Cyber Flower: Digital Valentine

Created by Tony DiCola

https://learn.adafruit.com/cyber-flower-digital-valentine

Last updated on 2024-06-03 02:17:48 PM EDT

Table of Contents

Overview

Roses are red,

violets are blue,

this flower changes color,

to show its love for you!

Give your valentine a truly special gift with the cyber flower digital valentine project! This is a beautiful artificial flower that glows different colors using a Gemma M0 (http://adafru.it/3501) and its built-in DotStar LED. If you pick up the cyber flower it detects your touch and animates a glowing heart beat to show its love. This flower will never die--as long as its batteries are charged.

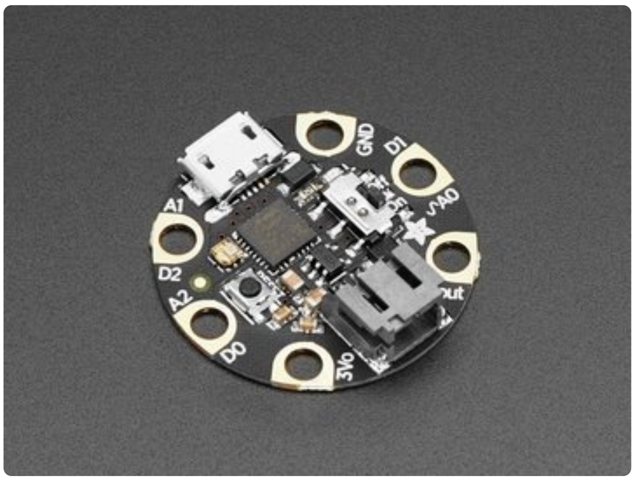
The cyber flower is a great demonstration of CircuitPython code. CircuitPython is a version MicroPython and the Python programming language which can run on boards like the Gemma M0. With simple Python code the cyber flower animates its DotStar LED. You can even modify the code with ease and no need to install software or complex tools.

Before you get started it will help to familiarize yourself with the Gemma M0 and CircuitPython (https://adafru.it/Cjs). Then continue on to learn about the parts and hardware needed to build this project.

Hardware

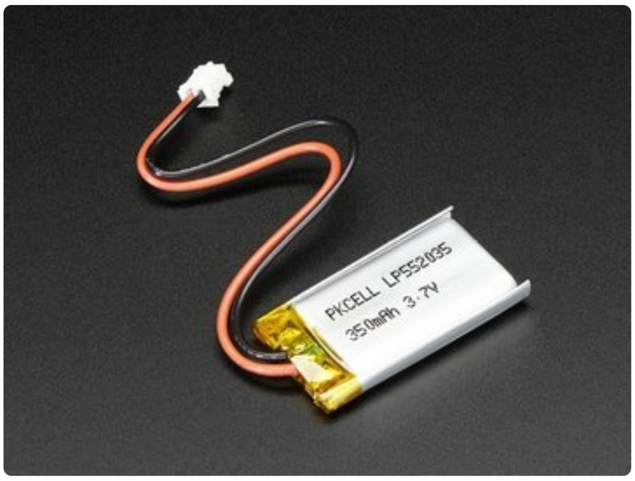
Parts

You'll need the following parts to build this project:

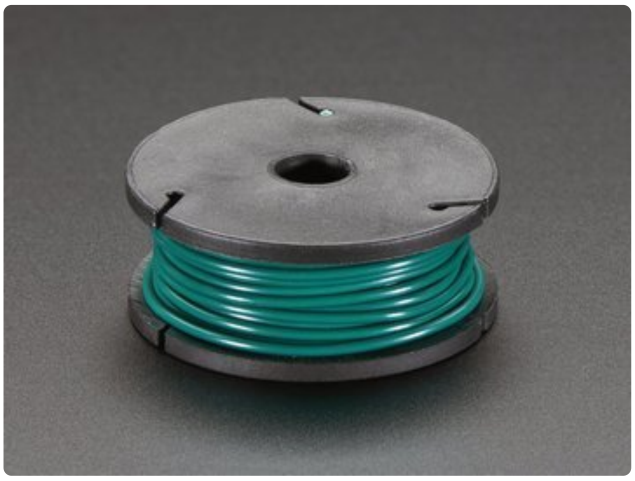


Gemma M0 (http://adafru.it/3501) - You need the Gemma M0 and not the older Gemma board for this project. The M0 version has a built-in DotStar LED that will light up the power by itself--no need to add any extra hardware! In addition the M0 version of Gemma can run CircuitPython code which is what this project is built to use!

