CircuitPython BLE Heart Rate Zone Trainer Display

Created by John Park

https://learn.adafruit.com/circuitpython-ble-heart-rate-monitor-gizmo

Last updated on 2023-08-29 03:59:46 PM EDT
# Table of Contents

## Overview
- Parts & Materials
- Heart Rate Monitor
- Optional
- Alt Version

## Understanding BLE
- BLE Basics
- Bluetooth LE Terms

## Heart Rate Service
- Heart Rate Characteristics
- nRF Connect View

## Build the Heart Rate Zone Trainer
- I2C Address Jumper
- Assembly
- Battery Power
- Optional Filter

## CircuitPython for Feather nRF52840
- Set up CircuitPython Quick Start!

## Code the Heart Rate Zone Trainer in CircuitPython
- Libraries
- Text Editor
- Code.py
- Code Explainer
- LED setup
- Max Heart Rate Variable
- Display Prep
- Seven Segment Display Use
- Max Heart Rate Display
- BLE Instance
- Display Functions
- Fresh Connection
- Device Info
- Heart Rate and Zone Percent

## Heart Rate Zone Trainer in Action
- Setup
- BPM and Percentage of Max Heart Rate

## CLUE Heart Rate Trainer
- Libraries
- Text Editor
- Code.py
- Code Explainer
- Connection
- Measurements
- Alarm
• In Use
• Setup
• Adjustment
• Alarm!
Overview

Heart rate zone training can be an excellent way to monitor workout intensity and increase your fitness and endurance. By spending certain periods of workout time at different percentages of your maximum heart rate. For example, warming up at 50-60%, then entering the "fitness zone" of 60-70% for a period, then going into the aerobic zone of 70-80%, finally peaking in the intense anaerobic zone of 80-90%.

Using a Bluetooth LE heart rate monitor armband or chest strap, you can send up-to-the moment heart rate data to a battery-powered Feather nRF5280 Bluefruit equipped with a pair of seven segment LED displays.

Place it on a wall or shelf where you can easily see it while you work out! Or carry it or hook it to your bike. CircuitPython makes it straightforward to connect to your heart rate monitor and stream the heart rate data and zone percentages to the displays.

Parts & Materials

Heart Rate Monitor
You'll need a heart rate monitor that supports Bluetooth Low Energy (BLE). I'm using the Scosche RHYTHM+ but you should be able to use any monitor that uses the Bluetooth SIG Heart Rate service standard.

These work by flashing green (and sometimes yellow) LEDs against your skin and then measuring the reflected light that returns. The color changes/darkens during the pulse of your heart thanks to all that blood sloshing around!
Adafruit Feather nRF52840 Express
The Adafruit Feather nRF52840 Express is the new Feather family member with Bluetooth Low Energy and native USB support featuring the nRF52840! It's... https://www.adafruit.com/product/4062

FeatherWing Tripler Mini Kit - Prototyping Add-on For Feathers
This is the FeatherWing Tripler - a prototyping add-on and more for all Feather boards. This is similar to our https://www.adafruit.com/product/3417

Adafruit 0.56" 4-Digit 7-Segment FeatherWing Display - Red
One segment? No way dude! 7-Segments for life!This is the Red Adafruit 0.56" 4-Digit 7-Segment Display w/ FeatherWing Combo Pack! We also have these combo... https://www.adafruit.com/product/3108

Adafruit 0.56" 4-Digit 7-Segment FeatherWing Display - Blue
One segment? No way dude! 7-Segments for life!This is the Blue Adafruit 0.56" 4-Digit 7-Segment Display w/ FeatherWing Combo Pack! We also have these... https://www.adafruit.com/product/3106
1 x **USB Cable**
USB A to Micro-B - 3 foot long

https://www.adafruit.com/product/592

---

**Optional**

Seven Segment LED displays look pretty good on their own, but they look even better with a colored gel filter in front of them!

I really like the way they look through a small piece of [LEE Filters CL797 Gel Filter Sheet - Deep Purple](https://www.adafruit.com/). I get mine from [Filmtools](https://www.filmtools.com).

---

**Alt Version**

You can build a CLUE variant of this project instead. All you'll need are a CLUE board instead of the Feather and seven segment displays.
Understanding BLE

BLE Basics

To understand how we communicate between the MagicLight Bulb and the Circuit Playground Bluefruit (CPB), it's first important to get an overview of how Bluetooth Low Energy (BLE) works in general.

The nRF52840 chip on the CPB uses Bluetooth Low Energy, or BLE. BLE is a wireless communication protocol used by many devices, including mobile devices. You can communicate between your CPB and peripherals such as the Magic Light, mobile devices, and even other CPB boards!

