Lab 9
WiFi connected Adafruit Feather Bluefruit Sense
using HTTP communication

Part 1 - Prepare the Feather for Wifi Connectivity

In this lab we will remove the need for an intermediary gateway (Bluefruit App on a cellphone) and use the Adafruit Airlift Featherwing to connect directly to a WiFi access point. Time to add a few more libraries to the lib folder on the Feather Sense. Your instructor will provide you the following libraries:

- **adafruit_esp32spi**
- **adafruit_requests.mpy**
- **adafruit_bus_device**

Copy the libraries into the lib folder on your Feather Sense.

The Featherwing Airlift has likely been sitting around waiting to be used. Start with a hardware check to make sure the Feather Sense can communicate with the Airlift. Copy the following code into the Mu editor and save it to the Feather Sense as code.py.

```
import board
import busio
from digitalio import DigitalInOut

from adafruit_esp32spi import adafruit_esp32spi

# assign SPI communication pins for the ESP32 Airlift
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready,
esp32_reset)

print("Running ESP32 SPI hardware test...")

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version.decode('utf-8'))
print("MAC addr:", [hex(i) for i in esp.MAC_address_actual])

for ap in esp.scan_networks():
    print("\t%s\t\tRSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))

print("Done!")
```

If the Feather Sense and the Airlift can communicate correctly you should immediately see some output in the Mu serial window.

What version of the firmware is running on the Airlift?

What is the MAC address of your Airlift?

List the access points your Airlift can currently “see”?

Which access point has the strongest signal?

Study the code. Add header information and inline comments as you trace your way through the code. Save the file as “lastname_lab9_part1.py”.

Show your working code on the Feather to your instructor.

Part 2 - Connect the Feather to the Internet

Connecting to access points and the internet usually (always?) includes the use of login id’s and passwords. To keep prying eyes away from your sensitive information, a specific circuitpython file, “secrets.py” will be used. This file will contain access point login information and other user specific information, such as AIO usernames and AIO keys. By keeping your “secrets” in a separate file, you can share your circuitpython code without accidentally giving away possible sensitive information.

Create a new CP file and copy the following code into it. Save it as “secrets.py”

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid' : 'access point name goes here',
    'password' : 'password to connect to access point goes here',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    }
```

The secrets file contains a python dictionary called secrets. The first key in the dictionary refers to the access point (ssid) your Feather will connect to for internet access. You saw the possible access points listed in part 1 of this lab. The second key value pair in the dictionary is for the password to connect to the ssid. We will add other key value pairs later. For now a third pair has been added for the timezone as an example.

Your instructor will provide you with proper credentials to create the secrets.py file correctly. Change the ssid and password values in the file and save it.

Now it is time to connect the Feather Sense to the internet. Copy the following code into a new Mu CP window and save it to the Feather as code.py.

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import board
import busio
from digitalio import DigitalInOut
import adafruit_requests as requests
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"

# AirLift Featherwing
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready,
esp32_reset)

requests.set_socket(socket, esp)

print("ESP32 SPI internet connection test...")

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version.decode("utf-8"))
print("MAC addr:", [hex(i) for i in esp.MAC_address_actual])

for ap in esp.scan_networks():
    print("\t%s\t\tRSSI: %d" % (str(ap["ssid"], "utf-8"), ap["rssi"]))

print("Connecting to AP...")
```

```
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
print(
    "IP lookup adafruit.com: %s" %
esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
)
print("Ping google.com: %d ms" % esp.ping("google.com"))

# esp._debug = True
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print("-" * 40)
print(r.text)
print("-" * 40)
r.close()

print()
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print("-" * 40)
print(r.json())
print("-" * 40)
r.close()

print("Done!")
```

In the Mu serial console you should see the code running just like it did in part 1 and then it attempts to connect to the access point you put in the secrets file. If it connects… the Feather starts getting information from around the internet!

What ip address was assigned to your Feather?

What is the ip address of adafruit.com?

How long did it take to ping google.com?

What text was fetched from TEXT_URL defined in the code?

What is the current price (rate) of bitcoin?

In this part of the lab you have not only connected to the internet but you also saw how to get the ip address of a hostname and how to ping a web server. The part 2 code also exposed you to some simple HTTP style commands using the adafruit_requests module.

Study the code. Add header information and inline comments as you trace your way through the code. Save the file as “lastname_lab9_part2.py”.

Show your working code on the Feather to your instructor. 

Part 3 - HTTP GET and POST

Two of the most basic HTTP commands are GET and POST. This part of the lab will show you a few more examples of these commands in action. We will make use of the website: https://httpbin.org This website provides a location for us to test GETting and POSTing some data. Copy the following code into a new editor tab in the Mu editor and save it to the Feather Sense as code.py.

```
# test adafruit_requests usage with an esp32spi_socket
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# AirLift Featherwing
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready,
esp32_reset)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
socket.set_interface(esp)
requests.set_socket(socket, esp)
```

```
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_GET_URL = "https://httpbin.org/get"
JSON_POST_URL = "https://httpbin.org/post"

print("Fetching text from %s" % TEXT_URL)
response = requests.get(TEXT_URL)
print("-" * 40)

print("Text Response: ", response.text)
print("-" * 40)
response.close()

# GET default JSON response from JSON_GET_URL
print("Fetching JSON data from %s" % JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print("-" * 40)

# print default JSON response
print("JSON Response: ", response.json())
print("-" * 40)
response.close()

# POST some random data to JSON_POST_URL
data = "cheese92"
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print("-" * 40)

# print the random data from the returned JSON
json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:", json_resp["data"])
print("-" * 40)
response.close()

# POST some random JSON data key:value pair to JSON_POST_URL
json_data = {"Temp": "67"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print("-" * 40)

# print the random JSON data key:value pair from the returned JSON
json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:", json_resp["json"])
print("-" * 40)

# Read Response's HTTP status code
print("Response HTTP Status Code: ", response.status_code)
print("-" * 40)

response.close()
```

Change the text data being POSTed and change the JSON data being POSTed to confirm that it works as expected.

Show your instructor your modified code's output.

Save the code as lastname_lab9_part3.py. Add header information and more inline comments.

CircuitPython provides an easy way to use simple HTTP commands but the examples shown all require a web server running somewhere waiting to be used by us. We could set up our own web server but that is beyond the scope of this course, or is it?

Part 4 - A Feather Sense + Airlift Web Server

We are going to program the Feather Sense to act as a web server. It is time to add another library to the Feather Sense lib folder. Your instructor will provide the following library:

- **adafruit_wsgi**

Copy the library to the lib folder on the Feather Sense. Create a new Mu editor tab and copy the following code. Save the code to the Feather as code.py.

```
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_esp32spi.adafruit_esp32spi_wifimanager as wifimanager
import adafruit_esp32spi.adafruit_esp32spi_wsgiserver as server
from adafruit_wsgi.wsgi_app import WSGIApp

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise



# AirLift Featherwing
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(
    spi, esp32_cs, esp32_ready, esp32_reset
)

# Use below for Most Boards
pixel = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)

## If you want to connect to wifi with secrets:
wifi = wifimanager.ESPSPI_WiFiManager(esp, secrets, pixel)
wifi.connect()

web_app = WSGIApp()
```

```
@web_app.route("/led_on/<r>/<g>/<b>")
def led_on(request, r, g, b):
    print("led on!")
    pixel.fill((int(r), int(g), int(b)))
    return ("200 OK", [], "Neopixel on with RGB set to:
{},{},{}".format(r,g,b))


@web_app.route("/led_off")
def led_off(request):
    print("led off!")
    pixel.fill(0)
    return ("200 OK", [], "Neopixel off!")

# Here we setup our server, passing in our web_app as the application
server.set_interface(esp)
wsgiServer = server.WSGIServer(80, application=web_app)

#print("MAC addr actual:", [hex(i) for i in esp.MAC_address_actual])

print("*"*50)
print("\tAirlift simple web server!")
print("*"*50)
print("open this IP in your browser: ", esp.pretty_ip(esp.ip_address))
print("*"*50)

# Start the server
wsgiServer.start()
while True:
    # Our main loop where we have the server poll for incoming requests
    try:
        wsgiServer.update_poll()
        # Could do any other background tasks here, like reading sensors
    except (ValueError, RuntimeError) as e:
        print("Failed to update server, restarting ESP32\n", e)
        wifi.reset()
        continue
```

In the Mu serial console, you should see the IP address of your new web server. The neopixel on the Feather Sense should be green when the web server starts. Open a browser on the computer or your phone and enter the IP address followed by “/led_off”. What happens?

