Adafruit QT Py Activity Timer and Hydration Reminder

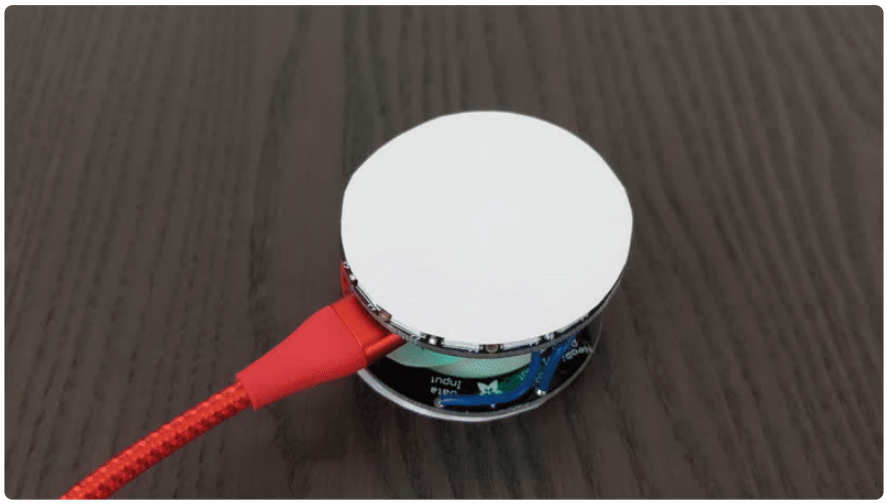
Created by Kattni Rembor

https://learn.adafruit.com/qt-py-activity-timer-and-hydration-reminder

Last updated on 2024-06-03 03:14:41 PM EDT

Table of Contents

Overview



It's easy to get focused on a task, and forget to take a break, or even eat lunch. Perhaps you find that you aren't drinking enough water throughout the day. Sometimes you simply need some assistance remembering things! This project is here to help.

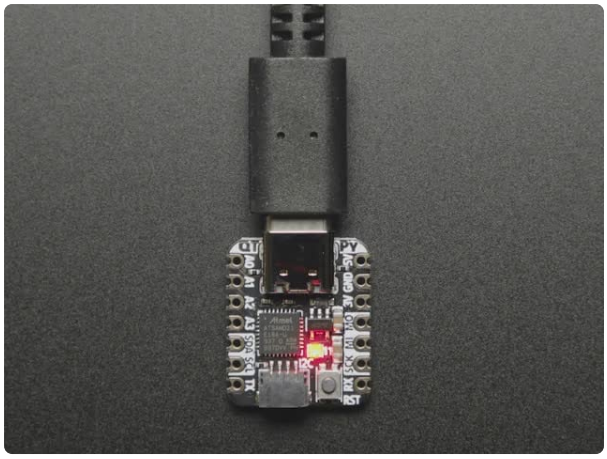
This guide will show you how to build a customisable LED timer using the Adafruit QT Py, the LIS3DH accelerometer, and two NeoPixel LED rings. It's meant to be compact and sit on your desk. The rings light up to let you know the timer is running. Set it on its side to turn off the timers and the LEDs.

For the activity timer, you'll set a duration for two different activities, such as 120 minutes of work and a 15 minute break. Plug it in, start the timer and it will gently fade between two colors to let you know time is passing. When it reaches the end it stays at that color for a bit before blinking at you to remind you that you haven't moved to the next task. Flip it over to begin the second timer, and the process repeats.

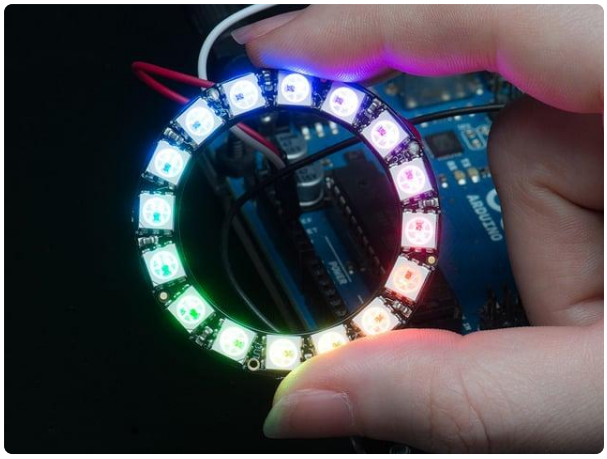
For the hydration reminder, you set the interval you'd like to be reminded to drink water, such as once every 60 minutes. Plug it in, start the timer, and the process is the same as the activity timer. In this case, when you flip it over, it counts down the same amount of time.

Parts

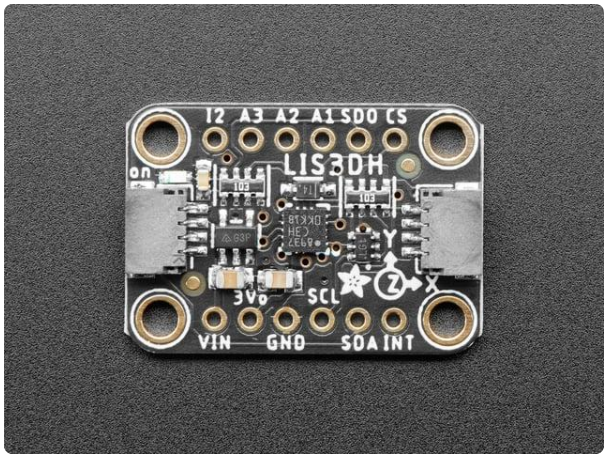
This build uses the 50mm long STEMMA QT cable listed below. The 100mm long STEMMA QT cable (http://adafru.it/4210)would also work, but would require taming the extra cable length.



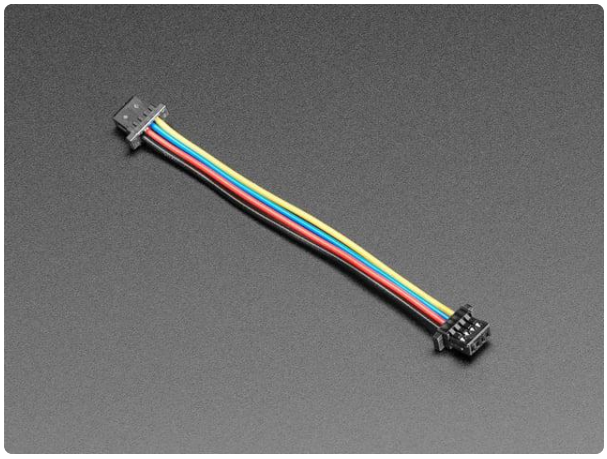
[Adafruit QT Py - SAMD21 Dev Board with STEMMA QT]
What a cutie pie! Or is it... a QT Py? This diminutive dev board comes with our favorite lil chip, the SAMD21 (as made famous in our GEMMA M0 and Trinket M0 boards).This time it...
https://www.adafruit.com/product/4600



[NeoPixel Ring - 16 x 5050 RGB LED with Integrated Drivers]
Round and round and round they go! 16 ultra bright smart LED NeoPixels are arranged in a circle with 1.75" (44.5mm) outer diameter. The rings are 'chainable' - connect the...
https://www.adafruit.com/product/1463



[Adafruit LIS3DH Triple-Axis Accelerometer (+-2g/4g/8g/16g)]
The LIS3DH is a very popular low power triple-axis accelerometer. It's low-cost, but has just about every 'extra' you'd want in...
https://www.adafruit.com/product/2809

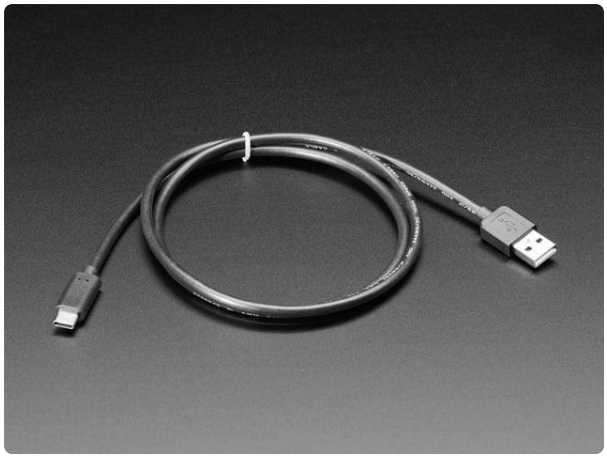


[STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long]
This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...
https://www.adafruit.com/product/4399

Hook-up Wire Spool Set - 22AWG Solid Core - 6 x 25 ft

Perfect for bread-boarding, free wiring, etc. This box contains 6 spools of solid-core wire. The wire is easy to solder to and when bent it keeps its shape pretty well. We like to have...

https://www.adafruit.com/product/1311

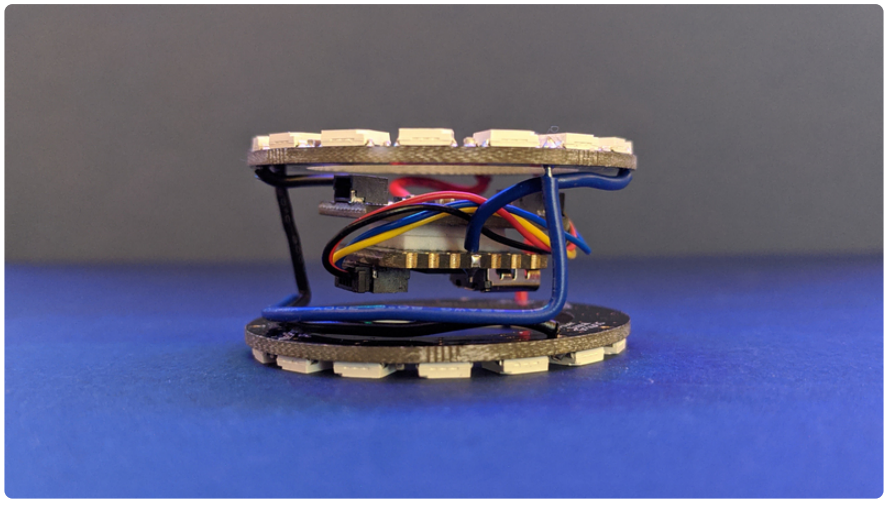


USB Type A to Type C Cable - approx 1 meter / 3 ft long

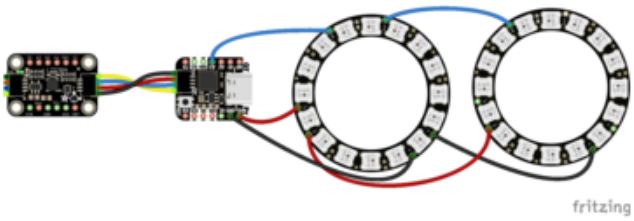
As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

https://www.adafruit.com/product/4474

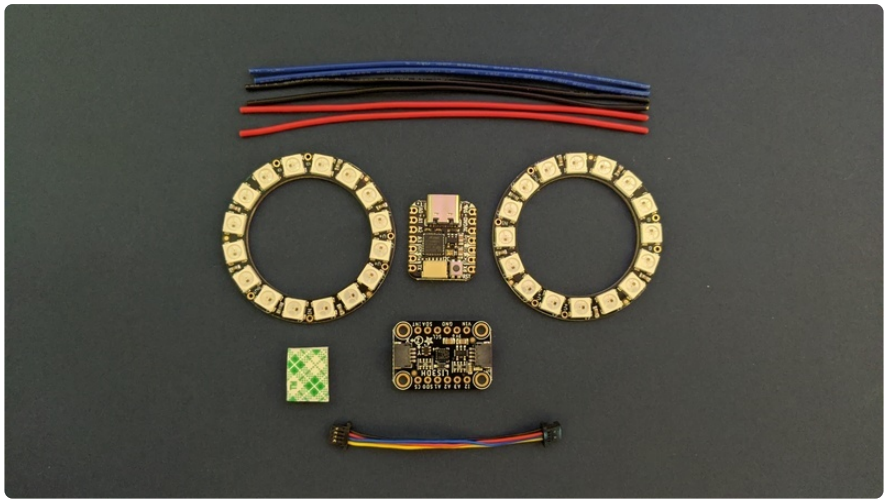
Assembly



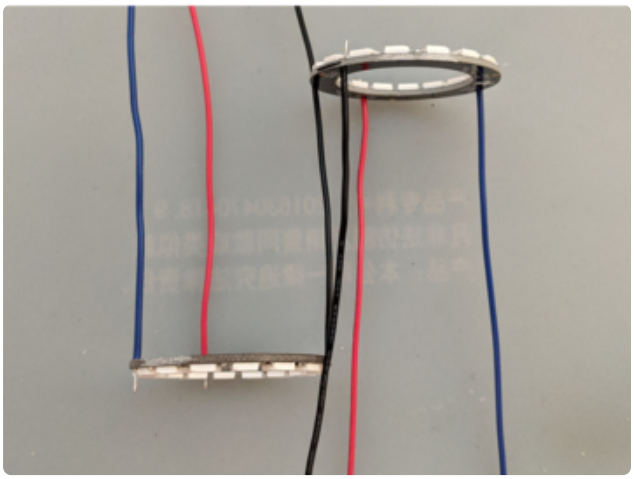
The following shows how the parts are wired together. Assembly of the timer involves some creative use of wire length to suspend the QT Py and LIS3DH between the two rings.

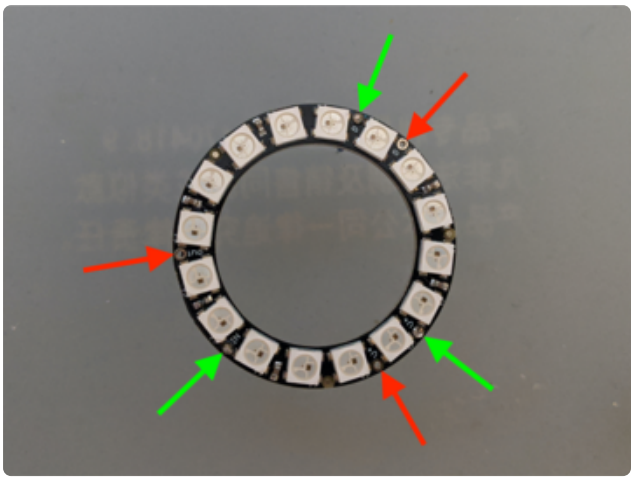


Data In on ring one to **A3 on QT Py**
V+ on ring one to **5V on QT Py**
Gnd on ring one to **GND on QT Py**
Gnd on ring two to **GND on ring one**
V+ on ring two to **V+ on ring one**
Data In on ring one to **Data Out on ring two**
LIS3DH to **QT Py** using **STEMMA QT cable**



Assembly



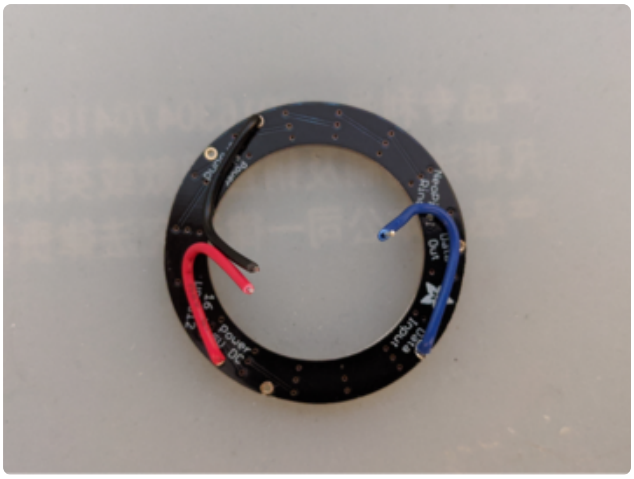


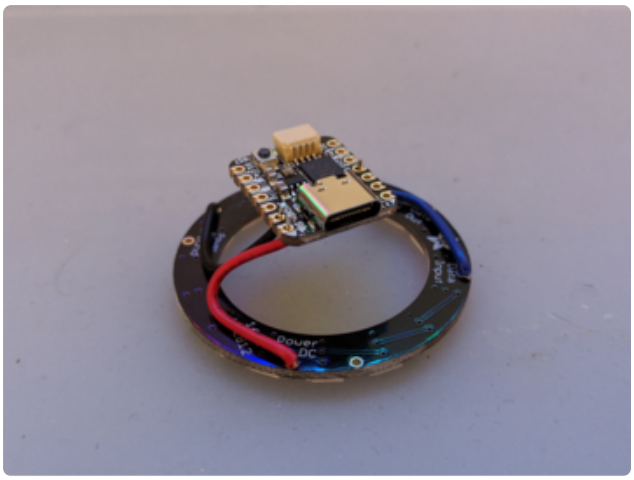
Begin by soldering the three wires to each ring, using piece of wire 5-6 inches in length.

Solder the **blue wire** to **In (data input)**, the **black wire** to **G (power signal ground)**, and the **red wire** to **V+ (power 5V DC)** on both rings. Solder the data input wire first, and follow the pattern indicated in the second image where green indicates the through-hole to use to solder the wire, and red indicates a through-hole to skip.

Trim any excess wire sticking up above the LEDs.

Choose a ring to be ring one.



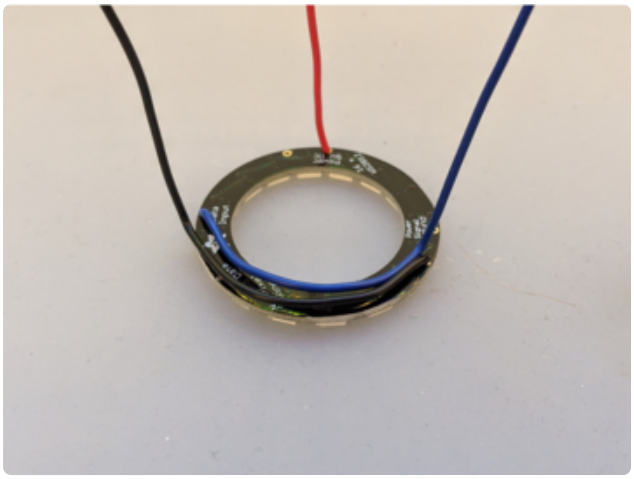


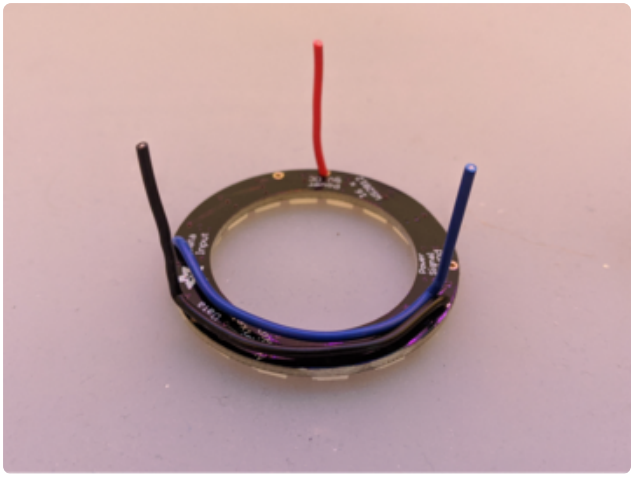
Bend the wires on ring one to match the locations of the pins on the QT Py, with enough wire to suspend the QT Py above the ring.

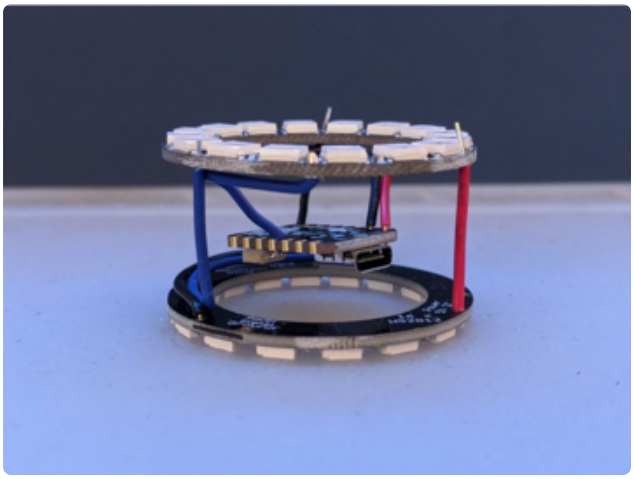
The **blue wire** should match **A3**, the **red wire** should match **5V** and the **black wire** should match **GND**.

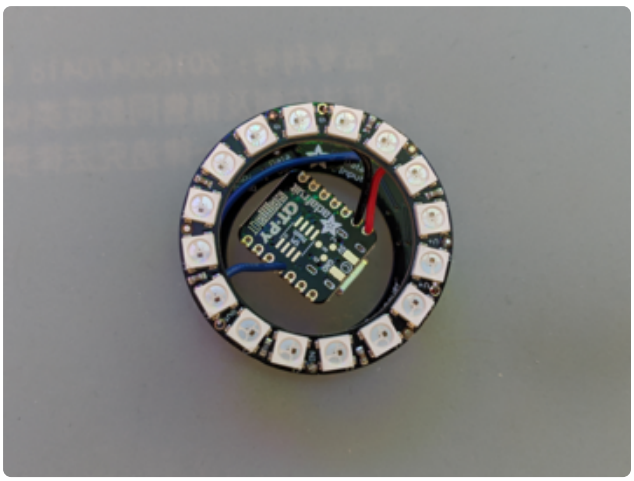
Make sure the QT Py is parallel to the ring - if you lay the ring flat on your desk, the QT Py should also be horizontal.

Solder the wires to the QT Py.









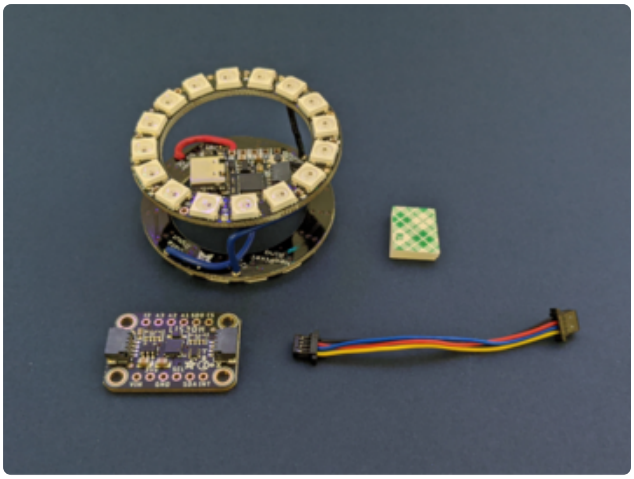
On ring two, bend the signal (blue) and ground (black) wires around the ring so they match the Data Out and GND pins on ring one. You can hold the rings together to line up the wires and bend them until they are correct.

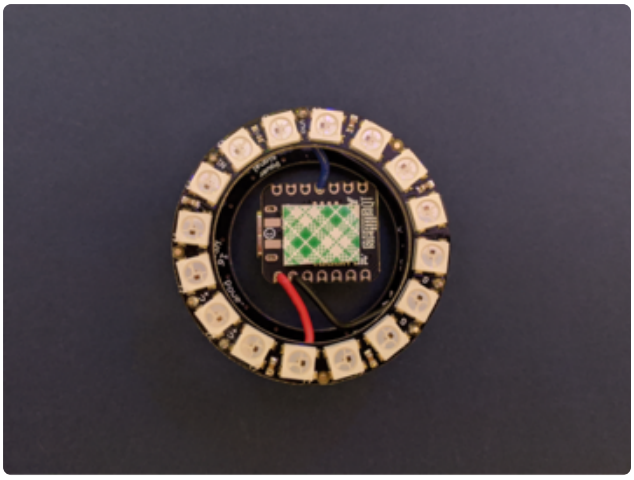
The red wire does not need to be bent and can connect directly from ring two to ring one.

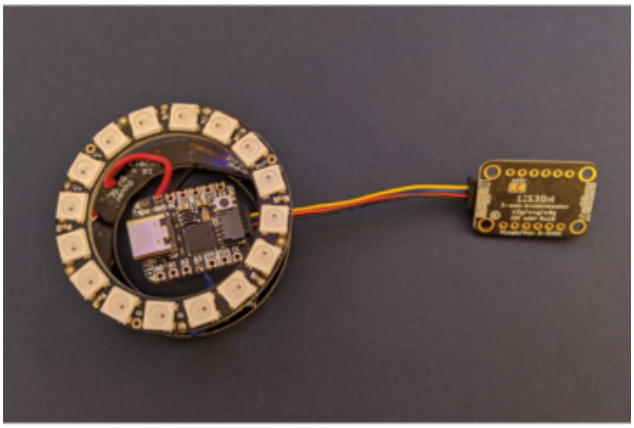
Trim the wires so less than an inch is sticking up from the ring. The lengths should all be equal.

Solder ring two to ring one with the **blue wire** to **Out (data output)**, the **black wire** to **G (power signal ground)**, and the **red wire** to **V+ (power 5V DC)**.

Trim any excess wire sticking up above the LEDs.







Gather the soldered QT Py and ring assembly, the double-sided foam tape, the STEMMA cable and the LIS3DH.

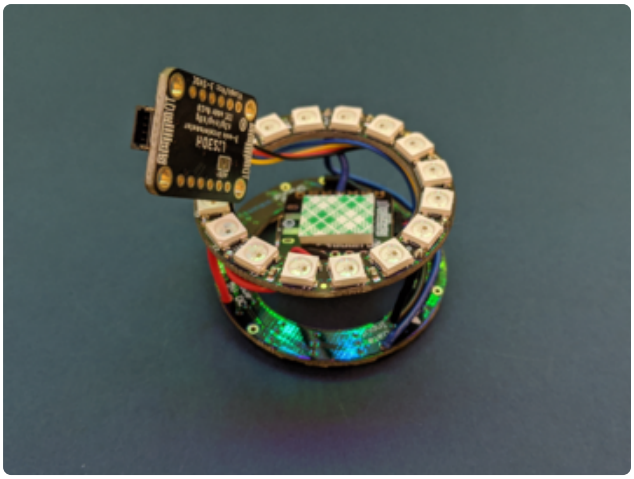
Attach the double-sided foam tape to the QT Py. This version required layering two pieces of foam tape together.

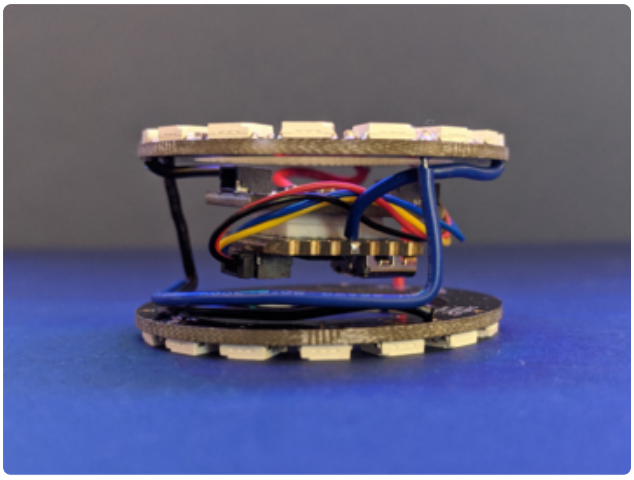
Plug the LIS3DH into the QT Py using the STEMMA QT connectors and cable.

Wind the cable and the LIS3DH through the rings so the cable is to the side of the LIS3DH and QT Py.

Use double-sided foam tape to attach the LIS3DH to the back of the QT Py. The LIS3DH should also be parallel to the rings - when you set the timer on the front of either of the rings, the LIS3DH and QT Py should both be horizontal.

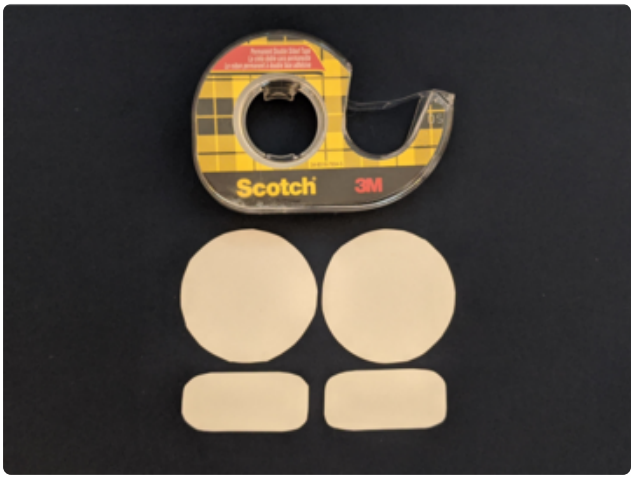
You're ready to start timing!





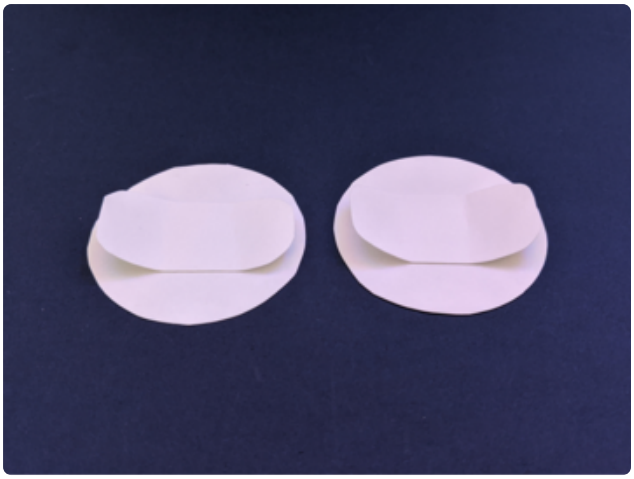
Optional Diffusers

NeoPixel LEDs are quite bright. It's possible to control the brightness using CircuitPython, but you may also want to add diffusers to the rings. Here is a simple way to add some physical diffusion to the LEDs on the timer.



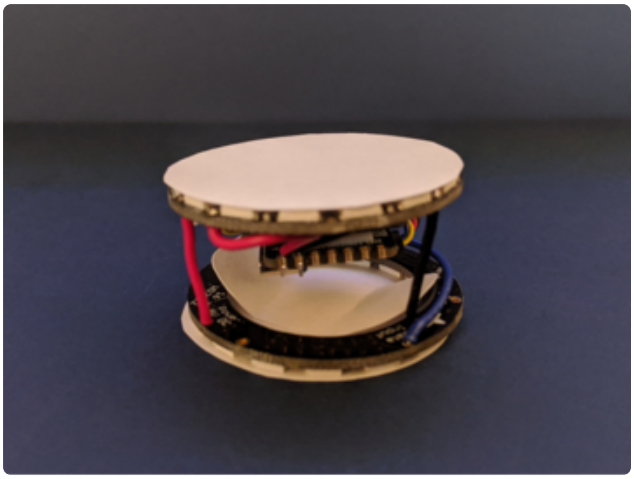
Get some double-sided tape and a piece of white or similarly light colored card stock, or other thicker paper product.

Use a ring as a guide to cut two circles out of card stock. Cut two strips the same length as the diameter of the circles, and round the corners of the strips.



Use a piece double-sided tape in the center of each strip to attach the strips to the circles.

Gently bend the strips away from the circles on each end to make two flaps on each diffuser.



Slip the flaps inside the ring to attach the diffuser to the ring. Orient them so the flaps are not blocking the USB port on the QT Py.

That's it!

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.

Set up CircuitPython Quick Start!

Follow this quick step-by-step for super-fast Python power :)

If you want to get started with your QT Py, and you have NOT soldered a chip to the back of it, download CircuitPython from the following link:

Download the latest version of CircuitPython for your QT Py from CircuitPython.org

https://adafru.it/NCB

If you have soldered a GD25Q16 SPI flash chip to the bottom of your board, you must use the Haxpress version of CircuitPython for the Adafruit QT Py for the flash to work! If you have NOT soldered a SPI flash chip to your QT Py, do NOT use this download! It will not give you 2MB of flash space without a chip!

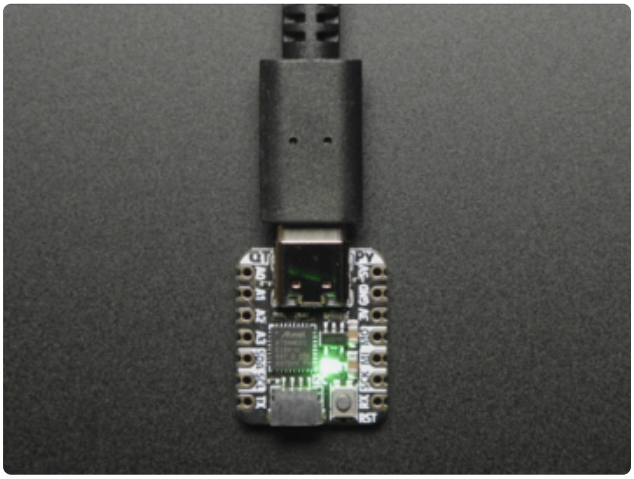
Download the latest version of CircuitPython for your QT Py Haxpress from CircuitPython.org

https://adafru.it/NCC

Click the link above and download the latest UF2 file.

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

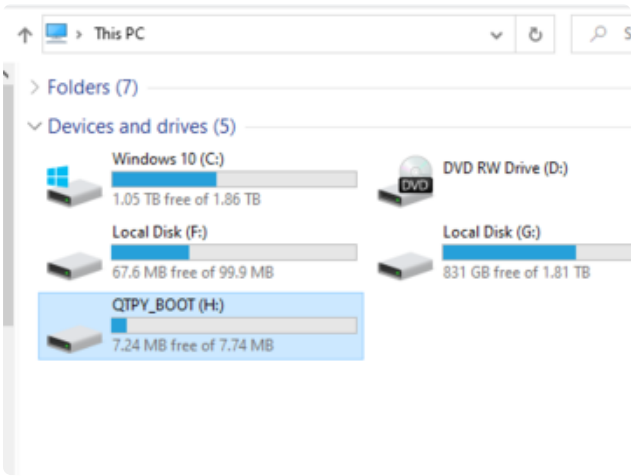
Plug your QT Py into your computer using a known-good USB cable.



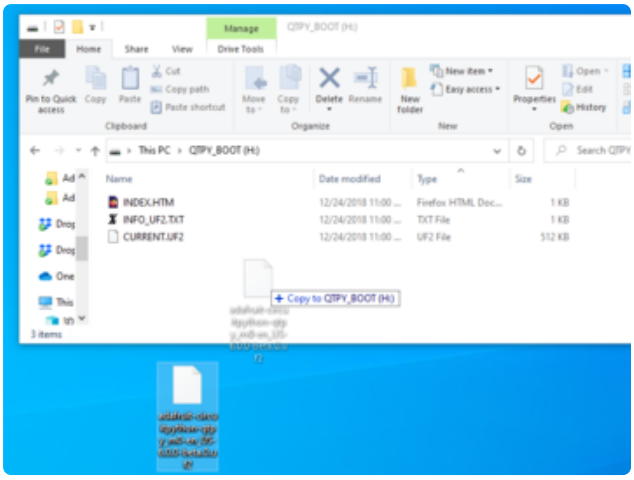
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

Double-click the small **RST (reset)** button, and you will see the NeoPixel RGB LED turn green. If it turns red, check the USB cable, try another USB port, etc.

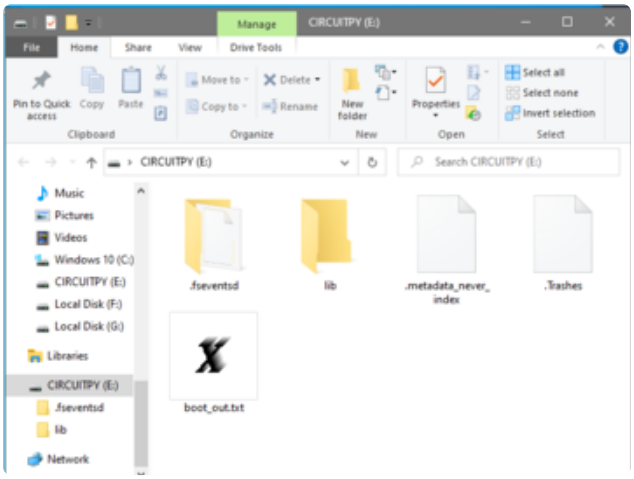
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



You will see a new disk drive appear called **QTPY_BOOT**.



Drag the **adafruit_circuitpython_etc.uf2** file to **QTPY_BOOT**



The red LED will flash. Then, the **QTPY_BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

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

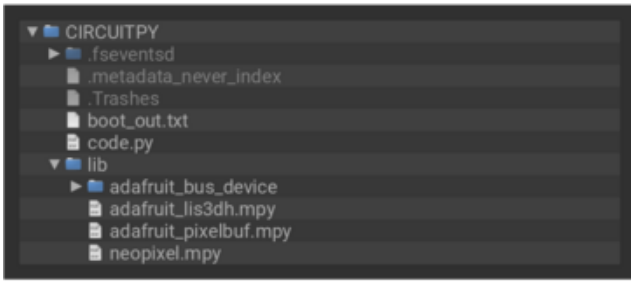
CircuitPython Libraries

The QT Py Activity Timer and Hydration Reminder requires some CircuitPython libraries to work. Complete the following steps to get your board ready.

CircuitPython Library Bundle

You'll need to install the Adafruit CircuitPython NeoPixel (https://adafru.it/yew) and Adafruit CircuitPython LIS3DH (https://adafru.it/uBs) libraries and their dependencies on your QT Py.

Carefully follow the steps to find and install these libraries from Adafruit's CircuitPython Library Bundle (https://adafru.it/ENC). Our CircuitPython starter guide has a great page on how to install libraries from the bundle (https://adafru.it/ABU).



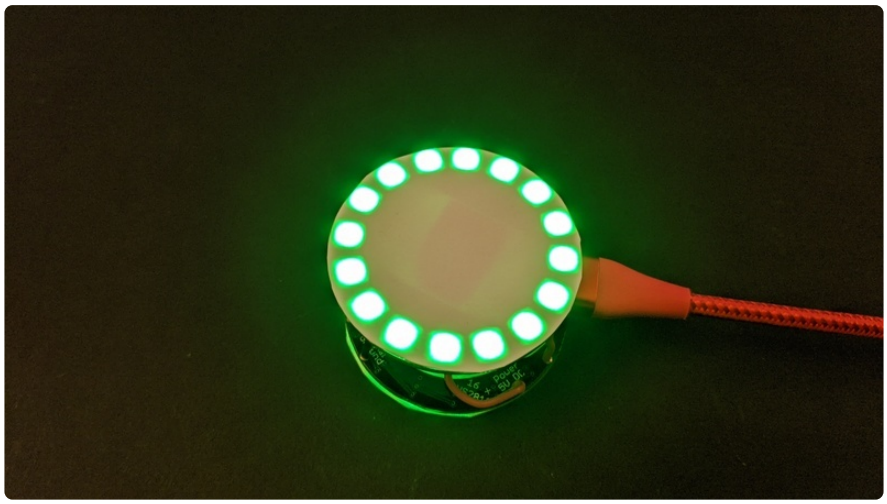
You'll want to manually install the following libraries by copying the files to the **lib** folder on your **CIRCUITPY** drive:

adafruit_bus_device
adafruit_lis3dh.mpy
adafruit_pypixelbuf.mpy
neopixel.mpy
Before continuing make sure your board's **lib** folder or root filesystem has the **adafruit_bus_device**, **adafruit_lis3dh.mpy**, **neopixel.mpy**,
and **adafruit_pypixelbuf.mpy** files copied over.

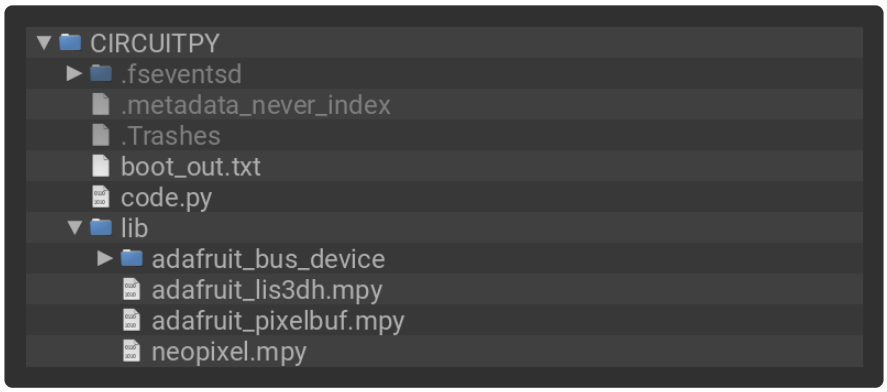
Library Install Troubleshooting

If you get an error indicating you have run out of space while copying files over, please refer to the Prevent & Remove MacOS Hidden Files section of the Troubleshooting page (https://adafru.it/Den) for information on resolving the issue.

Activity Timer



The QT Py Activity Timer is meant to time two separate activities, such as working and taking a break. Flip it to begin timing the first activity, such as working for 120 minutes. The timer will count down by fading the LEDs from one color to another, at which point it gives you a moment to flip it before blinking at you to let you know you haven't moved onto the next activity. Flip it to begin timing the next activity, such as a 15 minute break. The process repeats. You can customise the colors and timing to fit your aesthetics and needs. Flip it on its side to stop both timers and turn off the LEDs. Let's get timing!



Code

Save the following to your **CIRCUITPY** drive as **code.py**:

```
# SPDX-FileCopyrightText: 2020 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import adafruit_lis3dh
import neopixel
```

```
# Length of each activity, in minutes.
activity_one = 120
activity_two = 15

# Length of time, in minutes, before timer begins Red Alert blinking.
red_alert_delay = 2

# The color will fade from color_begin to color_complete. This is a gentle
indicator of how much
# time has passed. Set to any two colors.
color_begin = (0, 255, 0)  # Green. Set to any color to begin your countdown.
color_complete = (255, 0, 0)  # Red. Set to any color to end your countdown.
led_brightness = 0.2  # Set to a number between 0.0 and 1.0 to set LED brightness.

# Increase or decrease this to change the speed of the Red Alert blinking in
seconds.
red_alert_blink_speed = 0.5

# These are the thresholds the z-axis values must exceed for the orientation to be
considered
# "up" or "down". These values are valid if the LIS3DH breakout is mounted flat
inside the timer
# assembly. If you want the option to have the timer on an angle while timing, you
can calibrate
# these values by uncommenting the print(z) in orientation() to see the z-axis
values and update
# the thresholds to match the desired range.
down_threshold = -8
up_threshold = 8

# Number of LEDs. Default is 32 (2 x rings of 16 each). If you use a different form
of NeoPixels,
# update this to match the total number of pixels.
number_of_pixels = 32

# Set up accelerometer and LEDs.
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c)
pixels = neopixel.NeoPixel(board.A3, number_of_pixels, brightness=led_brightness)
pixels.fill(0)  # Turn off the LEDs if they're on.

STATE_IDLE = "Idle"
STATE_TIMING = "Timing"
STATE_TIMING_COMPLETE = "Timing complete"
RED_ALERT = "Red Alert"


def gradient(color_one, color_two, blend_weight):
    """
    Blend between two colors with a given ratio.

    :param color_one:  first color, as an (r,g,b) tuple
    :param color_two:  second color, as an (r,g,b) tuple
    :param blend_weight: Blend weight (ratio) of second color, 0.0 to 1.0
    """
    if blend_weight < 0.0:
        blend_weight = 0.0
    elif blend_weight > 1.0:
        blend_weight = 1.0
    initial_weight = 1.0 - blend_weight
    return (int(color_one[0] * initial_weight + color_two[0] * blend_weight),
            int(color_one[1] * initial_weight + color_two[1] * blend_weight),
            int(color_one[2] * initial_weight + color_two[2] * blend_weight))


# pylint: disable=global-statement
```

```
def set_state(state):
    global current_state, state_changed
    current_state = state
    state_changed = time.monotonic()
    print("Current state:", current_state)  # Print the current state.


def orientation():
    """Determines orientation based on z-axis values. Thresholds set above."""
    _, _, z = lis3dh.acceleration
    # print(z)  # Uncomment to print the z-axis value. Use to calibrate thresholds
if desired.
    if z < down_threshold:
        return 'down'
    if z > up_threshold:
        return 'up'
    return 'side'


def orientation_debounced():
    """
    Debounces the orientation changes. For a new orientation to be registered, the
timer must
    be in that orientation for the duration of the delay.
    """
    delay = 0.2
    initial_time = time.monotonic()
    initial_orientation = orientation()
    while True:
        new_orientation = orientation()
        if new_orientation != initial_orientation:
            initial_time = time.monotonic()
            initial_orientation = new_orientation
            continue
        if time.monotonic() - initial_time > delay:
            return new_orientation


def state_from_orientation():
    """Determines the state based on orientation."""
    global current_orientation
    new_orientation = orientation_debounced()
    if new_orientation != current_orientation:
        if new_orientation == 'side':
            set_state(STATE_IDLE)
            current_orientation = orientation_debounced()
            return
        set_state(STATE_TIMING)
        current_orientation = orientation_debounced()


def idle():
    """The idle state."""
    pixels.fill(0)
    state_from_orientation()


def timing():
    """The timing active state."""
    state_from_orientation()

    if current_orientation == 'up':
        activity_duration = activity_one * 60  # Converts seconds to minutes.
    elif current_orientation == 'down':
        activity_duration = activity_two * 60  # Converts seconds to minutes.
    else:
        return

    elapsed = time.monotonic() - state_changed
```

```
    if elapsed >= activity_duration:
        set_state(STATE_TIMING_COMPLETE)
        return

    blend = (elapsed / activity_duration)
    pixels.fill(gradient(color_begin, color_complete, blend))


def timing_complete():
    """The timing complete state."""
    pixels.fill(color_complete)

    state_from_orientation()

    elapsed = time.monotonic() - state_changed

    if elapsed >= (red_alert_delay * 60):  # Converts seconds to minutes.
        set_state(RED_ALERT)
        return


blink_is_on = False


def red_alert():
    """The Red Alert state."""
    global blink_is_on
    elapsed = time.monotonic() - state_changed
    if elapsed >= red_alert_blink_speed:
        set_state(RED_ALERT)
        blink_is_on = not blink_is_on
    if blink_is_on:
        pixels.fill(color_complete)
    else:
        pixels.fill(0)

    state_from_orientation()


current_orientation = orientation_debounced()  # Get the initial orientation.
state_changed = time.monotonic()  # Set an initial timestamp.
current_state = STATE_IDLE  # Set initial state to idle.

print("Start state:", current_state)  # Print the starting state.

while True:
    if current_state == STATE_IDLE:
        idle()
    if current_state == STATE_TIMING:
        timing()
    if current_state == STATE_TIMING_COMPLETE:
        timing_complete()
    if current_state == RED_ALERT:
        red_alert()
```

Settings and Customizations

The timer defaults to a 120 minute timer when facing up (LIS3DH on top), and 15 minutes when facing down (QT Py on top). Both timers begin green and fade to red, before entering Red Alert and beginning to blink.

If you're happy with the project as-is, you don't need to make any changes. If you'd like to customize it, you have the following options.

- `activity_one` - The length of time in minutes for the first activity. Defaults to 120 minutes (2 hours).
- `activity_two` - The length of time in minutes for the second activity. Defaults to 15 minutes.
- `red_alert_delay` - The length of time in minutes the timer shows the final color solid before blinking to remind you to move to the next activity. Defaults to 2 minutes. If you flip the the timer within this time, the blinking will not start.
- `color_begin` - The color the LEDs begin when the timer begins. Timer fades from `color_begin` to `color_complete` to indicate time passing.
- `color_complete` - The color the LEDs end when the timer ends. This is also the color that blinks during Red Alert. Timer fades from `color_begin` to `color_complete` to indicate time passing.
- `led_brightness` - The brightness of the LEDs. If you add a diffuser to the project, you may want to make them brighter. Must be a number between 0.0 and 1.0 where 0.0 is 0% (off), and 1.0 is 100% brightness, e.g. 0.3 would be 30% brightness.
- `red_alert_blink_speed` - This is the speed in seconds of the blinking during Red Alert. Defaults to blinking on and off every 0.5 seconds. If you'd like it to be faster, decrease this number. To slow it down, increase the number.

The z-axis values are used to determine the orientation of the timer. The thresholds used in the code assume the LIS3DH is mounted horizontally within the timer, and that the timer is sitting flat. As it doesn't weigh much, the USB cable can sometimes pull it on an angle. If you find the timer isn't running when you want it to because you can't get the timer oriented properly, you can alter these values based on the z-axis values you're getting.

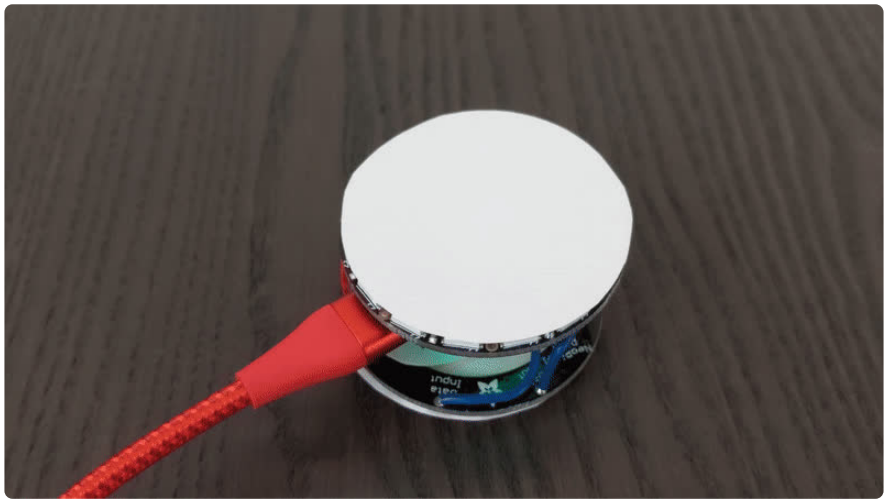
- `down_threshold` - The z-axis value must be less than this for the timer to start when in the down orientation. Defaults to -8.
- `up_threshold` - The z-axis value must be greater than this for the timer to start when in the up orientation. Defaults to 8.

If you used the parts suggested in the guide, you do not need to change the following. But, in the event you're using a different form of NeoPixels, you can update the following:

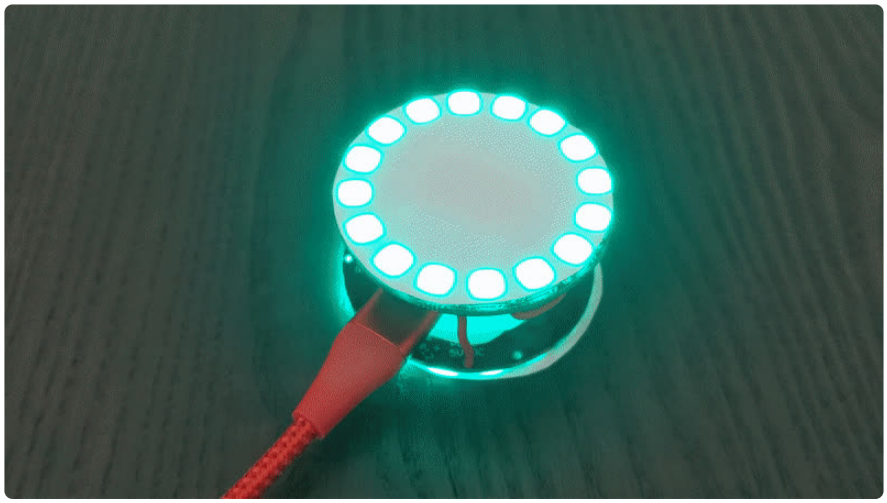
- `number_of_pixels` - The total number of LEDs connected to your QT Py.

Using the Activity Timer

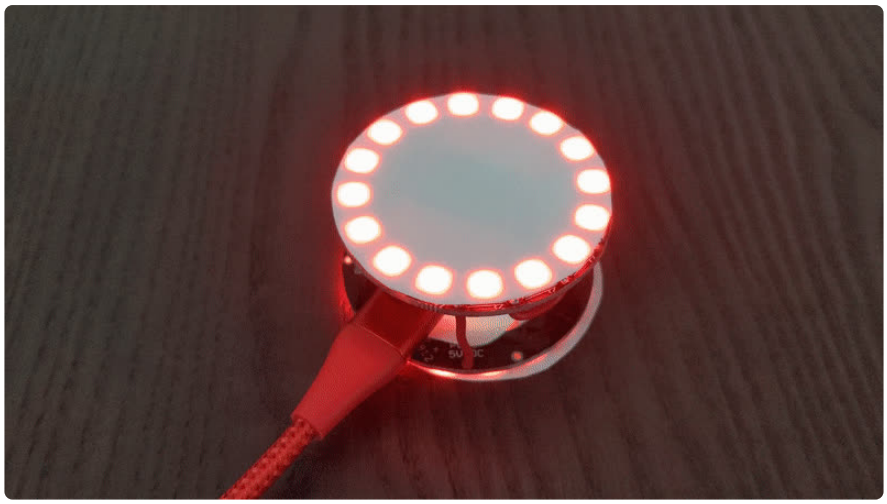
Plug it in to begin. It starts in idle mode. Flip it to the side that matches the activity you'd like to begin timing.



The timer will begin. By default it fades from green to red.

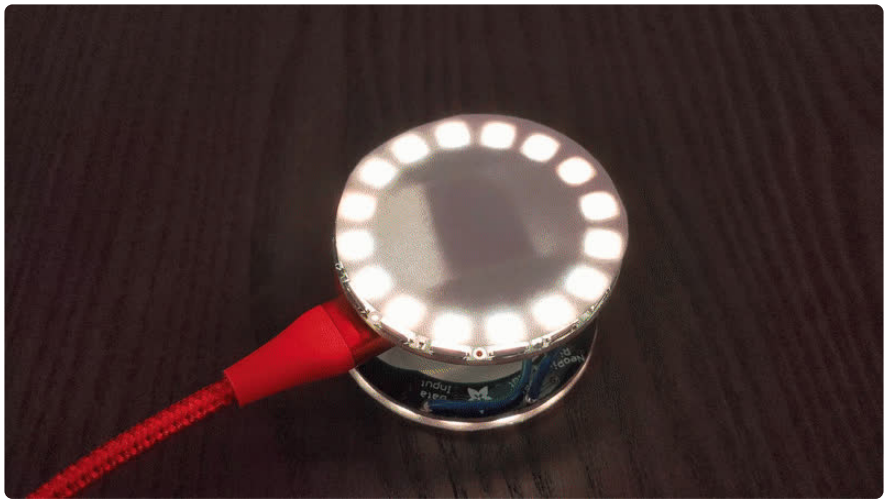


If you don't do anything when the timer is up, it will enter Red Alert mode and begin blinking.

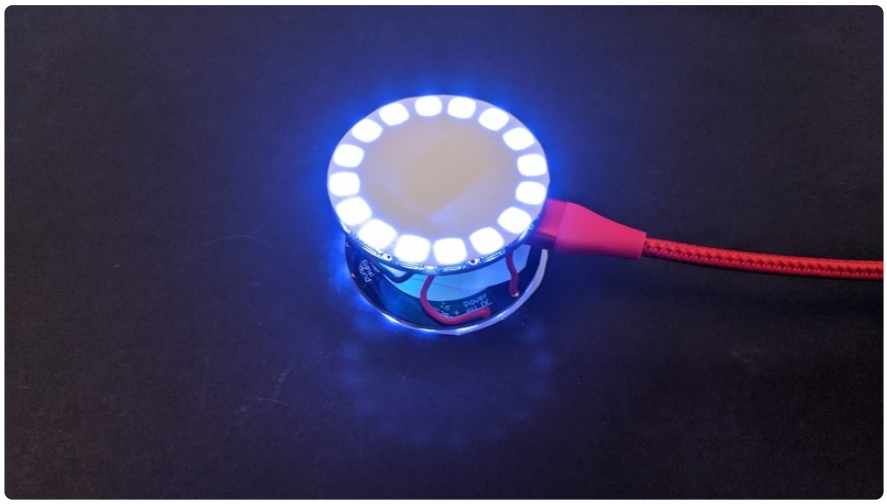


Flip it over at any time to start the timer for the next activity.

Flip it on its side to stop both of the timers and turn off the LEDs.



Hydration Reminder



The QT Py Hydration Reminder is designed to track a time interval to remind you to stay hydrated. Flip it to begin. The timer will count down by fading the LEDs from one color to another, at which point it gives you a moment to flip it before blinking at you to let you know you haven't taken a drink. Flip it to begin the next interval. The process repeats. You can customise the colors and timing to fit your aesthetics and needs. Flip it on its side to stop the timer and turn off the LEDs. Let's get hydrated!

Code

Save the following to your **CIRCUITPY** drive as **code.py**:

```
# SPDX-FileCopyrightText: 2020 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import adafruit_lis3dh
import neopixel

# The interval you would like to be reminded to drink water, in minutes.
hydration_reminder = 60
```

```
# Length of time, in minutes, before timer begins Red Alert blinking.
red_alert_delay = 2

# The color will fade from color_begin to color_complete. This is a gentle
indicator of how much
# time has passed. Set to any two colors.
color_begin = (0, 0, 255)  # Blue. Set to any color to begin your countdown.
color_complete = (255, 255, 255)  # White. Set to any color to end your countdown.
led_brightness = 0.2  # Set to a number between 0.0 and 1.0 to set LED brightness.

# Increase or decrease this to change the speed of the Red Alert blinking in
seconds.
red_alert_blink_speed = 0.5

# These are the thresholds the z-axis values must exceed for the orientation to be
considered
# "up" or "down". These values are valid if the LIS3DH breakout is mounted
horizontally inside the
# timer assembly. If you want the option to have the timer on an angle while
timing, you can
# calibrate these values by uncommenting the print(z) in orientation() to see the z-
axis values and
# update the thresholds to match the desired range.
down_threshold = -8
up_threshold = 8

# Number of LEDs. Default is 32 (2 x rings of 16 each). If you use a different form
of NeoPixels,
# update this to match the total number of pixels.
number_of_pixels = 32

# Set up accelerometer and LEDs.
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c)
pixels = neopixel.NeoPixel(board.A3, number_of_pixels, brightness=led_brightness)
pixels.fill(0)  # Turn off the LEDs if they're on.

STATE_IDLE = "Idle"
STATE_TIMING = "Timing"
STATE_TIMING_COMPLETE = "Timing complete"
RED_ALERT = "Red Alert"


def gradient(color_one, color_two, blend_weight):
    """
    Blend between two colors with a given ratio.

    :param color_one:  first color, as an (r,g,b) tuple
    :param color_two:  second color, as an (r,g,b) tuple
    :param blend_weight: Blend weight (ratio) of second color, 0.0 to 1.0
    """
    if blend_weight < 0.0:
        blend_weight = 0.0
    elif blend_weight > 1.0:
        blend_weight = 1.0
    initial_weight = 1.0 - blend_weight
    return (int(color_one[0] * initial_weight + color_two[0] * blend_weight),
            int(color_one[1] * initial_weight + color_two[1] * blend_weight),
            int(color_one[2] * initial_weight + color_two[2] * blend_weight))


# pylint: disable=global-statement
def set_state(state):
    global current_state, state_changed
    current_state = state
    state_changed = time.monotonic()
```

```
    print("Current state:", current_state)  # Print the current state.


def orientation():
    """Determines orientation based on z-axis values. Thresholds set above."""
    _, _, z = lis3dh.acceleration
    # print(z)  # Uncomment to print the z-axis value. Use to calibrate thresholds
if desired.
    if z < down_threshold:
        return 'down'
    if z > up_threshold:
        return 'up'
    return 'side'


def orientation_debounced():
    """
    Debounces the orientation changes. For a new orientation to be registered, the
timer must
    be in that orientation for the duration of the delay.
    """
    delay = 0.2
    initial_time = time.monotonic()
    initial_orientation = orientation()
    while True:
        new_orientation = orientation()
        if new_orientation != initial_orientation:
            initial_time = time.monotonic()
            initial_orientation = new_orientation
            continue
        if time.monotonic() - initial_time > delay:
            return new_orientation


def state_from_orientation():
    """Determines the state based on orientation."""
    global current_orientation
    new_orientation = orientation_debounced()
    if new_orientation != current_orientation:
        if new_orientation == 'side':
            set_state(STATE_IDLE)
            current_orientation = orientation_debounced()
            return
        set_state(STATE_TIMING)
        current_orientation = orientation_debounced()


def idle():
    """The idle state."""
    pixels.fill(0)
    state_from_orientation()


def timing():
    """The timing active state."""
    state_from_orientation()

    activity_duration = hydration_reminder * 60  # Converts minutes to seconds.

    elapsed = time.monotonic() - state_changed

    if elapsed >= activity_duration:
        set_state(STATE_TIMING_COMPLETE)
        return

    blend = (elapsed / activity_duration)
    pixels.fill(gradient(color_begin, color_complete, blend))
```

```
def timing_complete():
    """The timing complete state."""
    pixels.fill(color_complete)

    state_from_orientation()

    elapsed = time.monotonic() - state_changed

    if elapsed >= (red_alert_delay * 60):  # Converts minutes to seconds.
        set_state(RED_ALERT)
        return


blink_is_on = False


def red_alert():
    """The Red Alert state."""
    global blink_is_on
    elapsed = time.monotonic() - state_changed
    if elapsed >= red_alert_blink_speed:
        set_state(RED_ALERT)
        blink_is_on = not blink_is_on
    if blink_is_on:
        pixels.fill(color_complete)
    else:
        pixels.fill(0)

    state_from_orientation()


current_orientation = orientation_debounced()  # Get the initial orientation.
state_changed = time.monotonic()  # Set an initial timestamp.
current_state = STATE_IDLE  # Set initial state to idle.

print("Start state:", current_state)  # Print the starting state.

while True:
    if current_state == STATE_IDLE:
        idle()
    if current_state == STATE_TIMING:
        timing()
    if current_state == STATE_TIMING_COMPLETE:
        timing_complete()
    if current_state == RED_ALERT:
        red_alert()
```

Settings and Customizations

The timer defaults to a 120 minute timer when facing up (LIS3DH on top), and 15 minutes when facing down (QT Py on top). Both timers begin green and fade to red, before entering Red Alert and beginning to blink.

If you're happy with the project as-is, you don't need to make any changes. If you'd like to customize it, you have the following options.

- `hydration_reminder` - The length of the reminder interval in minutes. Defaults to 60 minutes (one hour).

- `red_alert_delay` - The length of time in minutes the timer shows the final color solid before blinking to remind you to take a drink. Defaults to 2 minutes. If you flip the the timer within this time, the blinking will not start.
- `color_begin` - The color the LEDs begin when the timer begins. Timer fades from `color_begin` to `color_complete` to indicate time passing.
- `color_complete` - The color the LEDs end when the timer ends. This is also the color that blinks during Red Alert. Timer fades from `color_begin` to `color_complete` to indicate time passing.
- `led_brightness` - The brightness of the LEDs. If you add a diffuser to the project, you may want to make them brighter. Must be a number between 0.0 and 1.0 where 0.0 is 0% (off), and 1.0 is 100% brightness, e.g. 0.3 would be 30% brightness.
- `red_alert_blink_speed` - This is the speed in seconds of the blinking during Red Alert. Defaults to blinking on and off every 0.5 seconds. If you'd like it to be faster, decrease this number. To slow it down, increase the number.

The z-axis values are used to determine the orientation of the timer. The thresholds used in the code assume the LIS3DH is mounted horizontally within the timer, and that the timer is sitting flat. As it doesn't weigh much, the USB cable can sometimes pull it on an angle. If you find the timer isn't running when you want it to because you can't get the timer oriented properly, you can alter these values based on the z-axis values you're getting.

- `down_threshold` - The z-axis value must be less than this for the timer to start when in the down orientation. Defaults to -8.
- `up_threshold` - The z-axis value must be greater than this for the timer to start when in the up orientation. Defaults to 8.

If you used the parts suggested in the guide, you do not need to change the following. But, in the event you're using a different form of NeoPixels, you can update the following:

- `number_of_pixels` - The total number of LEDs connected to your QT Py.

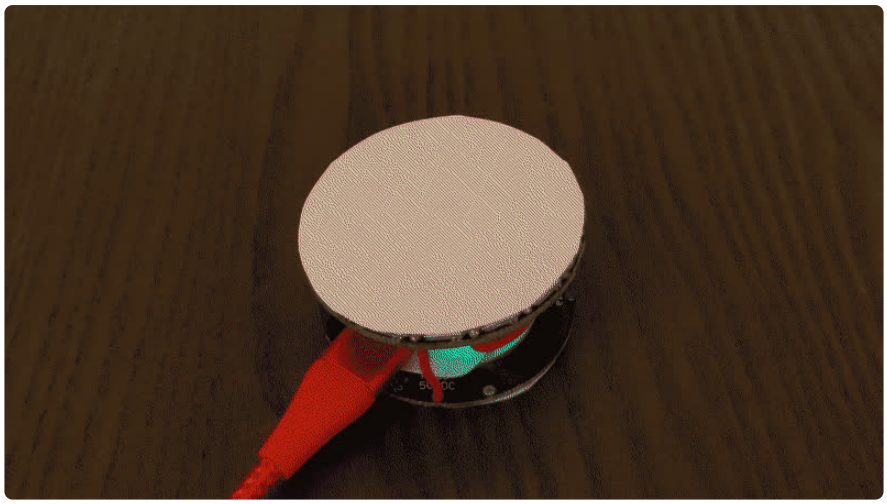
Use the Hydration Reminder

Plug it in to begin. It starts in idle mode. Flip it over to start the timer.

The timer will begin. By default, it will fade from blue to white.

If you don't do anything when the timer is up, it will enter Red Alert mode and begin blinking.

Flip it over at any time to restart the timer.



Flip it on its side to stop the timer and turn off the LEDs.