There are a few terms and concepts commonly used in BLE with which you may want to familiarize yourself. This will help you understand what your code is doing when you’re using CircuitPython and BLE.

Two major concepts to know about are the two modes of BLE devices:

- Broadcasting mode (also called GAP for Generic Access Profile)
- Connected device mode (also called GATT for Generic ATTribute Profile).

GAP mode deals with broadcasting peripheral advertisements, such as "I'm a device named LEDBlue-19592CBC", as well as advertising information necessary to establish
a dedicated device connection if desired. The peripheral may also be advertising available services.

GATT mode deals with communications and attribute transfer between two devices once they are connected, such as between a heart monitor and a phone, or between your CPB and the Magic Light.

![Bluetooth LE Terms Diagram]

Bluetooth LE Terms

GAP Mode

Device Roles:

- Peripheral - The low-power device that broadcasts advertisements. Examples of peripherals include: heart rate monitor, smart watch, fitness tracker, iBeacon, and the Magic Light. The CPB can also work as a peripheral.
- Central - The host "computer" that observes advertisements being broadcast by the Peripherals. This is often a mobile device such as a phone, tablet, desktop or laptop, but the CPB can also act as a central (which it will in this project).

Terms:

- Advertising - Information sent by the peripheral before a dedicated connection has been established. All nearby Centrals can observe these advertisements. When a peripheral device advertises, it may be transmitting the name of the device, describing its capabilities, and/or some other piece of data. Central can look for advertising peripherals to connect to, and use that information to
determine each peripheral's capabilities (or Services offered, more on that below).

GATT Mode

Device Roles:

- **Server** - In connected mode, a device may take on a new role as a Server, providing a Service available to clients. It can now send and receive data packets as requested by the Client device to which it now has a connection.
- **Client** - In connected mode, a device may also take on a new role as Client that can send requests to one or more of a Server's available Services to send and receive data packets.

**NOTE:** A device in GATT mode can take on the role of both Server and Client while connected to another device.

Terms:

- **Profile** - A pre-defined collection of Services that a BLE device can provide. For example, the Heart Rate Profile, or the Cycling Sensor (bike computer) Profile. These Profiles are defined by the Bluetooth Special Interest Group (SIG). For devices that don't fit into one of the pre-defined Profiles, the manufacturer creates their own Profile. For example, there is not a "Smart Bulb" profile, so the Magic Light manufacturer has created their own unique one.
- **Service** - A function the Server provides. For example, a heart rate monitor armband may have separate Services for Device Information, Battery Service, and Heart Rate itself. Each Service is comprised of collections of information called Characteristics. In the case of the Heart Rate Service, the two Characteristics are Heart Rate Measurement and Body Sensor Location. The peripheral advertises its services.
- **Characteristic** - A Characteristic is a container for the value, or attribute, of a piece of data along with any associated metadata, such as a human-readable name. A characteristic may be readable, writable, or both. For example, the Heart Rate Measurement Characteristic can be served up to the Client device and will report the heart rate measurement as a number, as well as the unit string "bpm" for beats-per-minute. The Magic Light Server has a Characteristic for the RGB value of the bulb which can be written to by the Central to change the color. Characteristics each have a Universal Unique Identifier (UUID) which is a 16-bit or 128-bit ID.
• Packet - Data transmitted by a device. BLE devices and host computers transmit and receive data in small bursts called packets.

This guide is another good introduction to the concepts of BLE, including GAP, GATT, Profiles, Services, and Characteristics.

Heart Rate Service

The Bluetooth Special Interest Group has a standardized GATT (Generitt ATTRibute Profile) for heart rate monitors called the Heart Rate profile. (You can see a list of all the GATT services here.)

This defines the commands and data that can be exchanged between the heart rate sensor device and the client device such as a phone, tablet, or BLE capable microcontroller (like we'll use in our project).

If you want to see how the Bluetooth SIG defines a GATT, such as the Heart Rate Service, you can look at the official XML file here.

Even better, run that URL through a code beautifier, such as codebeatify.org for a more human-readable version.

Heart Rate Characteristics

The Heart Rate service defines a three characteristics that can be served from the heart rate monitor (HRM) to a connected device.
Heart Rate Measurement

The most important for most needs is the Heart Rate Measurement Values characteristic which serves up the following information:

- Heart rate, in beats per minute (BPM)
- Contact -- if the device is in contact with the body or not
- Energy Expended, in kilojoules
- RR Intervals in 1024ths of seconds -- this is the measurement of intervals between beats

Not all heart monitors support all of the above characteristics, so it isn't uncommon to see 'None' returned for certain values.

