# Table of Contents

**Overview** 3

**Hardware** 4
- Parts
- Wiring

**Software** 6
- Setup MicroPython & WebREPL
- Load MicroPython Code
- Control Lights
- How It Works
Overview

This guide has been archived, there may be changes in the MicroPython API that makes this guide not work, and we won't be updating or supporting it.

Celebrate the holidays with MicroPython and smart holiday lights that are controlled from the web! This project shows you how to animate NeoPixels with MicroPython running on the ESP8266 WiFi microcontroller. Just pick the color and type of animation on a web page, click a button, and your holiday lights will spring to life with animated holiday cheer. The smart holiday lights will even remember their last animation run it on power up. This project is a great demonstration of the ESP8266 and MicroPython's strengths like easy WiFi access with the WebREPL and storing data in MicroPython's file system.

To follow this guide you'll want to be familiar with MicroPython by reading these guides:

- MicroPython Basics: How to Load MicroPython on a Board ()
- MicroPython Basics: Load Files & Run Code ()
- MicroPython Basics: ESP8266 WebREPL ()

See all the MicroPython guides in the learning system () for more information.

In addition you'll want to read the NeoPixel Uberguide () to understand the different types of pixels and considerations like how to power them.
Hardware

This guide has been archived, there may be changes in the MicroPython API that makes this guide not work, and we won't be updating or supporting it.

Parts

You'll need the following hardware for this guide:

- ESP8266 running MicroPython. The Feather HUZZAH ESP8266 is perfect if you don't have a board yet, but any ESP8266 board should work. See the loading MicroPython on a board guide for details on how to get MicroPython running on the board.
- NeoPixels. You can use any type of NeoPixels like strips, matrices, and more. However be sure you use RGB and not RGBW pixels as the code for this project assumes only RGB pixels. Three to four meters of a strip like the 30 pixels/meter NeoPixels are perfect for wrapping around a small wreath or tiny tree. Check out the NeoPixel Uberguide for more information on the different types of NeoPixels and how to power them.
- 5V power supply. You'll need a 5 volt power supply with enough current to drive all the pixels. Each NeoPixel can pull up to 60mA of current so be sure to get a large enough supply to support all your pixels! This 5V 10amp supply will handle up to about 150 NeoPixels at full brightness (grab a barrel jack to terminal block adapter to make connecting the supply easy). Check out the NeoPixel Uberguide for more power supply options.
- Large capacitor to protect pixels. As the NeoPixel Uberguide mentions you might want to add a large capacitor to the power connections to prevent damaging the pixels with a sudden rush of power. The 4700uF 10V electrolytic capacitor here should work great.
- Soldering tools. You might need to solder some of the NeoPixel and other connections so be sure to have a soldering iron, solder, and wire handy. If you're new to soldering check out the guide to excellent soldering too.

Wiring

Connect your hardware as shown below:
• Power supply ground/- to NeoPixel GND/ground and Feather HUZZAH ESP8266 GND/ground.
• Power supply 5V/+ to NeoPixel +5V and Feather HUZZAH ESP8266 USB pin. For other ESP8266 boards check its specs to see which pin can support a 5V power input that runs the board.
• Feather HUZZAH ESP8266 pin 15 to NeoPixel Din/input. You can use most other GPIO pins on the ESP8266 (except pin 16) but this guide will assume you're using pin 15.
• If you're using a capacitor to protect the pixels (highly recommended!) connect it as follows:
  ◦ Capacitor cathode (the pin on the side with the stripe) to power supply ground/-.
  ◦ Capacitor anode to power supply 5V/+.

If you're using a barrel jack to terminal block adapter it's easiest to connect the capacitor directly to the terminals like below (be sure the side with the stripe is connected to the negative/ground terminal):
Software

This guide has been archived, there may be changes in the MicroPython API that makes this guide not work, and we won't be updating or supporting it.