An artificial flower - Check your local craft store for a wide assortment of artificial flowers. A white color flower is best because it will show the true colors of the DotStar LED, however you can experiment with other colors too. Look for a flower with enough room to hold the Gemma M0 board in its center, like a rose, tulip, or similar flower.



Battery - A small lipoly battery like a 350mAh version (http://adafru.it/3501) is easy to hide between the petals of the flower. Or grab a battery extension cable (http://adafru.it/1131) and larger battery (http://adafru.it/1578) or 3xAAA battery pack (http://adafru.it/727) for more power options.



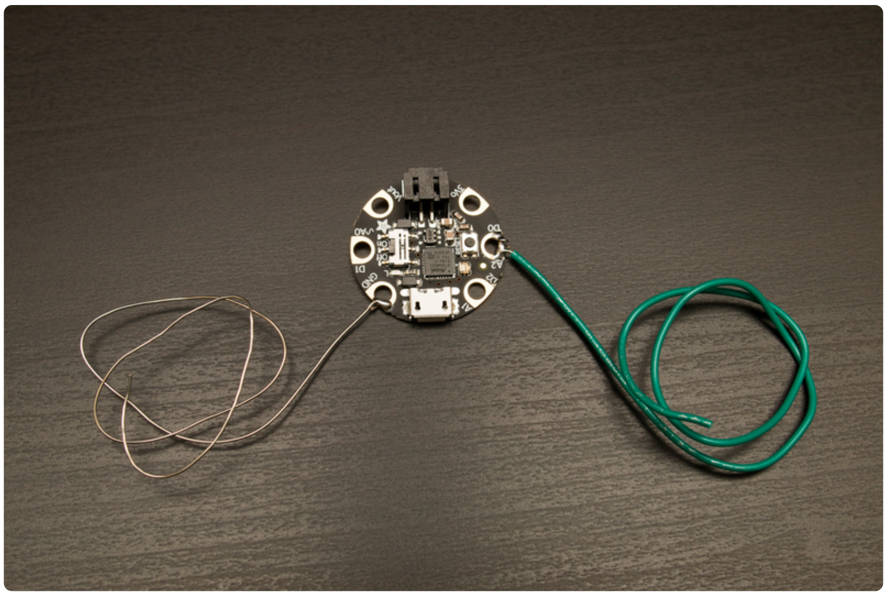
Solid core wire (http://adafru.it/2988) - Solid core wire is best for attaching to Gemma M0 and then wrapping around the stem of the flower. This wire is used to detect when someone touches the flower stem and will trigger a heart-beat animation on the LED. You don't have to attach this wire--if you don't then the touch sensing probably won't work but you'll still see nice color animations from the LED.

Soldering tools (http://adafru.it/136) - If you're attaching wires for capacitive sensing you'll need to solder them to pads on the Gemma M0 board. Check out the Adafruit Guide to Excellent Soldering (https://adafru.it/dxy) for more details on how to solder!

Assembly

Once you have the parts you can solder a few wires to the Gemma M0 board for capacitive touch sensing. This isn't absolutely necessary though and can be skipped if you aren't confident with soldering. Without the wires the flower won't detect touches but will still animate different colors. As an alternative you might be able to tape aluminum foil or copper tape to the pads of the Gemma M0 instead of soldering, but it might not be as robust or strong as soldering wires.