Body Sensor Location

Heart rate monitors will also include a characteristic for the intended location of the monitor on the body. This is built into the sensor firmware, not something that the device is determining on the fly! Standard values include:

- Wrist
- Chest
- Finger
- Hand
- Ear Lobe
- Foot
- Other

Heart Rate Control Point

If the HRM includes the Energy Expended feature, the heart rate control point characteristic is used to allow the client device to write control points to the HRM.

nRF Connect View

We can use the nRF Connect app from Nordic on iOS and Android to connect to a heart rate monitor and look at the service, characteristics, and data.
When we first connect to the device, we can see some data advertised including the device name, available services, connection parameters, manufacturer name, revision number, and more.

In this image we can see the HRM device (RHYTHM+) has been connected, and the Heart Rate Measurement characteristic is reporting its data.

If we request a read of the Body Sensor Location characteristic, we receive it as shown here:
Build the Heart Rate Zone Trainer

To begin, assemble the two seven segment display FeatherWings as shown here.

Solder male headers to the Feather nRF52840 as shown here.

Then, add the plain female headers to the FeatherWing Tripler as shown here.
I2C Address Jumper

In order to use two displays on one Feather, we need to give the boards unique I2C addresses.

Leave the red BPM display board at its default state, which will use 0x70 as its address.

For the blue heart rate zone percentage display, we'll solder the jumper pad marked A0, which will give the board the address 0x71.

I did this by soldering a small piece of wire across the two pads as shown here.
Assembly
It's best to install the Feather nRF52840 at the top of the tripler so the battery cable doesn't interfere with the displays. (This won't matter if you choose to power over USB instead.)

Then, place the red seven segment display at the middle position.

Place the blue display at the bottom position.
Battery Power
You can power the Heart Rate Zone Trainer from a LiPoly battery, by plugging it into the Feather's battery plug.

Use a little bit of double stick foam tape to adhere it to the back of the board, and snake the wire under the Feather to keep it out of the way.

To charge the battery, simply plug the Feather into USB power.

Optional Filter
One nice way to improve the look of your LED displays is with a small piece of colored gel filter designed for film and theatrical lighting. It hides the white unlit segments so the lit ones really stand out. In this project I'm using a deep purple gel which gives both the red and blue displays a very nice look. You can experiment with different filters if you use different display colors.
Cut a small section of gel from your sheet to fit the displays.

To adhere them to the displays, I used a couple of thin strips of 3M double stick transparent tape at the top and bottom of the display pair.

You can attach a strip across the top as shown here, and then trim away the excess with a hobby knife.

Repeat this for the bottom display's bottom edge, and then press the gel filter into place.
Once you code the Feather in CircuitPython on the next page you'll be able to see the beautiful impact of the filter on the displays!

CircuitPython for Feather nRF52840

CircuitPython is a derivative of MicroPython 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.

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 this board via CircuitPython.org
Click the link above to download the latest UF2 file.

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

Plug your Feather nRF52840 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 next to the USB connector on your board, and you will see the NeoPixel RGB LED turn green (identified by the arrow in the image). 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 FTHR840BOOT.

Drag the adafruit_circuitpython_etc.uf2 file to FTHR840BOOT.

The LED will flash. Then, the FTHR840BOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

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

©Adafruit Industries
Code the Heart Rate Zone Trainer in CircuitPython

Libraries
Once your Feather nRF52840 is set up with CircuitPython, you'll also need to add some library files. Follow this page for information on how to download and add libraries to your Feather.

From the library bundle you downloaded in that guide page, transfer the following libraries onto the Feather's /lib directory:

- adafruit_ble
- adafruit_bus_device
- adafruit_ht16k33
- adafruit_register
- adafruit_ble_heart_rate.mpy
- neopixel.mpy

Text Editor
Adafruit recommends using the Mu editor for using your CircuitPython code with the Feather boards. You can get more info in this guide.

Alternatively, you can use any text editor that saves files.

Code.py
Copy the code shown below, paste it into Mu. Plug your Feather into your computer via a known good USB cable. In your operating system's file explorer/finder, you should see a new flash drive named CIRCUITPY. Save the code from Mu to the Feather's CIRCUITPY drive as code.py

```python
# SPDX-FileCopyrightText: 2020 John Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

Heart Rate Trainer
Read heart rate data from a heart rate peripheral using the standard BLE Heart Rate service.
Displays BPM value to Seven Segment FeatherWing
```
import time
import board

import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.standard.device_info import DeviceInfoService
from adafruit_ble_heart_rate import HeartRateService
from digitalio import DigitalInOut, Direction

red_led = DigitalInOut(board.RED_LED)
red_led.direction = Direction.OUTPUT
red_led.value = True

blue_led = DigitalInOut(board.BLUE_LED)
blue_led.direction = Direction.OUTPUT
blue_led.value = False

# target heart rate for interval training
# Change this number depending on your max heart rate, usually figured
# as (220 - your age).
max_rate = 180

