Canary Nightlight
Created by Kattni Rembor

https://learn.adafruit.com/canary-nightlight

Last updated on 2023-10-19 04:58:32 PM EDT
# Table of Contents

## Overview
- Parts
- Powering the Canary from an Outlet

## Circuit Diagram
- Soldering
- Pins to Pins

## CAD Files
- CAD Parts List
- Build Volume
- Translucent Filament
- CAD Assembly
- Design Source Files

## Install CircuitPython
- CircuitPython Quickstart

## Get Started with Adafruit IO
- I have an Adafruit.com Account already
- Create an Adafruit Account (for Adafruit IO)

## Code the Canary
- Copy the Code and Libraries
- The settings.toml File
- Customize Your Canary
- Network-Down Detection
- Sleep and Wake Time Canary Colors
- Sleep and Wake Time
- Canary Brightness
- Time Check Interval
- Successful Ping Interval
- Blink Color
- Consecutive Ping Fail to Blink
- Network Down Reload Time
- Ping IP Address

## Code Walkthrough
- Modules and Libraries
- Customization
- Value Checks
- Hardware and Code Set Up
- NeoPixel Set Up
- Helper Functions
- Connect to WiFi
- Set Up Ping
- Connect to Adafruit IO and Retrieve the Time
- Variables
- The Loop
- Update the Current Time
- The try and except Block
- Time to Light Up LEDs
• If Network Down Detection is Disabled
• If Network Down Detection is Enabled

Assembly

• Headers for QT Py & BFF
• Install Headers to QT Py
• Secure Headers to QT Py
• QT Py & BFF Orientation
• Insert QT Py & BFF
• Trim Header Pins
• Short Header Pins
• Secure NeoPixel BFF to QT Py
• Soldered QT Py & BFF
• Bottom Cover
• Install QT Py & BFF
• Press Fit QT Py & BFF
• Installed PCBs
• Install Bird
• Bird Installation
• Final Build
Overview

Everyone needs a blue canary in the outlet by the light switch, ready to watch over them. How can you find a little glowing friend for yourself? Well, get the birdhouse in your soul ready! This canary night light is ready to move in and light it up.

This guide shows you how to build your own canary nightlight. The build involves a QT Py ESP32-S3, paired with a 5x5 NeoPixel Grid BFF, fitted into a beautiful 3D printed bird. The circuit diagram helps ensure that you're pairing the two boards properly. The CAD files are provided, along with info about what filaments work best for this print. After installing CircuitPython on your QT Py, you'll load the code, and learn how it works. Finally, you'll assemble the canary nightlight.

Blue light promotes wakefulness, and can be super helpful when it's time for you to be awake or you're trying to concentrate. The nightlight lights up blue during the day by default to help you with both.
On the other hand, bright blue light can often hinder you when you're trying to wind down or sleep. To avoid that, this canary is designed to turn red and dim during the evening and night to help you slumber.

What would a canary nightlight be if it wasn't watching over you? Well, this light also includes a feature that lets you know if your internet connection goes down. When the connection is down for a period of time, the canary will begin blinking red to notify you. It keeps blinking until the internet is back up. Wonder no more if it's you or the internet! This canary has your back.
The code for this project is super customisable so you can tweak it to your exact wants and needs. You can update the colors, brightness, timing, and more. The guide goes through everything you can change so you know your options.

![Image of a bird with LED lights](image)

**Parts**

This is a relatively simple build. To follow along with this guide, in addition to a 3D printer to print your canary (if you don't have a 3D printer, you can send the files to a printing company for manufacturing), you'll need the following parts.

Adafruit QT Py ESP32-S3 WiFi Dev Board with STEMMA QT
The ESP32-S3 has arrived in QT Py format - and what a great way to get started with this powerful new chip from Espressif! With dual 240 MHz cores, WiFi and BLE support, and native...
[https://www.adafruit.com/product/5426](https://www.adafruit.com/product/5426)
Adafruit 5x5 NeoPixel Grid BFF Add-On for QT Py and Xiao
Our QT Py boards are a great way to make very small microcontroller projects that pack a ton of power - and now we have a way for you to quickly add a glittering grid of 25 RGB...
https://www.adafruit.com/product/5646

Pink and Purple Woven USB A to USB C Cable - 1 meter long
This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also made for USB C for our modernized breakout boards, Feathers, and...
https://www.adafruit.com/product/5153

Powering the Canary from an Outlet
You'll need a USB wall plug if you plan to power it from an outlet, regardless of whether or not it's by a light switch. To properly power your canary, your wall plug needs to provide 5V, and at least 2A.

If you have a 5V USB-C iPhone power adapter sitting around, it is the best option for plugging in the canary. To complete this setup, you'll need to get a USB Type C plug to plug adapter (1) to go with it.
Alternatively, you can use any wall adapter that provides the required 5V 2A. If your wall plug is USB-A, then you'll need a USB-A to USB-C plug adapter. If your wall plug is USB-C, you'll need the USB Type C plug to plug adapter. Depending on the orientation your wall plug, you may also require a right angle USB type C adapter to complete your set up.

Attribution

This was inspired by one of Kattni's favorite songs from They Might Be Giants: Birdhouse In Your Soul. Check it out, and while you're at it, keep the nightlight on inside the birdhouse in your soul!

Circuit Diagram

The 5x5 NeoPixel Grid BFF is designed to be soldered directly to the QT Py ESP32-S3. No wires needed! All you need is two strips of header pins to make a tiny board sandwich.

Soldering

The first step is soldering the 5x5 NeoPixel Grid BFF to the QT Py ESP32-S3.

The BFF is designed to be soldered back-to-back with the QT Py.

The BFF must be oriented in the proper direction to work. The end of the BFF with all of the pads and jumpers is the side that should be closest to the USB connector on the QT Py. There are instructions on the back of the BFF that indicate the proper orientation. Follow them to be sure!

That's all there is to it! You’re ready to start coding your nightlight!
Pins to Pins

Even though there's no wiring needed, you may wonder what pins are being used by the BFF and how they coincide with the QT Py pins.

The diagram below is included only to show what pins the BFF is using and how they correspond to the QT Py pins. You do not need to include wires in this project!

BFF GND to QT Py GND
BFF 5V to QT Py 5V
BFF data pin to QT Py A3

CAD Files

CAD Parts List
STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material using PLA filament. Original design source may be downloaded using the links below.

- sphere-top.STL
- sphere-bottom.STL
- sphere-cover.STL

Download STLs.zip
Download CAD source
Build Volume
The parts require a 3D printer with a minimum build volume.

55mm (X) x 110mm (Y) x 66mm (Z)

Translucent Filament
Choose your desired color of filament. A white opaque colored filament will offer a soft-even diffusion while a "clear" transparent filament will give sharp and direct diffusion.
CAD Assembly
The QT Py and NeoPixel BFF boards are soldered together and press fitted into the bottom cover. The bird snap fits over the bottom cover with the notches lined up for the USB-C port.

These steps are shown on the Assembly page of this guide.

Design Source Files
The project assembly was designed in Fusion 360. This can be downloaded in different formats like STEP, STL and more. Electronic components like Adafruit's boards, displays, connectors and more can be downloaded from the Adafruit CAD parts GitHub Repo.

Install CircuitPython
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.

CircuitPython Quickstart
Follow this step-by-step to quickly get CircuitPython running on your board.

Download the latest version of CircuitPython for this board via circuitpython.org
Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

Double-click the reset button (highlighted in red above), and you will see the RGB status LED(s) turn green (highlighted in green above). If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

For this board, tap reset and wait for the LED to turn purple, and as soon as it turns purple, tap reset again. The second tap needs to happen while the LED is still purple.

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!
A lot of people end up using charge-only USB cables and it is very frustrating! Make sure you have a USB cable you know is good for data sync.

You will see a new disk drive appear called QTPYS3BOOT.

Drag the adafruit_circuitpython_etc.uf2 file to QTPYS3BOOT.

The BOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

That's it!

---

Get Started with Adafruit IO

Adafruit IO is integrated with your adafruit.com account so you don't need to create yet another online account! You need an Adafruit account to use Adafruit IO because
we want to make sure the data you upload is available to only you (unless you decide to publish your data).

I have an Adafruit.com Account already

If you already have an Adafruit account, then you already have access to Adafruit IO. It doesn't matter how you signed up, your account will make all three available.

To access Adafruit IO, simply visit https://io.adafruit.com to start streaming, logging, and interacting with your data.

Create an Adafruit Account (for Adafruit IO)

An Adafruit account makes Adafruit content and services available to you in one place. Your account provides access to the Adafruit shop, the Adafruit Learning System, and Adafruit IO. This means only one account, one username, and one password are necessary to engage with the content and services that Adafruit offers.

If you do not have an Adafruit account, signing up for a new Adafruit account only takes a couple of steps.


Click the Sign Up button under the "Need An Adafruit Account?" title, below the Sign In section.
This will take you to the Sign Up page.

Fill in the requested information, and click the Create Account button.

**SIGN UP**

The best way to shop with Adafruit is to create an account which allows you to shop faster, track the status of your current orders, review your previous orders and take advantage of our other member benefits.

This takes you to your Adafruit Account home page. From here, you can access all the features of your account.
You can also access the Adafruit content and services right from this page. Along the top of the page, you'll see a series of links beginning with "Shop". To access any of these, simply click the link.

For example, to begin working with Adafruit IO, click the IO link to the right of the rest of the links. This is the same for the other links as well.

That's all there is to creating a new Adafruit account, and navigating to Adafruit IO.

Code the Canary

The next step is to prepare the software on your QT Py ESP32-S3.

Copy the Code and Libraries

You need to copy the code and all of the necessary libraries to your QT Py.

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, and copy the entire lib folder and the code.py file to your CIRCUITPY drive.

Your CIRCUITPY drive should contain the following files and folders, as well as your settings.toml file.

```
# SPDX-FileCopyrightText: 2023 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT

===
CircuitPython Canary Day and Night Light with Optional Network-Down Detection

This project uses the QT Py ESP32-S3 with the NeoPixel 5x5 LED Grid BFF, along with a 3D printed bird. The LEDs light up different colors based on the time.

In the event that the internet connection fails, it will begin blinking red to notify you. If the initial test ping fails, and the subsequent pings fail over 30 times, the board will reset. Otherwise, the blinking will continue until the connection is back up. This feature is enabled by default. It can easily be disabled at the beginning of the code.

```python
import os
import ssl
import time
import ipaddress
import supervisor
import board
import wifi
import microcontroller
import socketpool
import adafruit_requests
import neopixel
from adafruit_io.adafruit_io import IO_HTTP

# ============ CUSTOMISATIONS ============
# Network-down detection enable or disable.
# By default, the network-down detection code, and the code that blinks when the network is down, are both enabled. If you wish to disable this feature,
# including the blinking, update this to False.
NETWORK_DOWN_DETECTION = True

# The basic canary colors.
# Red light at night is more conducive to sleep. Blue light in the morning is more conducive to waking up. The sleep color defaults to red to promote sleep. The wake color defaults to blue to promote wakefulness.
```
SLEEP_COLOR = (255, 0, 0)  # Red
WAKE_COLOR = (0, 0, 255)  # Blue

# Sleep time.
# This is the hour in 24-hour time at which the light should change to the
# desired color for the time you intend to sleep.
# Must be an integer between 0 and 23. Defaults to 20 (8pm).
SLEEP_TIME = 20

# Wake time.
# This is the hour in 24-hour time at which the light should change to the
# desired color for the time you intend to be awake.
# Must be an integer between 0 and 23. Defaults to 6 (6am).
WAKE_TIME = 6

# Canary brightness customisation.
# Brightness must be a float or integer between 0.0 and 1.0, where 0.0 is off, and
# 1.0 is max.
# This is the brightness of the canary during sleep time. It defaults to 0.2, or
# "20%".
# Increase or decrease this value to change the brightness.
SLEEP_BRIGHTNESS = 0.2
# This is the brightness of the canary during wake time. It defaults to 0.7, or
# "70%".
# Increase or decrease this value to change the brightness.
WAKE_BRIGHTNESS = 0.7

# Time check interval.
# This sets the time interval at which the code checks Adafruit IO for the current
time.
# This is included because Adafruit IO has rate limiting. It ensures that you do not
# hit the rate limit, and the time check does not get throttled.
# Defaults to 300 seconds (5 minutes). Must be an integer equal to or greater than
# 300.
# Increase this value to increase the time check interval. Do not decrease this
# value!
TIME_CHECK_INTERVAL = 300

if TIME_CHECK_INTERVAL < 300 or isinstance(TIME_CHECK_INTERVAL, float):
    # If is below the minimum or a float, raise this error and stop the code.
    raise ValueError("TIME_CHECK_INTERVAL must be a integer, and greater than 300!")

# Ping interval while ping is successful.
# This is the interval at which the code will send a ping while the network is up
# and the ping is successful. If for any reason you would prefer to slow down the ping
# interval, this value
# can be updated. Defaults to 1 second. Must be an integer equal to or greater than
# 1. Increase
# this value to increase the ping interval time. Do not decrease this value!
UP_PING_INTERVAL = 1

if UP_PING_INTERVAL < 1 or isinstance(UP_PING_INTERVAL, float):
    # If is below the minimum or a float, raise this error and stop the code.
    raise ValueError("UP_PING_INTERVAL must be a integer, and greater than 1!")

# The blink color.
# This is the color that the canary will blink to notify you that the network is
down.
# Defaults to red.
BLINK_COLOR = (255, 0, 0)

# Consecutive ping fail to blink.
# This value is the number of times ping will consecutively fail before the canary
begins blinking.
# If the blinking is happening too often, or if the network is often flaky, this
value can be increased to extend the number of failures it takes to begin blinking. 
# Defaults to 10. Must be an integer greater than 1.
CONSECUTIVE_PING_FAIL_TO_BLINK = 10

# The amount of time in seconds that needs to pass while the network is down AND 
# NETWORK DOWN DETECTION is DISABLED before the board resets to try again. 
# Defaults to 900 seconds, or 20 minutes. Must be an integer. Increase or decrease 
# this value to alter how long the network should be down in this specific case 
# before the board resets.
NETWORK_DOWN_RELOAD_TIME = 900

# IP address. 
# This is the IP address used to ping to verify that network connectivity is still present. 
# To switch to a different IP, update the following. Must be a valid IPV4 address as a 
# string (in quotes). Defaults to one of the OpenDNS IPs.
PING_IP = "208.67.222.222"

# ============ HARDWARE AND CODE SET UP ============
# Instantiate the NeoPixel object. 

# Create helper functions
def reload_on_error(delay, error_content=None, reload_type="reload"): 
    """Reset the board when an error is encountered."
    :param float delay: The delay in seconds before the board should reset.
    :param Exception error_content: The error encountered. Used to print the error before reset.
    :param str reload_type: The type of reload desired. Defaults to "reload", which invokes `supervisor.reload()` to soft-reload the board. To hard reset the board, set this to "reset", which invokes `microcontroller.reset()`.
    """
    if str(reload_type).lower().strip() not in ["reload", "reset"]: 
        raise ValueError("Invalid reload type:", reload_type)
    if error_content: 
        print("Error:
        " + str(error_content))
    if delay: 
        print("f" + " \n" + " microcontroller in {delay} seconds."
    time.sleep(delay)
    if reload_type == "reload": 
        supervisor.reload()
    if reload_type == "reset": 
        microcontroller.reset()

def color_time(current_hour):
    """Verifies what color the LEDs should be based on the time."
    :param current_hour: Provide a time, hour only. The `tm_hour` part of the `io.receive_time()` object is acceptable here.
    """
    if WAKE_TIME < SLEEP_TIME: 
        if WAKE_TIME <= current_hour < SLEEP_TIME: 
            pixels.brightness = WAKE_BRIGHTNESS 
            return WAKE_COLOR 
            pixels.brightness = SLEEP_BRIGHTNESS 
            return SLEEP_COLOR
if SLEEP_TIME <= current_hour < WAKE_TIME:
    pixels.brightness = SLEEP_BRIGHTNESS
    return SLEEP_COLOR
pixels.brightness = WAKE_BRIGHTNESS
    return WAKE_COLOR

def blink(color):
    """Blink the NeoPixel LEDs a specific color.
    :param tuple color: The color the LEDs will blink.
    """
    if color_time(sundial.tm_hour) == SLEEP_COLOR:
        pixels.brightness = SLEEP_BRIGHTNESS
    else:
        pixels.brightness = WAKE_BRIGHTNESS
    pixels.fill(color)
    time.sleep(0.5)
    pixels.fill((0, 0, 0))
    time.sleep(0.5)

# Connect to WiFi. This process can fail for various reasons. It is included in a try/except block to ensure the project continues running when unattended.
try:
    wifi.radio.connect(os.getenv("wifi_ssid"), os.getenv("wifi_password"))
    pool = socketpool.SocketPool(wifi.radio)
    requests = adafruit_requests.Session(pool, ssl.create_default_context())
except Exception as error:  # pylint: disable=broad-except
