# A logo with blue and red lines AI-generated content may be incorrect.

# NATIONAL TEXTILE UNIVERSITY

# FAISALABAD

# ASSIGNMENT

# INTERNET OF THINGS

# CREATING A WEB SERVER

# GROUP MEMBERS :

# FATIMA RIAZ 1342

# MARRIAM SAJJAD 1353

# SUMAIYA SHEHZADI 1376

# 1.Blink\_test.py

from machine import Pin from neopixel import NeoPixel import time

pin = Pin(48, Pin.OUT) # Change this if needed np = NeoPixel(pin, 1)

def test\_rgb():

colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 255), (0, 0, 0)] for color in colors: np[0] = color np.write() print(f"LED set to: {color}" time.sleep(1)

test\_rgb()

**Explaination:**

This MicroPython script controls a single NeoPixel (WS2812) RGB LED using an ESP32 microcontroller. It imports necessary modules, including Pin for GPIO control and NeoPixel for LED operation. The Pin(48, Pin.OUT) line sets up GPIO pin 48 as an output for the LED. A NeoPixel object (np) is created to control one LED. The test\_rgb() function cycles through five colors—red, green, blue, white, and off—by setting np[0] to each color, updating the LED with np.write(), and waiting for 1 second between changes. The script prints the active color to the console. However, there is a syntax error: the print statement is missing a closing parenthesis.

# 2.DHT11\_test.py

import dht import machine import time

# Define the GPIO pin where DHT11 is connected

DHT\_PIN = 4 # Change if using a different pin

# Initialize DHT11 sensor sensor = dht.DHT11(machine.Pin(DHT\_PIN))

while True:

try:

sensor.measure() # Get readings temp = sensor.temperature() # Read temperature (Celsius) hum = sensor.humidity() # Read humidity (%)

print(f"Temperature: {temp}°C, Humidity: {hum}%") except Exception as e:

print("Error reading DHT11:", e) time.sleep(2) # Wait 2 seconds before the next reading

**Description:**

This MicroPython script reads temperature and humidity data from a **DHT11** sensor using an ESP32 or ESP8266 microcontroller.

1. It imports necessary modules: dht for sensor communication, machine for GPIO control, and time for delays.
2. The **DHT11 sensor** is connected to GPIO pin **4** (can be changed if needed).
3. A DHT11 sensor object is created using sensor = dht.DHT11(machine.Pin(DHT\_PIN)).
4. Inside an infinite while True loop:
   * The sensor is **measured** using sensor.measure().
   * **Temperature** in Celsius is retrieved with sensor.temperature().
   * **Humidity** percentage is read with sensor.humidity().
   * The values are **printed** in a formatted string.
   * If an error occurs, it is caught in an except block and displayed.  A **2-second delay** is added before the next reading.

This script continuously monitors temperature and humidity, making it ideal for simple weather or indoor climate monitoring projects.

# Main.py

import network import socket import dht import machine import ssd1306 # OLED library

# WiFi Configuration

SSID = "Sweet home"

PASSWORD = "Ahmad456"

# Initialize WiFi in Station Mode

wifi = network.WLAN(network.STA\_IF)

wifi.active(True) wifi.connect(SSID, PASSWORD)

while not wifi.isconnected(): pass # Wait for connection print("Connected! IP Address:", wifi.ifconfig()[0])

# Initialize DHT11 Sensor dht\_pin = machine.Pin(4) # GPIO4 dht\_sensor = dht.DHT11(dht\_pin) # Initialize OLED Display (SSD1306) i2c = machine.SoftI2C(scl=machine.Pin(9), sda=machine.Pin(8)) oled = ssd1306.SSD1306\_I2C(128, 64, i2c) # Initialize RGB LED (Built-in NeoPixel) from neopixel import NeoPixel # Import NeoPixel library rgb\_pin = machine.Pin(48, machine.Pin.OUT) # Change pin if needed rgb\_led = NeoPixel(rgb\_pin, 1) # Only 1 LED on ESP32

# Function to Set RGB Color def set\_rgb\_color(r, g, b): rgb\_led[0] = (r, g, b) rgb\_led.write()

# Function to Read Temperature & Humidity def get\_sensor\_data(): dht\_sensor.measure() temp = dht\_sensor.temperature() hum = dht\_sensor.humidity() return temp, hum # HTML Web Page def generate\_webpage(temp, hum):

return """

<!DOCTYPE html>

<html>

<head>

<title>ESP32 Web Server</title>

<style> body { font-family: Arial; text-align: center; } .container { width: 80%%; margin: auto; } input { padding: 10px; font-size: 16px; } button { padding: 10px; font-size: 18px; margin-top: 10px; }

</style>

</head>

<body>

<div class="container">

<h1>ESP32 RGB LED & Sensor Web Server</h1>

<h2>Temperature: """ + str(temp) + """°C</h2>

<h2>Humidity: """ + str(hum) + """%</h2>

<h3>Set RGB Color</h3>

<form action="/" method="GET">

<input type="number" name="r" placeholder="Red (0-255)">

<input type="number" name="g" placeholder="Green (0-255)">

<input type="number" name="b" placeholder="Blue (0-255)">

<button type="submit">Set Color</button>

</form>

</div>

</body>

</html>

"""

# Start Web Server server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) server.bind(('', 80)) server.listen(5)

print("Web Server Started! Access it via browser.") while True:

conn, addr = server.accept() print("Client connected from", addr) request = conn.recv(1024).decode()

# Parse RGB Values if "GET /?" in request:

try:

params = request.split(" ")[1].split("?")[1].split("&") r = int(params[0].split("=")[1]) g = int(params[1].split("=")[1]) b = int(params[2].split("=")[1])

set\_rgb\_color(r, g, b)

oled.fill(0) oled.text("Color: R{} G{} B{}".format(r, g, b), 0, 20) oled.show() except: pass

# Get Sensor Data temp, hum = get\_sensor\_data()

# Send Webpage Response response = generate\_webpage(temp, hum) conn.send("HTTP/1.1 200 OK\nContent-Type: text/html\n\n" + response) conn.close()

**Description:**

**Explanation of Main.py**

This MicroPython script runs on an **ESP32** and does the following: 1. **WiFi Connection**

* It connects the ESP32 to a WiFi network using **Station mode** (STA\_IF).
* The WiFi credentials (SSID = "Sweet home", PASSWORD = "Ahmad456") are used to establish a connection.
* The script waits (while not wifi.isconnected()) until the ESP32 gets an IP address.

## DHT11 Temperature & Humidity Sensor

* The **DHT11** sensor is connected to GPIO4.
* It measures **temperature** and **humidity** values for display and web server updates.

## OLED Display (SSD1306, I2C)

* The OLED display is connected using **SoftI2C** (SCL: GPIO9, SDA: GPIO8).
* It is used to show temperature, humidity, and RGB LED status.

## RGB LED Control (NeoPixel)

* A **single NeoPixel RGB LED** is connected to **GPIO48**.
* The function set\_rgb\_color(r, g, b) allows changing LED colors dynamically.

## Web Server (Socket-based)

 A simple **HTTP web server** is created using the socket module, listening on port 80.  The server responds with an **HTML webpage** displaying:

 Temperature & Humidity readings.  A form for users to set RGB LED colors (Red, Green, Blue).

## Handling Web Requests

* The script checks for GET /?r=...&g=...&b=... requests in the URL.
* Extracts r, g, b values and updates the **RGB LED** accordingly.
* Displays the RGB color on the **OLED screen**.

## Client Connection Handling

* The server accepts client requests (conn, addr = server.accept()).  Extracts **sensor data**, processes requests, and sends an **HTML response**.
* The connection is then closed (conn.close()).

**Features:**

## ✅ Real-time Temperature & Humidity Monitoring ✅ Web-based RGB LED Control

✅ **OLED Display Updates** ✅ **WiFi Connectivity** **4. Wifi\_Connection.py** import network import time

WIFI\_SSID = "Sweet home" WIFI\_PASS = "Ahmad456"

def connect\_wifi(): wlan = network.WLAN(network.STA\_IF)

wlan.active(True) wlan.connect(WIFI\_SSID, WIFI\_PASS) for \_ in range(10): if wlan.isconnected():

print("Connected! IP:", wlan.ifconfig()[0]) return True time.sleep(1) print("Failed to connect.") return False

connect\_wifi()

**Description:**

This **MicroPython** script connects an **ESP32** to a WiFi network using **Station Mode (STA\_IF)**. It initializes the **WiFi interface**, activates it, and attempts to connect using the provided **SSID** and **password**. The script checks the connection status for up to **10 seconds**, printing the **assigned IP address** if successful. If it fails, it prints "Failed to connect." and stops. The connect\_wifi() function is called automatically when the script runs, ensuring **stable connectivity** before executing other tasks like **web servers or IoT functions**.