# Seven Segment FeatherWing setup
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
# microcontroller
display_A = Seg7x4(i2c, address=0x70)  # this will be the BPM display
display_A.fill(0)  # Clear the display

# display_A "b.P.M."
display_A.set_digit_raw(0, 0b11111100)
display_A.set_digit_raw(1, 0b11110011)
display_A.set_digit_raw(2, 0b00110011)
display_A.set_digit_raw(3, 0b10100111)

# display_B "Prct"
display_B.set_digit_raw(0, 0b01110011)
display_B.set_digit_raw(1, 0b01010000)
display_B.set_digit_raw(2, 0b01011000)
display_B.set_digit_raw(3, 0b01000110)
time.sleep(3)

display_A.fill(0)
for h in range(4):
    # display_B show maximum heart rate value
    display_B.fill(0)
display_B.print(max_rate)
time.sleep(2)

# PyLint can't find BLERadio for some reason so special case it here.
ble = adafruit_ble.BLERadio()  # pylint: disable=no-member

hr_connection = None

def display_SCAN():
    display_A.fill(0)
display_A.set_digit_raw(0, 0b01111100)
display_A.set_digit_raw(1, 0b01110011)
display_A.set_digit_raw(2, 0b00110011)
display_A.set_digit_raw(3, 0b10100111)
display_A.set_digit_raw(3, 0b00110111)

def display_bLE():
    display_B.fill(0)
    display_B.set_digit_raw(0, 0b00000000)
    display_B.set_digit_raw(1, 0b01111100)
    display_B.set_digit_raw(2, 0b00111000)
    display_B.set_digit_raw(3, 0b01111001)

def display_dots():  # "...."
    for j in range(4):
        display_A.set_digit_raw(j, 0b10000000)
        display_B.set_digit_raw(j, 0b10000000)

def display_dashes():  # "----"
    for k in range(4):
        display_A.set_digit_raw(k, 0b01000000)
        display_B.set_digit_raw(k, 0b01000000)

# Start with a fresh connection.
if ble.connected:
    display_SCAN()
    display_bLE()
    time.sleep(1)

for connection in ble.connections:
    if HeartRateService in connection:
        connection.disconnect()
        break

while True:
    print("Scanning...")
    red_led.value = True
    blue_led.value = False
    display SCAN()
    display_bLE()
    time.sleep(1)

for adv in ble.start_scan(ProvideServicesAdvertisement, timeout=5):
    if HeartRateService in adv.services:
        print("found a HeartRateService advertisement")
        hr_connection = ble.connect(adv)
        display dots()
        time.sleep(2)
        print("Connected")
        blue_led.value = True
        red_led.value = False
        break

# Stop scanning whether or not we are connected.
ble.stop_scan()
print("Stopped scan")
red_led.value = False
blue_led.value = True
time.sleep(0.5)

if hr_connection and hr_connection.connected:
    print("Fetch connection")
    if DeviceInfoService in hr_connection:
        dis = hr_connection[DeviceInfoService]
        try:
            manufacturer = dis.manufacturer
        except AttributeError:
            manufacturer = "(Manufacturer Not specified)"
        try:
            model_number = dis.model_number
        except AttributeError:
model_number = "(Model number not specified)"

print("Device:", manufacturer, model_number)
else:
    print("No device information")
hr_service = hr_connection[HeartRateService]
print("Location:", hr_service.location)

while hr_connection.connected:
    values = hr_service.measurement_values
    print(values)  # returns the full heart_rate data set
    if values:
        bpm = (values.heart_rate)
        if bpm is not 0:
            pct_target = (round(100*(bpm/max_rate)))
            display_A.fill(0)  # clear the display
            display_B.fill(0)
            if values.heart_rate is 0:
                display_dashes()
            else:
                display_A.fill(0)
                display_B.print(pct_target)
                time.sleep(0.1)
                display_A.print(bpm)
        time.sleep(0.9)
        display_A.set_digit_raw(0, 0b00000000)

Code Explainer

The code is doing a few fundamental things.

First, it loads the time and board libraries, as well as the necessary libraries to use BLE in general, and the HeartRateService in specific.

We also load the HT16K33 library to use the seven segment displays, and the digitalio library to use the Feather's on-board red and blue indicator LEDs.

import time
import board

import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.standard.device_info import DeviceInfoService
from adafruit_ble_heart_rate import HeartRateService
from adafruit_ht16k33.segments import Seg7x4
from digitalio import DigitalInOut, Direction

LED setup

Next, code to prepare the on-board LEDs and turn on the red one, while leaving the blue one turned off, until we start scanning for BLE devices.
# Feather on-board status LEDs setup
red_led = DigitalInOut(board.RED_LED)
red_led.direction = Direction.OUTPUT
red_led.value = True
blue_led = DigitalInOut(board.BLUE_LED)
blue_led.direction = Direction.OUTPUT
blue_led.value = False

## Max Heart Rate Variable

The `max_rate` variable is used to calculate your heart rate training zone percentages. You can change this to suit your maximum heart rate. The simplest way to calculate this is by subtracting your age from 220, but you can get much more specific numbers from a doctor or training specialist.

```python
max_rate = 180
```

## Display Prep

To prep for using the two displays, the `Set7x4` objects on the I2C bus are set using unique addresses. Remember, the second display's A0 pads was jumpered to set the address to 0x71, while leaving the first display at the default address 0x70.

```python
# Seven Segment FeatherWing setup
i2c = board.I2C()
display_A = Seg7x4(i2c, address=0x70)  # this will be the BPM display
display_A.brightness = 15
display_A.fill(0)  # Clear the display
# Second display has A0 address jumpered
display_B = Seg7x4(i2c, address=0x71)  # this will be the % target display
display_B.brightness = 15
display_B.fill(0)  # Clear the display
```

## Seven Segment Display Use

This guide includes a great intro to using the matrix displays with CircuitPython.

You can set the display in a few different ways:

- `display_A.print(1234)` will display 1234
- `display_A.set_digit_raw(3, 0b00000001)` will light up the top segment of the fourth digit (far right) only.
- `display[0] = '6'` will display a 6 on the first digit (far left)
During the startup sequence we'll display "b.P.M" on display A (the red one) and "Prct" on the display B (the blue one).

```python
# display_A "b.P.M."
display_A.set_digit_raw(0, 0b11111100)
display_A.set_digit_raw(1, 0b11111011)
display_A.set_digit_raw(2, 0b00110011)
display_A.set_digit_raw(3, 0b10100111)
# display_B "Prct"
display_B.set_digit_raw(0, 0b01110011)
display_B.set_digit_raw(1, 0b01010000)
display_B.set_digit_raw(2, 0b01011000)
display_B.set_digit_raw(3, 0b01000110)
time.sleep(3)
```

Here's a good image guide to the raw segment bitmask.

**Max Heart Rate Display**

Next, we'll set display A to "...." and show the max_rate value for a couple seconds on display B.

```python
display_A.fill(0)
for h in range(4):
    # display_A.set_digit_raw(h, 0b10000000)
    # display_B show maximum heart rate value
display_B.fill(0)
display_B.print(max_rate)
time.sleep(2)
```

**BLE Instance**

The BLE radio is defined next with `ble = adafruit_ble.BLERadio()`
Display Functions

We'll keep the code neat by creating some functions that can be used repeatedly to show certain messages on the displays.

```python
def display_SCAN():
    display_A.fill(0)
    display_A.set_digit_raw(0, 0b01101101)
    display_A.set_digit_raw(1, 0b00111001)
    display_A.set_digit_raw(2, 0b01110111)
    display_A.set_digit_raw(3, 0b00110111)

def display_bLE():
    display_B.fill(0)
    display_B.set_digit_raw(0, 0b00000000)
    display_B.set_digit_raw(1, 0b01111100)
    display_B.set_digit_raw(2, 0b00111000)
    display_B.set_digit_raw(3, 0b01111001)

def display_dots():  # "...."
    for j in range(4):
        display_A.set_digit_raw(j, 0b10000000)
        display_B.set_digit_raw(j, 0b10000000)

def display_dashes():  # "----"
    for k in range(4):
        display_A.set_digit_raw(k, 0b01000000)
        display_B.set_digit_raw(k, 0b01000000)
```

Fresh Connection

We scan for a BLE device with the Heart Rate Service being advertised, and set the displays and status LEDs to match.

When we connect, the displays switch to four dots .... and we stop scanning.

```python
# Start with a fresh connection.
if ble.connected:
    display_SCAN()
    display_bLE()
    time.sleep(1)

    for connection in ble.connections:
        if HeartRateService in connection:
            connection.disconnect()
            break

while True:
    print("Scanning...")
    red_led.value = True
    blue_led.value = False
    display_SCAN()
    display_bLE()
    time.sleep(1)

    for adv in ble.start_scan(ProvideServicesAdvertisement, timeout=5):
        if HeartRateService in adv.services:
```

©Adafruit Industries
print("found a HeartRateService advertisement")
hr_connection = ble.connect(adv)
display_dots()
time.sleep(2)
print("Connected")
blue_led.value = True
red_led.value = False
break

# Stop scanning whether or not we are connected.
ble.stop_scan()
print("Stopped scan")
red_led.value = False
blue_led.value = True
time.sleep(0.5)

Device Info

With the heart rate monitor connected, we'll request info that is displayed in the Mu REPL, if your Feather is connected to your computer over USB. This is purely informational for curiosity and debug purposes, and not displayed on the seven segment displays.

if hr_connection and hr_connection.connected:
    print("Fetch connection")
    if DeviceInfoService in hr_connection:
        dis = hr_connection[DeviceInfoService]
        try:
            manufacturer = dis.manufacturer
        except AttributeError:
            manufacturer = "(Manufacturer Not specified)"
        try:
            model_number = dis.model_number
        except AttributeError:
            model_number = "(Model number not specified)"
        print("Device:", manufacturer, model_number)
    else:
        print("No device information")
    hr_service = hr_connection[HeartRateService]
    print("Location:", hr_service.location)

Heart Rate and Zone Percent

We've reached the heart of the program! This is the code that loops over and over while the devices are connected.

First, we cast the heart rate service's measurement characteristic attributes that are sent as \texttt{values}, and then we cast the heart rate value itself as \texttt{bpm}.

We'll check at first for non-zero \texttt{bpm} readings, as the heart rate monitor sends a few zeros at first, and we'll ignore them so nobody gets too worried, and just display four dashes.
We'll create a `pct_target` variable that calculates the percentage of the `max_rate` based on current `bpm`.

The displays are cleared and then the percent value is shown on display B and the `bpm` is shown on display A. This display blinks each time the loop is run, and the whole process repeats every second.

```python
while hr_connection.connected:
    values = hr_service.measurement_values
    print(values)  # returns the full heart_rate data set
    if values:
        bpm = (values.heart_rate)
        if bpm is not 0:
            pct_target = (round(100*(bpm/max_rate)))
            display_A.fill(0)  # clear the display
            display_B.fill(0)
            if values.heart_rate is 0:
                display_dashes()
            else:
                display_A.fill(0)
                display_B.print(pct_target)
                time.sleep(0.1)
                display_A.print(bpm)
            time.sleep(0.9)
        display_A.set_digit_raw(0, 0b00000000)
```

On the next page we'll see it in action.

**Heart Rate Zone Trainer in Action**

![Heart Rate Zone Trainer](image)

You can use the Heart Rate Zone Trainer any time you want to do some exercise, and be aware of your heart rate and the zone percentage you're in. Set it on a surface...
where you can see it easily, and pay attention to how long you are in different heart rate training zones.

Setup
Plug in the battery (or a USB power cable).

The display will show the intro text "b.P.M." and "Prct" so you know which display is used for what.

Next, your max heart rate is displayed, in this case, 180.

The device will start scanning for a BLE heart rate monitor, displaying "SCAN bLE".

Turn on the heart rate monitor and strap it to your inner arm. At first the displays will show "...." as it connects, and "----" as the first zero bpm values are sent.
BPM and Percentage of Max Heart Rate
As the Heart Rate Monitor sends values, the Feather will display the bpm on top and the percentage of max on the bottom.

You can do some jumping jacks or run around a bit to get your heart rate up.

Here's an action video -- you'll see the rate going up as I ran for a few minutes.

CLUE Heart Rate Trainer
Here's a bonus version of the BLE Heart Rate Zone Trainer you can make with the CLUE board alone, no need for external displays!

Follow these instructions (1) to set up your CLUE with CircuitPython, then check out this page (1) for info on adding libraries.

Libraries

Once your CLUE is set up with CircuitPython and library files in general, we'll add some project specific libraries.

From the library bundle you downloaded in that guide page, transfer any additional libraries shown here onto the CLUE's /lib directory on the CIRCUITPY drive:

- adafruit_apds9960
- adafruit_ble
- adafruit_ble_heart_rate.mpy
- adafruit_bmp280.mpy
- adafruit_bus_device
- adafruit_clue.py
- adafruit_display_notification
- adafruit_display_shapes
- adafruit_display_text
- adafruit_lis3mdl.mpy
- adafruit_lsm6ds.mpy
- adafruit_register
- adafruit_sht31d.mpy
- neopixel.mpy
- simpleio.mpy

Text Editor

Adafruit recommends using the Mu editor for using your CircuitPython code with the Feather boards. You can get more info in this guide (1).