    # The exceptions raised by the 'wifi' module are not always clear. If you're receiving errors, check your SSID and password before continuing.
    print("Wifi connection failed.")
    reload_on_error(5, error)

# Set up IP address to use for pinging, as defined above.
ip_address = ipaddress.IPv4Address(PING_IP)
# Set up ping, and send initial ping.
wifi_ping = wifi.radio.ping(ip=ip_address)
# If the initial ping is unsuccessful, print the message.
if wifi_ping is None:
    print("Set up test-ping failed.")
    initial_ping = False
else:
    # Otherwise, print this message.
    print("Set up test-ping successful.")
    initial_ping = True

# Set up Adafruit IO. This will provide the current time through `io.receive_time()`.
io = IO_HTTP(os.getenv("aio_username"), os.getenv("aio_key"), requests)

# Retrieve the time on startup. This is included to verify that the Adafruit IO set up was successful. This process can fail for various reasons. It is included in a try/except block to ensure the project continues to run when unattended.
try:
    sundial = io.receive_time()  # Create the sundial variable to keep the time.
except Exception as error:  # pylint: disable=broad-except
    # If the time retrieval fails with an error, print the message.
    print("Adafruit IO set up and/or time retrieval failed.")
    # Then wait 5 seconds, and soft reload the board.
    reload_on_error(5, error)

# Initialise various time tracking variables.
check_time = 0
network_down_time = time.time()
ping_time = 0
ping_fail_time = time.time()

# Initialise network check variable.
network_check = 1

# Initialise ping fail count tracking.
ping_fail_count = 0

# ============ LOOP ============
while True:
    # Resets current_time to the current second every time through the loop.
    current_time = time.time()

    # WiFi and IO connections can fail arbitrarily. The bulk of the loop is
    # included in a
    # try/except block to ensure the project will continue to run unattended if any
    # failures do occur.
    try:
        # If this is the first run of the code or the time check interval has
        passed, continue.
        if not check_time or current_time - check_time >= TIME_CHECK_INTERVAL:
            # Send a single ping to test for network connectivity.
            network_check = wifi.radio.ping(ip=ip_address)
            # If there is network connectivity, run the time check code.
            if network_check is not None:
                # Reset `check_time` to continue tracking.
                check_time = time.time()
                # Retrieve the time and save it to `sundial`.
                sundial = io.receive_time()
                # Print the current date and time to the serial console.
                print(f"LED color time-check. Date and time: {sundial.tm_year}--" +
                      f"{sundial.tm_mon}: {sundial.tm_mday} {sundial.tm_hour}:" +
                      f"{sundial.tm_min:02}"")
                # Provide the current hour to the `color_time` function. The
                returned color is
                # provided to `pixels.fill()` to set the LED color.
                pixels.fill(color_time(sundial.tm_hour))
            else:
                print("Network check ping failed.")
        else:
            print("Network check ping has failed for over
                  {NETWORK_DOWN_RELOAD_TIME} seconds.")
            reload_on_error(3, reload_type="reset")

        # If network down detection is enabled, run the rest of the code.
        if NETWORK_DOWN_DETECTION:
            # If this is the first run of the code or up ping interval` time has
            passed, continue.
            if not ping_time or current_time - ping_time >= UP_PING_INTERVAL:
                # Reset `ping_time` to continue tracking.
                ping_time = time.time()
                # Ping to verify network connection.
                wifi_ping = wifi.radio.ping(ip=ip_address)
                # If the ping is successful, set the fail count to 0, and print IP
                if wifi_ping is not None:
                    ping_fail_count = 0
                    print(f"Pinging {ip_address}: {wifi_ping} ms")
                else:
                    print(f"Network ping check ping has failed.")
            else:
                # If the ping has failed, and it's been one second, continue with this
                code.
                if wifi_ping is None and current_time - ping_fail_time >= 1:
                    # Reset `ping_fail_time` to continue tracking.
```python
ping_fail_time = time.time()
# Add one to the failure count tracking.
ping_fail_count += 1
# Print the ping failure count.
print(f"Ping failed {ping_fail_count} times")
# If the ping fail count exceeds the value defined above, begin blinking the LED
# to indicate that the network is down.
if ping_fail_count > CONSECUTIVE_PING_FAIL_TO_BLINK:
    blink(BLINK_COLOR)
    # If the initial setup ping failed, it means the network connection was failing
    # from the beginning, and might require reloading the board. So, if the initial
    # ping failed and the ping_fail_count is greater than 30,
    # immediately soft reload
    # the board.
    if not initial_ping and ping_fail_count > 30:
        reload_on_error(0)
        # If the initial ping succeeded, the blinking will continue until
        # the connection is reestablished and the pings are once again successful.
        # ============ ERROR HANDLING ============
        # Since network-related code can fail arbitrarily in a variety of ways, this
        # final block is
        # included to reset the board when an error is encountered.
        # When the error is thrown, wait 10 seconds, and hard reset the board.
        except Exception as error:
            reload_on_error(10, error, reload_type="reset")
```

The settings.toml File

On your CIRCUITPY drive, you should find a settings.toml file. This is the file that is used to store your WiFi and Adafruit IO credentials, so you can use them in your code, without needing to add them directly to your code.py.

If you don't see the file, you can make one with any text editor on your computer by putting in the information below.

If you do not have an Adafruit IO account, see the previous page on how to create one.

Edit the settings.toml file to include the following. Update the information in quotes to match your credentials.