Setup MicroPython & WebREPL

First make sure you are running the latest version of ESP8266 MicroPython for your board. Check out the how to load MicroPython on a board guide () for more information on loading MicroPython onto the ESP8266 board.

Note that at the time of this guide's writing the current version of ESP8266 MicroPython is 1.8.6. Later versions might change the WebREPL so if you run into unexpected issues try going back to the 1.8.6 version.

Next setup the WebREPL on the ESP8266 board by following this guide (). Be aware the latest 1.8.6 version of MicroPython slightly changed how the WebREPL is setup so follow the guide () or the official instructions to enable the WebREPL (). Don't continue until you have your ESP8266 running MicroPython and can connect to its WebREPL!

Load MicroPython Code

After the WebREPL is enabled you'll need to download the MicroPython and webpage code for this project from its home on GitHub () by clicking the button below:
ESP8266 MicroPython smart holiday lights project code.
This will animate NeoPixels that can be controlled from the included
lights.html web page.
Author: Tony DiCola
License: MIT License
import machine
import neopixel
import utime
import ujson

# Static configuration that never changes:
PIXEL_PIN   = machine.Pin(15, machine.Pin.OUT)  # Pin connected to the NeoPixels.
PIXEL_COUNT = 32                                # Number of NeoPixels.
CONFIG_FILE = 'config.json'                     # Name of animation config file.

# Mirror the colors to make a ramp up and ramp down with no repeated colors.
def mirror(values):
    # Add the input values in reverse order to the end of the array.
    # However slice off the very first and very last items (the [1:-1] syntax)
    # to prevent the first and last values from repeating.
    # For example an input of:
    #  [1, 2, 3]
    # Returns:
    #  [1, 2, 3, 2]
    # Instead of returning:
    #  [1, 2, 3, 3, 2, 1]
    # Which would duplicate 3 and 1 as you loop through the elements.
    values.extend(list(reversed(values))[1:-1])
    return values

# Linear interpolation helper:
def _lerp(x, x0, x1, y0, y1):
    return y0 + (x - x0) * ((y1 - y0)/(x1 - x0))

# Animation functions:
def blank(config, np, pixel_count):  # pylint: disable=unused-argument, redefined-
    # Turn off all the pixels.
    np.fill((0,0,0))
    np.write()

def solid(config, np, pixel_count):  # pylint: disable=unused-argument, redefined-
    # Solid pulse of all pixels at the same color.
    colors = config['colors']
    elapsed = utime.ticks_ms() // config['period_ms']
    current = elapsed % len(colors)
    np.fill(colors[current])
    np.write()

def chase(config, np, pixel_count):  # pylint: disable=unused-argument, redefined-
    # Chasing animation of pixels through different colors.
    colors = config['colors']
    elapsed = utime.ticks_ms() // config['period_ms']
    for i in range(PIXEL_COUNT):
        current = (elapsed+i) % len(colors)
        np[i] = colors[current]
    np.write()
```python
def smooth(config, np, pixel_count):  # pylint: disable=unused-argument, redefined-outernames
    # Smooth pulse of all pixels at the same color. Interpolates inbetween colors
    # for smoother animation.
    colors = config['colors']
    period_ms = config['period_ms']
    ticks = utime.ticks_ms()
    step = ticks // period_ms
    offset = ticks % period_ms
    color0 = colors[step % len(colors)]
    color1 = colors[(step+1) % len(colors)]
    color = (int(_lerp(offset, 0, period_ms, color0[0], color1[0])),
             int(_lerp(offset, 0, period_ms, color0[1], color1[1])),
             int(_lerp(offset, 0, period_ms, color0[2], color1[2])))
    np.fill(color)
    np.write()

# Setup code:
# Initialize NeoPixels and turn them off.
np = neopixel.NeoPixel(PIXEL_PIN, PIXEL_COUNT)
np.fill((0,0,0))
np.write()

# Try loading the animation configuration, otherwise fall back to a blank default.
try:
    with open(CONFIG_FILE, 'r') as infile:
        config = ujson.loads(infile.read())
except OSError:
    # Couldn't load the config file, so fall back to a default blank animation.
    config = {
        'colors': [[0,0,0]],
        'mirror_colors': False,
        'period_ms': 250,
        'animation': 'blank'
    }

# Mirror the color array if necessary.
if config['mirror_colors']:
    config['colors'] = mirror(config['colors'])

# Determine which animation function should be called.
animation = globals().get(config['animation'], blank)

# Main loop code:
while True:
    animation(config, np, PIXEL_COUNT)
    utime.sleep(0.01)
```