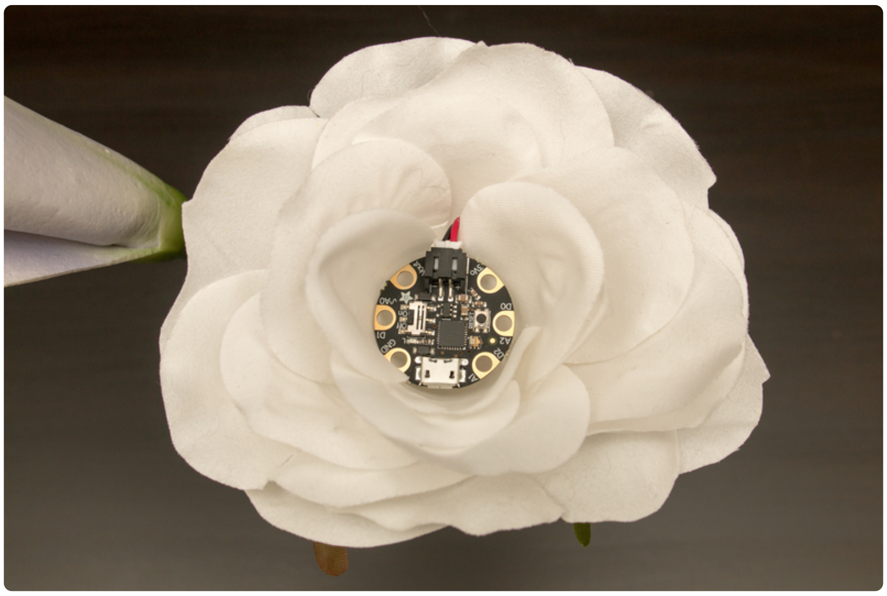
If you're soldering wires it's best to strip the insulation entirely off one length of wire and solder it to the board's ground pad. This will greatly help the capacitive touch sensing when running off of battery power because it depends on the person touching the flower being grounded to the battery. Then solder an insulated wire (i.e. **don't** strip off the insulation) to the D0 pad of the Gemma M0. Your Gemma M0 should have two wires soldered to it as shown below:



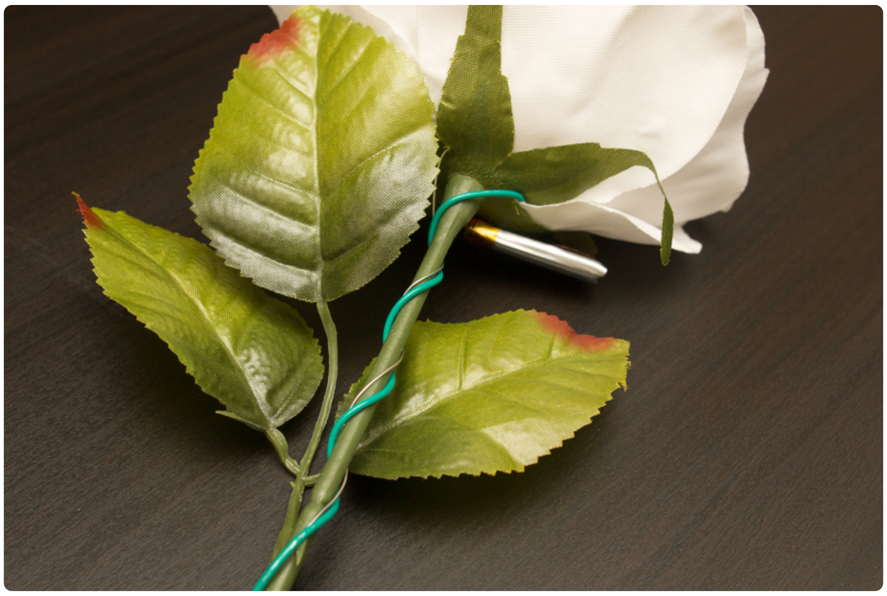
- **Bare / un-insulated wire soldered** to **Gemma M0 GND / ground**
- **Regular / insulated wire soldered** to **Gemma M0 D0.**

Next cut out the pistil from the center of your flower if you need to make room for the Gemma M0 board. Side cutters work well for this task:

Then nestle the Gemma M0 inside the center of the flower. If the flower is small enough it might hold the Gemma M0 in place without any assistance. Otherwise you might need to add tape, hot glue, or even sticky tack to hold the board in place:



Snake the wires through the petals and down the stem of the flower. If you're using wires for capacitive touch sensing wrap them around the stem in a double helix pattern. This way when someone touches the stem they'll touch both the uninsulated ground wire and the insulated capacitive touch sensing wire:



Be careful with the uninsulated ground wire to ensure it does not touch any other pin on the Gemma M0. If the ground wire touches the 3.3V or Vout pins it can short out the board and potentially damage it, the battery, or a computer connected to the Gemma M0! Take care to carefully route and move the ground wire!

Connect the battery and you're all set! The flower is ready to have code loaded onto it that will drive the DotStar LED!

Software

The software for this project is written in CircuitPython so it's easy to run and modify, just write the code straight to the board's USB drive! If you're new to CircuitPython and Gemma M0 first check out the Gemma M0 CircuitPython guide pages (https://adafru.it/Cjs). These will explain what CircuitPython is and how to load it onto your Gemma M0 board.

Before you continue make sure your Gemma M0 is running the latest version of CircuitPython by updating its UF2 firmware (https://adafru.it/zAl). Also be sure you can see the board's **CIRCUITPY** drive when you connect it to your computer with a USB cable. Once you see the **CIRCUITPY** drive you're all set and ready to load the project code.

Now you need to install two dependencies from the CircuitPython bundle. Like the bundle install page mentions (https://adafru.it/ABU) for non-express boards like Gemma M0 this means you have to download the bundle zip file, open it on your computer, and copy the following files and folders to your board's **CIRCUITPY** drive:

- **adafruit_dotstar.mpy**
- **adafruit_fancyled**

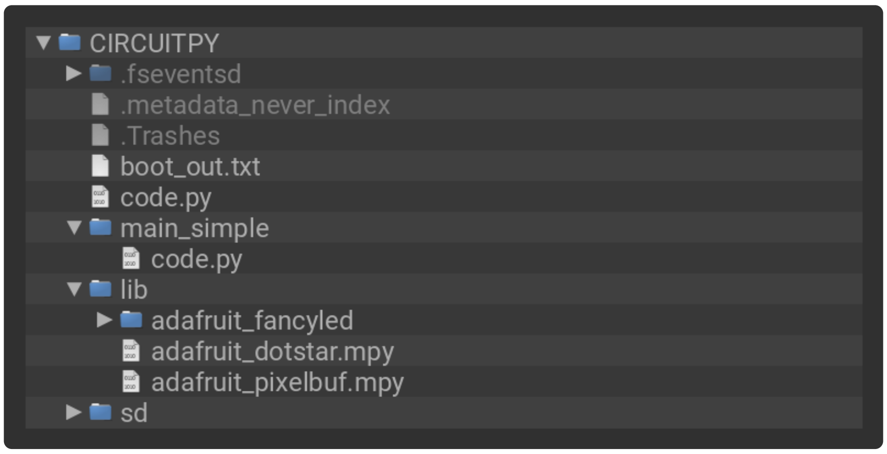
Be sure you've copied both these files and folders (and all the files inside the folders) to your **CIRCUITPY** drive before continuing!

```
# SPDX-FileCopyrightText: 2018 Tony DiCola for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Cyber Flower: Digital Valentine
#
# 'Roses are red,
#  Violets are blue,
#  This flower changes color,
#  To show its love for you.'
#
# Load this on a Gemma M0 running CircuitPython and it will smoothly animate
# the DotStar LED between different color hues.  Touch the D0 pad and it will
# cause the pixel to pulse like a heart beat.  You might need to also attach a
# wire to the ground pin to ensure capacitive touch sensing can work on battery
# power.  For example strip the insulation from a wire and solder it to ground,
# then solder a wire (with the insulation still attached) to D0, and wrap
# both wires around the stem of a flower like a double-helix.  When you touch
# the wires you'll ground yourself (touching the bare ground wire) and cause
# enough capacitance in the D0 wire (even though it's still insulated) to
# trigger the heartbeat.  Or just leave D0 unconnected to have a nicely
# animated lit-up flower!
#
# Note that on power-up the flower will wait about 5 seconds before turning on
# the LED.  During this time the board's red LED will flash and this is an
# indication that it's waiting to power on.  Place the flower down so nothing
# is touching it and then pick it up again after the DotStar LED starts
# animating.  This will ensure the capacitive touch sensing isn't accidentally
# calibrated with your body touching it (making it less accurate).
#
# Also note this depends on two external modules to be loaded on the Gemma M0:
#  - Adafruit CircuitPython DotStar:
# https://github.com/adafruit/Adafruit_CircuitPython_DotStar
#  - Adafruit CircuitPython FancyLED:
# https://github.com/adafruit/Adafruit_CircuitPython_FancyLED
#
# You _must_ have both adafruit_dotstar.mpy and the adafruit_fancyled folder
# and files within it on your board for this code to work!  If you run into
# trouble or can't get the dependencies see the main_simple.py code as an
# alternative that has no dependencies but slightly more complex code.
#
# Author: Tony DiCola
# License: MIT License
import math
import time

import adafruit_dotstar
import adafruit_fancyled.adafruit_fancyled as fancy
import board
import digitalio
import touchio

# Variables that control the code.  Try changing these to modify speed, color,
# etc.
START_DELAY = 5.0  # How many seconds to wait after power up before
# jumping into the animation and initializing the
# touch input.  This gives you time to take move your
# fingers off the flower so the capacitive touch
# sensing is better calibrated.  During the delay
# the small red LED on the board will flash.

TOUCH_PIN = board.D0  # The board pin to listen for touches and trigger the
# heart beat animation.  You can change this to any
# other pin like board.D2 or board.D1.  Make sure not
# to touch this pin as the board powers on or the
# capacitive sensing will get confused (just reset
# the board and try again).

BRIGHTNESS = 1.0  # The brightness of the colors.  Set this to a value
# anywhere within 0 and 1.0, where 1.0 is full bright.
# For example 0.5 would be half brightness.

RAINBOW_PERIOD_S = 18.0  # How many seconds it takes for the default rainbow
# cycle animation to perform a full cycle.  Increase
# this to slow down the animation or decrease to speed
# it up.

HEARTBEAT_BPM = 60.0  # Heartbeat animation beats per minute.  Increase to
# speed up the heartbeat, and decrease to slow down.

HEARTBEAT_HUE = 300.0  # The color hue to use when animating the heartbeat
# animation.  Pick a value in the range of 0 to 359
# degrees, see the hue spectrum here:
#   https://en.wikipedia.org/wiki/Hue
# A value of 300 is a nice pink color.

# First initialize the DotStar LED and turn it off.
dotstar = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1)
dotstar[0] = (0, 0, 0)

# Also make sure the on-board red LED is turned off.
red_led = digitalio.DigitalInOut(board.L)
red_led.switch_to_output(value=False)
```

```
# Wait the startup delay period while flashing the red LED.  This gives time
# to move your hand away from the flower/stem so the capacitive touch sensing
# is initialized and calibrated with a good non-touch starting state.
start = time.monotonic()
while time.monotonic() - start <= START_DELAY:
    # Blink the red LED on and off every half second.
    red_led.value = True
    time.sleep(0.5)
    red_led.value = False
    time.sleep(0.5)

# Setup the touch input.
touch = touchio.TouchIn(TOUCH_PIN)

# Convert periods to frequencies that are used later in animations.
rainbow_freq = 1.0 / RAINBOW_PERIOD_S

# Calculcate periods and values used by the heartbeat animation.
beat_period = 60.0 / HEARTBEAT_BPM
beat_quarter_period = beat_period / 4.0  # Quarter period controls the speed of
# the heartbeat drop-off (using an
# exponential decay function).
beat_phase = beat_period / 5.0  # Phase controls how long in-between


# the two parts of the heart beat
# (the 'ba-boom' of the beat).

# Handy function for linear interpolation of a value.  Pass in a value
# x that's within the range x0...x1 and a range y0...y1 to get an output value
# y that's proportionally within y0...y1 based on x within x0...x1.  Handy for
# transforming a value in one range to a value in another (like Arduino's map
# function).

# pylint: disable=redefined-outer-name
def lerp(x, x0, x1, y0, y1):
    return y0 + (x - x0) * ((y1 - y0) / (x1 - x0))


# Main loop below will run forever:
while True:
    # Get the current time at the start of the animation update.
    current = time.monotonic()
    # Now check if the touch input is being touched and choose a different
    # animation to run, either a rainbow cycle or heartbeat.
    if touch.value:
        # The touch input is being touched, so figure out the color using
        # a heartbeat animation.
        # This works using exponential decay of the color value (brightness)
        # over time:
        #   https://en.wikipedia.org/wiki/Exponential_decay
        # A heart beat is made of two sub-beats (the 'ba-boom') so two decay
        # functions are calculated using the same fall-off period but slightly
        # out of phase so one occurs a little bit after the other.
        t0 = current % beat_period
        t1 = (current + beat_phase) % beat_period
        x0 = math.pow(math.e, -t0 / beat_quarter_period)
        x1 = math.pow(math.e, -t1 / beat_quarter_period)
        # After calculating both exponential decay values pick the biggest one
        # as the secondary one will occur after the first.  Scale each by
        # the global brightness and then convert to RGB color using the fixed
        # hue but modulating the color value (brightness).  Luckily the result
        # of the exponential decay is a value that goes from 1.0 to 0.0 just
        # like we expect for full bright to zero brightness with HSV color
        # (i.e. no interpolation is necessary).
        val = max(x0, x1) * BRIGHTNESS
        color = fancy.gamma_adjust(fancy.CHSV(HEARTBEAT_HUE / 359.0, 1.0, val))
        dotstar[0] = color.pack()
    else:
        # The touch input is not being touched (touch.value is False) so
        # compute the hue with a smooth cycle over time.
        # First use the sine function to smoothly generate a value that goes
        # from -1.0 to 1.0 at a certain frequency to match the rainbow period.
        x = math.sin(2.0 * math.pi * rainbow_freq * current)
        # Then compute the hue by converting the sine wave value from something
        # that goes from -1.0 to 1.0 to instead go from 0 to 1.0 hue.
        hue = lerp(x, -1.0, 1.0, 0.0, 1.0)
        # Finally update the DotStar LED by converting the HSV color at the
        # specified hue to a RGB color the LED understands.
        color = fancy.gamma_adjust(fancy.CHSV(hue, 1.0, BRIGHTNESS))
        dotstar[0] = color.pack()
```

Copy this **code.py** to your Gemma M0's **CIRCUITPY** drive. Make sure to **overwrite** any main.py that's already on the board!

At this point be sure your **CIRCUITPY** drive has the cyber flower **code.py, adafruit_dotstar.mpy,** and **adafruit_fancyled** files and folder on it. You can delete any other files and folders on the drive (boot_out.txt is created by the board with debug data, you can ignore it):



```
▼ CIRCUITPY
    ► .fseventsd
      .metadata_never_index
      .Trashes
      boot_out.txt
      code.py
    ▼ main_simple
        code.py
    ▼ lib
        ► adafruit_fancyled
          adafruit_dotstar.mpy
          adafruit_pixelbuf.mpy
    ► sd
```

The flower code might start running as soon as you copy the file to your board. However just to be sure you can eject the **CIRCUITPY** drive from your computer, then press the board's reset button and the flower code should start running.

When the flower code first starts it will wait 5 seconds and flash the Gemma M0's tiny red LED on and off. This is a short delay that gives you time to stop touching the board and the D0 and ground wires. For the capacitive touch sensing to work you want the board to be 'calibrated' with nothing touching it, so during the initial startup delay put the flower down so nothing is touching it. Then after 5 seconds the red LED will stop flashing and the DotStar LED will turn on to animate different colors. At this point the flower code is running--try picking up the flower or touching the D0 wire to see the heart beat animation!

Note that if the flower is only powered by a battery (i.e. not connected to your computer) you might need to touch both the ground wire and D0 wire to trigger the heart beat animation. This is why assembling the flower with a bare / uninsulated wire connected to ground is best, it allows someone to touch the flower and simultaneously ground themselves and trigger the capacitive sensing.

Also be aware if you have trouble getting all the dependencies copied to your board and working, there's a variant that does not have any dependencies called main_simple.py (https://adafru.it/Cjw) You can download main_simple.py (https://adafru.it/Cjw) and rename it to **code.py**, then copy to your Gemma M0's **CIRCUITPY** drive. There are no other dependencies needed to use this version of the code (but it's a little harder to read and follow compared to the main code above).

Modify The Flower Code

Out of the box the flower code should work great without any modifications. However since it's all simple Python code you can easily change it with any text editor. The code is written with a few variables at the top that you can change to modify how the flower works. First make sure you're familiar with editing and running code on a CircuitPython board (https://adafru.it/Cjx).

If you open the flower **code.py** on the **CIRCUITPY** drive, notice these values at the top:

```
# Variables that control the code.  Try changing these to modify speed, color,
# etc.
START_DELAY = 5.0        # How many seconds to wait after power up before
                         # jumping into the animation and initializing the
                         # touch input.  This gives you time to take move your
                         # fingers off the flower so the capacitive touch
                         # sensing is better calibrated.  During the delay
                         # the small red LED on the board will flash.

TOUCH_PIN = board.D0     # The board pin to listen for touches and trigger the
                         # heart beat animation.  You can change this to any
                         # other pin like board.D2 or board.D1.  Make sure not
                         # to touch this pin as the board powers on or the
                         # capacitive sensing will get confused (just reset
                         # the board and try again).

BRIGHTNESS = 1.0         # The brightness of the colors.  Set this to a value
                         # anywhere within 0 and 1.0, where 1.0 is full bright.
                         # For example 0.5 would be half brightness.

RAINBOW_PERIOD_S = 18.0 # How many seconds it takes for the default rainbow
                         # cycle animation to perform a full cycle.  Increase
                         # this to slow down the animation or decrease to speed
                         # it up.

HEARTBEAT_BPM = 60.0     # Heartbeat animation beats per minute.  Increase to
                         # speed up the heartbeat, and decrease to slow down.

HEARTBEAT_HUE = 300.0    # The color hue to use when animating the heartbeat
                         # animation.  Pick a value in the range of 0 to 359
                         # degrees, see the hue spectrum here:
                         #   https://en.wikipedia.org/wiki/Hue
```

On the right you can see comments (text after a # character) which describe the values. Try changing one of these, like the **HEARTBEAT_HUE** value at the bottom. If you change **HEARTBEAT_HUE** from **300.0** to **180.0** it will change the color of the heart beat animation to a cyan/blue color. Modify the value and save the **code.py** file on your **CIRCUITPY** drive. The board should see the changed code and start running it again (remember it has a 5 second startup delay where it blinks the red LED). If you touch the D0 pad now the heart beat should be cyan/blue--cool!

Experiment with changing other values like **HEARTBEAT_BPM** to change how fast the heart beat animation runs, or **RAINBOW_PERIOD_S** to change how long it takes for the DotStar to cycle through a rainbow of colors. Each time you change a value just save the **code.py** on the board again and it should start running with the new code!

That's all there is to the cyber flower digital valentine project!