```toml
wifi_ssid = "your-wifi-ssid"
wifi_password = "your-wifi-password"
aio_username = "your-adafruit-io-username"
aio_key = "your-long-adafruit-io-key"
```

Once settings.toml is updated, you're ready to continue!
Customize Your Canary

There are a number of features in this project that you can customize including the nightlight colors! This section will go through each feature, and cover the default and how to update it.

Network-Down Detection

The canary is watching over you with this feature! The light includes a feature that verifies whether you still have network connectivity, and notifies you when your network is down by blinking. That way you don't have to sit there and wonder if it's you or the internet!

If you have the canary sitting on your desk in your office, this is perfect. However, if you're using this light in your bedroom, it may not be so desirable. Even though the blinking is dimmer while you sleep, you may not want it to blink at all. The first thing in this code that you can customize is whether that blink code will run.

The code defaults to the blinking feature being enabled. To disable this feature, you need to change the following variable to False.

```
NETWORK_DOWN_DETECTION = True
```

Sleep and Wake Time Canary Colors

Red light is conducive to sleep. Blue light is conducive to promoting wakefulness. Therefore, the canary default colors are red for when you intend to sleep, and blue for when you intend to be awake.

Obviously these choices have a purpose, but what's the point of a DIY light if you can't customize the colors? Here you can choose any color of the rainbow for your canary.

The sleep color is the color your canary will light up during the later-specified sleep time. The wake color is color during the later-specified wake time.

To update either color, you'll change the associated RGB color tuples () to the desired colors.
SLEEP_COLOR = (255, 0, 0)
WAKE_COLOR = (0, 0, 255)

Sleep and Wake Time

One of the main features of this project is that you can change the color that the canary lights up based on a specified time. This is where you set the times by providing an hour in 24-hour time; it must be an integer (whole number) between \(0\) and \(23\).

Sleep Time

This is the hour at which the light should change to the desired color for the time you intend to be sleepy or sleep. It defaults to \(20\), or 8pm.

SLEEP_TIME = 20

Wake Time

This is the hour at which the light should change to the desired color for the time you intend to promote wakefulness or be awake. It defaults to \(6\), or 6am.

WAKE_TIME = 6

Canary Brightness

The NeoPixel LEDs used in this project can get very bright. That said, how bright your canary should be while you're awake and asleep is an entirely personal choice.

NeoPixel brightness is provided as a float (a number with a decimal) between \(0.0\) and \(1.0\), where \(0.0\) is off, and \(1.0\) is max brightness. So, for example, if you wanted the LEDs to be quite dim, you might choose \(0.1\). Alternatively, if you want them much brighter than that but not max, you might choose \(0.8\).

The brightness during sleep time is defaults to \(0.2\). The brightness during wake time defaults to \(0.7\).

To increase or decrease these brightnesses, update the values as explained above.
SLEEP_BRIGHTNESS = 0.2
WAKE_BRIGHTNESS = 0.7

Time Check Interval

This sets the interval at which the code checks with Adafruit IO to receive the current time. It is required because Adafruit IO has rate-limiting in place, and ensures that you do not hit that limit and end up throttled. It defaults to 300 seconds, or 5 minutes.

If you wish to increase the delay between time checks, you can increase this value.

It must be a number of seconds greater than or equal to 300. If you enter a value less than 300, the code will fail with an error.

TIME_CHECK_INTERVAL = 300

Successful Ping Interval

This is the interval at which the code will send a ping specifically while the network is connected, and the pings are successful. This defaults to 1 second.

If for any reason you wish to slow down the delay between successful pings, you can increase this value.

This value must be 1 second or greater. The code will fail with an error if you decrease this value below 1.

UP_PING_INTERVAL = 1

Blink Color

When the network-down detection is enabled, your canary will blink when the network is down. The blink color defaults to red, but you can change it to any color of the rainbow.

To update the color, you'll change the RGB color tuple () to the desired color.

BLINK_COLOR = (255, 0, 0)
Consecutive Ping Fail to Blink

This value is the number of times the pings must consecutively fail before the canary begins blinking. It defaults to 10 consecutive failures.

Network connections in some places are flakier than others. If you feel like the blinking is happening too often, you can increase this value to extend the number of consecutive failures it takes to begin blinking.

CONSECUTIVE_PING_FAIL_TO_BLINK = 10

Network Down Reload Time

This is the amount of time in seconds that needs to pass before resetting the board, but only when network down detection is disabled. It is in place to ensure that if your network goes down, and you’re not tracking it to use the canary to notify you, the board will eventually reset to attempt to reestablish a connection. It defaults to 900 seconds, or 20 minutes.

You can update this to whatever time interval in seconds you want, but know that the network check only occurs every 5 minutes or more when network down detection is disabled. So, if you set it to something less than 5 minutes, it will reload the board on the first failure.

NETWORK_DOWN_RELOAD_TIME = 900

Ping IP Address

This is the IP address used with ping to verify that network connectivity is still present. It defaults to 208.67.222.222, which is one of the Open DNS IP addresses.

To switch, update this to a different IP. It must be a valid IPV4 address as a string (in quotes).

Be considerate when choosing an IP address. Make sure whatever you choose is prepared to handle the repeated pings. If you are concerned, remember you can decrease the ping time interval as well.
Code Walkthrough

You've customized all of the options you're interested in changing. Now, the rest of the code follows. Wondering what's going on with it? This page has you covered!

The first part of the code is only run once after the board starts up. It is used for getting everything set up.

Modules and Libraries

First, you import the necessary CircuitPython modules and libraries.

```python
import os
import ssl
import time
import ipaddress
import supervisor
import board
import wifi
import microcontroller
import socketpool
import adafruit_requests
import neopixel
from adafruit_io.adafruit_io import IO_HTTP
```

Customization

The customization details can be found in the Customize Your Canary section of the Code the Canary page.

Value Checks

There are two time values that should not be set below a certain value, and must be an integer (whole number). Therefore, there are two checks in the customization code to ensure they are set properly.
There are two things each check is looking for.

- Is the provided value is less than the required minimum?
- Is the provided value is a float?

If either or both of these conditions are met, the code will stop running and raise an exception. For the code to continue running, you must update the value to an integer within the appropriate range.

```python
if UP_PING_INTERVAL < 1 or isinstance(UP_PING_INTERVAL, float):
    raise ValueError("UP_PING_INTERVAL must be an integer, and greater than 1!")
```

## Hardware and Code Set Up

There is very little hardware set up, but a pretty solid chunk of code setup is necessary.

### NeoPixel Set Up

To tell the code where to look for the NeoPixel BFF and be able to control the NeoPixels, you need to instantiate the NeoPixel object. The BFF connects on pin A3, and there are 25 LEDs in the grid.

```python
```

### Helper Functions

There are three helper functions used in this code. In many situations, functions are included in demo code to create a simple way to use the same block of code multiple times later in the demo. They can also be used to make later code simpler by separating out a complicated code block, even if it's only going to be used once. The helper functions in this code fall into both categories.

### Reload on Error

This function is used to reset the board when an error occurs. Some errors you want to stop the code so you can fix the issue. Other errors occur because of an issue beyond the code or board, and can sometimes be resolved by resetting the board. Both types are included in this demo. This function is specifically used for the ones
where we want the board to reset to the code can continue running. This ensures that if an error occurs, such as the WiFi disconnects or the Adafruit IO connection fails, that you canary will simply reset and continue being a glowing friend.

The `reload_on_error` function has three possible arguments: `delay`, `error_content`, and `reload_type`.

- **delay** - This is the delay in seconds between when the error occurs and when the board resets. This is the only argument that is required for this function to work.
- **error_content** - If an error string is provided, this is used to print the error to the serial console. If nothing is provided, nothing is printed. In this demo, typically the exception will be `except Exception as error:`, and you would include `error` after a comma following the delay.
- **reload_type** - This determines which way this function will trigger a reload. There are two options: hard `reset` and soft `reload`. A hard reset is the same thing that happens when you press the physical reset button on the board. A soft reload is the same thing that happens when your code auto-reloads, or you press CTRL+C and CTRL+D in the serial console. Some issues are resolved with a soft reload, and others require a hard reset. Therefore, this function is designed to allow for both. It defaults to "reload" (the quotes are necessary), meaning, with no changes, it will trigger a soft reload. To trigger a hard reset, update this argument to "reset".

```python
def reload_on_error(delay, error_content=None, reload_type="reload"):
    if str(reload_type).lower().strip() not in ["reload", "reset"]:
        raise ValueError("Invalid reload type:" + str(reload_type))
    if error_content:
        print("Error:
" + str(error_content))
    if delay:
        print("{} microcontroller in {} seconds.".format(reload_type[0].upper() + reload_type[1:], delay))
        time.sleep(delay)
    if reload_type == "reload":
        supervisor.reload()
    if reload_type == "reset":
        microcontroller.reset()
```

**Color Time**

This function determines what color to illuminate your canary based on the current time. It is made up of many of the customisable variables from earlier in the code. This code is necessary because the wake time and sleep time might not always be in numerical order; sleep time could occur after wake time. This complicated the color check code significantly, and therefore made the most sense in a function.
The `color_time` function has one argument.

- **current_hour** - This is the current time as an hour value only. In this code, the value here is always `sundial.tm_hour` which is the hour value from the time returned when checking in with Adafruit IO's time service.

```python
def color_time(current_hour):
    if WAKE_TIME < SLEEP_TIME:
        if WAKE_TIME <= current_hour < SLEEP_TIME:
            pixels.brightness = WAKE_BRIGHTNESS
            return WAKE_COLOR
        pixels.brightness = SLEEP_BRIGHTNESS
        return SLEEP_COLOR
    if SLEEP_TIME <= current_hour < WAKE_TIME:
        pixels.brightness = SLEEP_BRIGHTNESS
        return SLEEP_COLOR
    pixels.brightness = WAKE_BRIGHTNESS
    return WAKE_COLOR
```

**Blink**

This is used to blink the canary a specified color. It uses the `color_time()` function to sort out whether to blink at the wake brightness of the sleep brightness; that way if the network goes down at night, it doesn't start blinking at full brightness.

The `blink` function has one argument.

- **color** - This is the color to blink the LEDs. The color used in the code is customised at the top of the program.

```python
def blink(color):
    if color_time(sundial.tm_hour) == SLEEP_COLOR:
        pixels.brightness = SLEEP_BRIGHTNESS
    else:
        pixels.brightness = WAKE_BRIGHTNESS
    pixels.fill(color)
    time.sleep(0.5)
    pixels.fill((0, 0, 0))
    time.sleep(0.5)
```

**Connect to WiFi**

This section connects to the WiFi using your credentials from the settings.toml file. Then it creates the socket pool for the requests object, and created the requests object needed for the Adafruit IO connection.
This process can fail for various reasons, and is therefore included in a `try/except` block. If the process fails, the board will print the error and soft reload following a five second delay. The errors thrown by this process are sometimes very unclear. If you’re receiving errors, check your SSID and password before continuing.

```python
try:
    wifi.radio.connect(os.getenv("wifi_ssid"), os.getenv("wifi_password"))
    pool = socketpool.SocketPool(wifi.radio)
    requests = adafruit_requests.Session(pool, ssl.create_default_context())
except Exception as error:
    print("Wifi connection failed.")
    reload_on_error(5, error)
```

Set Up Ping

Next is the ping IP set up and initial ping.

First, you’ll set up the IP address that the code will be pinging, as defined when going through the code customisations. Next, you send out a ping. If the initial ping is unsuccessful, print a message to the serial console.

The `initial_ping` variable is used later to determine whether to reload the board following a specific number of consecutive ping fails or not. If the ping fails, set it to `False`. If the initial ping is successful, set it to `True`.

```python
ip_address = ipaddress.IPv4Address(PING_IP)
wifi_ping = wifi.radio.ping(ip=ip_address)
if wifi_ping is None:
    print("Setup test-ping failed.")
    initial_ping = False
else:
    initial_ping = True
```

Connect to Adafruit IO and Retrieve the Time

First you connect to Adafruit IO using your credentials from your settings.toml file, and the `requests` object created previously.

Next, you test the connection was successful by checking for the current time. The Adafruit IO set up and connection can fail for various reasons, but that failure isn't evident until you try to use it. Therefore, this time check is included in a `try/except` block. If the connection fails, it will print the error, and soft reload the board following a five second delay.

```python
io = IO_HTTP(os.getenv("aio_username"), os.getenv("aio_key"), requests)
```
try:
    sundial = io.receive_time()
except Exception as error:
    print("Adafruit IO set up and/or time retrieval failed.")
    reload_on_error(5, error)

Variables

The last thing before the loop is initializing five variables for later use.

Time Tracking

There are four time-tracking variables.

- **check_time** - This is used to track the last time the code retrieves the time from Adafruit IO, which is then used to determine the next time the code should retrieve the time. It is initialized to 0 so it can also be used to verify the first run through the code.
- **network_down_time** - This is used to track the time, when the network is down, until the board should reset. This is only used when network down detection is disabled.
- **ping_time** - This is used to track the last time a successful ping occurred, which is then used to determine the next time the code should send a ping. It is initialized to 0 so it can also be used to verify the first run through the code.
- **ping_fail_time** - This is used to track the last time a failed ping occurred, which is then used to determine the next time the code should send a ping.

```python
ping_time = 0
check_time = 0
ping_fail_time = time.time()
```

Network Check

There is one check variable.

- **network_check** - This is used to determine whether to reset the board if the network is down and network down detection is disabled. It is initialized to 1 because its code trigger state is None (the state set when the network check ping fails), and therefore, if it is never set to None (because the network ping check is successful), the resulting reset code is never run.

```python
network_check = 1
```
Count Tracking

There is one count-tracking variable.

- **ping_fail_count**: This is used to keep track of the consecutive ping failures so the code knows when to blink or not blink.

```python
ping_fail_count = 0
```

The Loop

The last section of code is inside the `while True:` loop. This means it runs the first time the board starts up, and will continue looping until the board is reset.

```python
while True:
    [...]  # This is a way of indicating there is more code here.
    # It will be used again this section for this purpose.
```

Update the Current Time

The first line inside the loop sets the current time. This will happen every time the loop runs. `current_time` is used throughout the code along with the other time-tracking variables to determine time intervals in a non-blocking manner.

```python
current_time = time.time()
```

The `try` and `except` Block

The canary is designed to be plugged in and left alone. The majority of the code in the loop involves WiFi and Adafruit IO. Both of these can fail for various reasons, and if used standalone, the failures will cause the code to fail and stop running. This isn't conducive to a "set it and forget it" situation. So, to ensure the code keeps running, the rest of the code is wrapped in a `try / except` block.

The way this works is that the code in the `try` block will be run from top to bottom. If no errors are encountered, it will skip the `except` block, and start again at the top of the loop. If an error is encountered, it will skip the rest of the code in the `try` block, and immediately run the code in the `except` block.
The **try** block contains all of the code meant for the loop that might run into an error.

The **except** block is used to handle the encountered errors, known as exceptions. It contains the code that ensures everything continues to run. In this case, if an `Exception` occurs, the code will use the `reload_on_error()` function to print the error info to the serial console, wait ten seconds, and then hard reset the board. This will resolve the majority of errors that might occur, and allow the code to run fresh so it can continue on its way.

```python
[...]  
try:
    [...] # Code in the try block is nested here.
    except Exception as error:
        reload_on_error(10, error, reload_type="reset")
```

### Time to Light Up LEDs

This code runs when it is the first time running the code after start up, or if the time check interval has passed.

```python
if not check_time or current_time - check_time >= TIME_CHECK_INTERVAL:
```

### The Network Check Ping

If you run the Adafruit IO time check code when there is no network available, it will throw an exception, and in the case of this example, reset the board. That means the board would reset every 20 seconds or so, without running any of the other code in the **try** block.

If network down detection is enabled, this means that the network down code will never run. That code would be skipped, and the canary will never blink to let you know your network is down.

To handle this eventuality, the code performs the network check ping before every time check. The time check code only runs if the network check ping was successful. This ensures that all of the following code in the **try** block will be run, including the network down code.

```python
network_check = wifi.radio.ping(ip=ip_address)
```
A successful ping returns the number of milliseconds it took to receive the packet back from the pinged IP address. A failed ping returns None. These results are used throughout the rest of the code to determine when to run what code blocks.

Receive the Current Time

If the network check ping is successful, the time code is run.

You reset check time to continue tracking and determine the next time the time code should be run.

```python
check_time = time.time()
```

You check with Adafruit IO to receive the current time, and save it to the variable `sundial`, which allows you to access specific pieces of the time, such as seconds or hours.

```python
sundial = io.receive_time()
```

You print to the serial console a message including the date and time received from Adafruit IO, in a human-readable format.

This line is split into two parts because it is otherwise too long to adhere to code line length conventions. Due to how CircuitPython handles f-strings, you must explicitly "add" them together to concatenate them in the serial console. This is why there is a `+` at the end of the first line.

```python
print(f"LED color time check. Date and time: {sundial.tm_year}-{sundial.tm_mon}-" +
     f"{sundial.tm_mday} {sundial.tm_hour}:{sundial.tm_min:02}")
```

Light It Up

This lights up all of the LEDs to the proper color based on the time, using the `color_time` function. It lights up red or blue by default.

The function expects you to provide the current hour. You can use `sundial` to easily access and provide the hour received from Adafruit IO. The `color_time` function returns the appropriate color.
That color is provided to `pixels.fill()` which lights up all of the NeoPixels the same color.

```python
pixels.fill(color_time(sundial.tm_hour))
```

If there is no network available at start up, the canary will not light up. If network down detection is disabled, the following will occur. As the network continues to be unavailable, the canary will continue to be dark. The board will reload after a specified period of time to attempt to resolve the connection. If the issue is indeed with the network, the canary will remain dark until network connectivity is established.

Network Check Ping Failure

Immediately after a failed network check ping, the code prints the following to the serial console.

```python
print("Network check ping failed.")
```

If Network Down Detection is Disabled

Disabling the network down detection feature of your canary means the code will run the next block of code, and skip the rest of the code in the `try` block.

Repeated Ping Failures

If the network check ping continues to fail for longer than the specified network down reload time interval, the code will print a message to the serial console, wait 3 seconds, and hard reset the board.

This if statement is split into two lines, as it exceeds code line length conventions if included on one line. To indicate to the code that the line continues on the next line, a `\` is included at the end of the first line. This is standard for CircuitPython and Python.

```python
if not NETWORK_DOWN_DETECTION and network_check is None and \
current_time - network_down_time >= NETWORK_DOWN_RELOAD_TIME:
    print(f"Network check ping has failed for over \{NETWORK_DOWN_RELOAD_TIME\} seconds.")
    reload_on_error(3, reload_type="reset")
```
If Network Down Detection is Enabled

Leaving the network down detection feature of your canary enabled means the code will skip the previous block, and run everything else in the `try` block.

Ping for Consistent Network Status Verification

This code runs when it is the first time running the code after start up, or if the up ping time interval has passed.

```
if not ping_time or current_time - ping_time >= UP_PING_INTERVAL:
```

Reset the ping time to continue tracking when to perform the next ping.

```
ping_time = time.time()
```

Send the ping to the specified IP address, and save the result to `wifi_ping` to use in the rest of this section.

```
wifi_ping = wifi.radio.ping(ip=ip_address)
```

When the Ping is Successful

A successful ping means `wifi_ping` has been assigned a value, and is therefore equal to something other than `None`. Therefore the last section of code in this block is run.

Since the ping is successful, set the ping fail count to `0`. Print the IP address and the ping time to the serial console.

```
if wifi_ping is not None:
    ping_fail_count = 0
    print(f"Pinging {ip_address}: {wifi_ping} ms")
```

When the Ping Fails

A failed ping means `wifi_ping` is equal to `None`. This section of code runs if a ping fails and one second has passed.

```
if wifi_ping is None and current_time - ping_fail_time >= 1:
```
Reset the ping fail time to continue tracking the next time to run this section of code.

```python
ping_fail_time = time.time()
```

Add one to the ping fail count tracking, and print to the serial console how many times the ping has consecutively failed.

```python
ping_fail_count += 1
print(f"Ping failed {ping_fail_count} times")
```

If the ping fail count exceeds the consecutive ping fail to blink value, begin blinking the canary. It blinks red by default. The blink duration is on for 0.5 seconds, then off for 0.5 seconds, which is to say, it blinks for half a second every second.

```python
if ping_fail_count > CONSECUTIVE_PING_FAIL_TO_BLINK:
    blink(BLINK_COLOR)
```

If the set up ping was successful, the blinking will continue until the next successful ping resets the ping fail count.

If the set up ping failed, it means that the network connectivity was having issues from the beginning, and if it continues as such, a board reload may be needed to attempt to resolve the issue. If this situation occurs, and the ping fail count exceeds 30, the board will immediately soft reload.

```python
if not initial_ping and ping_fail_count > 30:
    reload_on_error(0)
```

If there is no network available at start up, the canary will not light up. If network down detection is enabled, the canary will remain dark until the specified number of consecutive ping failures occur, and then will begin blinking to indicate that the network is down. This also means that after 30 consecutive ping failures, the board will soft reload in an attempt to resolve the connectivity issue.
Assembly

Headers for QT Py & BFF
Use the strips of header pins that shipped with the boards and use flush snips to create two strips of 1 x 7 headers.

Install Headers to QT Py
Insert the two strips of headers into the pins on the bottom of the QT Py.

Secure Headers to QT Py
Solder all 13 pins to the QT Py using a soldering iron and solder wire. Using a breadboard can help keep the headers straight while soldering.
QT Py & BFF Orientation
Grab the NeoPixel 5x5 grid BFF board and orient it with the QT Py so the USB port is lined up with the USB label.

Insert QT Py & BFF
Fit the header pins into the NeoPixel BFF and seat the two boards together until they're flush with the PCBs.

Please use eye protection and be careful cutting header pins. Aiming at a trash bin might be best to avoid errant pins flying.

Trim Header Pins
Using flush diagonal cutters, trim the longer header pins so they're half their length.
Short Header Pins
Trimmed header pins will make soldering the pins easier and less messy.

Secure NeoPixel BFF to QT Py
Solder the short headers to the pins on the NeoPixel BFF.

Soldered QT Py & BFF
Take a moment to inspect the pins have been properly soldered together.
Bottom Cover
Get the bottom cover ready for installing the QTPy and BFF.

Install QT Py & BFF
Orient the QT Py and BFF so the USB port is lined up with the notch in the bottom cover.

Press Fit QT Py & BFF
Press the PCBs in between the walls of the bottom cover and push it down until the PCBs sit flush.
Installed PCBs
Plug in a USB-C cable to test the circuit.

Install Bird
Orient the bird so the notch at the bottom is lined up with the QT Py's USB port.

Bird Installation
Press fit the cover into the bottom of the bird until the lip sits flush with the bottom of the bird.
Final Build
Congratulations on your build!