# ESP32 Holiday Christmas Lights

Control your Holiday Lights using MicroPython and the ESP32 micro-controller connected to Neopixels and your WiFi.

The Web Page is Automatically Updated every 20 seconds, or you can Use the Update Button to Manually Update the Web Page.

Since the Web Server contains a "blocking call" when used with the ESP32, your new selection can take up to 20 seconds to load.

Note - Any Menu Function that has "Custom", the colors can be controlled with the "RGB" Sliders.

## Animation

Rainbow Color Theater Chase Pattern

## Custom Color Mix Control (Red-Green-Blue)

Red

Green

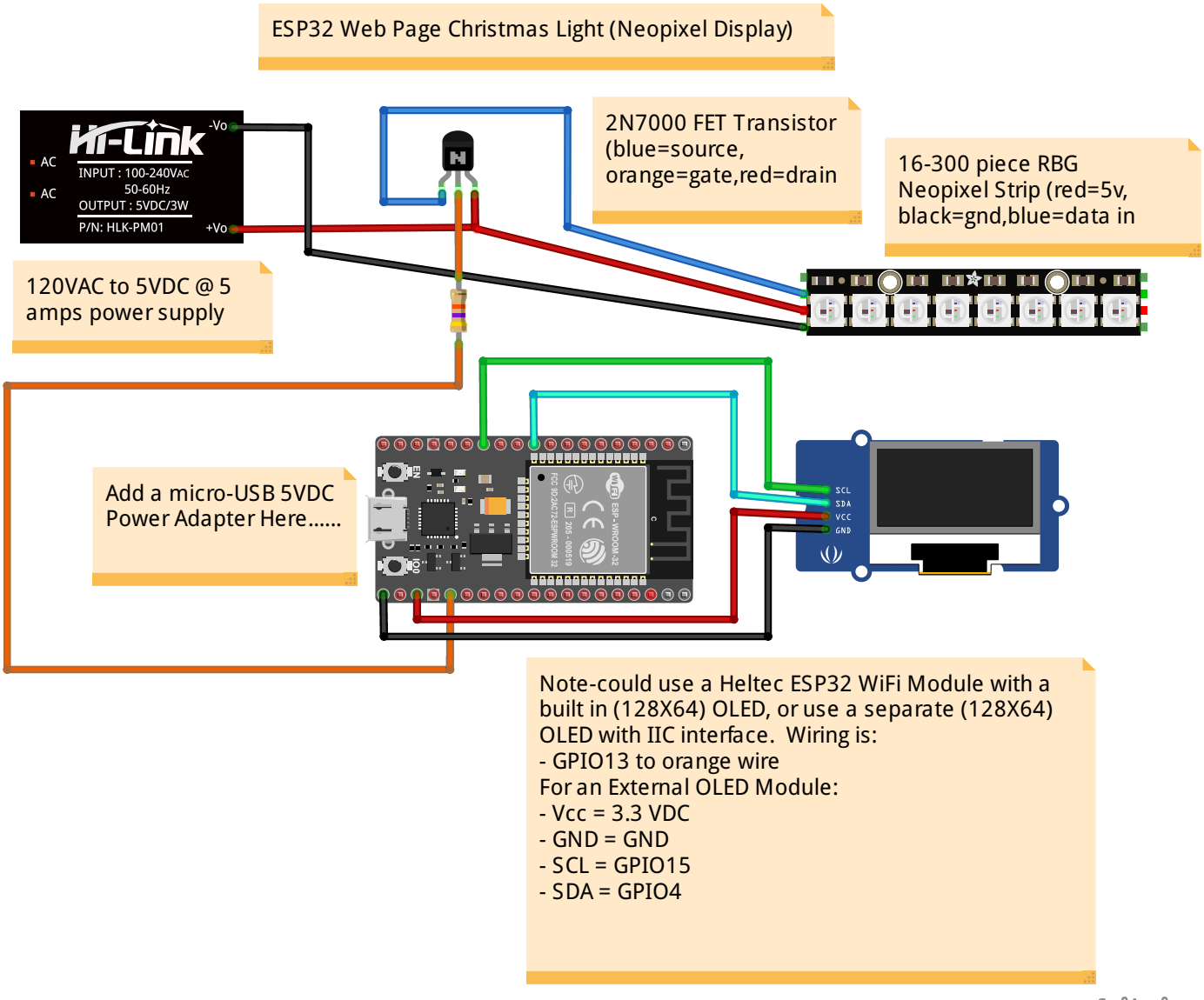
Blue

## Delay Time (mS)

### Time Slider

Value: 50

Update Lights


ESP32 Web Page Christmas Light (Neopixel Display)
Hi-Link
AC
AC
INPUT : 100-240VAC
50-60Hz
OUTPUT : 5VDC/3W
P/N: HLK-PM01
-Vo
+Vo
120VAC to 5VDC @ 5 amps power supply
2N7000 FET Transistor (blue=source, orange=gate,red=drain
16-300 piece RBG Neopixel Strip (red=5v, black=gnd,blue=data in
Add a micro-USB 5VDC Power Adapter Here.....
SCL
SDA
VCC
GND
Note-could use a Heltec ESP32 WiFi Module with a built in (128X64) OLED, or use a separate (128X64) OLED with IIC interface. Wiring is:
- GPIO13 to orange wire
For an External OLED Module:
- Vcc = 3.3 VDC
- GND = GND
- SCL = GPIO15
- SDA = GPIO4
fritzing

```
"""
WiFi Manager Program With Other Program (added)
===============================================
Notes:
-----
1) Reads SSID & Password from a web Page and will connect (if correct).
2) If not, starts an Access Point, and provides instructions on how to connect
   and enter SSID and Password as well as other instruction on the OLED Display.
3) Reboots to Leave Access Point, and Connects to existing WiFi.
4) Stores new Wifi SSID & Password Parameters to a wifi.dat file.
5) Monitors current status of Wifi Connection to ensure still available (need a while
loop)
6) Type "http://192.168.4.1" to get to web page to enter information
7) Place your program code in this main.py file
8) Install ssd1306.py, main.py, wifimanager.py, and microWebSrv.py as seperate files on
ESP32
    directory (boot.py already exists)
9) Store the HTML Page, favicon, and CSS file in a folder caled "www" and place on
ESP32 directory
10) Type "http://your ESP32 IP Address/xmaslights_3.html in a brower, and set setting
    then "update" on Web Page
11) If using neopixels > 16, be sure to change the "PIXEL_COUNT" number and use a
seperate power
    supply and driver transistor
    Note - The power supply voltage and drive must be the same voltage level and needs
    to be
    non-inverting.

Note - If you cannot connect to WiFi through a Web Browser, or get character
       encoding errors or issues, perform the following:

To Enter the WiFi SSID & Password, perform the following using the PUTTY Utility,
with a serial connection to th eserial port identified in "Windows", with a buad
rate of 115200, and type:
       f = open('wifi.dat', 'w')
       f.write('SSID;PASSWORD')
       f.close()
- Where SSID and Password (capitals only)is your information for your home network
  router.
- Also be sure to include the "ssid=" in the beginning, and the ";" between SSID
  and PASSWORD, and leave no spaces between words, (unless spaces are part of SSID
  and / or PASSWORD).
- If Putty will not let you write to this file, you may have to remove "main.py"
  file first with "Ampy" and reinstall after the CONFIG_FILE is written.
"""
# Import Modules
# --------------
import network
from time import sleep
import wifimanager # Start the Wifi Manager Program (this is a seperate file)

wlan = wifimanager.get_connection()
if wlan is None:
    print("Could not Initialize Network Connection.")
    while True:
        pass  # Stay here

# Shut off Access Point
# ---------------------
ap_if = network.WLAN(network.AP_IF)
sleep(5) # 5 second time delay
ap_if.active(False)

# Main Program Code Goes Below Here
# =================================
from microWebSrv import MicroWebSrv
from machine import Pin
```

```
from neopixel import NeoPixel
import ujson # Used if decoding Javascript Object Notation
import time
import uos #random number gen
from machine import Pin, I2C
from time import sleep
from ssd1306 import SSD1306_I2C

# Global Variables / Constants and Initial States
# -----------------------------------------------
PIXEL_COUNT = 16 # Number of NeoPixels.
PIXEL_PIN = 13 # ESP32 Neopixel driver pin
content1 = " " # Null String
menu = "blank"      # Default values are listed below
newdata = "blank"
Red = 127
Green = 127
Blue = 127
tdelay = 50
OLED_RST_PIN = 16

# Initialize Pixels:
# ------------------
np = NeoPixel(Pin(PIXEL_PIN), PIXEL_COUNT)

# OLED Initial Setup
# ------------------
OLED_RST = Pin(OLED_RST_PIN, Pin.OUT, value=1) # Sets up OLED RST to high
i2c = I2C(sda = Pin(4), scl = Pin(15)) # IIC Pins
display = SSD1306_I2C(128, 64, i2c) # IIC parameters for Display

# Generic Function to Display 6 lines on OLED
# -------------------------------------------
def Display_Stat(line1,line2,line3,line4,line5,line6,delaytime):
    display.fill(0) # Clear Previous Text
    display.text(line1,0,0,1) #Line 1
    display.text(line2,0,10,1) # Line 2
    display.text(line3,0,20,1) # Line 3
    display.text(line4,0,30,1) #Line 4
    display.text(line5,0,40,1) # Line 5
    display.text(line6,0,50,1) # Line 6
    display.show() # Display New Text
    sleep(delaytime) # Delay secs

# Opening Script Instructions:
# ----------------------------
L1 = "Server Started"
L2 = "Open Web Browser"
L3 = "  Type in: "
L4 = "ESP32 IP Address"
L5 = "/xmaslights_3"
L6 = ".html"
delayx = 5
Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function

# Amination Functions:
# ====================
def Blank(): # Turn off all the pixels.
    n = np.n
    for i in range(n):
        np[i] = (0, 0, 0)
    np.write()

def Chase(r, g, b, td): # Chase Routine (custom colors)
    n = np.n
    for i in range(256):
        for j in range(n):
```

```
130             np[j] = (0, 0, 0)
131         np[i % n] = (r, g, b)
132         np.write()
133         time.sleep_ms(td)
134     Blank() # Goto Function
135 
136 def Bounce(r, g, b, td): # Bounce routine (custom colors)
137     n = np.n
138     for i in range(256):
139         for j in range(n):
140             np[j] = (r, g, b)
141         if (i // n) % 2 == 0:
142             np[i % n] = (0, 0, 0)
143         else:
144             np[n - 1 - (i % n)] = (0, 0, 0)
145         np.write()
146         time.sleep_ms(td)
147     Blank() # Goto Function
148 
149 def Fade(r, g, b, td): # Fade routine (custom colors)
150     n = np.n
151     for i in range(0, 8 * 256, 8):
152         for j in range(n):
153             if (i // 256) % 2 == 0:
154                 r = i & 0xff
155                 g = i & 0xff
156                 b = i & 0xff
157             else:
158                 r = 255 - (i & 0xff)
159                 g = 255 - (i & 0xff)
160                 b = 255 - (i & 0xff)
161             np[j] = (r, g, b)
162         time.sleep_ms(td)
163         np.write()
164     Blank() # Goto Function
165 
166 def Solid(r, g, b, td): # Solid pulse of custom colors.
167     n = np.n
168     for k in range(128):
169         for i in range(n):
170             np[i] = (r, g, b)
171         np.write()
172         time.sleep_ms(td)
173         for j in range(n):
174             np[j] = (0, 0, 0)
175         np.write()
176         time.sleep_ms(td)
177     Blank() # Goto Function
178 
179 # Random Number Generator Function
180 # --------------------------------
181 def randrange(min_value, max_value):
182     magnitude = abs(max_value - min_value)
183     randbytes = uos.urandom(4)
184     offset = int((randbytes[3] << 24) | (randbytes[2] << 16) | (randbytes[1] << 8) |
    randbytes[0])
185     offset %= (magnitude+1)  # Offset by one to allow max_value to be included.
186     return min_value + offset
187 
188 def Random(r, g, b, td): # Random Chase colors.
189     n = np.n
190     for i in range(256):
191         r = randrange(0, 255)
192         g = randrange(0, 255)
193         b = randrange(0, 255)
194         #print("r = " + str(r)) # Activate next couple lines for debug
195         #print("g = " + str(g))
```

```
196             #print("b = " + str(b))
197             for j in range(n):
198                 np[j] = (0, 0, 0)
199             np[i % n] = (r, g, b)
200             np.write()
201             time.sleep_ms(td)
202         Blank() # Goto Function
203 
204     def theaterChase(r, g, b, td):
205          # Movie theater light style chaser animation. Custom Color
206         n = np.n
207         for j in range(128):
208             for q in range(2):
209                 for i in range(0, n, 2):
210                     np[i+q] = (r, g, b)
211                     np.write()
212                 time.sleep_ms(td)
213                 for i in range(0, n, 2):
214                     np[i+q] = (0, 0, 0)
215                     np.write()
216         Blank() # Goto Function
217 
218     def wheel(pos):
219         # Generate rainbow colors across 0-255 positions
220         if pos < 85:
221             return (pos * 3, 255 - pos * 3, 0)
222         elif pos < 170:
223             pos -= 85
224             return (255 - pos * 3, 0, pos * 3)
225         else:
226             pos -= 170
227             return (0, pos * 3, 255 - pos * 3)
228 
229     def rainbow(r, g, b, td):
230         # Draw rainbow that fades across all pixels at once
231         n = np.n
232         for j in range(256 * 4):
233             for i in range(n):
234                 r, g, b = wheel((i+j) & 255) # Position Function
235                 np[i] = (r, g, b)
236                 np.write()
237         time.sleep_ms(td)
238         Blank() # Goto Function
239 
240     def rainbowCycle(r, g, b, td):
241         # Draw rainbow that uniformly distributes itself across all pixels
242         n = np.n
243         for j in range(256 * 4):
244             for i in range(n):
245                 r, g, b = wheel((int(i * 256 / n) + j) & 255) # Position Function
246                 np[i] = (r, g, b)
247                 np.write()
248         time.sleep_ms(td)
249         Blank()# Goto Function
250 
251     def theaterChaseRainbow(r, g, b, td):
252         # Rainbow movie theater light style chaser animation
253         n = np.n
254         for j in range(120):
255             for q in range(2):
256                 for i in range(0, n, 2):
257                     r, g, b = wheel((i+j) % 255)
258                     np[i+q] = (r, g, b)
259                     np.write()
260                 time.sleep_ms(td)
261                 for i in range(0, n, 2):
262                     np[i+q] = (0, 0, 0)
```

```
                np.write()
    Blank() # Goto Function

# Data Extrapolation Function
# ---------------------------
def extrapolate_data(content):
    content1 = content.split(',')
    for i in range (len(content1)): # Search over entire string array
        print("For Command Data String is:") # Used to debug
        print(i, content1[i]) # Print all string data as separate elements
    # Reformat String Data to (Integer & String values)
    menu = str(content1[0]) # string menu choice
    Red = int(content1[1]) # Integer for red
    Green = int(content1[2]) # Integer for green
    Blue = int(content1[3]) # Integer for blue
    tdelay = int(content1[4]) # Integer for time delay
    print("")
    print("Final Global Values as Strings and Integers") # Used for debug
    print("Menu String Option = " + menu)
    print("Red Color Integer = " + str(Red))
    print("Green Color Integer = " + str(Green))
    print("Blue Color Integer = " + str(Blue))
    print("Time Delay Integer = " + str(tdelay))
    if (menu == "blank"):  # Goto Function
        print("Menu Choice is Blank(off)") # Used for debug
        L1 = "Main Menu"
        L2 = " "
        L3 = "Choice Is:"
        L4 = " "
        L5 = "Blank(Off)"
        L6 = " "
        delayx = 0.5
        Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display Function
        Blank()
    elif (menu == "solid"):  # Goto Function
        print("Menu Choice is Custom Solid Pattern") # Used for debug
        L1 = "Main Menu"
        L2 = " "
        L3 = "Choice Is:"
        L4 = " "
        L5 = "Custom Solid"
        L6 = "Pattern"
        delayx = 0.5
        Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display Function
        Solid(Red, Green, Blue, tdelay)
    elif (menu == "chase"):  # Goto Function
        print("Menu Choice is Custom Chase Pattern") # Used for debug
        L1 = "Main Menu"
        L2 = " "
        L3 = "Choice Is:"
        L4 = " "
        L5 = "Custom Chase"
        L6 = "Pattern"
        delayx = 0.5
        Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display Function
        Chase(Red, Green, Blue, tdelay)
    elif (menu == "random"):  # Goto Function
        print("Menu Choice is Random Color Chase Pattern") # Used for debug
        L1 = "Main Menu"
        L2 = " "
        L3 = "Choice Is:"
        L4 = " "
        L5 = "Random Color"
        L6 = "Chase Pattern"
        delayx = 0.5
        Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display Function
        Random(Red, Green, Blue, tdelay)
```

```
330         elif (menu == "fade"):  # Goto Function
331             print("Menu Choice is White Fade Pattern") # Used for debug
332             L1 = "Main Menu"
333             L2 = " "
334             L3 = "Choice Is:"
335             L4 = " "
336             L5 = "White Fade"
337             L6 = "Pattern"
338             delayx = 0.5
339             Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display Function
340             Fade(Red, Green, Blue, tdelay)
341         elif (menu == "bounce"):  # Goto Function
342             print("Menu Choice is Custom Bounce Pattern") # Used for debug
343             L1 = "Main Menu"
344             L2 = " "
345             L3 = "Choice Is:"
346             L4 = " "
347             L5 = "Custom Bounce"
348             L6 = "Pattern"
349             delayx = 0.5
350             Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display Function
351             Bounce(Red, Green, Blue, tdelay)
352         elif (menu == "tchase"):  # Goto Function
353             print("Menu Choice is Custom Theater Chase Pattern") # Used for debug
354             L1 = "Main Menu"
355             L2 = " "
356             L3 = "Choice Is:"
357             L4 = " "
358             L5 = "Custom Theater"
359             L6 = "Chase Pattern"
360             delayx = 0.5
361             Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display Function
362             theaterChase(Red, Green, Blue, tdelay)
363         elif (menu == "rain"):  # Goto Function
364             print("Menu Choice is Rainbow Color Pattern") # Used for debug
365             L1 = "Main Menu"
366             L2 = " "
367             L3 = "Choice Is:"
368             L4 = " "
369             L5 = "Rainbow Color"
370             L6 = "Pattern"
371             delayx = 0.5
372             Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display Function
373             rainbow(Red, Green, Blue, tdelay)
374         elif (menu == "raincyc"):  # Goto Function
375             print("Menu Choice is Rainbow Color Cycle Pattern") # Used for debug
376             L1 = "Main Menu"
377             L2 = " "
378             L3 = "Choice Is:"
379             L4 = " "
380             L5 = "Rainbow Color"
381             L6 = "Cycle Pattern"
382             delayx = 0.5
383             Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display Function
384             rainbowCycle(Red, Green, Blue, tdelay)
385         elif (menu == "raintheater"):  # Goto Function
386             print("Menu Choice is Rainbow Color Theater Chase Pattern") # Used for debug
387             L1 = "Main Menu"
388             L2 = " "
389             L3 = "Choice Is:"
390             L4 = " "
391             L5 = "Rainbow Color"
392             L6 = "Theater Chase"
393             delayx = 0.5
394             Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display Function
395             theaterChaseRainbow(Red, Green, Blue, tdelay)
396         else:
```

```
397             print("Menu Choice is the default state Blank") # Used for debug
398             Blank()
399
400     # Function to Start Client Server
401     # -------------------------------
402     def _httpHandlerxmasPost(httpClient, httpResponse):
403         content = httpClient.ReadRequestContent()  # Read data
404         print("ESP32 Response Is: " + str(content)) # Used to check HTML POST Data
405         print("")
406         try:
407             newdata = content.decode('UTF-8') # Convert byte data to string
408             httpResponse.WriteResponseOk()  # Write "OK" Header Information
409         except Exception as e:
410                                 print(e)
411         finally:
412             extrapolate_data(newdata) # Send Decoded HTML POST Data and goto Neopixel
                Function
413
414     # Bottom Page Script
415     # ------------------
416     routeHandlers = [ ( "/xmaslights_3", "POST",  _httpHandlerxmasPost ) ] # Route Path
417     srv = MicroWebSrv(routeHandlers=routeHandlers, webPath='/www/')
418     srv.Start(threaded=False) # No Threads
419
```

```
"""
Wifi Manager Routine
====================
"""

# Import Modules
# --------------
import network
import socket
import ure
import time
import os
import machine
from network import WLAN
from machine import Pin, I2C
from time import sleep
from ssd1306 import SSD1306_I2C

# Constants
# ---------
ap_ssid = "ESP32 WifiManager"
ap_password = "password"
ap_authmode = 3  # WPA2
WIFI_LED_PIN = 25 # ESP32 on board LED
OLED_RST_PIN = 16 # ESP32 OLED display Reset pin

# Initial Setup
# -------------
led = Pin(WIFI_LED_PIN, Pin.OUT, value=0) # Sets up LED and starts
OLED_RST = Pin(OLED_RST_PIN, Pin.OUT, value=1) # Sets up OLED RST to high
i2c = I2C(sda = Pin(4), scl = Pin(15)) # ESP32 OLED I2C Pins
display = SSD1306_I2C(128, 64, i2c) # ESP32 I2C display resolution

# Network Data File
# -----------------
NETWORK_PROFILES = 'wifi.dat'

# Access Point and Staion Network Class Objects
# ---------------------------------------------
ap = WLAN(network.AP_IF)
sta = WLAN(network.STA_IF)

# Generic Function to Display 6 lines on OLED
# -------------------------------------------
def Display_Stat(line1,line2,line3,line4,line5,line6,delaytime):
    display.fill(0) # Clear Previous Text
    display.text(line1,0,0,1) #Line 1
    display.text(line2,0,10,1) # Line 2
    display.text(line3,0,20,1) # Line 3
    display.text(line4,0,30,1) #Line 4
    display.text(line5,0,40,1) # Line 5
    display.text(line6,0,50,1) # Line 6
    display.show() # Display New Text
    sleep(delaytime) # Delay secs

# Function to Reboot the ESP32
# ----------------------------
def reboot():
    print("ESP32 is Rebooting....")
    L1 = "SSID and Pass"
    L2 = "Is Saved in a"
    L3 = ""
    L4 = "Configuration"
    L5 = "File"
    L6 = ""
    delayx = 3
    Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
```

```
68      machine.reset() # Hardware reset command
69
70  server_socket = None
71
72  # Function to return a working STA instance or "None" on the Main.py Program
73  # ----------------------------------------------------------------------------
74  def get_connection():
75      L1 = "Hello"
76      L2 = "Starting Wi-Fi"
77      L3 = "Let's Go..."
78      L4 = "Attemping to"
79      L5 = "Connect....."
80      L6 = "Please Wait"
81      delayx = 3
82      Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
83      if sta.isconnected(): # First check if there already is any connection:
84          led.value(1) # Turn on LED
85          return sta
86      connected = False
87      try: # ESP32 WiFi takes time, wait a bit and try once more
88          time.sleep(5) # 5 second delay
89          if sta.isconnected():
90              led.value(1) # Turn on LED
91              return sta
92          # Read known network profiles from file
93          profiles = read_profiles()
94          # Search WiFis networks in range
95          sta.active(True)
96          networks = sta.scan()
97          AUTHMODE = {0: "open", 1: "WEP", 2: "WPA-PSK", 3: "WPA2-PSK", 4: "WPA/WPA2-PSK"}
98          for ssid, bssid, channel, rssi, authmode, hidden in sorted(networks, key=lambda
            x: x[3], reverse=True):
99              ssid = ssid.decode('utf-8')
100             encrypted = authmode > 0
101             print("ssid: %s chan: %d rssi: %d authmode: %s" % (ssid, channel, rssi,
                AUTHMODE.get(authmode, '?')))
102             if encrypted:
103                 if ssid in profiles:
104                     password = profiles[ssid]
105                     connected = do_connect(ssid, password)
106                 else:
107                     print("skipping unknown encrypted network")
108             else:  # open
109                 connected = do_connect(ssid, None) # Unencrypted network
110             if connected:
111                 break
112     except OSError as e:
113         print("exception", str(e))
114
115     # start web server for connection manager:
116     if not connected:
117         connected = start()
118     return sta if connected else None
119
120 # Function to read Network SSID Profiles
121 # --------------------------------------
122 def read_profiles():
123     with open(NETWORK_PROFILES) as f:
124         lines = f.readlines()
125     profiles = {}
126     for line in lines:
127         ssid, password = line.strip("\n").split(";")
128         profiles[ssid] = password
129     return profiles
130
131 # Function to Write Network SSID Profiles
132 # ---------------------------------------
```

```
133 def write_profiles(profiles):
134     lines = []
135     for ssid, password in profiles.items():
136         lines.append("%s;%s\n" % (ssid, password))
137     with open(NETWORK_PROFILES, "w") as f:
138         f.write(''.join(lines))
139 
140 # Function to connect WiFi
141 # ------------------------
142 def do_connect(ssid, password):
143     sta.active(True)
144     if sta.isconnected():
145         return None
146     print('Trying to connect to %s...' % ssid)
147     sta.connect(ssid, password)
148     attempt = 0
149     for retry in range(11):
150         connected = sta.isconnected()
151         if connected:
152             break
153         L1 = "Trying to"
154         L2 = "Connect....."
155         L3 = ""
156         L4 = "Attempt: " + str(attempt)
157         L5 = ""
158         L6 = ""
159         delayx = 1
160         Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
161         attempt += 1
162         print('.', end='')
163     if connected:
164         print('\nConnected. Network config: ', sta.ifconfig())
165         led.value(1) # Turn on LED
166         #ap.active(False) # Turn off AP Mode
167         L1 = "WiFi Connected!"
168         L2 = "IP Address Is:"
169         L3 = str(sta.ifconfig())
170         L4 = ""
171         L5 = "Returning to"
172         L6 = "Main Program"
173         delayx = 3
174         Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
175     else:
176         print('\nFailed. Not Connected to: ' + ssid)
177         led.value(0) # Turn off LED
178         L1 = "Wi-Fi"
179         L2 = "Connection"
180         L3 = "Failed"
181         L4 = ""
182         L5 = ""
183         L6 = ""
184         delayx = 3
185         Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
186     return connected
187 
188 # Function to Send Web Page Total Response
189 # ----------------------------------------
190 def send_response(client, payload, status_code=200):
191     client.sendall("HTTP/1.0 {} OK\r\n".format(status_code))
192     client.sendall("Content-Type: text/html\r\n")
193     client.sendall("Content-Length: {}\r\n".format(len(payload)))
194     client.sendall("\r\n")
195     if len(payload) > 0:
196         client.sendall(payload)
197 
198 # Function to Display HTML Web Browser
199 # ------------------------------------
```

```
def handle_root(client):
    sta.active(True)
    ssids = sorted(ssid.decode('utf-8') for ssid, *_ in sta.scan())
    response = []
    response.append("""\
        <html>
          <body bgcolor="#00FFFF">
            <h1 style="color: #5e9ca0; text-align: center;">
                <span style="color: #ff0000;">
                    ESP32 WiFi Setup
                </span>
            </h1>
            <form action="configure" method="post">
                <table style="margin-left: auto; margin-right: auto;">
                    <tbody>
    """)
    for ssid in ssids:
        response.append("""\
                        <tr>
                            <td colspan="2">
                                <input type="radio" name="ssid" value="{0}" />{0}
                            </td>
                        </tr>
        """.format(ssid))

    response.append("""\
                        <tr>
                            <td>Password:</td>
                            <td><input name="password" type="password" /></td>
                        </tr>
                    </tbody>
                </table>
                <p style="text-align: center;">
                    <input type="submit" value="Submit" />
                </p>
            </form>
            <p> </p>
            <hr />
            <h5>
                <span style="color: #ff0000;">
                    Your ssid and password information will be saved into the
                    "%(filename)s" file in your ESP module for future usage.
                    Be careful about security!
                </span>
            </h5>
            <hr />
          </body>
        </html>
    """ % dict(filename=NETWORK_PROFILES))
    send_response(client, "\n".join(response))

# Function to Get SSID & Password
# -------------------------------
def handle_configure(client, request):
    match = ure.search("ssid=([^&]*)&password=(.*)", request)
    if match is None:
        send_response(client, "Parameters not found", status_code=400)
        return False
    try:
        ssid = match.group(1).decode("utf-8").replace("%3F", "?").replace("%21", "!")
        password = match.group(2).decode("utf-8").replace("%3F", "?").replace("%21", "!")
    except:
        ssid = match.group(1).replace("%3F", "?").replace("%21", "!")
        password = match.group(2).replace("%3F", "?").replace("%21", "!")
    if len(ssid) == 0:
        send_response(client, "SSID must be provided", status_code=400)
        return False
```

```
267         if do_connect(ssid, password):
268             response = """\
269                 <html>
270                     <center>
271                         <br><br>
272                         <h1 style="color: #5e9ca0; text-align: center;">
273                             <span style="color: #ff0000;">
274                                 ESP32 successfully connected to WiFi network %(ssid)s.
275                             </span>
276                         </h1>
277                         <br><br>
278                     </center>
279                 </html>
280             """ % dict(ssid=ssid)
281             send_response(client, response)
282             try:
283                 profiles = read_profiles()
284             except OSError:
285                 profiles = {}
286             profiles[ssid] = password
287             write_profiles(profiles)
288             return True
289         else:
290             response = """\
291                 <html>
292                     <center>
293                         <h1 style="color: #5e9ca0; text-align: center;">
294                             <span style="color: #ff0000;">
295                                 ESP32 could not connect to WiFi network %(ssid)s.
296                             </span>
297                         </h1>
298                         <br><br>
299                         <form>
300                             <input type="button" value="Go back!"
                                onclick="history.back()"></input>
301                         </form>
302                     </center>
303                 </html>
304             """ % dict(ssid=ssid)
305             send_response(client, response)
306             return False
307 
308     # Path Error Function
309     # -------------------
310     def handle_not_found(client, url):
311         send_response(client, "Path not found: {}".format(url), status_code=404)
312 
313     # Web Socket Close Function
314     # -------------------------
315     def stop():
316         global server_socket
317         if server_socket:
318             server_socket.close()
319             server_socket = None
320 
321     # Web Server Start Function
322     # -------------------------
323     def start(port=80):
324         global server_socket
325         addr = socket.getaddrinfo('0.0.0.0', port)[0][-1]
326         stop()
327         sta.active(True)
328         ap.active(True)
329         ap.config(essid=ap_ssid, password=ap_password, authmode=ap_authmode)
330         server_socket = socket.socket()
331         server_socket.bind(addr)
332         server_socket.listen(1)
```

```
333         print('Connect to WiFi ssid ' + ap_ssid + ', default password: ' + ap_password)
334         print('and access the ESP via your favorite web browser at 192.168.4.1.')
335         print('Listening on:', addr)
336         L1 = "Server Started"
337         L2 = "Connect to AP"
338         L3 = "Type 'password'"
339         L4 = "Open Web Browser"
340         L5 = "Type in Address"
341         L6 = "192.168.4.1"
342         delayx = 3
343         Display_Stat(L1,L2,L3,L4,L5,L6,delayx) # Goto Display function
344         while True:
345             if sta.isconnected():
346                 return True
347             client, addr = server_socket.accept()
348             print('client connected from', addr)
349             try:
350                 client.settimeout(5.0)
351                 request = b""
352                 try:
353                     while "\r\n\r\n" not in request:
354                         request += client.recv(512)
355                 except OSError:
356                     pass
357                 print("Request is: {}".format(request))
358                 if "HTTP" not in request:
359                     # skip invalid requests
360                     continue
361                 # version 1.9 compatibility
362                 try:
363                     url = ure.search("(?:GET|POST) /(.*?)(?:\\?.*?)? HTTP",
                        request).group(1).decode("utf-8").rstrip("/")
364                 except:
365                     url = ure.search("(?:GET|POST) /(.*?)(?:\\?.*?)? HTTP",
                        request).group(1).rstrip("/")
366                 print("URL is {}".format(url))
367                 if url == "":
368                     handle_root(client)
369                 elif url == "configure":
370                     handle_configure(client, request)
371                 else:
372                     handle_not_found(client, url)
373             finally:
374                 client.close()
375                 #reboot() # Goto Reboot function
376
377
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Light Control Functions Start Here -->
    <title>Holiday Lights</title>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
    shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css "
    integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
    crossorigin="anonymous">
  <style>
    .slidecontainer {
     width: 100%;
    }

    .slider {
    -webkit-appearance: none;
    width: 100%;
    height: 15px;
    border-radius: 5px;
    background: Silver;
    outline: none;
    opacity: 0.7;
    -webkit-transition: .2s;
    transition: opacity .2s;
    }

    .slider:hover {
    opacity: 1;
    }

    .slider::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    width: 25px;
    height: 25px;
    border-radius: 50%;
    background: black;
    cursor: pointer;
    }

    .slider::-moz-range-thumb {
    width: 25px;
    height: 25px;
    border-radius: 50%;
    background: #4CAF50;
    cursor: pointer;
    }
    </style>

    <style>
    #main {
     width: 100%;
     height: 20%;
     border-style: solid;
     border-color: black;
     }
    </style>
    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js "
    integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
```

```
      crossorigin="anonymous"></script>
      <script
      src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
      integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
      crossorigin="anonymous"></script>
      <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
      integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
      crossorigin="anonymous"></script>
    </head>
  <br>
   <body style="background-color:snow;" onload="newload()">
    <div class="container">
      <div class="jumbotron">
      <h1><b>ESP32 Holiday Christmas Lights</b></h1>
      <p>Control your Holiday Lights using MicroPython and the ESP32 micro-controller
      connected to Neopixels and your WiFi.</p>
      <p>The Web Page is Automatically Updated every 20 seconds, or you can Use the
      Update Button to Manually Update the Web Page.</p>
      <p>Since the Web Server contains a "blocking call" when used with the ESP32, your
      new selection can take up to 20 seconds to load.</p>
      <p>Note - Any Menu Function that has "Custom", the colors can be controlled with
      the "RGB" Sliders.</p>
      </div>
    </div>
  <br>
    <div class="container" align="center">
      <div class="form-group">
        <label for="animation"><h1><b>Animation</b></h1></label>
          <select class="form-control" onchange="mnuFunction()" id="animation">
             <option value="solid">Custom Solid Pattern</option>
             <option value="chase">Custom Chase Pattern</option>
             <option value="bounce">Custom Bounce Pattern</option>
             <option value="random">Random Color Chase Pattern</option>
             <option value="fade">White Fade Pattern</option>
             <option value="tchase">Custom Theater Chase Pattern</option>
             <option value="rain">Rainbow Color Pattern</option>
             <option value="raincyc">Rainbow Color Cycle Pattern</option>
             <option value="raintheater">Rainbow Color Theater Chase
             Pattern</option>
             <option value="blank">Blank (off)</option>
           </select>
      </div>
    </div>
    <script>
    // Global variables
      var new2 = 'solid';
      var r = 127;
      var g = 127;
      var b = 127;
      var sldr = 50;
      var data1;
    </script>
    <script>
     // Initial Menu Choice
     var init1 = document.getElementById("animation").value;
      console.log("Initial Menu Choice = " + init1); // displays initial menu choice
    // Choose Menu Function
     function mnuFunction() {
      new2 = document.getElementById("animation").value;
      console.log("Menu Selection = " + new2); // Displays Menu = The Choice
    }
    </script>
  <br>
   <div class="container">
    <!-- RBG Color Sliders -->
    <div align="center" ><h1><b>Custom Color Mix Control (Red-Green-Blue)</b></h1></div>
      <div data-role="content" class = "main" id = "main">
```

```
    <div data-role="fieldcontain" data-controltype="slider" class="redSlider">
        <label for="red" color="red">Red</label>
        <input id="red" type="range" name="redSlider" value="127" min="0" max="255"
        data-highlight="true" data-mini="true" onchange = "changeBackground()" />
    </div>
    <div data-role="fieldcontain" data-controltype="slider" class="greenSlider">
        <label for="green">Green</label>
        <input id="green" type="range" name="greenSlider" value="127" min="0"
        max="255" data-highlight="true" data-mini="true" onchange =
        "changeBackground()" />
    </div>
    <div data-role="fieldcontain" data-controltype="slider" class="blueSlider">
        <label for="blue">Blue</label>
        <input id="blue" type="range" name="blueSlider" value="127" min="0" max="255"
        data-highlight="true" data-mini="true" onchange = "changeBackground()" />
    </div>
</div>
  <script>
    // Sets Slider background to initial value upon startup
    var r1 = parseInt(document.getElementById('red').value, 10),
        g1 = parseInt(document.getElementById('green').value, 10),
        b1 = parseInt(document.getElementById('blue').value, 10);
    document.getElementById('main').style.backgroundColor = 'rgb('+r1+', '+g1+',
    '+b1+')';
    console.log("Initial RGB = " + 'rgb('+r1+', '+g1+', '+b1+')'); // displays RBG
    Initial Value
    // Sets Slider background to new value on a change
    function changeBackground () {
        r = parseInt(document.getElementById('red').value, 10),
        g = parseInt(document.getElementById('green').value, 10),
        b = parseInt(document.getElementById('blue').value, 10);

    console.log("Red = " + r); // displays red value change
    console.log("Green = " + g); // displays green value change
    console.log("Blue = " + b); // displays blue value change

    document.getElementById('main').style.backgroundColor = 'rgb('+r+', '+g+', '+b+')';
    console.log("RGB = " + 'rgb('+r+', '+g+', '+b+')'); // displays RBG composite change
    }
  </script>
<br>
<br>
    <!-- Time Delay Slider -->
    <div align="center" ><h1><b>Delay Time (mS)</b></h1></div>
      <p><h2 style="color:darkmagenta;"><b>Time Slider</b></h2></p>
        <div class="slidecontainer">
        <input type="range" min="1" max="100" value="50" class="slider" id="timeControl">
        <p>Value: <span id="timedemo"></span></p>
    </div>
    <script>
      // Reads initial value of time slider and all further changes in position
      var slider3 = document.getElementById("timeControl");
      var output3 = document.getElementById("timedemo");
        output3.innerHTML = slider3.value;
        console.log("Initial Time = " + slider3.value); // displays initial time value

      slider3.oninput = function() {
        output3.innerHTML = this.value;
           sldr = this.value;
        console.log("Output Time = " + this.value); // displays changing time
        incremental values
      }
    </script>
  </div>
<br>
   <!-- Update Button -->
    <div class="container" align="center" >
```

```
      <button class="btn btn-lg btn-primary" type="button" onclick="updtFunction()"
      id="update">Update Lights</button>
    </div>
    <script>
    // Update button function
    function updtFunction() {
      var new1 = document.getElementById("update").innerHTML;
      console.log("Button Status = " + new1); // Displays Update Button = Update Lights
      data1 = ''+new2+', '+r+', '+g+', '+b+', '+sldr+''; // Data Stream to be sent
      console.log("Manual Data Stream = " + data1); // Print data stream
      //var dataJSON = '{"red":' + e.r + ', "green":' + e.g + ', "blue":' + e.b + '}';
      var xhttp = new XMLHttpRequest();
      xhttp.open('POST', 'http://' + window.location.hostname + '/xmaslights_3', true);
      //xhttp.setRequestHeader('Content-type', 'application/json');
      //xhttp.send(colorJSON);
      xhttp.setRequestHeader('Content-type', '"application/x-www-form-urlencoded');
      xhttp.send(data1);
    }
    </script>
    <script>
      // Automatic Update Function
     function newload() {
       data1 = ''+new2+', '+r+', '+g+', '+b+', '+sldr+''; // Data Stream to be sent
       console.log("Auto Data Stream = " + data1); // Print data stream
       var xhttp = new XMLHttpRequest();
       xhttp.open('POST', 'http://' + window.location.hostname + '/xmaslights_3', true);
       xhttp.setRequestHeader('Content-type', '"application/x-www-form-urlencoded');
       xhttp.send(data1);
       // Start new timer (20 seconds)
       timeoutID = setTimeout('newload()', 20000);
}
    </script>
   <br>
   </body>
  </html>
```