Enter the IP address followed by “/led_on/255/0/0”. What happens?

Remote control over HTTP is pretty simple to do!

Study the code to understand what it is doing. Create the option “/blink” which will blink the neopixel five times.

Show your instructor your working code.

Save the code as lastname_lab9_part4.py. Add header information and more inline comments as necessary.

Part 5 - A Feather Sense + Airlift Web Server - getting sensor values

In part 4 you used HTTP to remotely control the Feather Sense. The Feather Sense has lots of available sensors. In part 5 you will use HTTP to retrieve sensor values from the Feather Sense.

Start with your code from part 4. Add the option “/temp” to the web server. When the web server receives this request, it should respond with the current temperature in degrees Fahrenheit from the bmp280 temperature sensor.

Show your instructor your working code.

Save the code as lastname_lab9_part5.py. Add header information and more inline comments as necessary.

Part 6 - A Feather Sense + Airlift Web Server - serving up an HTML page

In part 6 the Feather web server is going to serve an html webpage with some nice formatting and a few buttons. Starting with your code from part5, add the following function definition to your file:

```
def web_page():
    html = """<html><head><title>Feather Sense Web Server</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" href="data:,"><style>body { text-align: center; font-family:
"Trebuchet MS", Arial;}
    table { border-collapse: collapse; margin-left:auto; margin-right:auto; }
    th { padding: 12px; background-color: #0043af; color: white; }
    tr { border: 1px solid #ddd; padding: 12px; }
    tr:hover { background-color: #bcbcbc; }
    td { border: none; padding: 12px; }
    .sensor { color:white; font-weight: bold; background-color: #bcbcbc; padding: 1px;
    </style></head><body><h1>Feather Sense BMP280</h1>
    <table><tr><th>MEASUREMENT</th><th>VALUE</th></tr>
    <tr><td>Temp. Celsius</td><td><span class="sensor">""" +
str(round(bmp280.temperature, 2)) + """ C</span></td></tr>
    <tr><td>Temp. Fahrenheit</td><td><span class="sensor">""" +
str(round((bmp280.temperature) * (9/5) + 32, 2))  + """ F</span></td></tr>
    </table>
    <form method = "GET" action="/led_on/255/0/0">
            <button name = "LED" value = "/led_on/255/0/0" type="submit"
style="height:50px;width:100px">LED ON </button>
        </form>
        <form method = "GET" action="/led_off">
            <button name = "LED" value = "/led_off" type="submit"
style="height:50px;width:100px">LED OFF </button>
        </form>
        </body></html>"""
    return html
```

The function web_page() does nothing more than define a variable, html. This is the html code which will be sent when requested by a browser. If you look closely, right in the middle of the html, the bmp280 sensor is accessed to get the temperature value.

Add the following new web_app.route to your code:

```
@web_app.route("/page")
def page(request):  # pylint: disable=unused-argument
    print("page request received")
    return ("200 OK", [("Content-Type","text/html")], web_page())
```

Save the modified file to the Feather Sense as code.py. Try to access the new route, “/page” and see what happens. (Be patient…)

The Feather Sense can serve actual web pages too but it is pretty slow with our simple web server configuration.

Click on the “LED ON” button. What happens?

Go back to the “page” and click on the “LED OFF” button. What happens?

It is possible to have remote control and data retrieval combined on a single web page. The ”html” could be simplified or made more complicated. By following the example shown here in part 6 you should be able to create your own custom web interface if desired.

Use your pattern recognition abilities to modify the web_page() function and add two more rows to the table. Display the current barometric pressure on one row and display the current humidity on the second.

Show your instructor your working code.

Save the code as lastname_lab9_part6.py. Modify header information and add more inline comments as necessary.