Alternatively, you can use any text editor that saves files.
Copy the code shown below, paste it into Mu. Plug your CLUE into your computer via a known good USB cable. In your operating system's file explorer/finder, you should see a new flash drive named CIRCUITPY. Save the code from Mu to the CLUE's CIRCUITPY drive as code.py

```python
# SPDX-FileCopyrightText: 2020 John Park for Adafruit Industries
# SPDX-License-Identifier: MIT

Heart Rate Trainer
Read heart rate data from a heart rate peripheral using the standard BLE Heart Rate service.
Displays BPM value and percentage of max heart rate on CLUE

import time
from adafruit_clue import clue
import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.standard.device_info import DeviceInfoService
from adafruit_ble_heart_rate import HeartRateService

clue_data = clue.simple_text_display(title="Heart Rate", title_color = clue.PINK, title_scale=1, text_scale=3)
alarm_enable = True

# target heart rate for interval training
# Change this number depending on your max heart rate, usually figured
# as (220 - your age).
max_rate = 180

# PyLint can't find BLERadio for some reason so special case it here.
ble = adafruit_ble.BLERadio()    # pylint: disable=no-member
hr_connection = None

# Start with a fresh connection.
if ble.connected:
    print("SCAN")
    print("BLE")
    time.sleep(1)
    for connection in ble.connections:
        if HeartRateService in connection:
            connection.disconnect()
            break

while True:
    print("Scanning...")
    print("SCAN")
    print("BLE")
    time.sleep(1)
    clue_data[0].text = "BPM: ---"
    clue_data[0].color = ((30, 0, 0))
    clue_data[1].text = "Scanning..."
    clue_data[3].text = ""
    clue_data[1].color = ((130, 130, 0))
    clue_data.show()
```

©Adafruit Industries
for adv in ble.start_scan(ProvideServicesAdvertisement, timeout=5):
    if HeartRateService in adv.services:
        hr_connection = ble.connect(adv)
        #display_dots()
        print("....")
        time.sleep(2)
        print("Connected")
        break

    # Stop scanning whether or not we are connected.
    ble.stop_scan()
    print("Stopped scan")
    time.sleep(0.1)

if hr_connection and hr_connection.connected:
    print("Fetch connection")
    if DeviceInfoService in hr_connection:
        dis = hr_connection[DeviceInfoService]
        try:
            manufacturer = dis.manufacturer
        except AttributeError:
            manufacturer = "(Manufacturer Not specified)"
        try:
            model_number = dis.model_number
        except AttributeError:
            model_number = "(Model number not specified)"
        print("Device:", manufacturer, model_number)
    else:
        print("No device information")
    hr_service = hr_connection[HeartRateService]
    print("Location:", hr_service.location)

while hr_connection.connected:
    values = hr_service.measurement_values
    #print(values) # returns the full heart_rate data set
    if values:
        bpm = (values.heart_rate)
        if bpm is not 0:
            pct_target = (round(100*(bpm/max_rate)))
        if values.heart_rate is 0:
            print("----")
            clue_data[0].text = "BPM: ---"
            clue_data[0].color = (0, 0, 80)
            clue_data[1].text = "Target: --"
            clue_data[1].color = (0, 0, 80)
        else:
            clue_data[0].text = "BPM: {0:d}".format(bpm)
            clue_data[0].color = clue.RED
            clue_data[1].text = "Target: {0:d}%".format(pct_target)
            if pct_target < 90:
                alarm = False
                clue_data[1].color = clue.CYAN
            else:
                alarm = True
                clue_data[1].color = clue.RED

            clue_data[3].text = "Max HR: : {0:d}".format(max_rate)
            clue_data[3].color = clue.BLUE
            clue_data.show()

        if alarm and alarm_enable:
            clue.start_tone(2000)
        else:
            clue.stop_tone()

    # Inputs
    if clue.button_a:
if clue.touch_2:  # hold cap touch 2 for bigger change rate
    max_rate = max_rate - 10
else:
    max_rate = max_rate - 1
if clue.button_b:
    if clue.touch_2:
        max_rate = max_rate + 10
    else:
        max_rate = max_rate + 1

if clue.touch_0:
    alarm_enable = False
if clue.touch_1:
    alarm_enable = True

time.sleep(0.2)

---

Code Explainer

The code is doing a few fundamental things.

First, it loads the `time` and `board` libraries, as well as the necessary libraries to use BLE in general, and the `adafruit_ble_heart_rate` library in specific.

We also load the `adafruit_clue` library so we can take advantage of convenient commands that simplify using the CLUE's display.

The `clue_data` variable is created to instantiate the CLUE display object for simple text and titles.

We also set up the `BLERadio` so it can be used to communicate with the sensor.

```python
import time
from adafruit_clue import clue
import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.standard.device_info import DeviceInfoService
from adafruit_ble_heart_rate import HeartRateService

clue_data = clue.simple_text_display(title="Heart Rate", title_color = clue.PINK,
title_scale=1, text_scale=3)

alarm_enable = True

# target heart rate for interval training
# Change this number depending on your max heart rate, usually figured
# as (220 - your age).
max_rate = 180
```
Connection

Next, we scan for a BLE peripheral device advertising that it has the Heart Rate service.

We display the BPM, and "Scanning..." text, specifying their lines and colors on the CLUE TFT display.

When it is found, the CLUE will connect to it and then display the device name and other info. These are regular `print()` statements that show up in the REPL or other serial display, including the CLUE's display.

```python
print("Scanning...")
print("SCAN")
print("BLE")
time.sleep(1)
clue_data[0].text = "BPM: ---"
clue_data[0].color = ((30, 0, 0))
clue_data[1].text = "Scanning..."
clue_data[3].text = ""
clue_data[1].color = ((130, 130, 0))
clue_data.show()

for adv in ble.start_scan(ProvideServicesAdvertisement, timeout=5):
    if HeartRateService in adv.services:
        print("found a HeartRateService advertisement")
        hr_connection = ble.connect(adv)
        #display_dots()
        print("....")
        time.sleep(2)
        print("Connected")
        break

    # Stop scanning whether or not we are connected.
ble.stop_scan()
print("Stopped scan")
time.sleep(0.1)

if hr_connection and hr_connection.connected:
    print("Fetch connection")
    if DeviceInfoService in hr_connection:
        dis = hr_connection[DeviceInfoService]
        try:
            manufacturer = dis.manufacturer
        except AttributeError:
            manufacturer = "(Manufacturer Not specified)"
        try:
            model_number = dis.model_number
        except AttributeError:
            model_number = "(Model number not specified)"
        print("Device:", manufacturer, model_number)
        else:
            print("No device information")
        hr_service = hr_connection[HeartRateService]
        print("Location:", hr_service.location)
```

©Adafruit Industries
Measurements

Next we begin displaying the measurements. At first we show dashes while the data streams to the CLUE from the HRM, then switch to real data.

```python
while hr_connection.connected:
    values = hr_service.measurement_values
    #print(values)  # returns the full heart_rate data set
    if values:
        bpm = (values.heart_rate)
        if bpm is not 0:
            pct_target = (round(100*(bpm/max_rate)))
            if values.heart_rate is 0:
                print("----")
                clue_data[0].text = "BPM: ---"
                clue_data[0].color = ((80, 0, 0))
                clue_data[1].text = "Target: --"
                clue_data[1].color = ((0, 0, 80))
            else:
                clue_data[0].text = "BPM: {0:d}".format(bpm)
                clue_data[0].color = clue.RED
                clue_data[1].text = "Target: {0:d}%".format(pct_target)
        else:
            print("----")
            clue_data[0].text = "BPM: ---"
            clue_data[0].color = ((80, 0, 0))
            clue_data[1].text = "Target: --"
            clue_data[1].color = ((0, 0, 80))
    else:
        print("----")
        clue_data[0].text = "BPM: ---"
        clue_data[0].color = ((80, 0, 0))
        clue_data[1].text = "Target: --"
        clue_data[1].color = ((0, 0, 80))
```

Alarm

We'll do a calculation of the current BPM vs the Max HR and set off an alarm when it goes above 90%.

Well use the buttons and cap touch to adjust max heart rate and alarm on/off.

```python
if pct_target < 90:
    alarm = False
    clue_data[1].color = clue.CYAN
else:
    alarm = True
    clue_data[1].color = clue.RED
clue_data[3].text = "Max HR: : {0:d}".format(max_rate)
clue_data[3].color = clue.BLUE
clue_data.show()
if alarm and alarm_enable:
    clue.start_tone(2000)
else:
    clue.stop_tone()

# Inputs
if clue.button_a:
    if clue.touch_2:  # hold cap touch 2 for bigger change rate
        max_rate = max_rate -10
    else:
        max_rate = max_rate - 1
if clue.button_b:
    if clue.touch_2:
        max_rate = max_rate + 10
    else:
        max_rate = max_rate + 1
```
In Use

You can use the Heart Rate Zone Trainer any time you want to do some exercise, and be aware of your heart rate and the zone percentage you're in. Set it on a surface where you can see it easily, and pay attention to how long you are in different heart rate training zones.
Setup
Plug in the battery (or a USB power cable).

The display will show that it is scanning for a BLE HRM to connect to.

Turn on the heart rate monitor and strap it to your inner arm.

Next, the CLUE will connect and then show dashed lines for the BPM and Percent of target while the monitor begins streaming the data.

Next, your BPM, Target percentage, and max heart rate are displayed.
Adjustment
Since the CLUE has buttons and cap sense inputs, let's use them!

Press the B button to increase the max HR value one unit at a time.

The A button will decrease it.

To make larger changes, hold the 2 cap sense pad while using the buttons. This will increment the tens place.
Alarm!
If your heart rate goes above 90% of your max HR, the target percentage text will turn red and the alarm buzzer on the CLUE will sound!

Press cap touch 0 to turn off the alarm or touch 1 to turn it back on. Also, take a rest!