Open the archive and find the following files:

- lights.py - This is the MicroPython code that will power the project.
- lights.html - This is the webpage you'll use to control the lights and animation.

Open lights.py in a text editor and notice the configuration values at the top:

```bash
# Static configuration that never changes:
PIXEL_PIN   = machine.Pin(15, machine.Pin.OUT)  # Pin connected to the NeoPixels.
PIXEL_COUNT = 32                                # Number of NeoPixels.
CONFIG_FILE = 'config.json'                     # Name of animation config file.
```
Change the PIXEL_COUNT value to be the number of NeoPixels in your light strand/project.

If you’re using a different pin other than GPIO15 to connect to the NeoPixels be sure to change it in the PIXEL_PIN value.

The rest of the file can stay the same and doesn't need to be changed. Save the file and rename it to main.py so that it will run on boot of the board. Copy the file to the MicroPython board using either the WebREPL or a tool like ampy.

Control Lights

Now for some fun, you’re ready to control the lights from a web page! Open the lights.html page in your web browser (note that the page has been tested with the latest Chrome browser, but it should work with other modern web browsers like Safari and IE 8+).

You should see the page load and look like:

![Lights HTML Page](image)

Make sure your computer is connected to the ESP8266 board's WiFi network just like when you’re using the WebREPL (however be sure you aren’t also connected to the WebREPL in another tab or browser!).

Set the board URL to the URL you use to access the WebREPL (if you aren’t sure then leave the default value, it’s what most WebREPL connections use).

In the password field enter the password you set when setting up and accessing the WebREPL.
You can control the animation of the lights with the fields on the page:

- **Animation** - This drop-down controls if the lights will animate in a solid pulse, chase, smooth fade, or blank animation (no animation).
- **Animation Period** - This value controls the speed of the animation (in milliseconds), or how long the animation stays on any specific color. Small values will speed up the animation and large values will slow it down.
- **Colors** - This drop-down controls the range of colors that the lights will display. You can pick a blue/red, green/red, or blue/white color set. Look at the index.html code and How it Works video 3 below to see how to add more colors!
- **Mirror Colors** - This checkbox controls if the selected colors should ramp up and down smoothly. The colors by default just ramp up, like from blue up to red, but by keeping mirror colors checked you’ll see they ramp up and down, like from blue up to red and back down to blue again. Try turning mirroring on and off to see how it changes the animation.

Once you've picked your desired options click the Update Lights button. After a moment you should see the lights spring to life with the selected animation!

If the lights don’t update be sure you can connect to the board using the WebREPL (you might need to press Ctrl-C once connected to stop the main.py loop and get to the Python prompt). Be sure to use the exact same board URL and password with lights.html as you do when accessing the WebREPL. Also try pressing the reset button on the board to do a hard reset and ensure the MicroPython code & WebREPL are running again.

Try changing the animation values and pressing update lights again to see the lights change. The code for this project will actually remember the last animation it was given so you can even unplug the board, plug it back in, and see the lights jump back to their last animation.

That's all there is to using the ESP8266 MicroPython smart holiday lights! Happy holidays!

**How It Works**

If you’re curious how the web page controls the lights watch the following videos which explore in depth how the project was created and works: