (like those starting with **FC**, **FD**, **FE**) are private or local; Addresses starting with other prefixes like **2002:** and **2001:** are globally routable. In 2024, far from all ISPs and home networks support IPv6 internet connectivity. For more info consult resources like [Wikipedia \(https://adafru.it/1a4z\)](https://adafru.it/1a4z). If you're interested in global IPv6 connectivity you can use services like [Hurricane Electric \(https://adafru.it/1a4A\)](https://adafru.it/1a4A) to create an "IPv6 tunnel" (free as of 2024, but requires expertise and a compatible router or host computer to set up)

It's also important to be aware that, as currently implemented by Espressif, there are privacy concerns especially when these devices operate on the global IPv6 network: The device's unique identifier (its EUI-64 or MAC address) is used by default as part of its IPv6 address. This means that the device identity can be tracked across multiple networks by any service it connects to.

Enable IPv6 networking

Due to the privacy consideration, IPv6 networking is not automatically enabled. Instead, it must be explicitly enabled by a call to **start_dhcp_client** with the **ipv6=True** argument specified:

```
wifi.start_dhcp_client(ipv6=True)
```

Check IP addresses

The read-only **addresses** property of the **wifi.radio** object holds all addresses, including IPv4 and IPv6 addresses:

```
>>> wifi.radio.addresses
('FE80::7EDF:A1FF:FE00:518C', 'FD5F:3F5C:FE50:0:7EDF:A1FF:FE00:518C', '10.0.3.96')
```

The **wifi.radio.dns** servers can be IPv4 or IPv6:

```
>>> wifi.radio.dns
('FD5F:3F5C:FE50::1',)
```

```
>>> wifi.radio.dns = ("1.1.1.1",)
>>> wifi.radio.dns
('1.1.1.1',)
```

Ping v6 networks

`wifi.radio.ping` accepts v6 addresses and names:

```
>>> wifi.radio.ping("google.com")
0.043
>>> wifi.radio.ping("ipv6.google.com")
0.048
```

Create & use IPv6 sockets

Use the address family `socket.AF_INET6`. After the socket is created, use methods like `connect`, `send`, `recvfrom_into`, etc just like for IPv4 sockets. This code snippet shows communicating with a private-network NTP server; this IPv6 address will not work on your network:

```
>>> ntp_addr = ("fd5f:3f5c:fe50::20e", 123)
>>> PACKET_SIZE = 48
>>>
>>> buf = bytearray(PACKET_SIZE)
>>> with socket.socket(socket.AF_INET6, socket.SOCK_DGRAM) as s:
...     s.settimeout(1)
...     buf[0] = 0b0010_0011
...     s.sendto(buf, ntp_addr)
...     print(s.recvfrom_into(buf))
...     print(buf)
...
48
(48, ('fd5f:3f5c:fe50::20e', 123))
bytearray(b'$\x01\x03\xeb\x00\x00\x00\x00\x00\x00\x00GPS\x00\xeaA0h\x07s;
\xc0\x00\x00\x00\x00\x00\x00\x00\xeaA0n\xeb4\x82-\xeaA0n\xebAU\xb1')
```

Getting The Date & Time

A very common need for projects is to know the current date and time. Especially when you want to deep sleep until an event, or you want to change your display based on what day, time, date, etc. it is

Determining the correct local time is really really hard. There are various time zones, Daylight Savings dates, leap seconds, etc. Trying to get NTP time and then back-calculating what the local time is, is extraordinarily hard on a microcontroller just isn't worth the effort and it will get out of sync as laws change anyways.

For that reason, we have the free adafruit.io time service. **Free for anyone with a free adafruit.io account.** You do need an account because we have to keep accidentally mis-programmed-board from overwhelming adafruit.io and lock them out temporarily. Again, it's free!

There are other services like WorldTimeAPI, but we don't use those for our guides because they are nice people and we don't want to accidentally overload their site. Also, there's a chance it may eventually go down or also require an account.

Step 1) Make an Adafruit account

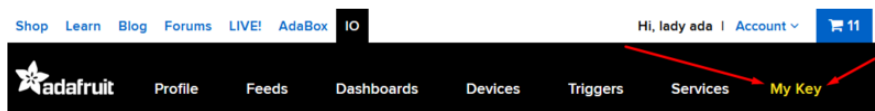
It's free! Visit <https://accounts.adafruit.com/> (<https://adafru.it/dyy>) to register and make an account if you do not already have one

Step 2) Sign into Adafruit IO

Head over to io.adafruit.com (<https://adafru.it/fsU>) and click **Sign In** to log into IO using your Adafruit account. It's free and fast to join.

Step 3) Get your Adafruit IO Key

Click on **My Key** in the top bar



You will get a popup with your **Username** and **Key** (In this screenshot, we've covered it with red blocks)

YOUR ADAFRUIT IO KEY ✕

Your Adafruit IO Key should be kept in a safe place and treated with the same care as your Adafruit username and password. People who have access to your Adafruit IO Key can view all of your data, create new feeds for your account, and manipulate your active feeds.

If you need to regenerate a new Adafruit IO Key, all of your existing programs and scripts will need to be manually changed to the new key.

Username

Active Key

REGENERATE KEY

[Hide Code Samples](#)

Go to the **settings.toml** file on your CIRCUITPY drive and add three lines for **AIO_USERNAME**, **ADAFRUIT_AIO_KEY** and **TIMEZONE** so you get something like the following:

```
# 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  
  
CIRCUITPY_WIFI_SSID = "your-wifi-ssid"
```

```

CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"
ADAFRUIT_AIO_USERNAME = "your-adafruit-io-username"
ADAFRUIT_AIO_KEY = "your-adafruit-io-key"
# Timezone names from http://worldtimeapi.org/timezones
TIMEZONE="America/New_York"

```

The timezone is optional, if you don't have that entry, adafruit.io will guess your timezone based on geographic IP address lookup. You can visit <http://worldtimeapi.org/timezones> (<https://adafru.it/EcP>) to see all the time zones available (even though we do not use Worldtime for time-keeping, we do use the same time zone table).

Step 4) Upload Test Python Code

This code is like the Internet Test code from before, but this time it will connect to adafruit.io and get the local time

```

import ipaddress
import os
import ssl
import wifi
import socketpool
import adafruit_requests
import secrets

# Get our username, key and desired timezone
ssid = os.getenv("CIRCUITPY_WIFI_SSID")
password = os.getenv("CIRCUITPY_WIFI_PASSWORD")
aio_username = os.getenv("ADAFRUIT_AIO_USERNAME")
aio_key = os.getenv("ADAFRUIT_AIO_KEY")
timezone = os.getenv("TIMEZONE")
TIME_URL = f"https://io.adafruit.com/api/v2/{aio_username}/integrations/time/strftime?x-aio-key={aio_key}&tz={timezone}"
TIME_URL += "&fmt=%25Y-%25m-%25d+%25H%3A%25M%3A%25S.%25L+%25j+%25u+%25z+%25Z"

print("ESP32-S2 Adafruit IO Time test")

print("My MAC addr:", [hex(i) for i in wifi.radio.mac_address])

print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
        network.rssi, network.channel))
wifi.radio.stop_scanning_networks()

print("Connecting to", ssid)
wifi.radio.connect(ssid, password)
print(f"Connected to {ssid}!")
print("My IP address is", wifi.radio.ipv4_address)

ipv4 = ipaddress.ip_address("8.8.4.4")
print("Ping google.com:", wifi.radio.ping(ipv4), "ms")

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())

print("Fetching text from", TIME_URL)
response = requests.get(TIME_URL)
print("-" * 40)

```

```
print(response.text)
print("-" * 40)
```

After running this, you will see something like the below text. We have blocked out the part with the secret username and key data!

```
Connecting to adafruit
Connected to adafruit!
My IP address is 10.0.1.148
Ping google.com: 0.008000 ms
Fetching text from https://io.adafruit.com/api/v2/[REDACTED]/integrations/time/strftime?x-aio-
key=[REDACTED]&fmt=%25Y-%25m-%25d+%25H%3A%25M%3A%25S.%25L+%25j+%25u+%25z+%25Z
-----
2020-12-05 18:51:32.145 340 6 -0500 EST
-----
```

Note at the end you will get the date, time, and your timezone! If so, you have correctly configured your **settings.toml** and can continue to the next steps!

Code the MagTag Quote Clock

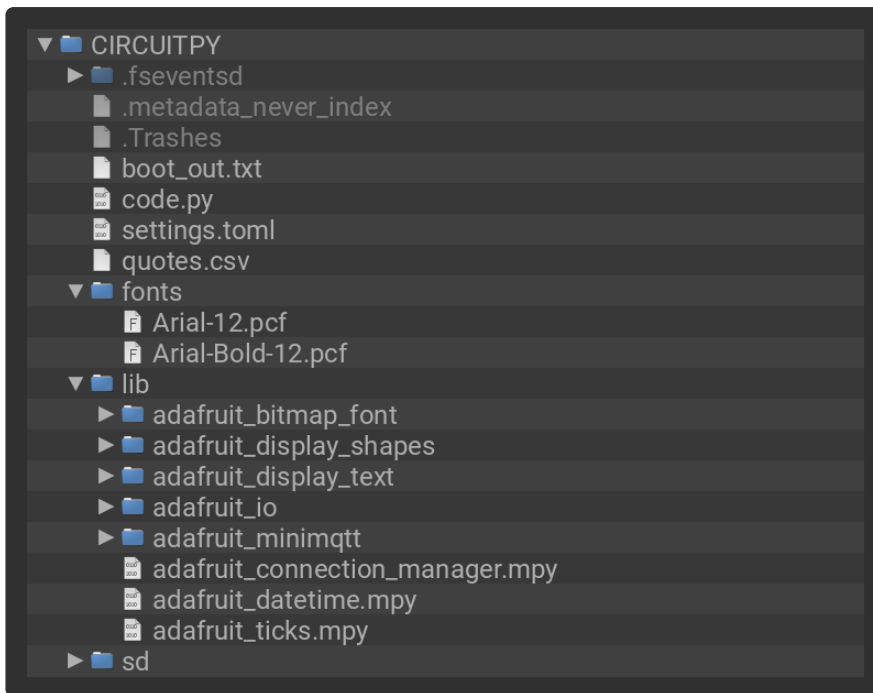
Installing Project Code

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

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **literary-clock/** and then click on the directory that matches the version of CircuitPython you're using.

Connect your MagTag board to your computer via a known good USB data+power cable. The board should show up as a thumb drive named **CIRCUITPY** in Explorer or Finder (depending on your operating system). Copy the contents of that directory to your **CIRCUITPY** drive.

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



```
# SPDX-FileCopyrightText: 2022 Eva Herrada for Adafruit Industries
# SPDX-License-Identifier: MIT
```

```
import time

import ssl
import gc
import socketpool
import wifi
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT
import adafruit_datetime
import adafruit_display_text
from adafruit_display_text import label
import board
from adafruit_bitmap_font import bitmap_font
import displayio
from adafruit_display_shapes.rect import Rect

UTC_OFFSET = -4

quotes = {}
with open("quotes.csv", "r", encoding="UTF-8") as F:
    for quote_line in F:
        split = quote_line.split("|")
        quotes[split[0]] = split[1:]

display = board.DISPLAY
splash = displayio.Group()
display.root_group = splash

arial = bitmap_font.load_font("fonts/Arial-12.pcf")
bold = bitmap_font.load_font("fonts/Arial-Bold-12.pcf")
LINE_SPACING = 0.8
HEIGHT = arial.get_bounding_box()[1]
QUOTE_X = 10
QUOTE_Y = 7

rect = Rect(0, 0, 296, 128, fill=0xFFFFFF, outline=0xFFFFFF)
splash.append(rect)

quote = label.Label(
```



```

        font=arial,
        x=QUOTE_X,
        y=QUOTE_Y,
        color=0x000000,
        line_spacing=LINE_SPACING,
    )

    splash.append(quote)
    time_label = label.Label(
        font=bold,
        color=0x000000,
        line_spacing=LINE_SPACING,
    )
    splash.append(time_label)

    time_label_2 = label.Label(
        font=bold,
        color=0x000000,
        line_spacing=LINE_SPACING,
    )
    splash.append(time_label_2)

    after_label = label.Label(
        font=arial,
        color=0x000000,
        line_spacing=LINE_SPACING,
    )
    splash.append(after_label)

    after_label_2 = label.Label(
        font=arial,
        color=0x000000,
        line_spacing=LINE_SPACING,
    )
    splash.append(after_label_2)

    author_label = label.Label(
        font=arial, x=QUOTE_X, y=115, color=0x000000, line_spacing=LINE_SPACING
    )
    splash.append(author_label)

    try:
        from secrets import secrets
    except ImportError:
        print("WiFi secrets are kept in secrets.py, please add them there!")
        raise

    aio_username = secrets["aio_username"]
    aio_key = secrets["aio_key"]

    print(f"Connecting to {secrets['ssid']}")
    wifi.radio.connect(secrets["ssid"], secrets["password"])
    print(f"Connected to {secrets['ssid']}!")

    def get_width(font, text):
        return sum(font.get_glyph(ord(c)).shift_x for c in text)

    def smart_split(text, font, width):
        words = ""
        spl = text.split(" ")
        for i, word in enumerate(spl):
            words += f" {word}"
            lwidth = get_width(font, words)
            if width + lwidth > 276:
                spl[i] = "\n" + spl[i]
                text = " ".join(spl)
                break

```

```

return text

def connected(client): # pylint: disable=unused-argument
    io.subscribe_to_time("iso")

def disconnected(client): # pylint: disable=unused-argument
    print("Disconnected from Adafruit IO!")

def update_text(hour_min):
    quote.text = (
        time_label.text
    ) = time_label_2.text = after_label.text = after_label_2.text = ""

    before, time_text, after = quotes[hour_min][0].split("^")
    text = adafruit_display_text.wrap_text_to_pixels(before, 276, font=arial)
    quote.text = "\n".join(text)

    for line in text:
        width = get_width(arial, line)

    time_text = smart_split(time_text, bold, width)

    split_time = time_text.split("\n")
    if time_text[0] != "\n":
        time_label.x = time_x = QUOTE_X + width
        time_label.y = time_y = QUOTE_Y + int((len(text) - 1) * HEIGHT *
LINE_SPACING)
        time_label.text = split_time[0]
    if "\n" in time_text:
        time_label_2.x = time_x = QUOTE_X
        time_label_2.y = time_y = QUOTE_Y + int(len(text) * HEIGHT * LINE_SPACING)
        wrapped = adafruit_display_text.wrap_text_to_pixels(
            split_time[1], 276, font=arial
        )
        time_label_2.text = "\n".join(wrapped)
    width = get_width(bold, split_time[-1]) + time_x - QUOTE_X

    if after:
        after = smart_split(after, arial, width)

        split_after = after.split("\n")
        if after[0] != "\n":
            after_label.x = QUOTE_X + width
            after_label.y = time_y
            after_label.text = split_after[0]
        if "\n" in after:
            after_label_2.x = QUOTE_X
            after_label_2.y = time_y + int(HEIGHT * LINE_SPACING)
            wrapped = adafruit_display_text.wrap_text_to_pixels(
                split_after[1], 276, font=arial
            )
            after_label_2.text = "\n".join(wrapped)

    author = f"{quotes[hour_min][2]} - {quotes[hour_min][1]}"
    author_label.text = adafruit_display_text.wrap_text_to_pixels(
        author, 276, font=arial
    )[0]
    time.sleep(display.time_to_refresh + 0.1)
    display.refresh()

LAST = None

def message(client, feed_id, payload): # pylint: disable=unused-argument
    global LAST # pylint: disable=global-statement

```

```

        timezone = adafruit_datetime.timezone.utc
        timezone._offset = adafruit_datetime.timedelta( # pylint: disable=protected-
access
            seconds=UTC_OFFSET * 3600
        )
        datetime = adafruit_datetime.datetime.fromisoformat(payload[:-1]).replace(
            tzinfo=timezone
        )
        local_datetime = datetime.tzinfo.fromutc(datetime)
        print(local_datetime)
        hour_min = f"{local_datetime.hour:02}:{local_datetime.minute:02}"
        if local_datetime.minute != LAST:
            if hour_min in quotes:
                update_text(hour_min)

        LAST = local_datetime.minute
        gc.collect()

# Create a socket pool
pool = socketpool.SocketPool(wifi.radio)

# Initialize a new MQTT Client object
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    port=1883,
    username=secrets["aio_username"],
    password=secrets["aio_key"],
    socket_pool=pool,
    ssl_context=ssl.create_default_context(),
)

# Initialize an Adafruit IO MQTT Client
io = IO_MQTT(mqtt_client)

# Connect the callback methods defined above to Adafruit IO
io.on_connect = connected
io.on_disconnect = disconnected
io.on_message = message

# Connect to Adafruit IO
print("Connecting to Adafruit IO...")
io.connect()

while True:
    try:
        io.loop()
    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        io.reconnect()
        continue
    time.sleep(1)

```

Code run-through

The code starts out by importing all the libraries it needs - quite a lot in this case.

```

import time

import ssl
import gc
import socketpool
import wifi
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT

```

```
import adafruit_datetime
import adafruit_display_text
from adafruit_display_text import label
import board
from adafruit_bitmap_font import bitmap_font
import displayio
from adafruit_display_shapes.rect import Rect
```

Then, it sets the UTC offset - you should modify this to your current local UTC offset (you can find that here: <https://www.timeanddate.com/time/zone/timezone/utc> (<https://adafru.it/18aE>)).

It then imports the quotes file and starts to set up the display and fonts.

```
UTC_OFFSET = -4

quotes = {}
with open("quotes.csv", "r", encoding="UTF-8") as F:
    for quote_line in F:
        split = quote_line.split("|")
        quotes[split[0]] = split[1:]

display = board.DISPLAY
splash = displayio.Group()
display.root_group = splash

arial = bitmap_font.load_font("fonts/Arial-12.pcf")
bold = bitmap_font.load_font("fonts/Arial-Bold-12.pcf")
LINE_SPACING = 0.8
HEIGHT = arial.get_bounding_box()[1]
QUOTE_X = 10
QUOTE_Y = 7
```

Now, the display background and text labels are set up and added to the display group.

```
rect = Rect(0, 0, 296, 128, fill=0xFFFFFF, outline=0xFFFFFF)
splash.append(rect)

quote = label.Label(
    font=arial,
    x=QUOTE_X,
    y=QUOTE_Y,
    color=0x000000,
    line_spacing=LINE_SPACING,
)

splash.append(quote)
time_label = label.Label(
    font=bold,
    color=0x000000,
    line_spacing=LINE_SPACING,
)
splash.append(time_label)

time_label_2 = label.Label(
    font=bold,
    color=0x000000,
    line_spacing=LINE_SPACING,
)
splash.append(time_label_2)
```

```

after_label = label.Label(
    font=arial,
    color=0x000000,
    line_spacing=LINE_SPACING,
)
splash.append(after_label)

after_label_2 = label.Label(
    font=arial,
    color=0x000000,
    line_spacing=LINE_SPACING,
)
splash.append(after_label_2)

author_label = label.Label(
    font=arial, x=QUOTE_X, y=115, color=0x000000, line_spacing=LINE_SPACING
)
splash.append(author_label)

```

After that, the MagTag attempts to connect to the internet.

```

try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

aio_username = secrets["aio_username"]
aio_key = secrets["aio_key"]

print(f"Connecting to {secrets['ssid']}")
wifi.radio.connect(secrets["ssid"], secrets["password"])
print(f"Connected to {secrets['ssid']}!")

```

At this point, we start defining a few helper functions.

The first one, `get_width`, is used to get the width of a string, in pixels, when passed the string and the font the string will be displayed in.

The next one, `smart_split`, is used to tell the code when to wrap a line when it's not the first label being used in a block of text. This is necessary since the code uses multiple fonts (bold and normal Arial 12pt.) in the same text block.

The last two are functions that are run when Adafruit IO is initially connected to - it subscribes the user to the ISO formatted time feed - and when it is disconnected from, respectively.

```

def get_width(font, text):
    return sum(font.get_glyph(ord(c)).shift_x for c in text)

def smart_split(text, font, width):
    words = ""
    spl = text.split(" ")
    for i, word in enumerate(spl):
        words += f" {word}"
        lwidth = get_width(font, words)
        if width + lwidth > 276:
            spl[i] = "\n" + spl[i]

```

```

        text = " ".join(spl)
        break
    return text

def connected(client): # pylint: disable=unused-argument
    io.subscribe_to_time("iso")

def disconnected(client): # pylint: disable=unused-argument
    print("Disconnected from Adafruit IO!")

```

This function is run whenever the quote to be displayed is updated. It's a bit complicated but a very important part of this project.

It starts by wiping all of the labels since we don't use every single label every time.

It then goes on to separate the different parts of the quote so it can set one part of that as bold and the rest as normal and sets the text of the part of the quote prior to the time.

Then the code for setting the location of the time text and the text after the time text is run, which account for the possibility that the first line of that may need to be wrapped over to the next line.

Finally, the display is refreshed with the new quote.

```

def update_text(hour_min):
    quote.text = (
        time_label.text
    ) = time_label_2.text = after_label.text = after_label_2.text = ""

    before, time_text, after = quotes[hour_min][0].split("^")
    text = adafruit_display_text.wrap_text_to_pixels(before, 276, font=arial)
    quote.text = "\n".join(text)

    for line in text:
        width = get_width(arial, line)

    time_text = smart_split(time_text, bold, width)

    split_time = time_text.split("\n")
    if time_text[0] != "\n":
        time_label.x = time_x = QUOTE_X + width
        time_label.y = time_y = QUOTE_Y + int((len(text) - 1) * HEIGHT *
LINE_SPACING)
        time_label.text = split_time[0]
    if "\n" in time_text:
        time_label_2.x = time_x = QUOTE_X
        time_label_2.y = time_y = QUOTE_Y + int(len(text) * HEIGHT * LINE_SPACING)
        wrapped = adafruit_display_text.wrap_text_to_pixels(
            split_time[1], 276, font=arial
        )
        time_label_2.text = "\n".join(wrapped)
    width = get_width(bold, split_time[-1]) + time_x - QUOTE_X

    if after:
        after = smart_split(after, arial, width)

        split_after = after.split("\n")

```



```

if after[0] != "\n":
    after_label.x = QUOTE_X + width
    after_label.y = time_y
    after_label.text = split_after[0]
if "\n" in after:
    after_label_2.x = QUOTE_X
    after_label_2.y = time_y + int(HEIGHT * LINE_SPACING)
    wrapped = adafruit_display_text.wrap_text_to_pixels(
        split_after[1], 276, font=arial
    )
    after_label_2.text = "\n".join(wrapped)

author = f"{quotes[hour_min][2]} - {quotes[hour_min][1]}"
author_label.text = adafruit_display_text.wrap_text_to_pixels(
    author, 276, font=arial
)[0]
time.sleep(display.time_to_refresh + 0.1)
display.refresh()

```

This function is run whenever the IO feed gets a new value, so roughly once a second. It starts by converting the received UTC time into the local time.

Then it checks to see if the time received is the same hour and minute as the last time received and if a quote entry exists for said time. If it isn't the same time and a quote does exist, the code then sends the time to the function above to update the quote.

```

LAST = None

def message(client, feed_id, payload): # pylint: disable=unused-argument
    global LAST # pylint: disable=global-statement
    timezone = adafruit_datetime.timezone.utc
    timezone._offset = adafruit_datetime.timedelta( # pylint: disable=protected-
access
        seconds=UTC_OFFSET * 3600
    )
    datetime = adafruit_datetime.datetime.fromisoformat(payload[:-1]).replace(
        tzinfo=timezone
    )
    local_datetime = datetime.tzinfo.fromutc(datetime)
    print(local_datetime)
    hour_min = f"{local_datetime.hour:02}:{local_datetime.minute:02}"
    if local_datetime.minute != LAST:
        if hour_min in quotes:
            update_text(hour_min)

    LAST = local_datetime.minute
    gc.collect()

```

However, before any of those functions can be used, the code needs to set up the Adafruit IO MQTT connection, which the following code does.

```

# Create a socket pool
pool = socketpool.SocketPool(wifi.radio)

# Initialize a new MQTT Client object
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    port=1883,
    username=secrets["aio_username"],
    password=secrets["aio_key"],

```

```

        socket_pool=pool,
        ssl_context=ssl.create_default_context(),
    )

# Initialize an Adafruit IO MQTT Client
io = IO_MQTT(mqtt_client)

# Connect the callback methods defined above to Adafruit IO
io.on_connect = connected
io.on_disconnect = disconnected
io.on_message = message

# Connect to Adafruit IO
print("Connecting to Adafruit IO...")
io.connect()

```

After it is connected the code runs through this loop to continually check for a new feed update from the Adafruit IO time feed.

```

while True:
    try:
        io.loop()
    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        io.reconnect()
        continue
    time.sleep(1)

```

Using The MagTag Quote Clock

If you've already loaded all the code onto the board, all that's left to do is update the UTC offset on line 20 of `code.py` to reflect your local time. UTC is 4 hours ahead of my local time, so I set my UTC offset to `-4`.

Find your UTC offset here

<https://adafru.it/18aE>

Time Zone in UTC, Time Zone (Coordinated Universal Time)

Time/General

Time Zone

DST Changes



20:01:00 UTC
Friday, November 4, 2022

Fullscreen

2022 Time Zones - UTC

Current: UTC

Current Offset: No UTC/GMT offset

Difference: 4 hours ahead of Baltimore



After that, plug your MagTag into a power source and you should be good to go!

