CLUE Step Counter with ST LSM6DS33

Created by Liz Clark

https://learn.adafruit.com/clue-step-counter-st-lsm6ds33

Last updated on 2024-06-03 03:06:04 PM EDT

Table of Contents

Overview 3

- How It Works
- Parts
- Project Video

CircuitPython on CLUE 5

- Set up CircuitPython Quick Start!

Coding the CLUE Step Counter 8

- Additional Libraries
- Code

CircuitPython Code Walkthrough 12

- Setup
- Graphics
- Pedometer Setup
- The Loop
- Counting Steps
- Calculating Average Steps Per Hour
- Updating the Progress Bar
- Adjusting Screen Brightness

Final Assembly and Use 18

Overview

You can clue into your daily step count with the CLUE board! Pair it with a wearable case and you have a DIY step counter. A lot of step counters require an app to see your data, but with this project you can see your daily steps without any fears about your data security.

How It Works

The CLUE's on-board accelerometer has a built-in pedometer. With the CircuitPython CLUE and LSM6DS33 libraries, you can access the pedometer to count your steps with just a few lines of code.

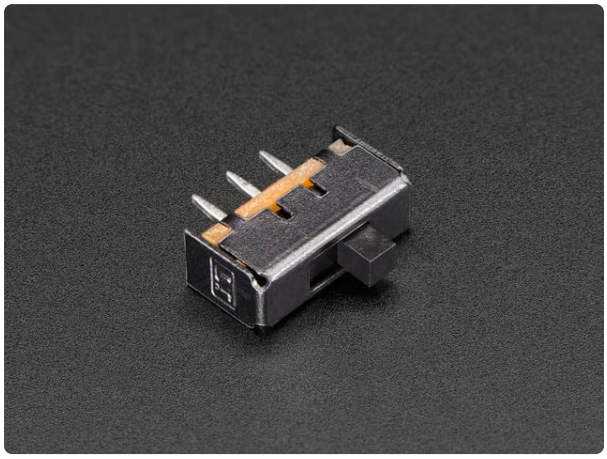
In addition to monitoring your total step count, you can also track your progress towards your step goal and see how many steps per hour you're taking. To keep your battery life going strong, you can also adjust the CLUE's display brightness using the A and B buttons on either side of the screen.

Parts

[Adafruit CLUE - nRF52840 Express with Bluetooth LE](https://www.adafruit.com/product/4500)

Do you feel like you just don't have a CLUE? Well, we can help with that - get a CLUE here at Adafruit by picking up this sensor-packed development board. We wanted to build some...

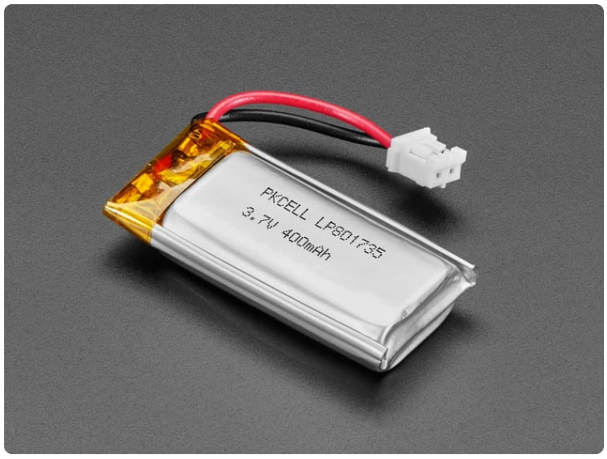
https://www.adafruit.com/product/4500



[Breadboard-friendly SPDT Slide Switch](https://www.adafruit.com/product/805)

These nice switches are perfect for use with breadboard and perfboard projects. They have 0.1" spacing and snap in nicely into a solderless breadboard. They're easy to switch...

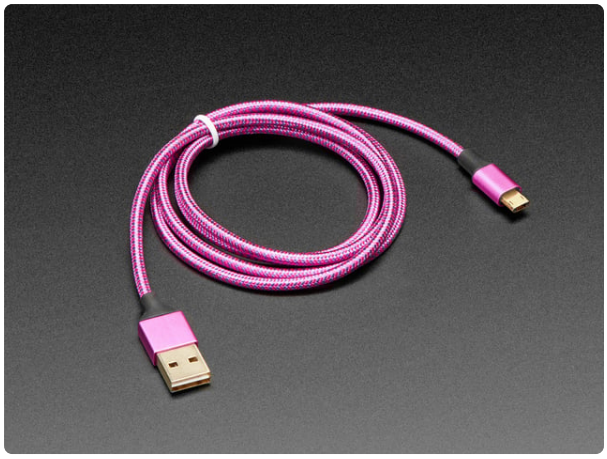
https://www.adafruit.com/product/805



[Lithium Ion Polymer Battery Ideal For Feathers - 3.7V 400mAh](https://www.adafruit.com/product/3898)

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/3898



Fully Reversible Pink/Purple USB A to micro B Cable - 1m long
This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also fully reversible! That's right, you will save seconds a day by...
https://www.adafruit.com/product/4111

Project Video

CircuitPython on CLUE

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

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

Set up CircuitPython Quick Start!

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

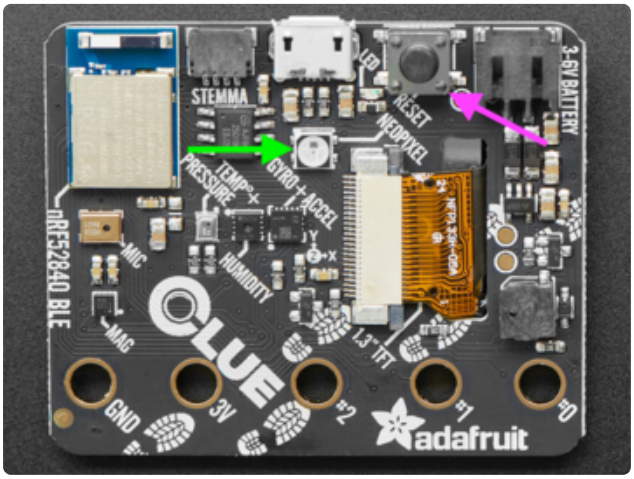
Download the latest version of CircuitPython for CLUE from circuitpython.org

https://adafru.it/IHF

Click the link above to download the latest version of CircuitPython for the CLUE.

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

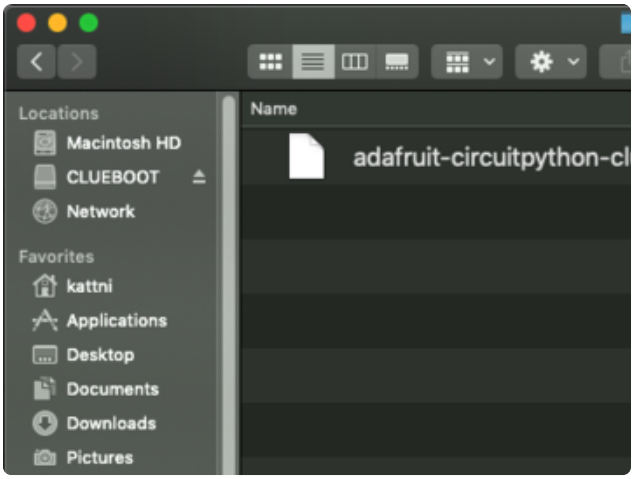
Plug your CLUE into your computer using a known-good USB cable.



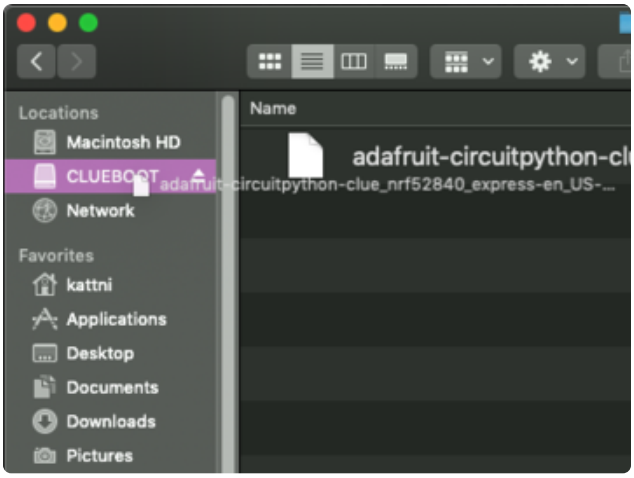
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

Double-click the **Reset** button on the top (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

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

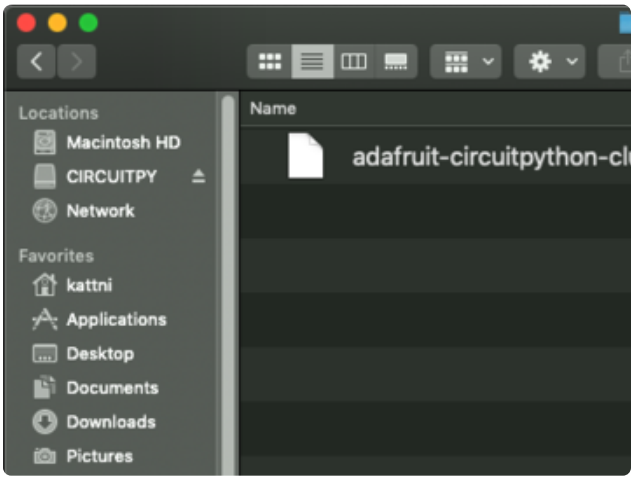


You will see a new disk drive appear called **CLUEBOOT**.



Drag the **adafruit-circuitpython-clue-etc.uf2** file to **CLUEBOOT.**

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

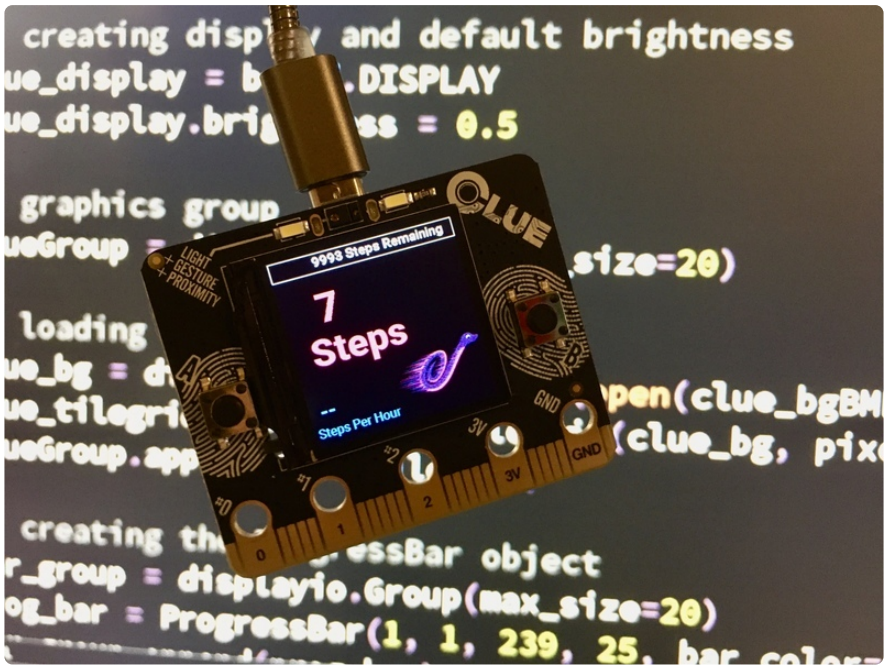


If this is the first time you're installing CircuitPython or you're doing a completely fresh install after erasing the filesystem, you will have two files - **boot_out.txt**, and **code.py**, and one folder - **lib** on your **CIRCUITPY** drive.

If CircuitPython was already installed, the files present before reloading CircuitPython should still be present on your **CIRCUITPY** drive. Loading CircuitPython will not create new files if there was already a CircuitPython filesystem present.

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

Coding the CLUE Step Counter

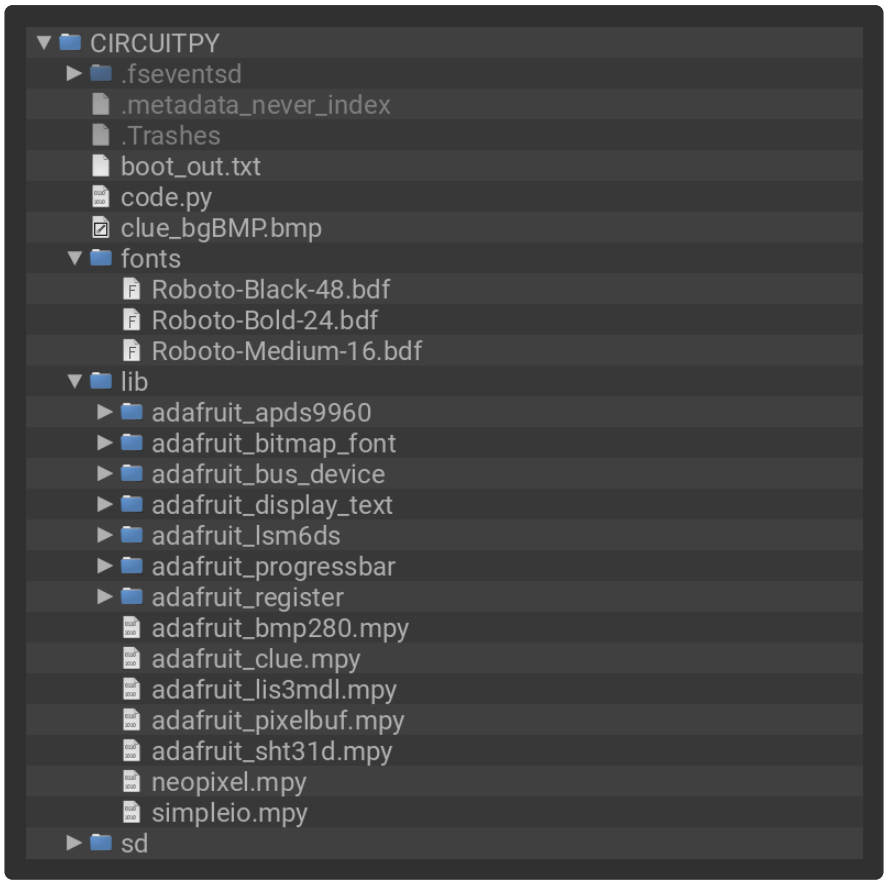


Additional Libraries

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 **Clue_Step_Counter/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

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



Code

```
# SPDX-FileCopyrightText: 2020 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import displayio
from adafruit_clue import clue
from simpleio import map_range
from adafruit_bitmap_font import bitmap_font
from adafruit_lsm6ds.lsm6ds33 import LSM6DS33
from adafruit_lsm6ds import Rate, AccelRange
from adafruit_progressbar.progressbar import ProgressBar
from adafruit_display_text.label import Label

#  turns off onboard NeoPixel to conserve battery
clue.pixel.brightness = (0.0)

#  accessing the Clue's accelerometer
sensor = LSM6DS33(board.I2C())

#  step goal
step_goal = 10000

#  onboard button states
a_state = False
b_state = False

#  array to adjust screen brightness
bright_level = [0, 0.5, 1]
```

```
countdown = 0 #  variable for the step goal progress bar
clock = 0 #  variable used to keep track of time for the steps per hour counter
clock_count = 0 #  holds the number of hours that the step counter has been running
clock_check = 0 #  holds the result of the clock divided by 3600 seconds (1 hour)
last_step = 0 #  state used to properly counter steps
mono = time.monotonic() #  time.monotonic() device
mode = 1 #  state used to track screen brightness
steps_log = 0 #  holds total steps to check for steps per hour
steps_remaining = 0 #  holds the remaining steps needed to reach the step goal
sph = 0 #  holds steps per hour

#  variables to hold file locations for background and fonts
clue_bgBMP = "/clue_bgBMP.bmp"
small_font = "/fonts/Roboto-Medium-16.bdf"
med_font = "/fonts/Roboto-Bold-24.bdf"
big_font = "/fonts/Roboto-Black-48.bdf"

#  glyphs for fonts
glyphs = b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-,.: '

#  loading bitmap fonts
small_font = bitmap_font.load_font(small_font)
small_font.load_glyphs(glyphs)
med_font = bitmap_font.load_font(med_font)
med_font.load_glyphs(glyphs)
big_font = bitmap_font.load_font(big_font)
big_font.load_glyphs(glyphs)

#  creating display and default brightness
clue_display = board.DISPLAY
clue_display.brightness = 0.5

#  graphics group
clueGroup = displayio.Group()

#  loading bitmap background
# CircuitPython 6 & 7 compatible
clue_bg = displayio.OnDiskBitmap(open(clue_bgBMP, "rb"))
clue_tilegrid = displayio.TileGrid(
    clue_bg, pixel_shader=getattr(clue_bg, 'pixel_shader',
displayio.ColorConverter())
)
clueGroup.append(clue_tilegrid)

# # CircuitPython 7+ compatible
# clue_bg = displayio.OnDiskBitmap(clue_bgBMP)
# clue_tilegrid = displayio.TileGrid(clue_bg, pixel_shader=clue_bg.pixel_shader)
# clueGroup.append(clue_tilegrid)

#  creating the ProgressBar object
bar_group = displayio.Group()
prog_bar = ProgressBar(1, 1, 239, 25, bar_color=0x652f8f)
bar_group.append(prog_bar)

clueGroup.append(bar_group)

#  text for step goal
steps_countdown = Label(small_font, text='%d Steps Remaining' % step_goal,
color=clue.WHITE)
steps_countdown.x = 55
steps_countdown.y = 12

#  text for steps
text_steps = Label(big_font, text="0     ", color=0xe90e8b)
text_steps.x = 45
text_steps.y = 70

#  text for steps per hour
text_sph = Label(med_font, text=" -- ", color=0x29abe2)
```

```
text_sph.x = 8
text_sph.y = 195

#  adding all text to the display group
clueGroup.append(text_sph)
clueGroup.append(steps_countdown)
clueGroup.append(text_steps)

#  sending display group to the display at startup
clue_display.root_group = clueGroup

#  setting up the accelerometer and pedometer
sensor.accelerometer_range = AccelRange.RANGE_2G
sensor.accelerometer_data_rate = Rate.RATE_26_HZ
sensor.gyro_data_rate = Rate.RATE_SHUTDOWN
sensor.pedometer_enable = True

while True:

    #  button debouncing
    if not clue.button_a and not a_state:
        a_state = True
    if not clue.button_b and not b_state:
        b_state = True

    #  setting up steps to hold step count
    steps = sensor.pedometer_steps

    #  creating the data for the ProgressBar
    countdown = map_range(steps, 0, step_goal, 0.0, 1.0)

    #  actual counting of the steps
    #  if a step is taken:
    if abs(steps-last_step) > 1:
        step_time = time.monotonic()
        #  updates last_step
        last_step = steps
        #  updates the display
        text_steps.text = '%d' % steps
        clock = step_time - mono

        #  logging steps per hour
        if clock > 3600:
            #  gets number of hours to add to total
            clock_check = clock / 3600
            #  logs the step count as of that hour
            steps_log = steps
            #  adds the hours to get a new hours total
            clock_count += round(clock_check)
            #  divides steps by hours to get steps per hour
            sph = steps_log / clock_count
            #  adds the sph to the display
            text_sph.text = '%d' % sph
            #  resets clock to count to the next hour again
            clock = 0
            mono = time.monotonic()

        #  adjusting countdown to step goal
        prog_bar.progress = float(countdown)

    #  displaying countdown to step goal
    if step_goal - steps > 0:
        steps_remaining = step_goal - steps
        steps_countdown.text = '%d Steps Remaining' % steps_remaining
    else:
        steps_countdown.text = 'Steps Goal Met!'

    #  adjusting screen brightness, a button decreases brightness
    if clue.button_a and a_state:
```

```
        mode -= 1
        a_state = False
        if mode < 0:
            mode = 0
            clue_display.brightness = bright_level[mode]
        else:
            clue_display.brightness = bright_level[mode]
    #  b button increases brightness
    if clue.button_b and b_state:
        mode += 1
        b_state = False
        if mode > 2:
            mode = 2
            clue_display.brightness = bright_level[mode]
        else:
            clue_display.brightness = bright_level[mode]

    time.sleep(0.001)
```

CircuitPython Code Walkthrough

Setup

The CircuitPython code begins by importing the libraries.

```
import time
import board
import displayio
from adafruit_clue import clue
from simpleio import map_range
from adafruit_bitmap_font import bitmap_font
from adafruit_lsm6ds import LSM6DS33, Rate, AccelRange
from adafruit_progressbar import ProgressBar
from adafruit_display_text.label import Label
```

After the libraries are imported, the on-board NeoPixel is turned off. This is done in an effort to conserve battery life.

```
clue.pixel.brightness = (0.0)
```

Next, the CLUE's accelerometer is defined and will be called as `sensor` in the code. There are built-in functions in the CLUE's CircuitPython library for common accelerometer uses but this is how you can access it directly.

```
sensor = LSM6DS33(board.I2C())
```

Then, the overall step goal is setup. You can edit this number to match your goal.

```
step_goal = 10000
```

Some state machines are created for the physical A and B buttons on the front of the CLUE. These will be used to adjust the brightness of the screen.

```
a_state = False
b_state = False
```

Speaking of screen brightness, this is followed by an array called `bright_level`, which holds the three brightness level options for the screen: `0` (off), `0.5` (half brightness) and `1` (full brightness).

```
bright_level = [0, 0.5, 1]
```

This is followed by various states and variables that will be used in the main loop. Their purposes are commented next to them.

```
countdown = 0 #  variable for the step goal progress bar
clock = 0 #  variable used to keep track of time for the steps per hour counter
clock_count = 0 #  holds the number of hours that the step counter has been running
clock_check = 0 #  holds the result of the clock divided by 3600 seconds (1 hour)
last_step = 0 #  state used to properly counter steps
mono = time.monotonic() #  time.monotonic() device
mode = 1 #  state used to track screen brightness
steps_log = 0 #  holds total steps to check for steps per hour
steps_remaining = 0 #  holds the remaining steps needed to reach the step goal
sph = 0 #  holds steps per hour
```

Graphics

Up next is the graphics setup. First, the on-board file locations for the graphical background and the fonts are defined.

```
clue_bgBMP = "/clue_bgBMP.bmp"
small_font = "/fonts/Roboto-Medium-16.bdf"
med_font = "/fonts/Roboto-Bold-24.bdf"
big_font = "/fonts/Roboto-Black-48.bdf"
```

Next, the glyphs are defined that will be used with the bitmap fonts and the setup for the bitmap fonts is completed.

```
glyphs = b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-,.: '

small_font = bitmap_font.load_font(small_font)
small_font.load_glyphs(glyphs)
med_font = bitmap_font.load_font(med_font)
med_font.load_glyphs(glyphs)
big_font = bitmap_font.load_font(big_font)
big_font.load_glyphs(glyphs)
```

The CLUE's display is setup to be `clue_display` and the default brightness is set to half.

```
clue_display = board.DISPLAY
clue_display.brightness = 0.5
```

Next, the graphics group is created along with the `tilegrid`. This will hold the bitmap background.

```
clueGroup = displayio.Group()

clue_bg = displayio.OnDiskBitmap(open("/clue_bgBMP.bmp", "rb"))
clue_tilegrid = displayio.TileGrid(clue_bg, pixel_shader=getattr(clue_bg,
'pixel_shader', displayio.ColorConverter()))
clueGroup.append(clue_tilegrid)
```

Following the bitmap background, a `ProgressBar` is created. This bar will illustrate your progress to hitting your step goal at the top of the display.

```
bar_group = displayio.Group()
prog_bar = ProgressBar, (11, 239, 25, bar_color=0x652f8f)
bar_group.append(prog_bar)

clueGroup.append(bar_group)
```

For more info on the progress bar library, check out this documentation on GitHub

https://adafru.it/Kpb

The final graphical elements are text objects. They'll hold the text for steps remaining, the step count and steps per hour. The text objects are added to the `clueGroup` graphic and then the `clueGroup` is shown on the CLUE's display on boot with `clue_display.root_group = clueGroup`.

```
steps_countdown = Label(small_font, text='%d Steps Remaining' % step_goal,
color=clue.WHITE)
steps_countdown.x = 55
steps_countdown.y = 12

text_steps = Label(big_font, text="0     ", color=0xe90e8b)
text_steps.x = 45
text_steps.y = 70

text_sph = Label(med_font, text=" -- ", color=0x29abe2)
text_sph.x = 8
text_sph.y = 195

clueGroup.append(text_sph)
clueGroup.append(steps_countdown)
clueGroup.append(text_steps)

clue_display.root_group = clueGroup
```

Pedometer Setup

The final lines of code before the loop setup the accelerometer on the CLUE board and enable the pedometer.

```
sensor.accelerometer_range = AccelRange.RANGE_2G
sensor.accelerometer_data_rate = Rate.RATE_26_HZ
sensor.gyro_data_rate = Rate.RATE_SHUTDOWN
sensor.pedometer_enable = True
```

The Loop

After all the setup, the loop begins with button debouncing for the A and B buttons located on the front of the CLUE.

```
while True:
    if not clue.button_a and not a_state:
        a_state = True
    if not clue.button_b and not b_state:
        b_state = True
```

This is followed by setting up the variable `steps` to hold the step count being collected by the CLUE's pedometer.

```
steps = sensor.pedometer_steps
```

Next, `countdown` is setup to hold what will be the progress bar's data that will show how close you are to your steps goal. This utilizes the `map_range` function, which is handy for mapping different value ranges so that they play well together.

You'll see that there are five values in the parentheses: `steps`, `0`, `step_goal`, `0.0` and `1.0`. Ignoring `steps` for a moment, the value range of `0` to `step_goal` is being mapped to `0.0` and `1.0`, which is the range that the ProgressBar library looks for when creating a ProgressBar object. The first item, in this case `steps`, is what will be subtracted from the total value range (in this case `step_goal`).

In summary, the `step_goal` range is being mapped to the ProgressBar's range, which will have your total steps at the time subtracted in order to display the progress towards your goal.

```
countdown = map_range(steps, 0, step_goal, 0.0, 1.0)
```

Counting Steps

This leads to the actual step counting. It begins with an `if` statement that checks if the step count has changed.

```
if abs(steps-last_step) &gt; 1:
```

If it has, then `time.monotonic()` is logged as `step_time` and `last_steps` is updated to hold the value of `steps`.

```
step_time = time.monotonic()
last_step = steps
```

Additionally, the display is updated to show the current step count. The `text_steps` text object displays the value being held by `steps`.

```
text_steps.text = '%d' % steps
```

This is followed by some time tracking that will be used to calculate your steps per hour. `clock` is setup to hold the difference between the two `time.monotonic()` devices `step_time` and `mono`. `mono` is logged before the loop and by doing this you can track how long steps have actually been taking place rather than the CLUE's time that it has been operating.

```
clock = step_time - mono
```

Calculating Average Steps Per Hour

To get the steps per hour calculation, there is an `if` statement that checks if `steps` have been counted for more than an hour, or 3600 seconds. If they have, then the steps per hour will be updated. This is done by having `clock_check` hold the value of `clock` divided by `3600`. This is done so that you can get a count of how many hours have passed.

Your step count is then held by `steps_log`. Another variable, `clock_count`, is increased by the rounded result of `clock_check`. This means that the total hour count is increased. The steps per hour, being held by `sph`, is then updated to divide `steps_log`, which is the total steps you've taken, by the `clock_count`, which is the total number of hours that you've been counting steps. The result of this calculation is then pushed to the `text_sph` object to update the Clue's display.

Finally, `clock` is reset to `0` to begin counting down to the next hour and `mono` is updated to hold `time.monotonic()`.

```
if clock &gt; 3600:
    clock_check = clock / 3600
    steps_log = steps
    clock_count += round(clock_check)
    print('hour count: %d' % clock_count)
    sph = steps_log / clock_count
    text_sph.text = '%d' % sph
    clock = 0
    mono = time.monotonic()
```

Updating the Progress Bar

The progress bar is updated by setting `bar.progress` to hold the value of `countdown` as a float. This simultaneously updates the display and the math going on behind the scenes with the ProgressBar library.

```
bar.progress = float(countdown)
```

Up next is a new `if` statement, this time checking to see if you have reached your steps goal. If you haven't, then the remaining steps are stored in the variable `steps_remaining`. This value is then stored in the `steps_countdown` text object and is sent to the display to be shown over the progress bar.

However, if the goal has been met, then the `steps_countdown` text will read "Steps Goal Met!".

```
if step_goal - steps &gt; 0:
        steps_remaining = step_goal - steps
        steps_countdown.text = '%d Steps Remaining' % steps_remaining
else:
        steps_countdown.text = 'Steps Goal Met!'
```

Adjusting Screen Brightness

The last portion of the loop is for adjusting the CLUE's screen brightness. This will come in handy for preserving battery life. If you press the A button, then the brightness will decrease and if you press the B button, the brightness will increase.

There are three brightness levels, which are held in the `bright_level` array. The variable `mode` is being used to track the current brightness level and position in the `bright_level` array. If A is pressed, then `mode` is decreased by `1` and if B is pressed, then `mode` is increased by `1`. The result is a change in the screen's

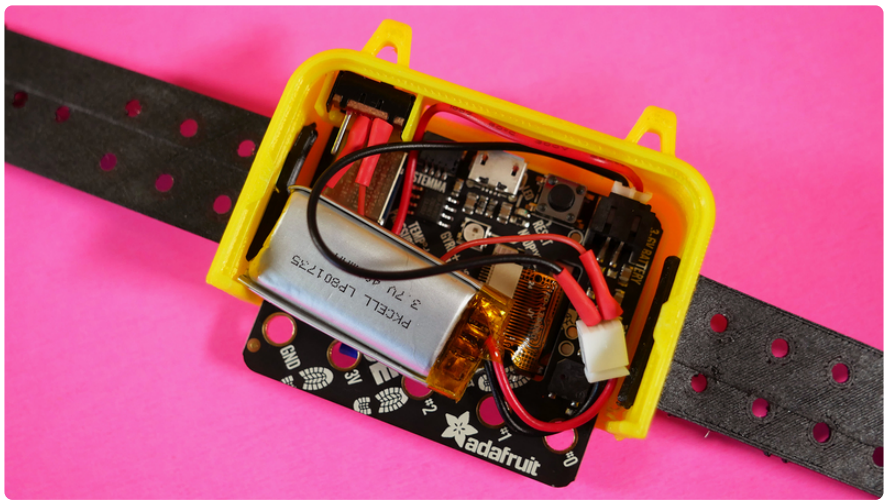
brightness level. However, if `mode` is less than `0` or greater than `2`, the brightness does not change since you will no longer be in range of the `bright_level` array.

```
if clue.button_a and a_state:
        mode -= 1
        a_state = False
        if mode &lt; 0:
            mode = 0
            clue_display.brightness = bright_level[mode]
        else:
            clue_display.brightness = bright_level[mode]

if clue.button_b and b_state:
        mode += 1
        b_state = False
        if mode &gt; 2:
            mode = 2
            clue_display.brightness = bright_level[mode]
        else:
            clue_display.brightness = bright_level[mode]
```

Final Assembly and Use

After loading the CircuitPython code and files onto the CLUE, you're ready to get stepping. For best results, you'll want to house your CLUE in a wearable case. The Ruiz brothers have a great 3D printed design and Learn Guide for a CLUE case that you wear on your wrist like a watch. This is the perfect way to count your steps. The guide also details how to install an on/off switch for the CLUE with a lipo battery.



Link to the CLUE Case Learn Guide by the Ruiz Brothers

https://adafru.it/Kpc

After printing and assembling your case, your CLUE step counter is complete! You can track your steps all day long without worrying about the security of your health data

or location tracking. The battery should last at least a full day so you won't miss a single step.